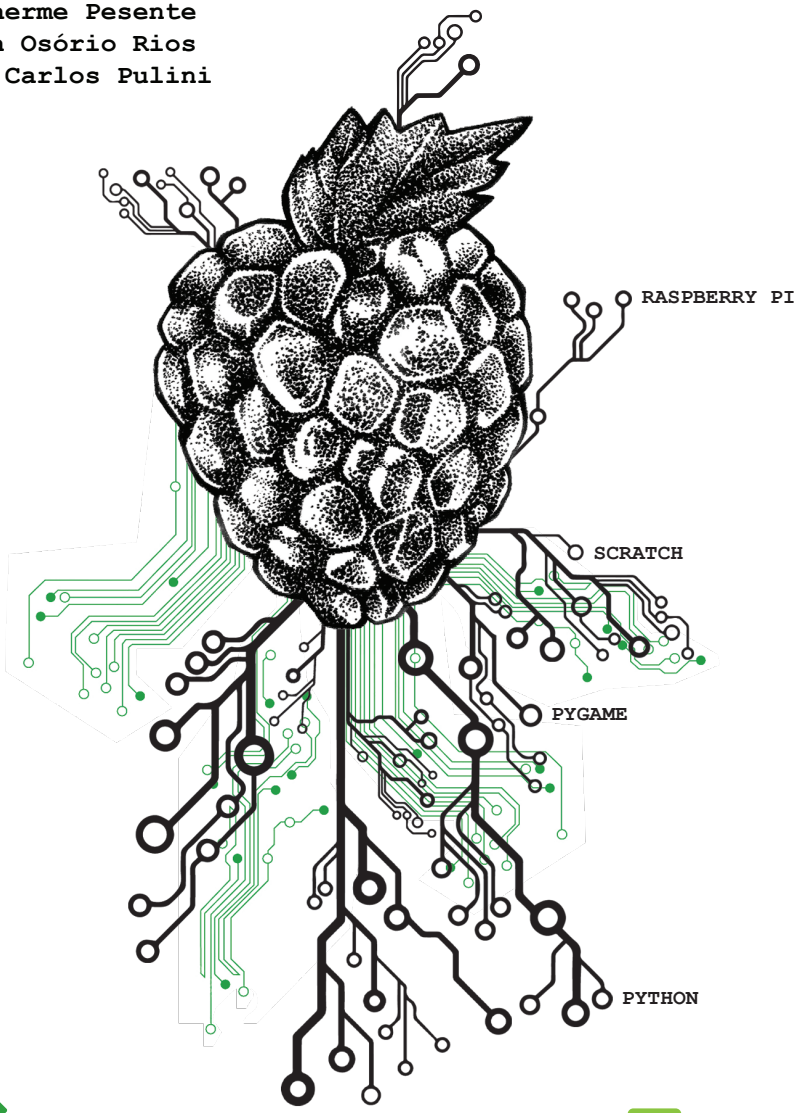
PROGRAMAÇÃO PARA LEIGOS COM

RASPBERRY PI

Elivelto Ebermam
Guilherme Pesente
Renan Osório Rios
Igor Carlos Pulini



Edifes
Editora do Ifes

editoraIFPB

Elivelto Ebermam
Guilherme Moraes Pesente
Renan Osório Rios
Igor Carlos Pulini

PROGRAMAÇÃO PARA LEIGOS COM
RASPBERRY PI

Edifes
Editora do Ifes

João Pessoa, 2017.

Capa: Wanessa Paiva Sobral
Revisão de texto: Antonio Jonas Pinotti
Colaboração: Thalyson da Silva Cordeiro

Dados Internacionais de Catalogação na Publicação
Bibliotecária Rossanna dos Santos Santana Rubim – CRB6- ES 403

9964 Programação para leigos com Raspberry Pi / Elivelto Ebermam... [et al.]. – Vitória, ES : Edifes ; João Pessoa, PB : Editora IFPB, 2017.
290 p. : il. ; 21 cm.

Inclui bibliografia.
ISBN 97885_____(broch.).
ISBN 97885_____(e-book).

1. Raspberry Pi (Computador) – Programação 2. Microcomputadores. I. Título.

CDD 22 – 005.1

Editora do Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo
Rua Barão de Mauá, nº 30 – Jucutuquara
29040-860 – Vitória – ES
www.edifes.ifes.edu.br
editora@ifes.edu.br

Editora do Instituto Federal de Educação, Ciência e Tecnologia do IFPB
Av. João da Mata, 256 – Jaguaribe
58015-020 – João Pessoa – PB
editora.ifpb.edu.br
editora@ifpb.edu.br

Sumário

Dedico este livro a todos que me apoiaram e a meus pais, fontes da minha inspiração, especialmente minha mãe, Rosimar, que me incentivou e não deixou que eu perdesse o foco.

Guilherme Moraes Pesente

Dedico este livro a minha namorada, Sheila, meus pais, Avelino e Evani, minha irmã, Elciele, minha avó, Otília, e a meu tio, Arlindo, que sempre me incentivaram e apoiaram.

Elivelto Ebermam

AGRADECIMENTOS

Primeiramente agradecemos a Deus por toda inspiração nos dada. Agradecemos também ao nosso orientador, Prof. Dr. Renan Osório Rios, e ao nosso coorientador, Prof. MSc. Igor Carlos Pulini. Agradecemos ao Prof. MSc. Antonio Jonas Pinotti pela revisão ortográfica e ao aluno Thallyson Cordeiro pela arte desenvolvida.

Ao finalizar este livro, esperamos que todos os que o utilizem consigam de forma simples e prática compreender, aprender e se apaixonar por programação e desenolvimento de Projetos/Softwares. Que através dos softwares Scratch, Python, Pygame e o Hardware Raspberry Pi, vocês possam compreender os conceitos da computação e que os exemplos demonstrados sejam capazes de moldar o raciocínio lógico dentro de cada pessoa.

Todo o projeto do desenvolvimento do livro foi feito no Ifes (Instituto Federal do Espírito Santo) – Campus Colatina. Contamos com o apoio da instituição, dos professores, da COINFO (Coordenadoria de Informática) e da Fundação de Amparo à Pesquisa e Inovação (FAPES). Todos se esforçaram ao máximo para que este projeto fosse executado com sucesso. Foi disponibilizado o Laboratório LIA (Laboratório de Informática Aplicada), que hoje conta com diversos projetos em desenvolvimento e que possui uma página (`http://col.ifes.edu.br/lia/`) com todos os trabalhos descritos e seus respectivos pesquisadores e orientadores.

Com as linguagens de programação Scratch, Python e Pygame, acreditamos que os alunos que utilizarem este livro poderão, de forma clara e prática, aprender os conceitos básicos

da programação, como comandos de decisão, repetição, entre outros.

Por fim, gostaríamos de agradecer a todos os envolvidos no projeto.

PREFÁCIO

Quando vemos um jogo de última geração nos perguntamos: como ele é feito? Bom, além de todo o design gráfico presente, alguém (normalmente uma equipe) teve que programar todas as ações do jogo. Claro que essa tarefa é complexa, mas nada nos impede de começarmos por jogos e aplicativos mais simples.

Aprender a programar pode ser uma atividade simples e divertida se iniciada da forma correta. Escolher a primeira linguagem de programação é muito importante, pois algumas não são adequadas. Elas apresentam uma sintaxe complexa e isso pode complicar o aprendizado e desestimulá-lo.

A disciplina de programação algumas vezes se torna a grande responsável pela evasão de alunos em cursos de computação. Eles acabam perdendo tempo tentando entender a sintaxe, enquanto o mais importante é aprender a lógica de programação.

A programação não deve ser vista como algo árduo, mas como uma arte. Ela pode ser utilizada em praticamente todas as áreas. Pode-se construir desde aplicações simples até simulações do mundo real e jogos complexos.

Mais do que propriamente ensinar, o objetivo deste livro é estimular o leitor a gostar de programar. Para isso foram escolhidas ferramentas simples e exemplos que estimulem o leitor a programar. Assim, o aprendizado flui naturalmente e de forma divertida.

A linguagem inicial escolhida foi Scratch. Através dela serão construídos jogos interessantes sem precisar digitar

nenhuma linha de código, apenas arrastando e soltando blocos de comandos prontos.

Na próxima fase, é mostrado como fazer pequenos aplicativos por meio de linhas de código. A linguagem utilizada é Python, que apresenta uma sintaxe simples e organizada.

Também é possível construir jogos em Python e isso é feito com auxílio da biblioteca Pygame. Será mostrado como construir a parte gráfica e lógica de vários jogos.

Assim, inicia-se a caminhada em programação. Mais do que entender como funcionam os jogos e programas, será possível construir os seus próprios.

APRESENTAÇÃO

A programação de Softwares para computadores é considerada por muitos alunos o "terror" ao ingressarem em cursos de informática e até mesmo, muitas vezes, em cursos que em dado momento utilizará a programação de computadores. Muitos alunos enxergam os cursos de informática como sendo cursos triviais, no qual não encontrarão muitas dificuldades no decorrer dos semestres.

Como em toda disciplina acadêmica, todo o conhecimento é passado gradativamente, porém, muitos alunos no decorrer dos estudos não conseguem absorver esse conhecimento nas etapas propostas. Em alguns casos, as ferramentas de ensino utilizadas no início do aprendizado do aluno são consideradas inapropriadas, pois espera-se que os alunos já possuam algum tipo de conhecimento em linguagens de programação. Gomes, Henriques e Mendes (2008) fazem essa afirmação. Os mesmos consideram as ferramentas introdutórias a programação como sendo ferramentas que possuem uma sintaxe grande e complexa, sendo mais apropriadas para ambientes que desenvolvam grandes projetos.

Outro problema encontrado é que em alguns casos os alunos entram nos cursos de informática sem possuírem sede de conhecimento. Eben Upton e Gareth Halfacree, no livro *Raspberry Pi - Manual do Usuário*, notaram que, no ano de 2005, alunos que desejavam cursar Ciência da Computação na Universidade de Cambridge tinham baixo conhecimento em linguagens mais simples, como PHP, e pouco conhecimento em linguagem de marcação, como HTML, diferentemente dos alunos que concorriam a uma vaga nos anos de 1990. Estes possuíam conhecimento em diversas linguagens de

programação, conhecimento em hardware e muitas vezes conheciam linguagens difíceis de se trabalhar, como o Assembly.

Com o passar dos anos, muitos estudos foram feitos a respeito do assunto. Podemos encontrar diversas literaturas que indicam o uso de ferramentas de ensino lúdico como sendo essencial para o ensino da linguagem de programação de computadores nos anos iniciais para alunos dos cursos de informática ou que utilizarão linguagem de programação em diversos outros cursos. Dessa forma, surgiu a seguinte pergunta: como podemos contribuir no aprendizado dos alunos nas disciplinas de programação nos anos iniciais?

Seymour Papert defende a teoria do construcionismo, para a qual cada indivíduo é capaz de construir o próprio conhecimento. Tomando como base a teoria de Papert, resolvemos desenvolver este livro utilizando as linguagens de programação Scratch, Python e sua biblioteca Pygame, além da placa lógica Raspberry Pi. O livro propõe o ensino de programação de computadores de uma forma mais simples e clara, tendo como apoio as linguagens de programação citadas acima. Essas linguagens podem ser utilizadas através da placa lógica Raspberry Pi ou até mesmo em um computador pessoal. Tais linguagens encaixam no conceito de Software Livre, ou seja, não é preciso pagar para utilizar e o resultado pode ser copiado, distribuído, alterado, melhorado e executado sem que haja nenhuma restrição.

O livro foi dividido em 4 (quatro) capítulos, tendo como objetivo a construção gradativa e sequencial do conhecimento do aluno. O primeiro capítulo, “Raspberry Pi”, estuda a placa lógica Raspberry Pi. Essa placa foi desenvolvida para que os alunos pudessem conhecer melhor o hardware de um

computador. Através da mesma é possível desenvolver diversos projetos, desde um videogame até um cluster (para isso precisará de diversas placas lógica Raspberry Pi). Diferentemente de um computador que possua um gabinete e mais barata que um notebook, essa placa precisa apenas de alguns periféricos para que a mesma funcione perfeitamente.

No segundo capítulo, "Scratch", daremos início ao aprendizado em programação. Esse capítulo foca no desenvolvimento da lógica de programação, dessa forma, o aluno utilizará uma linguagem para desenvolvimento de projetos lúdicos através de uma plataforma que utiliza comandos prontos de arrastar e soltar. Com isso, o aluno construirá todos os comandos do seu projeto como se estivesse montando um brinquedo LEGO.

A linguagem Scratch é indicada para o início ao aprendizado em programação justamente por possuir os comandos prontos, divididos por blocos (Movimento, Som, Eventos, Controle, Aparência etc.). Isso tornará fácil o aprendizado do aluno, pois ele terá em sua tela todos os comandos necessários para o desenvolvimento do seu projeto, além de ver de uma forma bem ilustrativa seu projeto sendo desenvolvido e executado lado a lado com os comandos.

Já no terceiro capítulo, "Python", será notada a primeira diferença no desenvolvimento dos projetos de programação: os alunos passarão a utilizar uma linguagem de programação que precisa que os códigos sejam escritos através de linhas de comandos. O Python está sendo indicado como a segunda linguagem de programação, visto que os alunos já despertaram a lógica de programação de computadores através da linguagem Scratch.

Para o desenvolvimento de projetos através da linguagem Python, deverá ser utilizado o Shell, que é o local onde poderemos escrever as linhas de comandos na linguagem Python. Utilizando o Python, os alunos podem começar a se familiarizar com uma linguagem cujos comandos devem ser escritos, pois a grande maioria das linguagens de programação dependem de comandos escritos, diferente do Scratch, no qual utilizamos comandos prontos de arrastar e soltar.

O Python é uma linguagem bem desenvolvida. Para que um projeto possa ser rodado corretamente, os estudantes, além de colocar os comandos corretos, terão que saber indentar (colocar os espaçamentos corretos) o programa, contribuindo assim para que seja criado um programa mais limpo e fácil de entender.

O nosso quarto e último capítulo, "Pygame", traz uma extensão do Python, sua biblioteca Pygame. Após o aluno despertar sua lógica de programação no Scratch, aprender a escrever os códigos através de linhas de comandos e desenvolver um projeto com uma indentação correta, o aluno poderá utilizar uma biblioteca muito rica em funções para o desenvolvimento de jogos lúdicos através de linhas de comandos.

O Pygame encerra este livro com chave de ouro, pois, com o passar do tempo e com um melhor conhecimento de linguagens de programação, o aluno se sentirá motivado para desenvolver um projeto mais rico em detalhes, tornando seu trabalho mais complexo.

Todo este material foi desenvolvido com o objetivo de transmitir o conhecimento em linguagem de programação. Muitas ferramentas foram estudadas e encontramos nas linguagens Scratch, Python e Pygame o melhor para que os

alunos possam aprender sobre o fantástico mundo da programação de computadores.

Bons estudos!

RASPBERRY PI - INTRODUÇÃO

Em 2006, Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft resolveram criar um computador pequeno e acessível para crianças no laboratório da University of Cambridge, na Inglaterra. Eben Upton, diretor de Estudos em Ciência da Computação na universidade, havia observado que os alunos que se candidatavam a participar do laboratório de Ciências da Computação da Universidade não apresentavam as mesmas habilidades e domínio das máquinas que tinham os alunos da década de 1990. Naquela época, jovens de 17 anos que desejavam cursar essas disciplinas já chegavam à faculdade com conhecimento de linguagens de programação e do funcionamento do hardware; alguns até trabalhavam com a linguagem Assembly[1].

O primeiro protótipo do pequeno computador surgiu na mesa da cozinha da casa de Eben. Ele e seus amigos começaram a soldar, em uma protoboard[2] com um chip Atmel[3], alguns

[1] Assembly: linguagem próxima ao código de máquina, utilizada para programar microprocessadores e microcontroladores.

[2] Protoboard: também conhecida como Matriz de Contatos, é uma placa com centenas de furos para alojar componentes eletrônicos, utilizada para fazer testes de projetos.

[3] Chip Atmel: microcontrolador de 8 bits, o primeiro a utilizar memória flash.

outros chips baratos de microcontroladores[4] para monitorar um aparelho de TV. O projeto contava com apenas 512 K de memória RAM, atingindo poucos MIPS[5] de processamento.

O grande desafio era tornar esse pequeno computador atrativo aos olhos das crianças, já acostumadas, na época, com jogos eletrônicos sofisticados e iPads.

Tempos depois, discutindo sobre o projeto e o estado geral do ensino da computação, Eben Upton e seus companheiros de laboratório, Rob Mullins, Alan Mycroft, Jack Lang, Pete Lomas e David Braben resolveram criar a Raspberry Pi Foundation. O nome Raspberry Pi foi escolhido em equipe. "Raspberry" é a fruta framboesa; a escolha seguia a tradição de colocar nome de frutas em empresas e em computadores – Apple, Tangerine, Apricot... "PI" é uma abreviação de Python, a linguagem de programação mais indicada para o aprendizado em programação no Raspberry Pi.

A Raspberry Pi Foundation passou logo a contar com vários colaboradores e voluntários. A sua equipe original possuia figuras proeminentes em suas áreas de interesse:

- Rob Mullins, professor da Universidade de Cambridge, dedica-se à arquitetura de computadores,

[4] Microcontroladores: microprocessadores que podem ser programados para diversas funções da automação: controle de estado, de fluxo e de periféricos.

[5] MIPS – milhões de instruções por segundo: uma das medidas utilizadas para comparar o desempenho de computadores.

design VLSI[6], redes on-chip de interconexão, chips multiprocessadores e processamento paralelo.

- Allan Mycroft, professor da Universidade de Cambridge, estuda linguagens de programação, otimização e implementação de programas. Foi cofundador da Associação Europeia de Linguagens de Programação e Sistemas, além de já ter trabalhado no AT & T Labs e Intel Research.
- Jack Lang, empresário e business angel[7], é Entrepreneur in Residence e Fellow na Cambridge Judge Business School, além de ter sido fundador da Electronic Share Information Ltd.
- Pete Lomas, diretor de Engenharia da Norcott Technologies e ex-professor de engenharia da computação da The University of Manchester, foi o responsável pelo design de hardware e implementação do Raspberry Pi.
- David Braben, CEO da Frontier Developments, uma grande empresa desenvolvedora de jogos, foi o responsável pelo design gráfico do Raspberry Pi.

Essa foi a equipe responsável pelo desenvolvimento desse pequeno computador que hoje ajuda a diversas crianças,

---

[6] VLSI: processo de criação de circuitos integrados por combinação de milhares de transistores em um único chip.

[7] Business Angel: investidor que arrisca seu capital, financiando pequenas empresas de ponta, em troca de participação na empresa ou de retorno financeiro; hoje existem várias associações desse tipo de investidor espalhadas pelo mundo.

adolescentes, jovens e adultos a aprenderem a programar e estenderem seus conhecimentos de hardware e software.

CONHECENDO O EQUIPAMENTO

O Raspberry Pi logo se tornou famoso pelas suas duas principais características: o tamanho (tem as dimensões de um cartão de crédito) e seu baixo custo (US$ 35). Ao ser anunciado em 2012, já superava as expectativas de venda por conter essas características incomuns; como poderia uma placa de tão pequena se tornar um computador?

O Raspberry Pi prometia ser tão promissor para o ensino de lógica de programação, de hardware e de software, que a poderosa Google doou cerca de 15 mil Raspberry Pi para escolas carentes do Reino Unido a fim de incentivar o surgimento de futuros possíveis programadores e desenvolvedores.

O Raspberry Pi, mesmo sendo pequeno e possuindo configurações limitadas em relação aos computadores, iPads e notebooks, não se intimida quando o assunto é "O que posso fazer com meu Raspberry Pi?". Através desse pequeno dispositivo conseguimos navegar pela web, assistir a vídeos, trabalhar com robótica e até mesmo criar clusters[8], por um custo muito menor em relação a outros computadores.

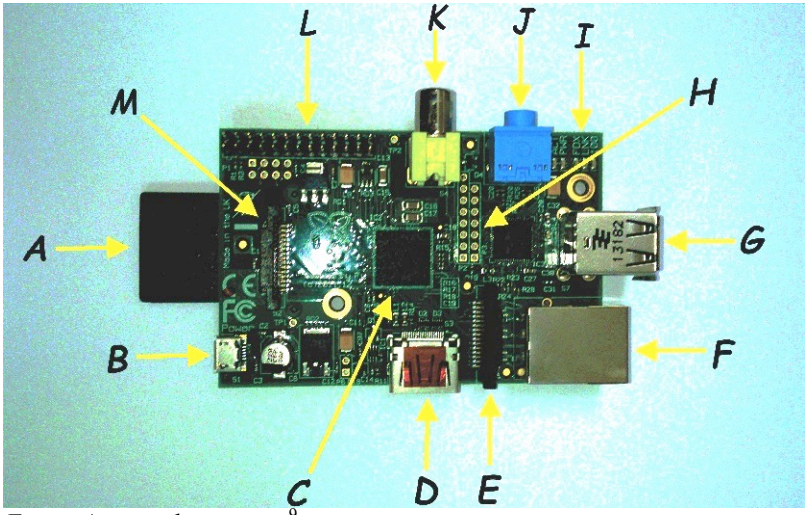
[8] Cluster: conjunto de computadores processando uma mesma aplicação.

Algumas linguagens de programação acompanham o Raspberry: Scratch, Python e Pygames. O Scratch é uma ferramenta desenvolvida no MIT para o ensino inicial de lógica de programação de forma lúdica. Python é uma linguagem mais avançada, com a qual o estudante já pode desenvolver projetos mais ambiciosos, como robôs e clusters. Pygames é uma biblioteca de rotinas em Python que trabalha com orientação a objetos desenvolvida para facilitar a criação de jogos. Além dessas, podem ser ainda utilizadas outras linguagens: C, Ruby, Java e Perl.

O sistema operacional (SO) integrado ao Raspberry Pi é o Raspbian, variante de uma distribuição Debian do Linux, disponibilizado para download no site do Raspberry Pi (http://www.raspberrypi.org/downloads). A escolha por um sistema GNU de distribuição Linux, já que se trata de código aberto (open source) e grátis, permite que a placa seja mais barata, induz à programação e melhoria do SO por parte de colaboradores e incita os alunos a um maior envolvimento com hardware e software. Entretanto, podem ser usados outros SO, como se verá no tópico **SISTEMA OPERACIONAL**.

A placa possui várias portas, comuns em computadores, para áudio, vídeo e dados: HDMI, USB, Ethernet e GPIO. Matt Richardson e Shawn Wallace, no livro *Primeiros Passos com o Raspberry Pi*, apresentam os seus principais componentes, mostrados na Figura 1.

Figura 1 – Placa Raspberry Pi B



Fonte: Acervo dos autores[9].

A. **Slot para cartão de memória Micro SD (Secure Digital[10])**: O Raspbian e os dados que você precisar armazenar estarão gravados nesse cartão, que pode ter até 64 GB de capacidade.

B. **Fonte de alimentação (entrada de energia)**: O Raspberry não trabalha com interruptores de alimentação de energia. Para o seu funcionamento é utilizado um

[9] Da Figura 1 em diante, as imagens que não apresentarem fonte e referência são do acervo dos próprios autores.

[10] Cartão SD (*Secure Digital Card*): utilizado para armazenar arquivos como imagens, músicas, vídeos, documentos, entre outros.

cabo USB de 5v, como o de um telefone celular, que irá entrar em uma porta USB; essa entrada só funciona para receber a carga que irá ligar o Raspberry, não é uma porta USB adicional. A escolha do cabo USB para levar energia até o Pi se deve ao seu baixo custo e à facilidade de ser encontrado.

C. **Processador**: O coração do Raspberry Pi bate com o mesmo processador que encontramos no iPhone 3G[11] e no Kindle 2[12], um processador de 700 MHz de 32 bits construído sobre a arquitetura ARM11[13]. Os chips ARM possuem variações de arquitetura com diferentes núcleos; dessa forma, temos também variações de capacidade e preço. O segundo lote do Raspberry Model B possuía 512 MB de memória RAM, enquanto o modelo A tinha 256 MB.

D. **Porta HDMI (High-Definition Multimedia Interface)**: Por meio da porta HDMI conseguimos transmitir, com alta qualidade, áudio e vídeo por um só cabo até um monitor. O Raspberry Pi suporta cerca de catorze tipo de resoluções de vídeo. Através de adaptadores externos

---

[11] iPhone 3G: smartphone desenvolvido pela Apple Inc., lançado em 11 de julho de 2008.

[12] Kindle 2: leitor digital desenvolvido pela Amazon, lançado em 9 de fevereiro de 2009.

[13] Arquitetura ARM: arquitetura de processadores de 32 bits utilizados em sistemas embarcados (sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla).

pode-se converter vídeo em DVI e jogar a imagem para monitores de modelo antigo.

E. Conector de Interface Serial para Câmera: Por meio dessa porta podemos conectar um cabo serial de câmera e assim transmitir a imagem para um monitor.

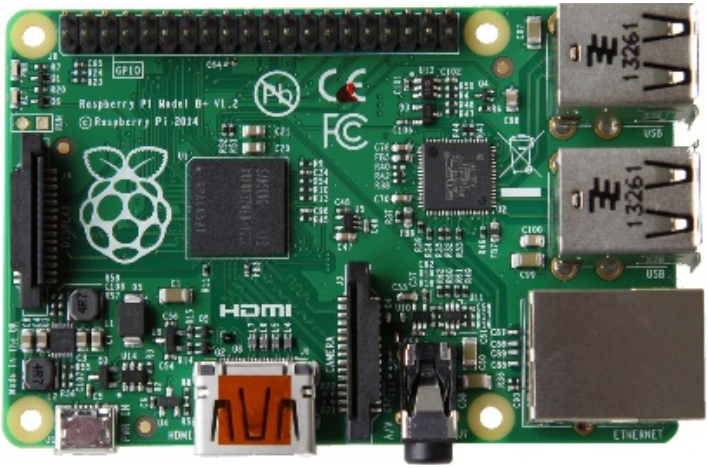
F. Porta Ethernet: Diferentemente do modelo A, o modelo B do Raspberry Pi possui porta Ethernet[14] para o padrão RJ-45[15]. Pode-se também utilizar redes Wi-Fi, mas para isso devemos utilizar uma das portas USB e nela conectar o "Dongle Wi-Pi (Wireless Internet Platform for Interoperability)".

G. Portas USB (Universal Serial Bus): O modelo A possuía apenas uma porta USB; o modelo B possui duas portas UBS 2.0; a versão mais atual, o modelo B+, possui quatro portas USB 2.0. A quantidade de portas USB ainda pode ser expandida com um hub USB para a utilização de mais periféricos. A Figura 2 mostra o modelo B+.

[14] Ethernet: arquitetura de interconexão para redes locais.

[15] Padrão RJ-45: padrão de conector para cabos de rede.

Figura 2 - Placa Raspberry Pi B+



Fonte: Raspberry Pi (www.raspberrypi.org).

H. **Conectores P2 e P3**: Essas duas linhas perfuradas na placa são os conectores JTAG (Joint Test Action Group), utilizados para testes de chips Broadcom (P2) e o de rede LAN9512 (P3). Por serem de natureza proprietária, esses conectores dificilmente serão utilizados em seus projetos.

I. **LED:** O status de funcionamento da placa é mostrado em cinco LEDs, cujos significados estão detalhados na Tabela 1.

Tabela 1 – LEDs.

LED	Cor	Descrição
ACT	Verde	Acende quando o cartão SD é acessado.
PWR	Vermelho	Conectado à alimentação de 3,3 V.
FDX	Verde	ON (ligado) se o adaptador de rede é full-duplex.
LNK	Verde	Luz indicando atividade de rede.
100	Amarelo	ON (ligado) se a conexão de rede for de 100Mbps.

Fonte: Adaptado de Matt Richardson e Shawn Wallace (2013).

J. Saída de áudio analógico: Destina-se aos reprodutores de áudio (amplificadores, caixas de som etc.). Essa saída possui apenas 3,5mm e conduz cargas de alta impedância. Caso utilize fone de ouvido ou alto-falante, nessa porta sem alimentação elétrica independente haverá queda na qualidade do áudio, diferentemente do cabo HDMI, que, ao transmitir o áudio para o monitor, não apresenta nenhuma perda de qualidade.

K. Saída de vídeo: Trata-se de um conector do tipo RCA para fornecer sinais de vídeo composto NTSC[16] ou PAL[17]. Essa saída de vídeo tem baixa qualidade, sendo preferível usar a porta HDMI quando for possível.

[16] NTSC: sistema de televisão analógico.

[17] PAL: padrão de codificação de cores para televisão analógica.

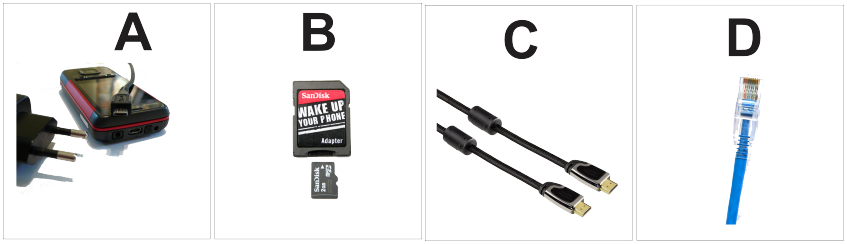
L. **Pinos de Entrada e Saída (GPIO) para uso geral**: São 26 pinos GPIO (General Purpose Input/Output) para comunicação com outros dispositivos externos; podem ser usados, por exemplo, para controle de equipamentos de automação.

M. **Conector de Interface Serial do Display (DSI)**: Esse conector foi projetado para a utilização de um cabo flat de 15 pinos para a comunicação com uma tela LCD ou OLED ou uma WebCam.

PERIFÉRICOS

Alguns periféricos são necessários para se poder trabalhar com o Raspberry Pi. Há uma lista de equipamentos compatíveis em <http://elinux.org/RPi_Verified_Peripherals>; é aconselhável verificá-la antes de adquirir qualquer periférico. A Figura 3 mostra os periféricos mínimos necessários.

Figura 3 – Perifericos necessários para o funcionamento do Raspberry Pi



Fonte: Wikimedia (wikimedia.org).

A. **Fonte de alimentação**: Deve ser utilizado um adaptador micro USB que forneça 5V e um mínimo de 700mA de corrente. Podem ser usados carregadores de celular, desde que tenham estas características. O Raspberry Pi pode até funcionar com uma fonte sem essas especificações, porém podem ocorrer falhas imprevisíveis e travamentos.

B. **Cartão Micro SD**: Para o armazenamento do SO, programas e dados do usuário deve ser utilizado um cartão SD de pelo menos 4GB de Classe 4, que possui uma velocidade de transferência de 4MB/seg. Nas versões anteriores do Raspberry Pi não era aconselhável utilizar cartões de Classe superior, principalmente os da Classe 6, que, embora com velocidade maior, apresentavam estabilidade inferior.

C. **Cabo HDMI**: Cabos HDMI são utilizados para monitores que possuem esse tipo de entrada. Caso seja necessário conectar um monitor DVI, pode ser usado um adaptador.

D. **Cabo Ethernet**: Hoje quase já não utilizamos os cabos Ethernet para comunicação com rede e com a Internet, já que as placas e redes wireless vêm ganhando a preferência dos usuários. Porém, para o melhor aproveitamento do Raspberry, é aconselhável a utilização do cabo de rede, pois os cabos são menos sujeitos a falhas e perda de velocidade, ao contrário das placas wireless, que encontram dificuldade com barreiras (paredes, árvores, janelas).

Esses são os periféricos que julgamos "obrigatórios" e que já permitem usufruir bastante dos recursos do Raspberry Pi. Entretanto, é possível turbinar seu pequeno computador com vários periféricos opcionais. A lista a seguir apresenta alguns interessantes:

- **Hub USB**: pode ser utilizado quando for necessário ligar vários dispositivos de entrada (pen drive, joystick, mouse wireless etc.), pois na sua versão B o Raspberry tem apenas duas portas USB.
- **Modulo de Câmera**: O módulo de câmera para o Raspberry Pi (2,5 cm x 2,5 cm) é capaz de captar imagens fixas a 5 MP e gravar vídeos nos modos 1080p30, 720p60 e VGA90.
- **Adaptador Wi-Fi**: Muitos dos adaptadores utilizados para localização e utilização de redes Wi-Fi funcionam com a placa Raspberry Pi. Como já foi observado, é importante verificar sua compatibilidade.
- **Periféricos para o GPIO**: Esse grupo de pinos pode ser utilizado para controlar peças de hardware em aplicações de automação. Podem ser ligados sensores, switches, LEDs, resistores[18], jumpers[19] e vários outros dispositivos.
- **Display LCD**: Com as conexões permitidas pelos pinos GPIO também podem ser ligados monitores LCDs para recebimento de mensagens e imagens gráficas.

---

[18] Resistor: conversor de energia elétrica em energia térmica.

[19] Jumpers: conectores que permitem ligar os pinos da placa.

GABINETE (CASE)

O gabinete do Raspberry também é um dos itens que devem estar em sua lista de compras. É necessário para proteger o equipamento de poeira e mantê-lo em uma posição fixa, pois os periféricos, cabos e conectores ligados à placa têm pesos diferentes, desequilibrando o conjunto, o que pode provocar mau contato e mau funcionamento.

O gabinete pode ser de acrílico, papelão, madeira etc. Você pode adquiri-lo ou fabricá-lo usando uma impressora 3D ou cortadora a laser (há vários projetos disponíveis na internet). A Figura 4 mostra um modelo transparente.

Figura 4 – Case Raspberry Pi

CUIDADOS NO MANUSEIO

Equipamentos eletrônicos exigem algum cuidado no seu manuseio. O Raspberry Pi é uma placa pequena e frágil, assim, maiores cuidados devem ser tomados ao utilizá-lo. Equipamentos de proteção são fundamentais, como o gabinete (case). O ambiente não pode ser úmido nem muito quente para não danificar os circuitos. Cuidado ao manusear os periféricos, ao plugá-los e desplugá-los, principalmente o cartão SD, que pode se corromper. Cuidados com a limpeza também são importantes, pois a poeira pode causar problemas em seus circuitos.

O processador Broadcom SoC baseado em ARM do Raspberry também não é de grande capacidade. Sendo assim, não queira executar programas muito grandes ou muito pesados; o desempenho será baixo e poderá ocorrer travamento.

SISTEMA OPERACIONAL

Como já informado no tópico **CONHECENDO O EQUIPAMENTO**, o SO presente no Raspberry Pi é o Raspbian, baseado no Debian, uma das mais antigas distribuições Linux, desenvolvida por Ian Murdock em 1993. Esse foi o sistema utilizado para o desenvolvimento dos exemplos encontrados neste livro (ver Figura 5) e é a distribuição "oficialmente recomendada" pela Pi Foundation. Entretanto, outros podem ser utilizados; a lista a seguir é retirada do livro de Richardson e Wallace (2013):

- **Linux Education Raspberry Pi da Adafruit (Occidentalis)**: A Adafruit desenvolveu uma variante baseada no Raspbian, que chamou de Occidentalis. Esse SO inclui ferramentas e drivers[20] que serão úteis para o ensino da eletrônica. Mais informações podem ser encontradas no site da Adafruit: <https://learn.adafruit.com/adafruit-raspberry-pi-educational-linux-distro>**.**
- **Arch Linux**: O sistema Arch Linux foi desenvolvido para computadores com base na arquitetura ARM; dessa forma, suporta o Raspberry Pi desde seu lançamento (https://www.archlinux.org/).
- **Xbian**: O Xbian também é uma variante baseada no Raspbian para utilização do Raspberry como um centro de mídia. Duas outras distribuições semelhantes ao Xbian e com a mesma finalidade são OpenELEC (Open Embedded Linux Entertainment Center) e Raspbmc. Podem ser encontradas nos sites: <http://xbian.org>, <http://openelec.tv/> e <http://www.raspbmc.com>.
- **QtonPi**: Essa distribuição foi baseada no Qt5, um framework multiplataforma utilizado para desenvolvimento de aplicativos. Seus aplicativos podem ser utilizados em diversos tipos de hardware e software com pouquíssima (ou nenhuma) alteração de código. Mais informações no site <http://qt-project.org/wiki/Qt-RaspberryPi>.

---

[20] Driver: programa ou conjunto de programas que permitem a comunicação entre o SO e um dispositivo de hardware ou software.

Figura 5 – Sistemas Operacionais para Raspberry Pi

Para usuários que desejam uma interface gráfica, Eben Upton e Gareth Halfacree, no livro *Raspberry Pi - Manual do Usuário*, mostram que pode ser instalado o Lightweight X11 Desktop Environment (LXDE), uma interface gráfica simpática e completa para o usuário não ter dificuldade no uso do Raspbian.

INSTALAÇÃO DO SISTEMA OPERACIONAL

O SO que escolhemos, o Raspbian, será instalado num cartão SD. São necessários dois passos para essa instalação, feitos em um outro computador qualquer: (1) formatar o cartão e (2) fazer a gravação do SO propriamente dita. Utilizaremos dois programas para isto:

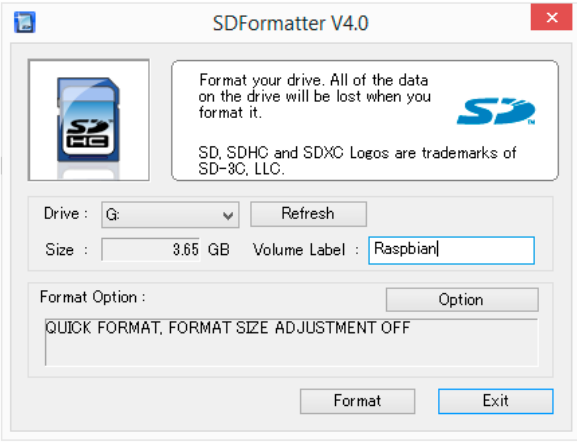
- **SDFormatter**: para a formatação do cartão SD: baixar de <https://www.sdcard.org/ downloads/>.
- **Win32 Disk Imager**: para a gravação do SO: baixar de <http://sourceforge.net/projects/win32diskimager/files/latest/download>.

SDFORMATTER

O SDFormatter é utilizado para formatar todos os tipos de cartão de memória (SD, SDHC e SDXC) e sua utilização é simples e prática. Após baixar o software, inicie a sua execução. Surge a tela inicial, mostrada na Figura 6.

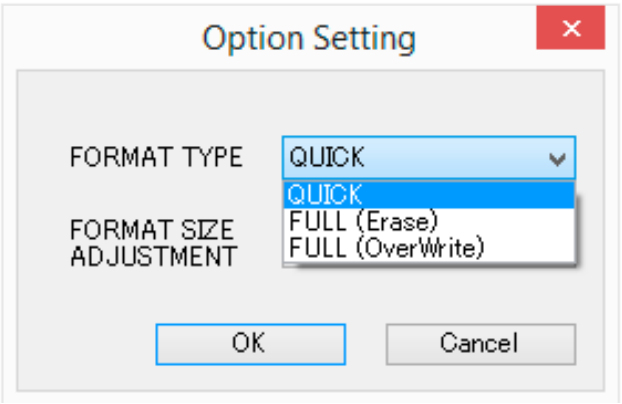
Na caixa Drive, escolha a unidade em que aparece o cartão SD; ao selecioná-la será mostrado o tamanho da unidade (Size). Digite o nome (Volume Label) que deseja dar ao cartão SD; Raspbian pode ser um bom nome.

Figura 6 – Tela inicial do SDFormatter



Em seguida, clique em Option para escolher o tipo de formatação. Na caixa Format Type são dadas três opções, mostradas na Figura 7:

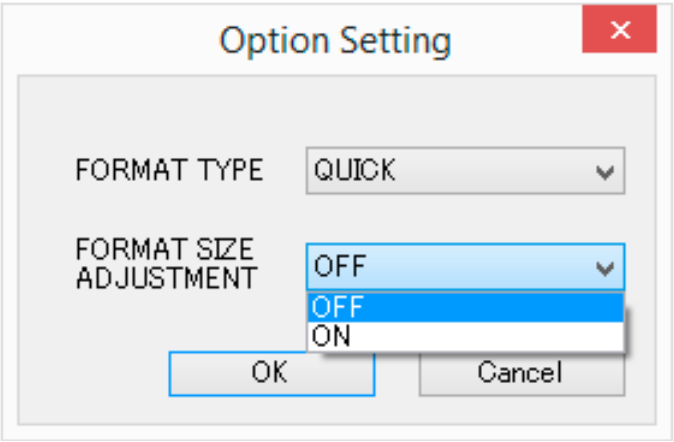
Figura 7 – Option Settings SDFormatter



- **QUICK**: Esta opção é utilizada para formatação rápida: apenas são inicializados os parâmetros do sistema de arquivos do cartão.
- **FULL (Erase)**: Formatação do tipo completo. Além de inicializar os parâmetros de sistema de arquivos no cartão, também são inicializadas as áreas de arquivos no cartão.
- **FULL (OverWrite)**: Formatação ainda mais completa (e a mais demorada) que o método anterior. Os dados já existentes no cartão são sobrescritos. Esta formatação impede a recuperação dos dados.

Depois, é necessário ajustar a formatação ao tamanho do cartão SD, na caixa Format size adjustment. São dadas duas opções, mostradas na Figura 8:

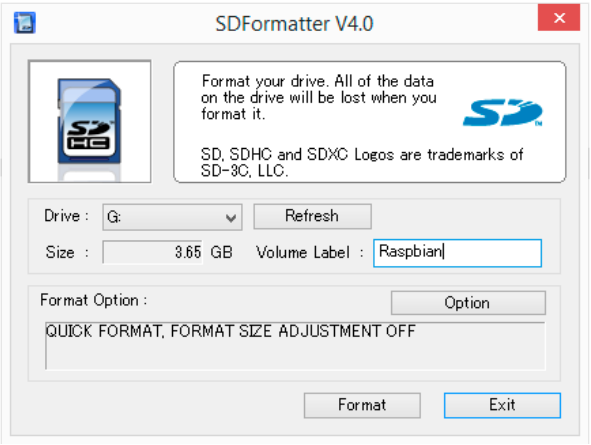
Figura 8 – Format Size Adjustment



- **ON**: Esta opção irá ajustar a capacidade do cartão a um valor múltiplo da capacidade de um cilindro; assim, ela pode reduzir a capacidade final do cartão.
- **OFF**: Esta opção é a aconselhada para a formatação no nosso caso e na maioria dos outros casos, pois ela não irá diminuir a capacidade do cartão[21].

Clicando em OK, voltamos à tela inicial do software. A caixa **Format Option** mostra as opções que selecionamos (ver Figura 9). Podemos clicar em **Format**.

Figura 9 – Format Option

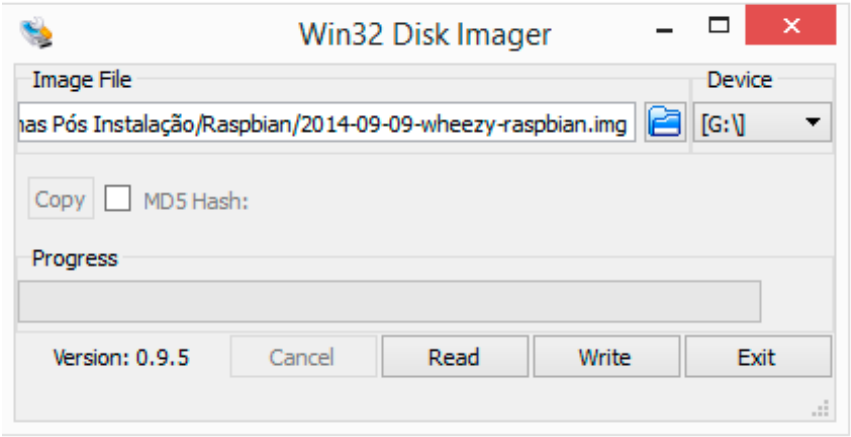


[21] Se você for utilizar o cartão para outra finalidade, é possível que tenha que utilizar a opção ON.

WIN32 DISK IMAGER

Com este software faremos a gravação do SO Raspbian em nosso cartão SD. Sua utilização também é simples. Iniciando a execução do programa, é mostrada a tela da Figura 10. Na caixa **Image File**, selecione o Raspbian. Na caixa **Device**, selecione a unidade em que se encontra o cartão SD. Em seguida, clique em **Write** e aguarde o término da gravação.

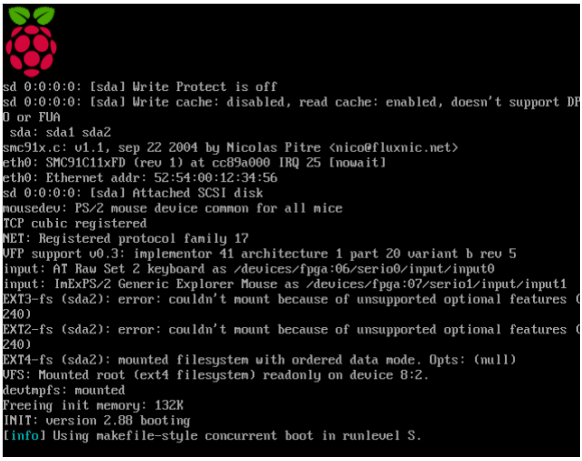
Figura 10 – Wind Disk Imager



CONFIGURANDO O RASPBERRY PI

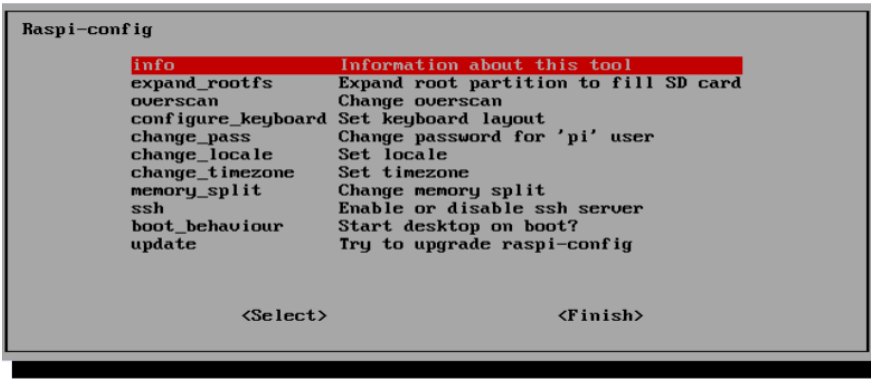
Com o sistema operacional instalado em nosso cartão SD, iremos agora inicializar nosso Raspberry Pi. Para isso conecte os periféricos que irá usar ao Raspberry (os imprescindíveis são o cartão SD e o monitor); conecte-o também a uma fonte de energia. Na inicialização, uma tela com diversas informações é carregada e a partir daí o Raspbian estará sendo carregado (ver Figura 11).

Figura 11 – Inicialização do Raspbian



Na primeira vez em que for inicializado, o Raspberry exibirá a ferramenta raspi-config (ver Figura 12). Caso precise executar a configuração outra vez, digite na linha de comando: `sudo raspi-config`. As opções do programa permitem as seguintes alterações:

Figura 12 – Raspi-config



- **Expand_rootfs**: Permite ampliar o sistema de arquivos para que possa ser utilizada toda a capacidade do cartão SD.
- **Overscan**: Quando utilizado um monitor de alta definição, há a possibilidade de o texto ultrapassar os lados da tela, desalinhando a exibição e perdendo parte da imagem. Para corrigir esse problema, ative o Overscan e altere os valores para que a imagem seja alinhada à tela. Utilize valores positivos quando a imagem sair da tela e valores negativos caso apareçam bordas pretas em torno dela.
- **Configure_keyboard**: O teclado vem pré-configurado com o layout britânico (UK). Caso queira alterar, selecione outro, de acordo com a língua que irá utilizar.
- **Change_pass**: Permite alterar a senha e o usuário.

- **Change_locale**: O Pi vem configurado com a localização do Reino Unido, utilizando seu estilo de codificação UTF-8 (en_GB.UTF-8). Selecione o país de sua localização. Exemplo: Brasil UTF-8 (pt_BR.UTF-8).
- **Chang_timezone**: Para definição de fuso horário, selecione a sua região e a cidade de sua localização.
- **Memory_split**: Permite alterar a quantidade de memória que a GPU (unidade gráfica) e a CPU irão utilizar. Deixe esta opção como padrão (default).
- **SSH**: Permite habilitar a opção de acessar o Pi via SSH (Secure Shell) remotamente pela rede. Deixar esta opção desabilitada permite economizar recursos.
- **Boot Behaviour**: Esta opção permite pular a digitação de usuário e senha na inicialização (selecionando YES). Se deixar NO, deverão ser digitados estes três comandos (o login e a senha podem ser alterados na opção anterior **Change_pass**):

  Raspberrypi login: `pi`
  Password: `raspberry`
  pi@raspberrypi ~ $ `startx`

- **Update**: Esta opção permite atualizar alguns utilitários de forma automática, estando conectado à Internet.

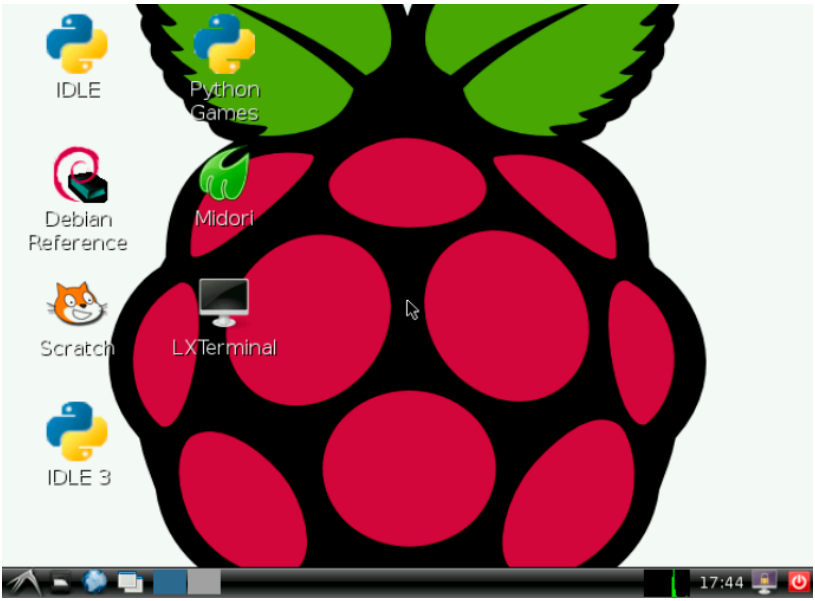
Ao finalizar suas configurações, utilize a tecla **TAB** e selecione **Finish** (concluir). Poderá ocorrer de o SO reiniciar sozinho. Se isso não acontecer e for a primeira inicialização, digite o comando:

pi@raspberrypi ~ $ `sudo reboot`

Isso irá forçar a inicialização com as novas configurações.

A Figura 13 mostra a tela inicial do Raspbian. No *desktop* já encontramos as ferramentas de programação Scratch e Python, além do navegador Midori e do LXTerminal, que serão abordados ao longo do livro.

Figura 13 – Desktop Raspbian



DESLIGANDO SEU RASPBERRY PI (SHUTTING DOWN)

O Raspberry Pi não tem um interruptor de energia. Para desligá-lo, devemos usar o botão **Logout** no canto inferior direito da tela ou executando o seguinte comando no shell[22]:

pi@raspberrypi ~ $ `sudo shutdown -h now`

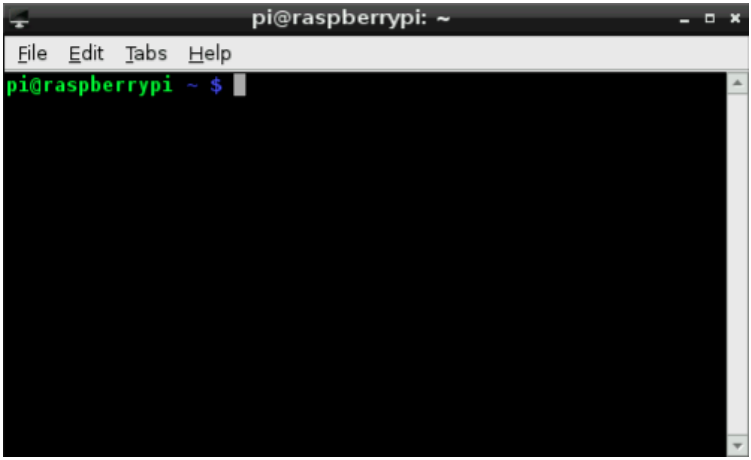
Faça o desligamento de seu Pi corretamente, pois, se somente puxar o plugue da tomada, o cartão SD pode ser corrompido e parar de funcionar.

UTILIZANDO O SHELL

O Raspbian, como todos os SOs de distribuição Linux, utiliza o shell para executar os comandos de instalação, alteração, remoção de programas etc. O programa que irá fornecer acesso ao shell é o LXTerminal, mostrado na Figura 14.

[22] Shell: programa para interação de usuário e máquina através de linhas de código.

Figura 14 - Shell



O shell guarda um histórico de comandos que pode ser muito útil para facilitar a digitação. Por exemplo, você só precisa começar a digitar um comando e apertar a tecla TAB para o restante do comando aparecer: isso é muito útil quando você está à procura de um diretório. Outra facilidade importante é poder voltar os comandos digitados: caso a execução de um comando tenha dado erro, basta apertar a tecla com a seta para cima e reaparecerá o comando anterior; deve-se então corrigir apenas a parte errada e apertar o *Enter*. Comandos anteriores podem ser repetidos apenas usando as teclas: seta para cima, seta para baixo e *Enter*.

PRINCIPAIS DIRETÓRIOS

O Raspbian monta uma estrutura padrão de diretórios (ou pastas), cujo conteúdo e finalidade são mostrados na Tabela 2.

Tabela 2 – Diretórios Raspbian

Diretório	Descrição
/	Raiz
/bin	Programas e comandos que todos os usuários podem executar
/boot	Arquivos necessários no momento da inicialização (boot)
/dev	Arquivos especiais que representam os dispositivos do sistema
/etc	Arquivos de configuração
/etc/init.d	*Scripts* para inicializar os serviços
/etc/X11	Arquivos de configuração X11
/home	Diretório pessoal dos usuários
/home/pi	Diretório pessoal para o usuário pi
/lib	Módulos ou drivers do *kernel*
/media	Pontos de montagem para mídias removíveis
/proc	Diretório virtual com informações sobre os processos em execução e o SO
/sbin	Programas para manutenção de sistema
/sys	Diretório especial no Raspberry Pi que representa os dispositivos de hardware

/tmp	Espaço para programas criarem arquivos temporários
/usr	Programas e dados utilizáveis por todos os usuários
/usr/bin	A maioria dos programas no sistema operacional reside aqui
/usr/games	Sim, jogos
/usr/lib	Bibliotecas para suportar os programas comuns
/usr/local	Software específico para esta máquina
/usr/sbin	Mais programas de administração de sistemas
/usr/share	Recursos que são compartilhados entre aplicativos, como ícones ou fontes
/usr/src	Linux é open source, aqui está o código-fonte
/var	Logs de sistema e arquivos spool
/var/backups	Cópias de backup de todos os arquivos de sistema mais importantes
/var/cache	Qualquer programa que armazena dados (como o apt-get ou um navegador web) os armazena aqui
/var/log	Todos os logs do sistema e logs de serviços individuais
/var/mail	Todos os emails de usuários são armazenados aqui se você estiver configurado para lidar com emails
/var/spool	Dados aguardando para serem processados (por exemplo, email de entrada, trabalhos de impressão)

Fonte: Adaptado de Matt Richardson e Shwan Wallace (2013).

FUNÇÕES BÁSICAS

O Shell do Raspbian possui a mesma função e os mesmos comandos de um sistema Linux. O prompt **pi@raspberrypi ~ $** mostrado a cada linha de comandos solicitada compõe-se de quatro partes com o seguinte significado:

- **pi@**: Nome do usuário que está acessando o sistema; o default é **pi**.
- **raspberrypi**: Nome do computador: o default é **raspberrypi**.
- **~**: Diretório inicial do *shell* (home).
- **$**: Final do prompt. O texto do comando será digitado à frente desse cifrão; o comando é executado ao pressionar a tecla "Enter".

Existem vários comandos, utilizados para instalar novos programas, alterar arquivos, listar diretórios etc. Para nos movimentarmos dentro da árvore de diretórios, usamos o comando **cd**. Exemplos:

Ir para o diretório /etc/calendar: **pi@raspberrypi ~ $** `cd /etc/calendar`

Retornar a um diretório anterior: **pi@raspberrypi ~ $** `cd .. ou cd ~`

Retornar ao diretório raiz (*home*): **pi@raspberrypi ~ $** `cd /`

Ao chegar a um diretório, você pode não saber se o próximo diretório está exatamente no local onde você se encontra ou dentro de outro diretório num nível abaixo. O comando **ls** permite ver o conteúdo (diretórios e arquivos) de qualquer diretório. Exemplo: Para chegar ao diretório do python, devemos passar pelos diretórios */var/lib/python*. Caso não se lembre que esse diretório se encontra dentro do diretório */lib*, você pode utilizar o comando **ls** para listar o conteúdo de cada um destes diretórios. Exemplo:

```
pi@raspberrypi ~ $ cd /var/lib/python
pi@raspberrypi ~ $ ls python
        python3.2_installed
```

Caso queira uma lista mais detalhada do conteúdo do seu diretório, basta utilizar o comando **ls –l** que ele trará, além do nome da pasta ou do arquivo, suas permissões, data de alteração e tamanho dos arquivos. Exemplo:

```
pi@raspberrypi ~ $ cd /var/lib/python
pi@raspberrypi ~ $ ls -l python
        -rw-r--r-- 1 root 0 Jul 15 2012 python3.2_installed
```

Quando utilizamos o parâmetro **–a**, estaremos listando, além dos arquivos normais, também os arquivos que estejam invisíveis para os usuários.

```
pi@raspberrypi ~ $ cd /var/lib/python
pi@raspberrypi ~ $ ls -la python
        drwxr-xr-x 2 root root 4096 Jul 15 2012.
        drwxr-xr-x 34 root root 4096 Jul 15 2012..
        -rw-r--r-- 1 root 0 Jul 15 2012 python3.2_installed
```

Para renomear arquivos ou diretórios (exceto os diretórios padrões), devemos utilizar o comando **mv**. Por exemplo, vamos utilizar o comando **touch** para criar um arquivo vazio e depois trocar seu nome:

pi@raspberrypi ~ $ `touch primeiro_arquivo`

pi@raspberrypi ~ $ `ls`

Desktop primeiro_arquivo python_games

pi@raspberrypi ~ $ `mv primeiro_arquivo arquivo_renomeado`

pi@raspberrypi ~ $ `ls`

arquivo_renomeado Desktop python_games

Para criar um novo diretório devemos utilizar o comando **mkdir**.

pi@raspberrypi ~ $ `mkdir meuDir`

pi@raspberrypi ~ $ `ls`

arquivo_renomeado Desktop meuDir python_games

Para remover um arquivo, podemos utilizar o comando **rm**. Para excluir diretórios vazios, utilizamos o comando **rmdir**. Para excluir diretórios contendo arquivos ou outros diretórios: **rm –r**; o parâmetro **-r** indica que o comando irá excluir recursivamente os arquivos do diretório.

pi@raspberrypi ~ $ `ls`

arquivo_renomeado Desktop meuDir python_games

```
pi@raspberrypi ~ $ rm arquivo_renomeado
pi@raspberrypi ~ $ ls
        Desktop        meuDir        python_games
pi@raspberrypi ~ $ rmdir meuDir/
pi@raspberrypi ~ $ ls
        Desktop        python_games
```

PERMISSÕES

O Raspbian é um SO que aceita vários usuários utilizando o mesmo computador. Para que não haja confusão, o ideal é que cada usuário possua os seus próprios arquivos e não que todos os usuários tenham acesso a todos os arquivos.

Desses usuários, apenas um possui permissão de alterar qualquer arquivo: é o root, ou super-admin. Não é aconselhável fazer `login` com a conta do `root`. Os usuários comuns, que não possuem todos os privilégios de um super-admin, podem utilizar o comando **sudo**. Esse comando permite instalar programas e fazer alterações em arquivos, sem que se esteja logado como `root`.

Cada arquivo pertence a um único usuário ou a um determinado grupo. Os comandos chow e chgrp permitem alterar o proprietário ou o grupo de um arquivo. Os arquivos também possuem um conjunto de "permissões" que indicam se o arquivo pode ser lido, sobrescrito ou executado por um usuário ou grupo. A Figura 15 mostra um exemplo destas permissões.

Figura 15 – Permissões

Fonte: Adaptado de Matt Richardson e Shwan Wallace (2013).

Para configurar as permissões de forma individual, utilizamos o comando **chmod**. A Tabela 3 relaciona os tipos de permissões que podem ser definidas.

Tabela 3 – Permissões CHMOD

Opção	Significado
u	usuário
g	Grupo
o	outros fora do grupo
a	tudo / todos
r	permissão de leitura
w	permissão de gravação
x	permissão de execução
+	adicionar permissão
-	remover permissão

Fonte: Adaptado de Matt Richardson e Shwan Wallace (2013).

CONFIGURANDO A REDE

A configuração de rede é tão simples como em qualquer distribuição Linux. Caso haja um servidor DHCP[23] na rede, o Raspberry não precisa ser configurado, pois pegará um endereço IP na rede automaticamente; porém, se sua rede possui configuração manual de endereço IP, é necessário configurá-lo antes do acesso à rede.

As informações sobre a configuração de uma rede ficam armazenadas em um arquivo chamado `interfaces`, localizado na pasta `/etc/network`; somente o usuário root pode editá-lo, pois qualquer erro nele pode impedir o acesso da sua máquina à rede. Para editar esse arquivo, digite o seguinte comando no LXTerminal:

```
sudo nano /etc/network/interfaces
```

Nesse exemplo, estamos utilizando o editor padrão do Raspbian, o nano. A Figura 16 mostra o conteúdo do arquivo a ser alterado.

[23] DHCP: Dynamic Host Configuration Protocol. O servidor DHCP atribui automaticamente um endereço IP às máquinas que entram na rede.

Figura 16 – Interface de Rede no Raspbian

Na linha `iface eth0 inet dhcp`, substitua `dhcp` por `static`. Digite depois as outras linhas. O arquivo deverá ficar assim:

iface eth0 inet static

address 192.168.0.2 //Endereço IP que sua máquina irá receber

netmask 255.255.255.0 //Máscara padrão

gateway 192.168.0.1 //Endereço IP do seu roteador ou modem ADSL.

Ao finalizar a digitação, aperte CTRL+O para salvar as alterações e depois CTRL+X para encerrar o nano e retornar ao LXTerminal. Para validar sua alteração na rede, reinicie o serviço de rede digitando o seguinte comando:

```
sudo /etc/init.d/networking restart
```

Feito isso, seu Pi está configurado para acessar a sua rede e a Internet.

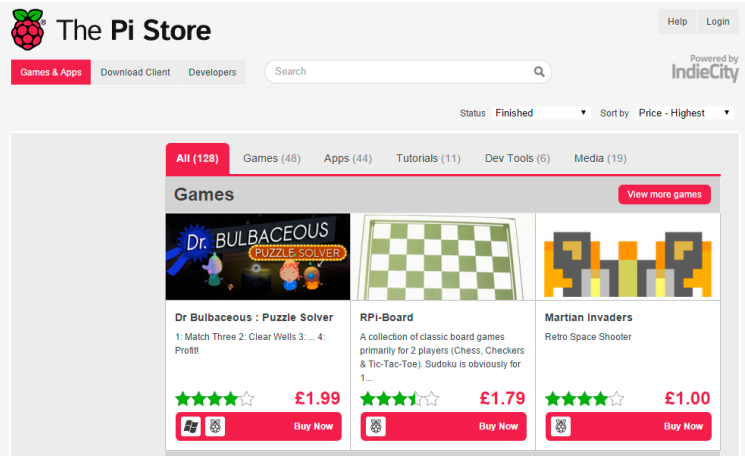
INSTALANDO SOFTWARES ADCIONAIS

Vimos que o Pi já vem com alguns softwares instalados, como o Python, Scratch e Midori. Caso deseje instalar um novo, há duas maneiras. A primeira utiliza o próprio `shell`, executando os comandos `apt-get e install`, que efetuam o download e a instalação:

pi@raspberrypi ~ $ `sudo apt-get install mc`

Outra maneira é através da loja Pi Store (`http://store.raspberrypi.com/`). É necessário efetuar um cadastro na loja para poder efetuar o download. A Figura 17 ilustra a interface inicial da Pi Store.

Figura 17 – Pi Store



Além de permitir baixar novas ferramentas, games e programas, a Pi Store também admite que você envie os seus programas para que outros usuários os utilizem.

SCRATCH

Scratch é uma plataforma de desenvolvimento com uma linguagem de programação gráfica desenvolvida pelo Media Lab do MIT (Massachusetts Institute of Technology) em 2003, baseada no LOGO, linguagem já existente também produzida pelo MIT. Sua publicação e disseminação, porém, ocorreram apenas a partir de 2007, quando o Scratch passou a ser conhecido como linguagem de programação e começou a ser utilizado por instituições de ensino.

Grande parte das linguagens de programação são baseadas em texto (instruções escritas com uma série de regras específicas); para qualquer função que tenha que executar o computador deverá ler aquele texto e processá-lo. Como exemplo, vamos mostrar o famoso *Hello World*!!! escrito em algumas linguagens conhecidas:

```
printf("Hello World!!!");                    (em C++)
frase = input("Hello World!!!")              (em Python)
System.out.print(“Hello World!!!”);          (em Java)
```

Para o iniciante em programação, aprender todas essas sintaxes pode ser muito complicado e maçante. Diferentemente dessas linguagens que utilizam linhas de comandos, o Scratch

possui linguagem gráfica com comandos prontos para "arrastar e soltar", facilitando assim o aprendizado e ajudando na construção do conhecimento. A Figura 18 mostra como fica o `Hello World!!!` no Scratch.

Figura 18 – Hello Word!

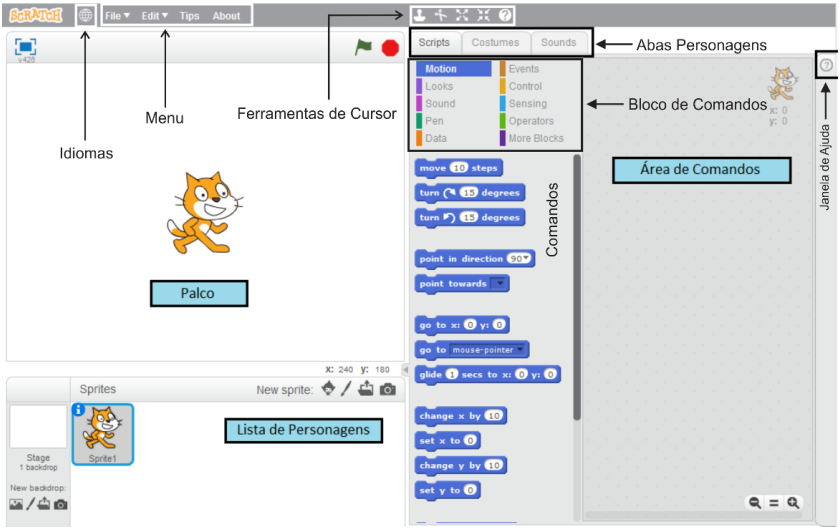
Utilizamos poucos comandos em nosso Sprite[24]: um comando de iniciar e um para mostrar a mensagem. Para cada personagem criado na plataforma Scratch, deverão ser escritos comandos para o seu movimento dentro do palco. Por exemplo: ao fazer um jogo de corrida para dois jogadores, criamos separadamente a lógica para cada personagem; a lógica criada para o primeiro não influenciará o segundo e nenhum dos dois personagens executará comandos feitos para o palco. Com isso, torna-se mais simples o entendimento do programa, pois cada personagem é um objeto programado separadamente.

O Scratch pode ser utilizado on-line ou em nosso computador. Na primeira forma, devemos acessar o site do Scratch (`http://scratch.mit.edu/`) e clicar nos links

[24] Sprite: ator ou personagem, dentro da linguagem.

`Criar` ou `Experimente`; na outra forma, podemos instalá-lo em nosso computador, baixando-o de `http://scratch.mit.edu/scratch2download/`. A tela apresentada é a mesma mostrada na Figura 19. A interface para o programador é altamente intuitiva e de fácil aprendizado. A tela é dividida em três painéis, que permitem trabalhar com nossos atores. A língua padrão é o inglês, porém no ícone em forma de globo podemos alterar para Português (Brasil).

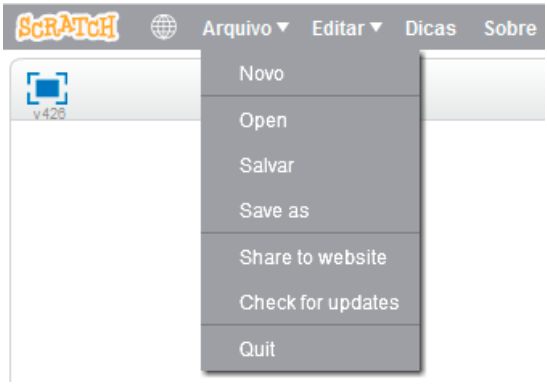
Figura 19 – Plataforma Scratch



MENU

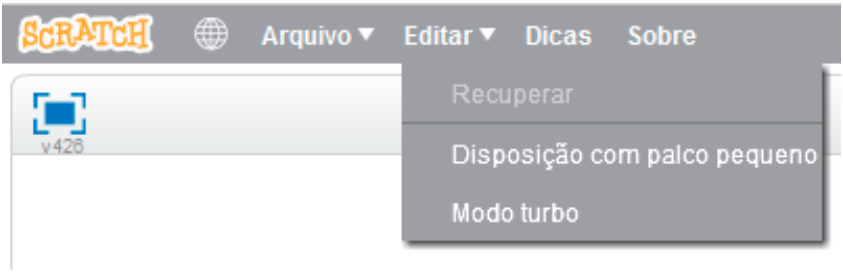
Ao lado do ícone do idioma temos o menu com quatro opções: Arquivo, Editar, Dicas e Sobre. O menu Arquivo abre os seguintes submenus (Figura 20): `Novo` (Novo projeto), `Open` (Abrir projeto existente), `Salvar` (Salvar projeto existente no formato `.sb2`, diferente da versão anterior. sb), `Save as` (Salvar novo projeto), `Share to website` (Compartilhar projeto no site do Scratch `http://scratch.mit.edu/`), `Check for updates` (Verificar atualizações da plataforma) e `Quit` (Sair da plataforma).

Figura 20 – Menu Arquivos



O menu `Editar` traz as opções (Figura 21): `Recuperar` (Recupera o ultimo bloco de código excluído do projeto), `Disposição com palco pequeno` (Diminui a área do palco e aumenta a área de código) e `Modo turbo` (Aumenta a velocidade de execução de alguns comandos).

Figura 21 – Menu Editar

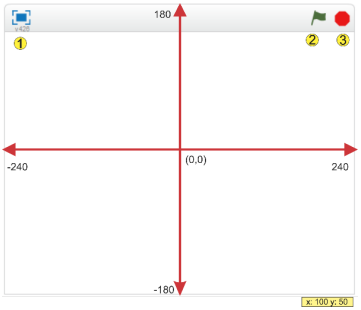


O menu *Dicas* irá abrir uma aba na lateral da tela, para que o usuário possa obter ajuda na solução de alguma dúvida. O menu *Sobre* direciona para o site do Scratch (`http://scratch.mit.edu/about/`).

PALCO

O Palco (Figura 22) é o local onde os personagens irão se movimentar e interagir. É um retângulo com as dimensões contadas em passos: 480 de largura e 360 de altura. O centro do palco é a posição x = 0 e y = 0.

Figura 22 – Palco



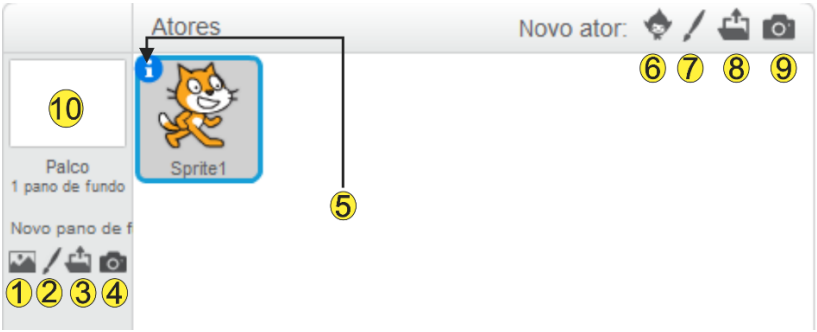
Para saber o valor de (X,Y), basta movimentar o mouse pelo palco; as coordenadas aparecerão no seu canto inferior direito. Além disto, há três ícones com a seguintes funções:

- **Ícone Tela**: opção de apresentação (1): permite expandir o palco e ocupar toda a tela com o jogo. Clicando novamente no ícone, o palco volta ao seu tamanho normal.
- Os ícones de `Bandeira verde` (2) e `Bolinha vermelha` (3) permitem que você inicie e pare o jogo a qualquer momento.

ATORES

As figuras dos personagens se posicionam na área denominada `Atores`; para nós, `Personagem` é a mesma coisa que `Ator`. Quando é criado um novo projeto, o único personagem que aparece é a do gato símbolo do Scratch, como mostra a Figura 23.

Figura 23 - Atores



A área destinada aos atores possui diversas funcionalidades (numeradas de 1 a 10 na figura) para facilitar o desenvolvimento dos projetos:

1. Escolher palcos já prontos da biblioteca Scratch.
2. Desenhar seu próprio palco.
3. Carregar cenário a partir do arquivo.
4. Tirar uma foto através da câmera do computador para utilizar como palco.
5. Obter as informações sobre o ator, clicando no botão *i* sobre sua figura.
6. Escolher um ator da biblioteca Scratch.
7. Pintar um novo ator.
8. Importar um novo ator.
9. Tirar uma foto através da câmera do computador para utilizar como ator.
10. Mostrar uma miniatura do Palco.

Para obter informações sobre o ator podemos clicar sobre a letra i como mencionado acima. Podemos ainda manuseá-los com um menu de contexto: dando um clique duplo sobre o personagem, ele ficará piscando, envolto em um retângulo de bordas azuis; clique sobre ele com o botão direito; deverá surgir um menu popup (Figura 24).

Figura 24 – Informações sobre o Ator

O menu apresenta estas opções: `info` (fornece informações do personagem), `duplicar` (permite duplicar o personagem), `apagar` (exclui o personagem), `salvar em arquivo local` (salva o personagem em seu computador) e `esconder` (retira o personagem do palco).

A barra de `ferramentas de cursor` (situada na faixa superior – ver Figura 19) é outro recurso útil quando se vai utilizar um grande número de personagens: permite duplicar, modificar tamanho ou obter informações sobre os objetos (Figura 25).

Figura 25 – Ferramentas edição do Ator

A ABA ROTEIROS

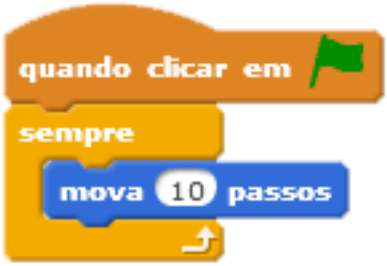
A aba Roteiros (ou Scripts) é a que permite escrever as instruções do programa. Os blocos com os comandos que podem ser utilizados estão divididos em dez categorias: Movimento (`Motion`), Aparência (`Looks`), Som (`Sound`), Caneta (`Pen`), Variáveis (`Data`), Eventos (`Events`), Controle (`Control`), Sensores (`Sensing`), Operadores (`Operators`) e mais blocos (`More Blocks`) (Figura 26).

Figura 26 – Aba de comandos (Roteiros)

Para programar o comportamento de um objeto, basta arrastar o comando desejado para a área de `comandos` (no lado direito da tela – ver Figura 19). Como exemplo, vamos utilizar três funções do nosso roteiro para fazer o gatinho andar 10 passos: Eventos, Controle e Movimento. De `Eventos`,

usamos a função que diz que toda vez que a bandeira verde for clicada o jogo começará; de `Controle`, usamos a repetição `sempre`; de `Movimento`, usamos `mova 10 passos`. O programa montado é mostrado na Figura 27.

Figura 27 – Primeiro exemplo de comandos no Scratch



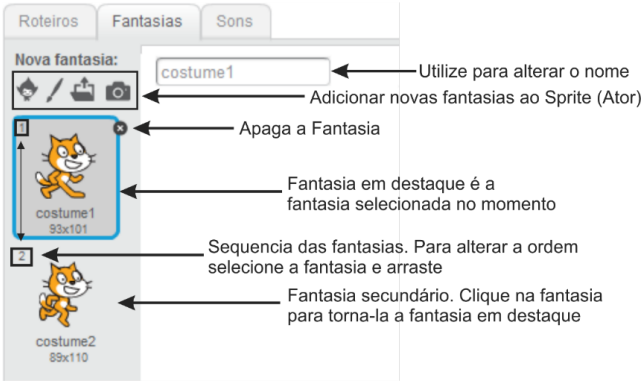
O exemplo é bem simples, pois tem apenas o intuito de mostrar a facilidade de manuseio da ferramenta: poucos comandos pré-definidos determinam o que o ator irá fazer. Com o passar do tempo virá o conhecimento de todos os comandos e dos outros recursos; com o aprimoramento da técnica, novos projetos mais complexos poderão ser desenvolvidos, envolvendo mais personagens, mais código e mais lógica de programação.

A ABA FANTASIA

A aparência de um ator pode ser alterada através da aba `Fantasias` (Figura 28). É possível alterar a fantasia de um ator já existente ou criar novas. Cada `Ator`, criado ou importado, pode ter várias `Fantasias`. Para que ele apareça

de outra forma deve-se duplicar o ator e então, modificar sua aparência ou importar novas fantasias de sua biblioteca ou de seu computador. Por exemplo: quando um personagem precisa executar vários movimentos, para cada movimento (andar, pular, abaixar etc.), deve-se criar ou importar uma nova fantasia; dessa forma os movimentos parecerão mais reais. Observe na Figura 28 que o gatinho é apresentado em dois momentos diferentes no seu caminhar.

Figura 28 – Aba Fantasias



Outras funções são disponibilizadas na aba Fantasias; observe a Figura 28. A mesma barra de ferramentas encontrada para os atores também existe aqui, só que aplicando-se à `aparência do ator`, e não ao ator. Cada aparência também tem o seu menu de contexto; clique sobre uma com o botão direito e surgirão as seguintes opções:

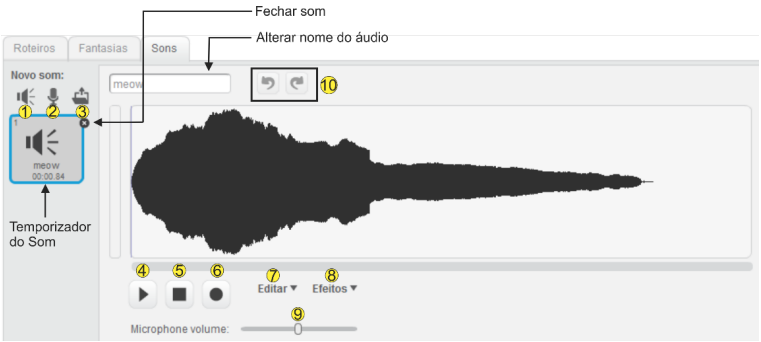
- **Duplicar**: Duplica a figura selecionada, jogando-a abaixo da última. Podemos trocá-la de posição arrastando-a para outro local; também podemos modificar sua aparência.

- **Apagar**: Apaga a figura selecionada. A aparência pode ser recuperada usando o menu `Editar` e depois `Recuperar`.
- **Salvar em arquivo local**: Salva o personagem no local desejado. Três formatos diferentes podem ser usados: `.svg`, `.png` ou `.bmp`. O default é no formato `.svg` (scalable vector graphics) e pode ser aberto com qualquer navegador.

A ABA SONS

Para deixar os programas mais divertidos, é possível fazer com que emitam sons. Isso faz com que os jogos imaginados fiquem bem mais dinâmicos e prendam a atenção do usuário. Diversos eventos podem provocar a emissão de sons: apertar uma tecla do computador, efetuar determinado movimento de um personagem, encostar um objeto em outro etc. A Figura 29 mostra a aba Sons; as funções permitidas foram numeradas na figura e estão comentadas a seguir.

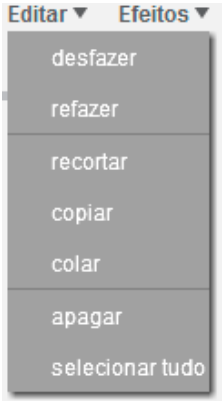
Figura 29 – Aba Sons



1. **Escolher um som da Biblioteca**: São fornecidos aproximadamente 94 sons na biblioteca padrão.
2. **Gravar um novo som**: Permite gravar um novo som na biblioteca, através de microfone.
3. **Carregar som a partir de arquivo**: Permite importar um arquivo de som. Alguns sons já são ligados a determinados personagens ou tipos de jogos, como no Street Fighter; se estiver usando um personagem conhecido é interessante usar seu som original.
4. **Botão Play**: Permite escutar o som antes de utilizá-lo em seu projeto.
5. **Botão Stop**: Permite parar a música antes de seu término.
6. **Botão Gravação**: Permite mixagem de sons: Podemos gravar em cima de um arquivo de som já existente (o som ficará gravado no início do arquivo).
7. **Editar**: Este menu abre um submenu com várias outras funcionalidades, como mostra a Figura 30.
    - **Desfazer**: Desfaz as alterações que tenham sido feitas no som.
    - **Refazer**: Refaz alterações que tenham sido desfeitas.
    - **Recortar**: Permite recortar a parte selecionada de um som; para selecionar, clique no início da faixa desejada e arraste o mouse até o ponto final desejado; depois, clique no menu `Editar/recortar`.
    - **Copiar**: Permite copiar a parte selecionada de um som; o processo é o mesmo de `Recortar`.

- **Colar**: Permite colar a parte que foi copiada ou recortada.
- **Apagar**: Permite apagar a parte selecionada de um som; o processo é o mesmo de Recortar.
- **Selecionar tudo**: Seleciona todo o som ou gravação.

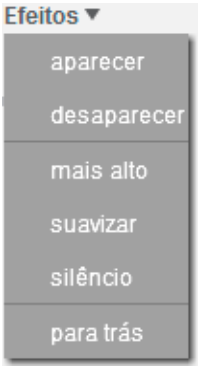
Figura 30 – Menu Editar



8. **Efeitos:** Este menu acrescenta algumas funcionalidades para a mixagem do som do ator (ver Figura 31).
    - **Aparecer**: Usando esta opção, o volume do som começa baixo e vai aumentando progressivamente até o final de seu uso.

- **Desaparecer:** Inverso da anterior. Nesta opção o volume do som começa alto e vai se esvanecendo.
- **Mais Alto**: Permite aumentar o volume.
- **Suavizar**: Permite diminuir o volume.
- **Silêncio**: Remove todo o som.
- **Para trás**: Com essa opção é possível fazer o efeito reverso do som.

Figura 31 – Menu Efeitos



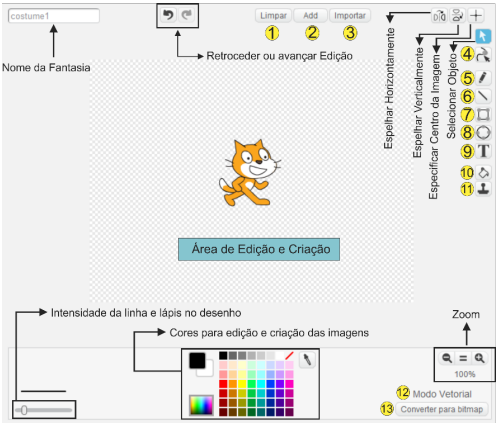
9. **Microphone volume** (volume do microfone): Permite definir o volume do som que será gravado pelo microfone.
10. **Voltar e Avançar**: Como nos editores de texto, com esta opção é possível voltar e recuperar alguma alteração feita erradamente ou avançar para uma alteração já feita.

Com essas funções disponibilizadas pela aba Som é possível fazer diversas combinações e mixagens, permitindo dotar os projetos com um fundo sonoro bem adequado a cada situação.

PAINT EDITOR

As fantasias e panos de fundo (`backdrops`) que aparecem na aba Fantasias podem ser editadas pela ferramenta `Paint Editor` (Figura 32). Não é necessário criar ou editar obrigatoriamente todas as fantasias com essa ferramenta, pois já vimos que é possível importar personagens e cenários para o palco (Figura 23/Figura 28); porém, esse editor fornecido tem vários recursos interessantes que podem ser utilizados para criar ou adaptar palcos e personagens bem ao gosto de sua criatividade. A Figura 32 mostra os recursos numerados, que são comentados a seguir.

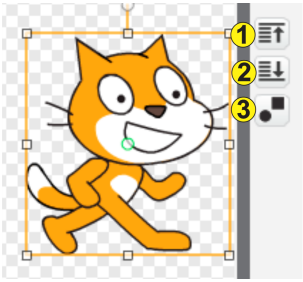
Figura 32 – Paint Editor



1. **Limpar**: Limpa toda tela, permitindo iniciar um novo desenho.
2. **Add**: Permite adicionar um personagem já existente na biblioteca.
3. **Importar**: Permite importar personagens ou imagens para o pano de fundo do palco.
4. **Remodelar**: Permite editar (remodelar) o seu ator, podendo alterar a face do personagem, o corpo, os membros.
5. **Lápis**: Permite desenhar a mão livre em seu palco ou criar seu próprio personagem, de acordo com sua criatividade.
6. **Linha**: Permite traçar linhas de forma mais precisa, sem muitas imperfeições.
7. **Retângulo**: Para desenhar objetos em forma de retângulo ou quadrado.
8. **Elipse**: Outra ferramenta para auxiliar no desenho de formas geométricas.
9. **Texto**: Permite escrever textos sobre o palco ou sobre algum objeto em uso.
10. **Colorir uma forma**: Conhecida como "balde de tinta" em alguns editores de imagem, essa opção permite preencher uma figura ou parte dela, agilizando o processo de coloração dos quadros.
11. **Duplicar**: Permite clicar sobre uma forma ou imagem para duplicá-la.
12. **Modo Vetorial**: Modo de edição do Scratch 2.0.
13. **Converter para bitmap**: Modo de edição do Scratch 1.4.

Essas são as funções oferecidas pelo `Paint Editor`. Porém, existem ainda três opções que podem ser utilizadas quando se seleciona um personagem ou objeto (Figura 33):

Figura 33 – Ator selecionado para edição



1. **Avançar uma Camada**: Essa opção faz aparecer (traz para frente) alguma camada que esteja escondida em seu objeto.
2. **Descer um nível**: Essa opção permite esconder (leva para trás) parte de uma imagem.
3. **Desagrupar**: Permite, com um clique, dividir a imagem em várias partes sem desordenar o objeto. Por exemplo: a imagem do Gato, ao clicar nessa opção, fica toda desagrupada; assim pode ser movimentada só a boca ou só o nariz. Só não esqueça que, como todas as partes do corpo estão desagrupadas, se você selecionar somente o olho, somente ele se moverá.

APRENDENDO A PROGRAMAR COM SCRATCH

Vamos iniciar nossa prática de programação. Desenvolveremos três projetos com níveis crescentes de complexidade, de forma que você possa ir adquirindo expertise e possa desenvolver novos projetos cada vez mais sofisticados. Você pode consultar também milhões de exemplos no site do Scratch (https://scratch.mit.edu/), além de poder publicar as suas criações. Na época da elaboração deste livro, o site compartilhava 10.000.233 projetos.

DESENVOLVIMENTO DO APLICATIVO - NÍVEL FÁCIL

Nosso primeiro projeto, que classificamos "fácil", vai se chamar "Labirinto". O objetivo do jogador é passar pelo labirinto até chegar ao portal (Linha de chegada do Labirinto); o jogador também não pode burlar o jogo cortando caminho. Se ele encostar no muro do labirinto, deve voltar e escolher outra direção. Quando o ator chegar até a saída, será mostrado o tempo que demorou para fazer o percurso.

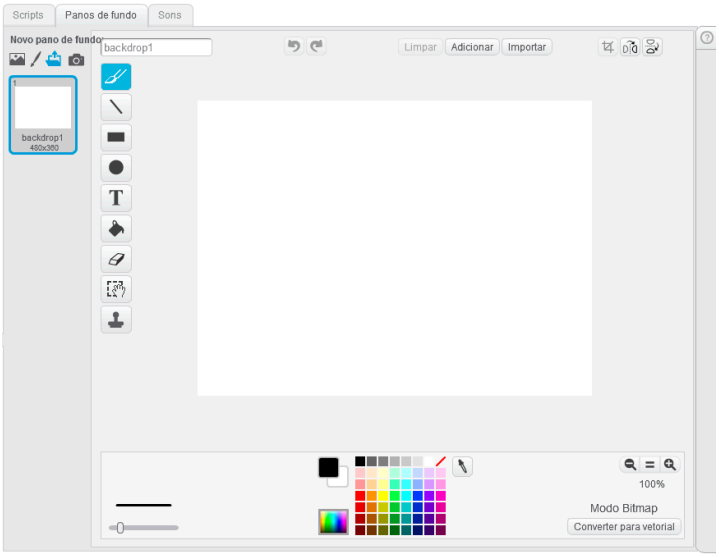
Iniciamos executando o aplicativo Scratch, que se encontra na área de trabalho. Podemos iniciar montando o palco, que será um labirinto. O palco é o fundo de tela onde se movimentam os personagens. Para montá-lo você pode desenhar o ambiente ou importar alguma imagem pronta.

Para importar, você pode fazer o seguinte: no canto inferior esquerdo, clique sobre "Carregar cenário a

partir do arquivo" (ver Figura 23, ícone 3); na janela que se abre, faça o caminho do arquivo desejado, clique sobre ele. Você pode também usar uma imagem da biblioteca; neste caso, use o ícone Palco (ver Figura 23, ícone 1) e selecione uma das opções apresentadas.

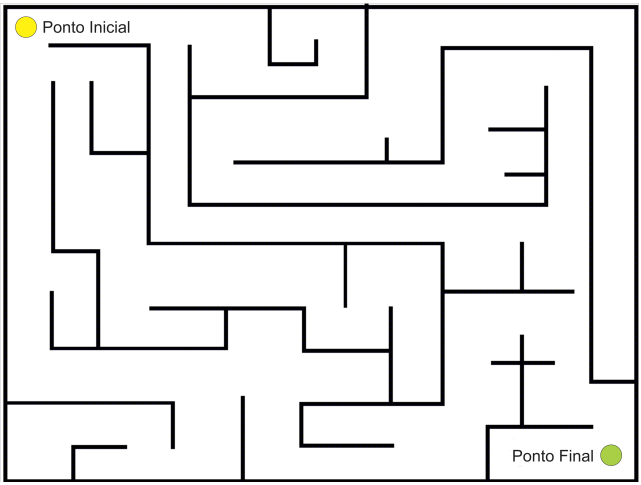
No nosso caso, vamos desenhar um labirinto como o da Figura 34. A bolinha percorrerá diversos caminhos dentro do labirinto e deverá chegar à abertura. Para isso, vá no canto inferior esquerdo, clique no ícone Pintar novo cenário (ver Figura 23, ícone pincel). Na área de desenho que se abre à direita (Figura 34), utilize as ferramentas de desenho Retângulo e Linha e, usando a sua criatividade, desenhe o labirinto, como mostra a Figura 34.

Figura 34 – Importar Pano de Fundo



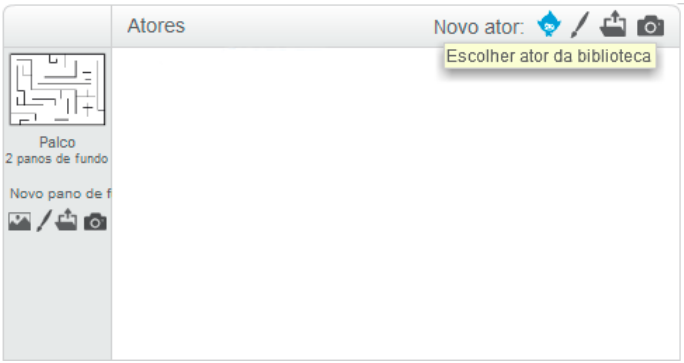
No desenho (Figura 35), não esqueça que o labirinto tem de apresentar ao menos um caminho válido que a bolinha deverá seguir, começando em um ponto inicial (entrada) e terminando em um ponto final (saída). Caso queira salvar a imagem do palco, basta ir para a tela de apresentação e clicar com o botão direito do mouse e selecionar a opção "`Save Picture of Stage`".

Figura 35 – Palco labirinto



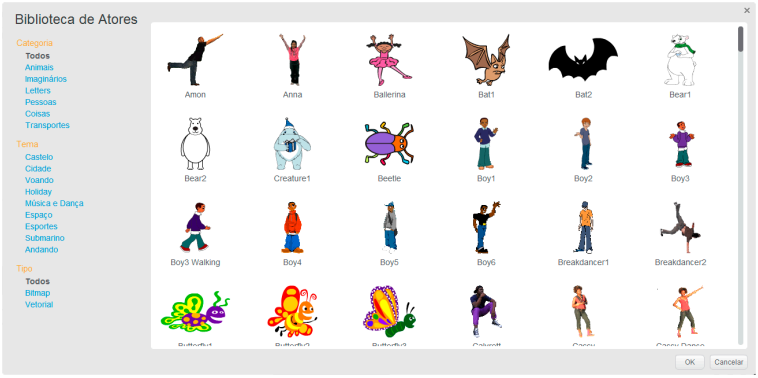
Com o palco criado, podemos criar o personagem que passará pelo labirinto; podemos desenhá-lo ou importá-lo da biblioteca, da mesma forma que fizemos para o palco. Já que aprendemos a desenhar ao montar o cenário, agora vamos importar uma imagem da biblioteca. Para isso, vá no canto inferior esquerdo, na área Atores, Novo Ator, clique no ícone "Escolher ator da biblioteca", como mostra a Figura 36.

Figura 36 – Importar Ator



A biblioteca é bem rica: possui diversos tipos de personagens separados por Categoria, Tema e Tipo (ver Figura 37). Selecione o personagem que desejar para continuar o jogo e clique em Ok.

Figura 37 – Biblioteca personagens



Com o novo Ator criado, podemos lhe atribuir as funções. Para isso, selecione o ator e, na faixa superior da tela, escolha a opção Roteiros (ou Scripts). Nesta aba há um conjunto de comandos que ficam separados por funções: Movimento, Controle, Sensores, Aparência etc. (Figura 38).

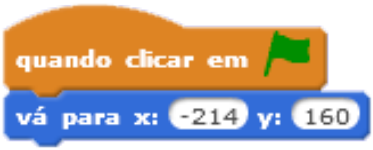
Figura 38 – Menu comandos

Clique na função Eventos, selecione o comando `Quando clicado` e arraste-o para o centro da tela: toda vez que o botão "`Bandeira verde`" for acionado, os outros comandos do programa serão executados (Figura 39).

Figura 39 – Função de iniciar comandos

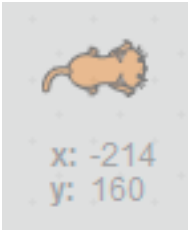
Nos passos seguintes vamos explicar as instruções do jogo, perguntar o nome do usuário, criar um temporizador e os comandos de controle do personagem. Vamos começar definindo a posição inicial do personagem no palco: clique na função Movimento, arraste a instrução `Vá para x:... y:...` e mude os valores de x e de y (veja a Figura 40).

Figura 40 – Definir posição ator no palco



As posições x: -214 y: 160 correspondem ao lado esquerdo superior do palco. A posição de cada personagem é dada sempre por um par de coordenadas (x, y). Para mudar os valores de x e y, basta arrastar o objeto pelo palco; enquanto é arrastado, pode ser vista à direita da tela a figura do objeto e, abaixo dele, as suas coordenadas (x, y), como mostra a Figura 41.

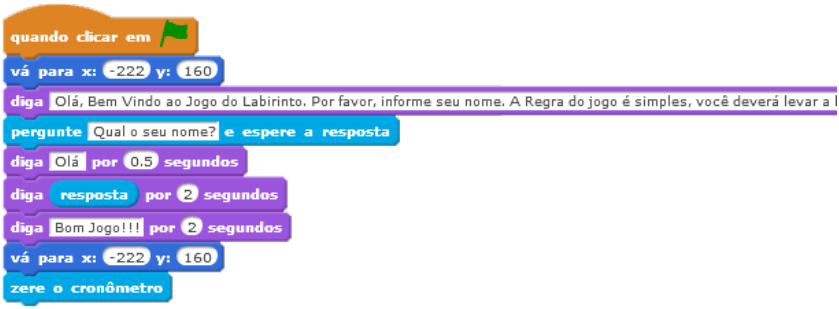
Figura 41 – Definição da posição do persongem através do eixo x e y



Para mostrar as instruções do jogo, ainda na aba Roteiros, clique na função Aparência; depois arraste a instrução Diga e escreva as instruções para o jogador como na Figura 42. Para perguntar o nome do jogador, clique na função `Sensores` e arraste a instrução `Pergunte... e espere a resposta`; você pode escrever outra coisa ao invés de `Qual é o seu nome?`. Clique novamente na função `Aparência` e arraste a instrução `Diga... por... segundos`; você pode alterar o texto `Olá` e o `tempo`. Novamente vá até a função Aparência e arraste o comando `Diga... por... segundos`; agora vá até a função sensores e arraste o comando `resposta` para dentro do comando `Diga... por... segundos`; através desses comandandos, o ator irá retornar o nome que foi informado no início do jogo. Iremos novamente posicionar o nosso personagem para a posição `x: -214 y: 160`, para que o jogador não movimente o personagem durante a exibição das intruções do jogo.

Nesse momento deve ser zerado o cronômetro para iniciar a contagem para um novo jogo: clique na função Sensores e arraste a instrução Zere o cronômetro. No final do jogo será mostrada uma mensagem com o tempo decorrido. Também deve ser repetido o comando de posicionamento inicial do ator, pois o jogador pode ter movimentado o objeto enquanto lê as instruções. O ator deve ser posicionado no início do caminho válido para o labirinto. Veja todo o script na Figura 42.

Figura 42 – Comandos do início do jogo 1



Como bom observador, você já deve ter notado que os comandos são coloridos, como mostra a Figura 42, e que utilizam as mesmas cores colocadas antes dos menus das funções, como mostram as Figura 39 e Figura 40. Isso facilita ver um `script` grande já pronto (como o da Figura 44) e saber de qual parte do menu de funções os comandos foram retirados.

Agora vamos trabalhar com os movimentos do personagem: andar será a única ação feita por ele. Como queremos que ele ande continuamente até conseguir sair, clique sobre a figura dele, depois clique na função Controle e arraste a instrução Sempre. Agora iremos definir a condição para nosso personagem se mover dentro do palco, para isso vá até a função controle e arraste a instrução `Se`; agora iremos definir qual tecla irá mover o personagem para a direita, esquerda, para cima e para baixo. Vá para a função `Sensores` e escolha a instrução `tecla...pressionada` e escolha a tecla que utilizará para movimentar seu personagem para o lado esquerdo; nós utilizamos a tecla seta para esquerda. Com nosso primeiro lado definido, iremos dizer que direção pertence ao lado esquerdo,

para isso vá até a função Movimento e arraste a instrução `aponte para a direção...graus` e posicione a direção que pertence ao lado esquerdo, nesse caso, `-90 graus`. Selecione novamente a aba `Roteiros`, clique na função `Movimento`; arraste a instrução `Mova... passos` para dentro da abertura de `Sempre`; nesse comando definimos que o personagem vai andar quando apertarmos uma das setas; digite 5 na quantidade de passos. Repita o procedimento para as outras três direções restantes. Para adiantar o processo e não precisar escrever todo o comando a mão novamente, vá ao primeiro código e clique com o botão direto e escolha a opção duplicar (conforme a Figura 43), para poder duplicar o código e alterá-lo sem que seja preciso reescrever o código.

Figura 43 – Comando duplicar

Quando o personagem encostar nos muros do labirinto, ele tem que retornar (o jogador não pode trapacear atravessando uma parede). Para isto precisamos utilizar comandos de decisão e repetição: ao encostar na linha do labirinto, ele deve voltar 5 passos para ficar dentro dos limites permitidos. Para usar comandos de decisão, clique na função Sensores e arraste a instrução `Tocando na cor ...?`; em seguida clique no quadradinho da cor e selecione a cor desejada na caixa de cores que surge. O cursor do mouse fará com que a caixa de cores mude de acordo com a cor sobre a qual ele passar. O resultado da criação desses comandos é mostrado na Figura 44.

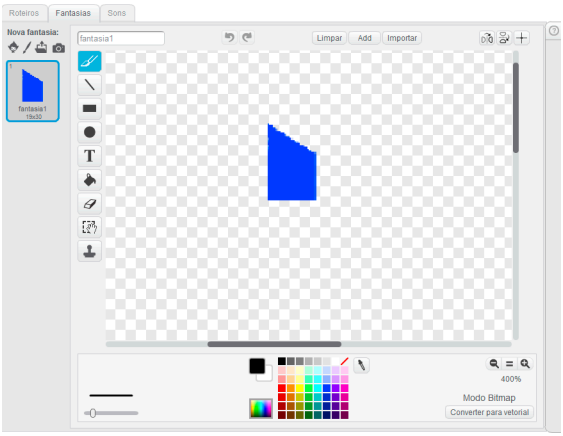
Figura 44 – Comandos do ator

Agora vamos criar a saída do labirinto: desenhamos um portal; quando o gato encostar nele o jogo acabará e será tocada a música da vitória. O portal deve ser um ator. Como já importamos um ator da biblioteca, neste caso vamos desenhá-lo. No canto esquerdo inferior da tela, em Atores, Novo Ator, clique em Pintar Novo Ator (ícone Pincel), como mostra a Figura 45.

Figura 45 – Pintar novo ator

Agora use a sua criatividade e desenhe o portal; pode ser algo como mostra a Figura 46.

Figura 46 – Paint Editor Scratch



Com o portal criado, vamos colocar suas funções. O primeiro comando do novo ator é o mesmo usado para o primeiro ator (veja as Figura 40 e Figura 41): clicar na bandeira verde de início do jogo (`Eventos`) e posicionar-se (`Movimento`); o portal deve ser posicionado na posição final de saída do labirinto. O próximo comando será de `Controle` (a repetição `Sempre`). O próximo comando define o final da partida: toda vez que nosso personagem (Gato) encostar no Portal, será dada a mensagem de vitória, será mostrado o tempo com que foi feito o percurso e será tocada uma música. A Figura 47 mostra todos os comandos completos. Tente completar o `script`: os comandos saem das funções Eventos, Movimento, Controle, Sensores, Som, Aparência e novamente Controle.

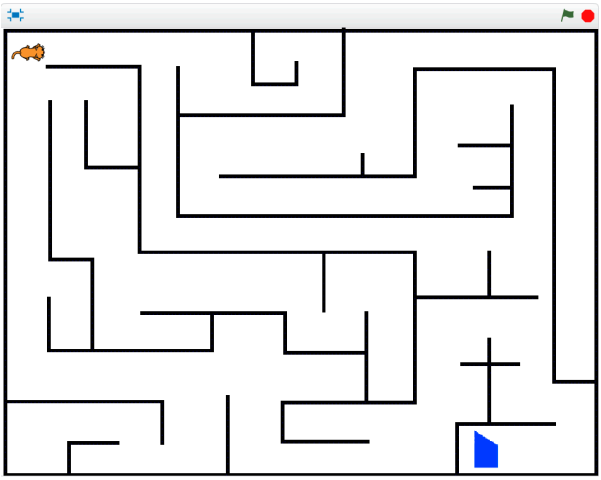
Figura 47 – Comandos do ator Portal

Já vimos no item ABA SONS que podemos colocar sons no projeto: a partir da biblioteca, importando um som de nossa preferência ou gravando um som. Vimos até que temos recursos para editar os sons que vamos utilizar. Vamos colocar o som no nosso projeto, importando-o da biblioteca. Selecione a função Som e arraste o comando `Toque o Som...` Agora clique na aba `Sons`; depois clique no ícone `Carregar som a partir de arquivo`, faça o caminho da biblioteca do Scratch e selecione o som `Congratulations`, como mostra a Figura 48.

Figura 48 – Menu sons

Desta forma concluímos o primeiro projeto. O jogo sendo utilizado está mostado na Figura 49. Observe os pontos de início e de término. Para jogar, clique na bandeira verde e utilize as setas do teclado para comandar o personagem.

Figura 49 – Jogo Finalizado



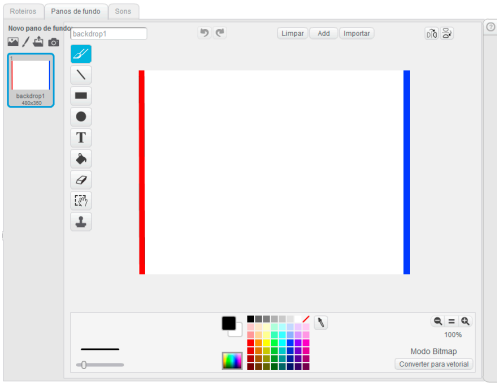
DESENVOLVIMENTO DO APLICATIVO - NÍVEL MÉDIO

Agora vamos criar um jogo de Pingue-Pongue para se jogar em dupla, que podemos considerar de nível intermediário. Este projeto é bastante interessante pois trabalha praticamente com todas as funções da plataforma Scratch. O objetivo do jogo também é simples: quem conseguir jogar a bolinha além da raquete do adversário marca ponto; quem conseguir fazer 5 pontos é o ganhador.

Em primeiro lugar vamos criar o palco: ele terá duas linhas atrás das raquetes, representando o limite até onde a bolinha chegará; sempre que a bolinha encostar nessas linhas o ponto será do adversário. Seguindo o roteiro das Figura 34 e

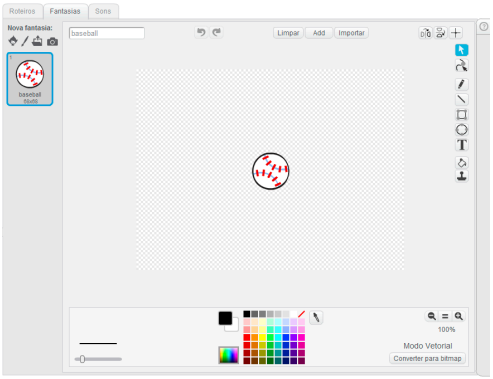
Figura 35, desenhe o palco como o da Figura 50. As cores vermelha e azul são importantes, pois serão usadas como controle para marcar ponto para cada jogador.

Figura 50 – Criação do Palco



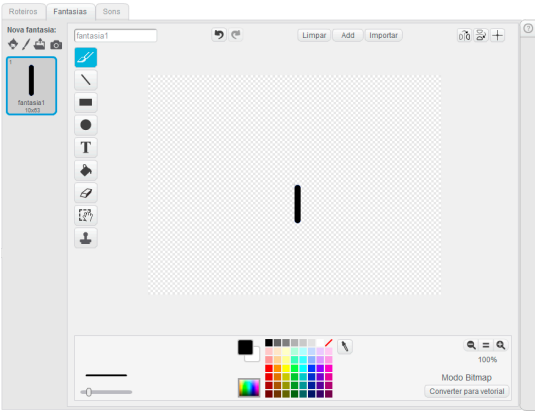
A bolinha para utilizar no jogo é um ator; vamos importá-lo da biblioteca, seguindo o roteiro mostrado na Figura 36/Figura 37. A bolinha importada está mostrada na Figura 51.

Figura 51 – Personagem bola



Agora vamos criar as raquetes; basta criar uma e depois duplicá-la. Vá à área Atores (lado inferior esquerdo) e clique no ícone `Pintar novo ator` (pincel). Como fez com o palco na Figura 50, desenhe a raquete mostrada na Figura 52.

Figura 52 – Persongem raquete

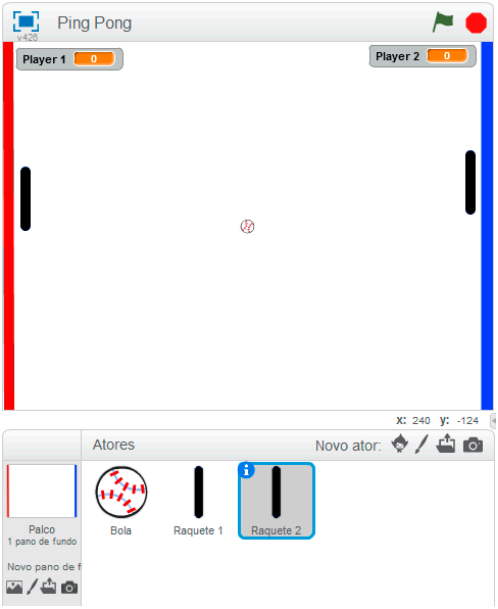


Para duplicar a raquete, na área de Atores, clique sobre ela com o botão direito; no menu popup aberto, clique em `duplicar`, ver Figura 53.

Figura 53 – Duplicar personagem raquete 1

A opção `duplicar` cria um novo personagem na área de Atores (ver Figura 53). A segunda raquete não terá vinculo algum com a primeira raquete. Arraste as raquetes e o placar (variáveis Player 1 Player 2) pelo palco, deixando-os posicionados como mostrado na Figura 54.

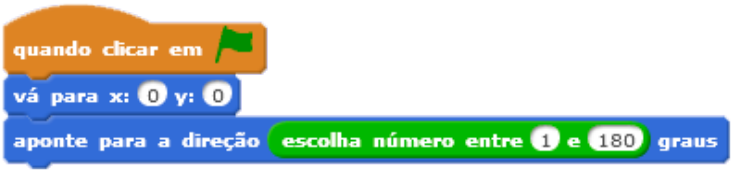
Figura 54 – Visão do palco com as duas raquetes



Criados o palco e os personagens, vamos programar a bolinha e depois as raquetes. No início do jogo a bolinha deverá estar no centro do palco. Coloque portanto os comandos `Quando clicar em` e `Vá para x:... e y:...` (como foi feito nas figuras 40 e 41) e digite 0 para x e y (ver Figura 55). Em seguida arraste a instrução `Aponte para a`

direção 90 graus (da função Movimento); clique na caixinha para digitar os graus (90) e, dentro desse comando, vá à função Operadores e arraste o comando Escolha um número entre 1 e 10; digite 180 no lugar do 10. Veja o resultado na Figura 55.

Figura 55 – Primeira Função do personagem Bola

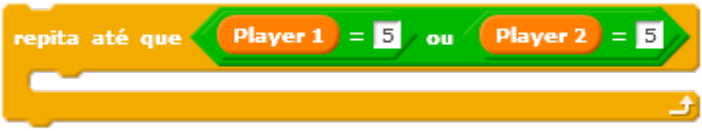


Agora vamos definir o placar para dois jogadores (serão duas variáveis). Para isso selecione a função Variáveis, clique em Criar uma variável e digite Player 1; faça o mesmo para Player 2. Lembre-se que as variáveis só serão criadas para a bolinha de pingue-pongue, as mesmas servirão de contador para o placar do jogo. Deixe marcadas as duas caixas de verificação ao lado das variáveis (ver Figura 56) para que elas apareçam no Palco; arraste-as no Palco para colocá-las na posição que você deseja. As duas variáveis (que representam o placar de cada jogador) devem começar o jogo valendo zero. Para isso, arraste duas vezes o comando mude Player para 0 para o script; mude a primeira instrução para o Player 1 e para 0 (zero); mude a segunda instrução para o Player 2 e para 0 (zero). Arraste também a instrução espere 1 seg da função Controle e altere o tempo para 1.5 segundos. Deixe tudo como a Figura 56.

Figura 56 – Placar: variáveis contadoras de pontos

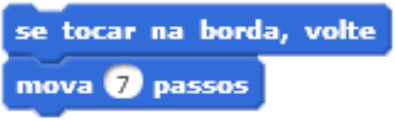
O jogo deve durar até que um dos jogadores (`Player 1` ou `Player2`) atinja 5 pontos. Para isso, arraste para o `script` o comando `Repita até que` que se encontra nas funções `Controle`. Há um hexágono após o `até que`: arraste o operador `ou` para o hexágono (também da função Controle); agora passamos a ter dois hexágonos. Arraste o operador = para o primeiro hexágono. Para a primeira caixinha do `ou` arraste o `Player 1` (da função `Variáveis`); na segunda caixinha, digite 5. Repita o procedimento para o `Player 2`: arraste o operador = para o segundo hexágono; para a primeira caixinha do `ou` arraste o `Player 2` (da função Variáveis); na segunda caixinha, digite 5. Veja como deve ficar na Figura 57.

Figura 57 – Comandos de decisão



Dentro do comando `Repita até que` ocorrerá a maior parte da lógica do projeto. A bolinha deve voltar toda vez que encostar na borda; se não colocarmos esse comando, a bolinha ficará presa na borda, sem voltar ao jogo. Para isso, arraste para a abertura do `Repita até que` o comando `Se tocar na borda, volte` (da função Movimento): esse é um comando de decisão que realiza uma única instrução: voltar. A instrução seguinte `Mova 7 passos` (também da função Movimento) será sempre executada, mesmo se a bolinha não tocar na borda. Veja a Figura 58.

Figura 58 – Comandos de decisão I

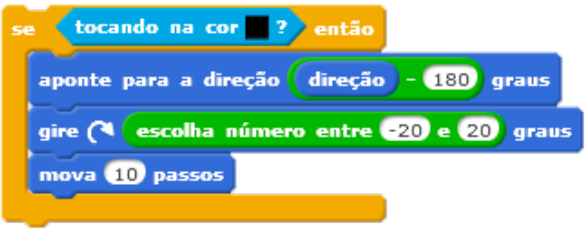


Ainda dentro da abertura do `Repita que` vamos colocar a reação da bolinha quando encostar na raquete (cor preta). Arraste a instrução `Se... então` (da função Controle); arraste a instrução `Tocando na cor ?` para o hexágono; coloque a cor preta na caixinha (ver Figura 59). Para selecionar a cor dentro da caixinha, clique sobre a cor que está dentro da caixinha e arraste o mouse até a cor preta dentro do

palco.. Tocando na cor preta, a bolinha deve voltar. Para programar isso, faça o seguinte: arraste a instrução Aponte para a direção... graus (função Movimento); arraste em seguida o operador menos (função Operadores) para a elipse com o valor dos graus (ficamos agora com duas elipses para preencher); para a primeira elipse, arraste a instrução direção (da função Movimento); na segunda elipse digite 180 (ver Figura 59).

Queremos que, na volta, a bolinha faça um movimento aleatório para sua trajetória ficar imprevisível e o jogo ficar mais movimentado. Arraste a instrução gire... graus (da função Movimento); depois arraste o operador Escolha número entre... e... (da função Operadores); digite -20 na primeira elipse e 20 na segunda (ver Figura 59). Em seguida queremos que a bolinha ande naquela direção: arraste a instrução Mova... passos (função Movimento) e digite 10 na elipse. Confira o bloco completo de instruções na Figura 59.

Figura 59 – Comandos de decisão III

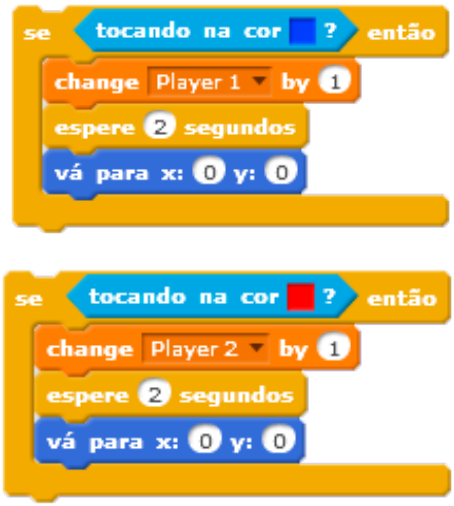


Se a bolinha tocar nos limites do palco (ver Figura 60) devemos marcar ponto para o adversário e voltar a bolinha para o centro do palco. Para isso, arraste novamente os comandos Se... então e Tocando na cor... (como feito na

Figura 60); coloque a cor `azul`. Esse comando deve ficar logo abaixo do `Se` da Figura 60, ainda dentro do `Repita até`. Agora arraste a instrução `Adicione a...` (da função Variáveis) para dentro da abertura do `Se... então`; selecione `Player 1` para a variável a ser somada (ver Figura 60). Arraste instrução `Espere... seg` (função Controle); digite 2 para os segundos. Arraste `Vá para x:... y:...` (função Movimento) e digite 0 para ambos. Confira com a Figura 60.

Observe, pela Figura 60, que as instruções para a contagem do Player 2 são quase idênticas. Repita, portanto, os passos do parágrafo anterior, trocando apenas o `player` e a `cor`.

Figura 60 – Comandos de decisão IV

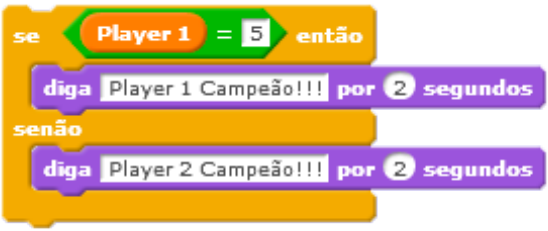


Quando qualquer um, Player 1 ou Player 2, atingir 5 pontos, o programa sairá do comando Repita até que, pois é isso que esse comando controla. Aí precisamos ver quem fez os 5 pontos. Para isso, arraste outra instrução Se... então, senão... (da função Controle) para depois do comando Repita até que. Arraste para o hexágono do Se... o operador = (da função Operadores); para a primeira caixinha do = arraste o Player 1 (da função Variáveis); na segunda caixinha, digite 5. Você fez algo parecido na

Figura 57. Confira com a Figura 61.

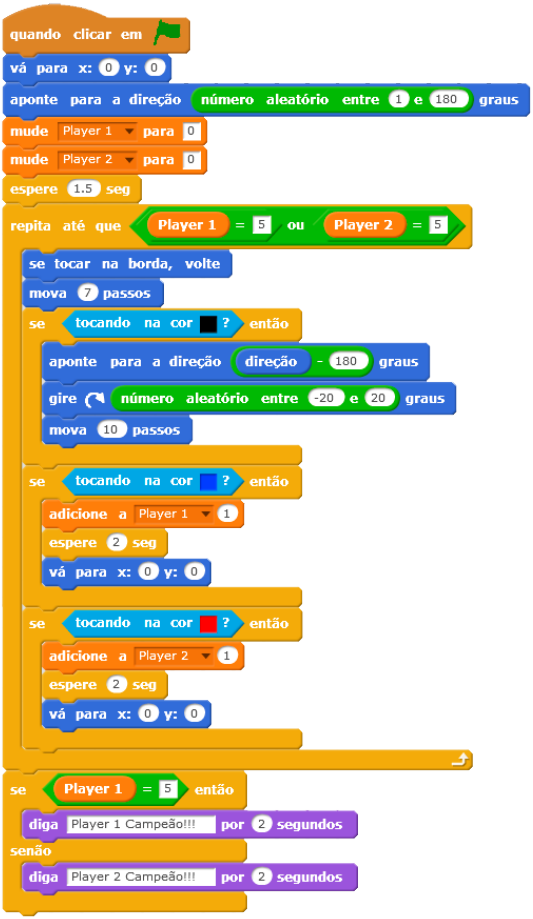
Agora arraste para a primeira abertura do Se... então, senão... a instrução Diga... por... segundos; digite "Player 1 Campeão!!!" e 2 na caixinha de segundos. Clique com o botão direito sobre esta instrução e, no menu popup, clique em Duplicar. Arraste a instrução duplicada para a abertura do Senão e clique novamente para soltar a instrução copiada; digite "Player 2" no lugar do "Player 1".

Figura 61 – Comandos de decisão V



Esses são os comandos definidos para o ator bolinha. A visão completa do seu script está mostrada na Figura 62. Como temos três atores no palco (a bolinha e duas raquetes), precisamos escrever os scripts das raquetes.

Figura 62 – Visão de todos os comandos



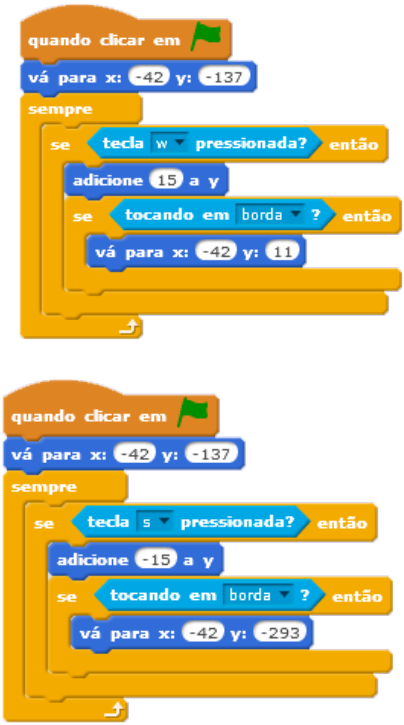
As raquetes só poderão se movimentar para cima e para baixo. A primeira utilizará as teclas `W` para subir e `S` para descer; a segunda utilizará as teclas `Seta para cima` para subir e `Seta para baixo` para descer. Precisamos de dois blocos de comandos para cada raquete. A programação da primeira raquete está mostrada na Figura 63.

Você já viu os comandos `Quando clicar` em e `Vá para x:... e y:...` na Figura 40 e Figura 41 e o comando `Sempre` na Figura 44. O conjunto de comandos a seguir programa a raquete para subir quando pressionada a tecla `W`. O comando `Se... então` foi visto na Figura 60, mas aqui temos novidades: para o hexágono arraste o comando `Tecla... pressionada` (da função Sensores) e selecione `W` para a tecla; dentro da abertura do `Se... então` entrarão duas instruções: `Adicione... a y` (da função Movimento) e outra `Se... então`. Digite 15 na primeira instrução; isso modifica a posição da raquete, fazendo-a subir, pois aumenta o valor de sua coordenada y. Como a raquete subiu, a instrução seguinte `Se... então` verifica se ela tocou na borda: arraste o comando `Tocando em... ?` (da função Sensores) para o hexágono e selecione borda. Arraste outra instrução `Vá para x:... e y:...` para a segunda `Se... então` e digite -42 e 11. Confira pela Figura 63.

O outro conjunto de instruções programa a raquete para descer quando pressionada a tecla `S` e verifica se tocou na borda de baixo. Você pode duplicar todo o conjunto: clique na instrução `Se... então` com o botão direito e depois em Duplicar. Arraste a instrução duplicada para baixo do `Se... então`, ainda dentro do `Sempre` e clique para soltá-la. Faça as alterações devidas: selecione `S` no lugar do `W`; digite -15 no

lugar do 15; digite -293 no lugar do 11. Confira tudo pela Figura 63.

Figura 63 – Comandos do Player 1



Observe que, mantendo constante o valor de x em -42 e variando o y, a raquete só se movimenta na vertical (para cima e para baixo).

Para o Player 2 as únicas mudanças são: as teclas Seta para cima e Seta para caixo e as posições x e y da raquete, como

mostra a Figura 64. Como o grupo de instruções é parecido, você pode duplicá-lo: clique com o botão direito na instrução Quando clicar em... e selecione Duplicar: todo o grupo de instruções "encaixadas" será duplicado. Arraste o grupo de instruções para a área Atores e clique para soltá-lo em cima do ator Raquete 2. Clique agora sobre o ator Raquete 2 para que o seu script (o duplicado) seja mostrado. Altere as teclas e as posições x e y, conforme mostrado na Figura 64.

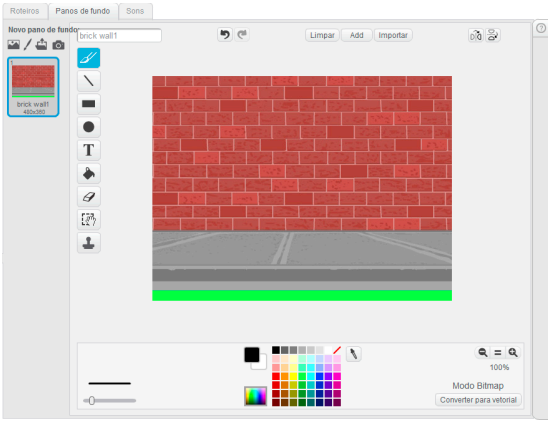
Figura 64 – Comandos do Player 2

DESENVOLVIMENTO DO APLICATIVO - NÍVEL DIFÍCIL

Vamos agora criar um jogo de nível difícil, que chamaremos Quebra Blocos: o jogo terá um Menu, tela de `game over` e será controlado pelo mouse. O interessante do jogo quebra bloco é que o mesmo trabalha com praticamente todas as funções encontradas na plataforma Scratch e o jogador irá se divertir bastante ao desenvolver o jogo e ao jogar, pois o objetivo principal do jogo é conseguir quebrar todos os blocos no menor tempo possível.

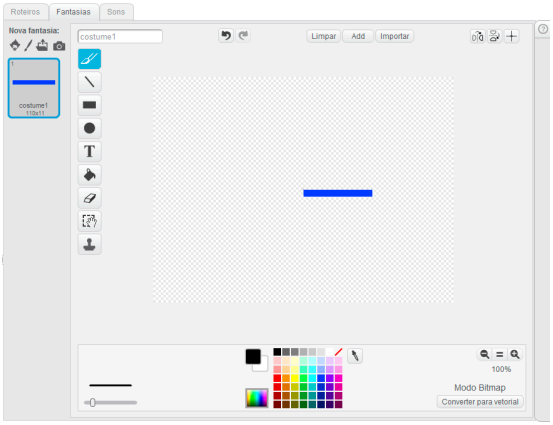
Vamos importar o palco da biblioteca: vá para o canto esquerdo embaixo e clique no ícone `Escolher pano de fundo`; são mostrados vários modelos da biblioteca, escolha um (pode ser o `Brick wall 1`) e clique em Ok. Usando o `Paint Editor` (ver item PAINT EDITOR) desenhe uma linha verde na parte de baixo do muro; observe que há uma barra de rolagem embaixo do palco; lembre-se de desenhar a linha em toda a sua extensão. Essa linha será o limite: toda vez que a bolinha encostar nela, passando assim pela raquete, será enviada uma mensagem de Game Over e o jogo terminará. Veja como fica na Figura 65.

Figura 65 – Palco do jogo Quebra blocos



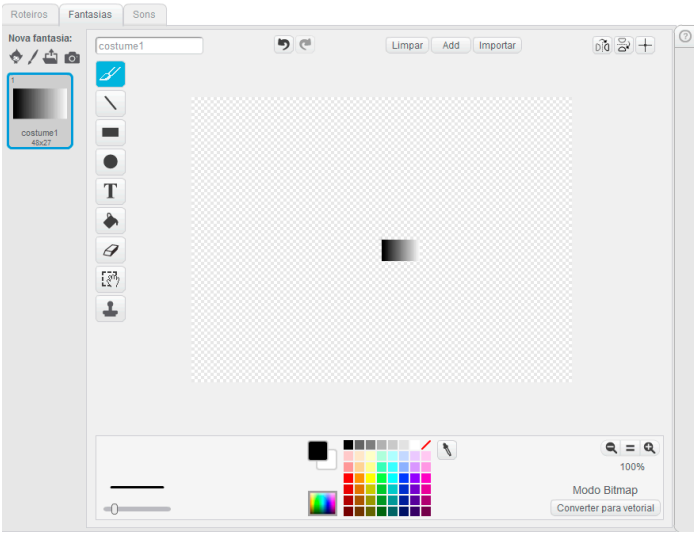
Agora vamos criar a raquete (nosso primeiro ator) que irá rebater a bolinha para quebrar os blocos. Repita o que você já fez na Figura 52; agora a raquete é horizontal e tem a cor azul (ver Figura 66).

Figura 66 – Criação do personagem Raquete



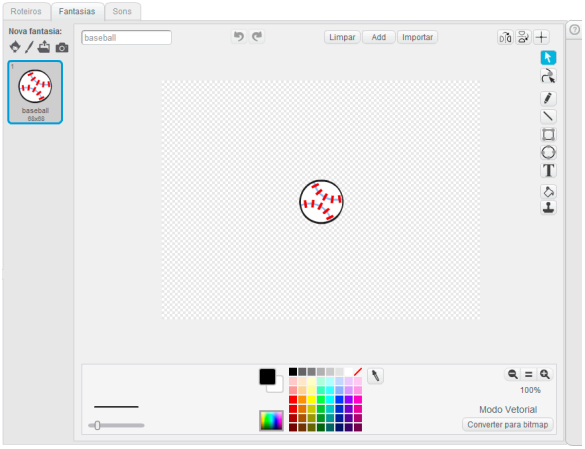
Os blocos que serão destruídos pela bolinha também são atores. Assim como fez com a raquete, desenhe um bloco, como mostra a Figura 67. Duplique os blocos, como fez na Figura 53 (no nosso caso, ficamos com 18 blocos, ver Figura 69). Dê os nomes aos blocos numerando-os: Bloco 1, Bloco 2, Bloco 3... e defina as cores para os blocos para termos uma estrutura heterogênea.

Figura 67 – Criação bloco



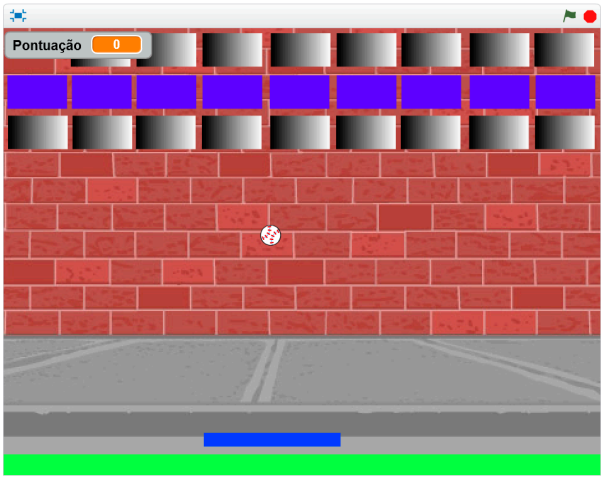
Nosso outro ator é a bolinha; vamos importá-lo da biblioteca, seguindo o roteiro mostrado na Figura 36, Figura 37 e Figura 51. A bolinha importada está mostrada na Figura 68.

Figura 68 – Personagem Bola



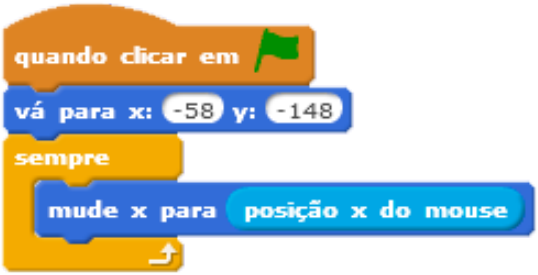
Depois de criar todos os atores, arraste-os no palco, deixando-os na posição mostrada na Figura 69.

Figura 69 – Visão palco



Agora precisamos programar cada um desses personagens. O primeiro `script` será o da raquete: ela andará apenas no eixo x, seguindo o mouse e deverá rebater a bolinha. Clique no ator `Raquete` e na aba `Scripts` coloque as duas instruções de início `Quando clicar em` e `Vá para x:... e y:...` (como foi feito nas Figura 40 e Figura 41); mude a posição inicial para `x: -58 e y: -148`. Coloque depois a instrução `Sempre` (ver Figura 44). Arraste a instrução `Mude x para...` (da função Movimento) para a abertura do `Sempre`; arraste em seguida a instrução `Posição x do mouse` (da função `Sensores`) para a caixa de digitação. Com isso, a raquete se manterá na mesma linha (eixo y), mas seguirá o mouse à direita e à esquerda (variando o eixo x). Observe como fica na Figura 70.

Figura 70 – Comandos de decisão raquete



Apenas isso constitui o `script` da raquete. Os blocos também têm um comportamento simples: quando o jogo começa, eles se mostram em uma determinada posição; se forem tocados pela bolinha, se escondem. Para isso, vamos usar dois

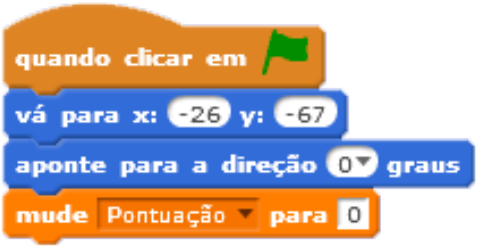
comandos novos: Mostre e Esconda (da função Aparência). Clique em um bloco para iniciar seu script. Coloque o comando Quando clicar em... (da função Eventos), depois arraste o comando Mostre (da função Aparência) e o Vá para x:... y:... (da função Movimento) e digite -186 e 176. Ver Figura 71.

Coloque a instrução Sempre (da função Controle) e arraste, para dentro da sua abertura, um comando Se... então (da função Controle); arraste a instrução Tocando em...? (da função Sensores) para depois do Se... e selecione Bola na caixa de seleção. Dentro do Se... então, coloque dois comandos: Espere... seg (da função Controle) e Esconda (da função Aparência); digite 0.2 para os segundos (duplique o Script para cada bloco e altere apenas a posição de cada). Observe como deve ficar, na Figura 71.

Figura 71 – Script de um bloco

Agora vamos desenvolver as funções do personagem principal, a bolinha. Além de se movimentar em qualquer direção, a bolinha também controlará a pontuação do jogador: toda vez que destruir um dos blocos, somará um ponto para o jogador. Clique na bolinha, na área dos Atores, e em `Scripts`. Coloque o comando `Quando clicar em...` (da função Eventos) e o `Vá para x:... y:...` (da função Movimento) e digite -26 e 67. Dessa forma a bolinha ficará acima da raquete. Arraste a instrução `Aponte para a direção... graus` (da função Movimento) e selecione 0 para os graus; essa direção faz a bolinha ir para cima. Na função Variáveis, crie uma variável com o nome Pontuação (você criou duas variáveis na Figura 56). Arraste a instrução `Mude... para...` para o placar começar com zero. Veja como fica tudo na Figura 72.

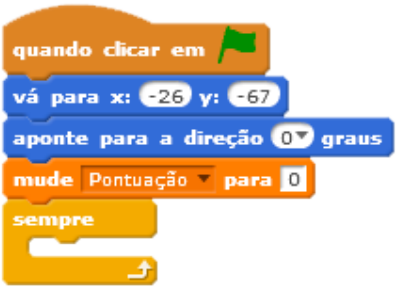
Figura 72 – Comandos iniciais da bola



O jogo termina em três situações: se forem destruídos todos os blocos, se terminar o tempo ou se a bolinha tocar na linha verde. Vamos colocar a função de `loop Sempre`, como

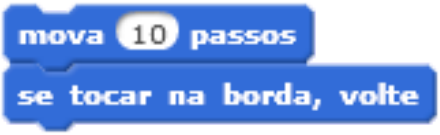
na Figura 73; dentro da abertura desse comando `Sempre` é que virão os outros blocos de comandos a seguir.

Figura 73 – Comando Sempre



Arraste as duas instruções `Mova ... passos` e `Se tocar na borda, volte` (da função Movimento); isto define a velocidade da bolinha e a mantém dentro das bordas do palco; ver Figura 74.

Figura 74 – Comandos personagem bola



O próximo bloco acrescentará 1 ponto ao placar se um bloco for destruído; vai dar um pouco de trabalho porque são muitos blocos. Arraste o comando `Se... então` (da função Controle). Arraste, para o hexágono depois do `Se`, o operador

Ou (da função Operadores); agora arraste, para o primeiro hexágono do Ou, a instrução Tocando em...? (da função Sensores) e selecione o Bloco 1. Observe como fica na Figura 75.

Agora é necessário testar os outros blocos. Para o hexágono vazio do Ou arraste outro operador Ou; para o primeiro hexágono vazio arraste outra instrução Tocando em...? e selecione o Bloco 2. Para o hexágono vazio do Ou arraste outro operador Ou; para o primeiro hexágono vazio arraste outra instrução Tocando em...? e selecione o Bloco 3. Como temos 18 blocos, você deverá repetir este processo para os 17 primeiros blocos. O último operador Ou receberá em cada hexágono uma instrução Tocando em...?, uma delas para o Bloco 17 e outra para o Bloco 18. Observe como fica o comando Se... então na Figura 75.

Dentro do comando Se... então colocaremos três comandos, que serão executados quando um bloco for destruído: somar 1 ponto no placar, fazer voltar a bolinha e tocar uma música. Arraste a instrução Adicione a... 1, da função Variáveis; selecione Pontuação. Arraste a instrução Aponte para a direção... graus; complete-a colocando um operador menos do mesmo jeito que você fez para a Figura 59. Arraste a instrução Toque o som... da função Som e importe uma música da biblioteca (você fez isso nas Figura 47 e Figura 48). Veja os três comandos na Figura 75.

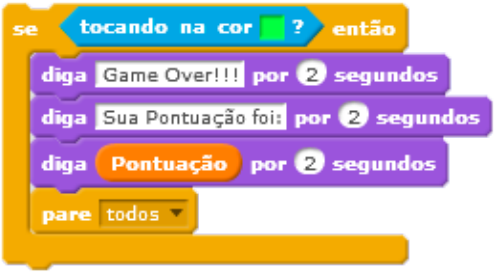
Figura 1 – Comandos de decisão

Quando a bolinha tocar a raquete, deve mudar de direção e fazer um movimento aleatório; já fizemos isso com a bolinha de pingue-pongue (ver Figura 59). Arraste outra instrução `Se... então`; depois arraste, para o hexágono do `Se`, a instrução `Tocando em...?` (da função Sensores) e selecione `Raquete`. Arraste para dentro do `Se... então` a instrução `Aponte para a direção... graus`; complete-a colocando um operador `menos` do mesmo jeito que você fez para a Figura 75; agora, porém, a subtração é invertida: compare as Figura 75 e Figura 76. Em seguida, arraste a instrução `Gire... graus` (da função Movimento) e complete-a, como você fez na Figura 59. Veje como fica na Figura 76.

Figura 76 – Comandos de decisão II

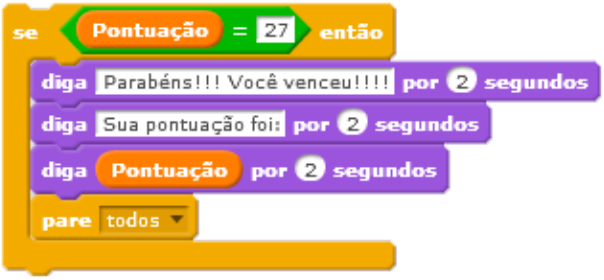
Se a bolinha passar pela raquete e encostar na linha verde, o jogo termina e é mostrada a pontuação do jogador. Os comandos para fazer isso estão mostrados na Figura 77. Você já utilizou todos eles: o `Se... então`, com `Tocando na cor...` na Figura 60; o `Diga... por... segundos` e o `Pare...` na Figura 47.

Figura 77 – Comandos de decisão III



Caso a pontuação atinja o máximo, exibimos a mensagem de Parabéns e a pontuação obtida pelo jogador. Os comandos para fazer isso estão na Figura 78. Você já utilizou todos eles: o `Se... então` com uma comparação na Figura 61; os demais, na figura anterior, podem ser duplicados.

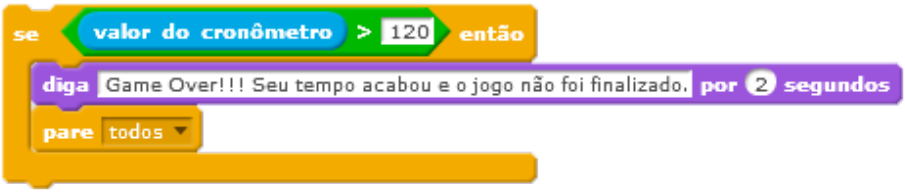
Figura 78 – Comandos de decisão IV



Ainda é preciso vigiar o cronômetro: vamos limitar o tempo de jogo em dois minutos; se exceder, `Game over!`. Arraste outra função `Se... então` (da função Controle);

arraste para o hexágono depois do `Se` o operador `Maior >` (da função Operadores); agora arraste para o primeiro hexágono do > a instrução `Valor do cronômetro` (da função Sensores); digite 120 no segundo hexágono do >. Duplique as duas instruções `Diga... por... segundos` e `Pare...` e coloque-as dentro da abertura do `Se... então`. Veja como fica na Figura 79.

Figura 79 – Comandos de decisão V



PYTHON

INTRODUÇÃO

Uma linguagem de programação é necessária para comunicar-se com o computador. Dessa forma é que ele recebe instruções e realiza o que o programador pediu. Porém, o computador não é capaz de compreender a linguagem humana. Basicamente ele só reconhece dois valores: 0 (sem passagem de corrente elétrica) e 1 (com passagem de corrente elétrica), o que tornaria impossível a comunicação com os humanos. As linguagens de programação foram inventadas para que possamos falar de um jeito humano e, mediante alguma tradução, o computador possa entender na linguagem dele.

O Scratch, que acabamos de ver, pode ser entendido como uma linguagem de programação, só que com dois propósitos específicos: criar jogos e utilizar uma interface visual para arrastar os comandos de forma a esconder a complexidade das linguagens de programação de uso geral.

Existem dezenas de linguagens de programação de uso geral; cada uma delas possui padrões ou regras de codificação específicos, que chamamos `sintaxe`. Você pode entender isso como a gramática da língua. A diferença é que se houver um erro de sintaxe, nem que seja um ponto ou uma vírgula, o comando não será traduzido para o computador: haverá um erro de sintaxe. No Scratch, você dificilmente comete um erro de sintaxe, pois escreve pouco: os comandos já estão prontos, você completa pouca coisa. Em linguagens de uso geral podem ocorrer vários erros de sintaxe.

Além do mais, o computador realiza somente as ações que foram programadas. É você quem tem que garantir que todas as instruções necessárias foram colocadas e na ordem certa; chamamos isso de `lógica do programa`. O Scratch não pode evitar que você cometa erros de lógica; nenhuma linguagem pode. Você descobre isso quando está testando ou usando o programa.

A linguagem Python apresenta uma sintaxe de alto nível; dizemos isso porque sua forma de escrita é bem próxima da linguagem humana. Ela foi concebida em 1889 por Guido Van Rossum no Instituto de Pesquisa para Matemática e Ciências da Computação nos Países Baixos e foi anunciada em 1991. A linguagem é clara e objetiva, evitando o uso de caracteres especiais no código. As suas variáveis são de tipagem dinâmica; ou seja, não é necessário declarar o tipo delas: adotam o tipo do valor que receberem. Outra característica importante é a indentação do código, ou seja, a forma com que o texto é organizado: ele deve ter seus blocos organizados com o espaçamento correto, caso contrário não irá funcionar. Além disso, Python suporta orientação a objetos.

INSTALAÇÃO

Para desenvolver em Python, primeiramente é necessário baixar sua plataforma de desenvolvimento. Ela está disponível no site oficial (python.org).

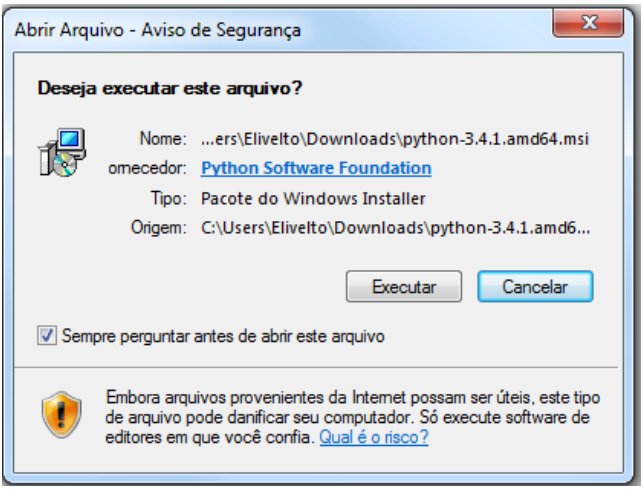
Para baixar e instalar o Python siga os seguintes passos:

1 – Acesse o link <`https://www.python.org/downloads/release/python-341/`>.

2 – Desça pela página até a seção File e selecione a opção que deseja baixar. Aqui será baixada a opção `Windows x86 MSI installer.`

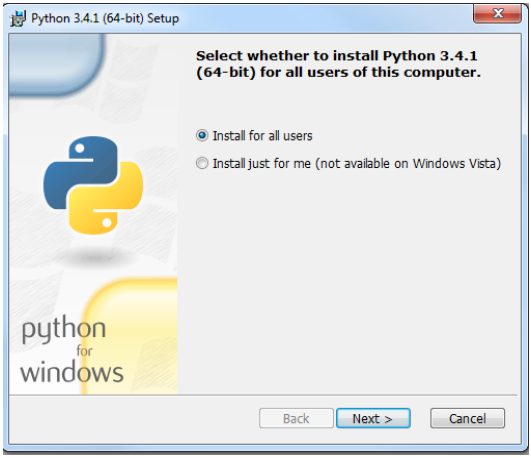
3 – Depois de baixado o arquivo, dê dois cliques sobre ele para executá-lo. Surge a tela da Figura 80. Clique no botão Executar.

Figura 80 – Processo de instalação do Python 3.4.1 I



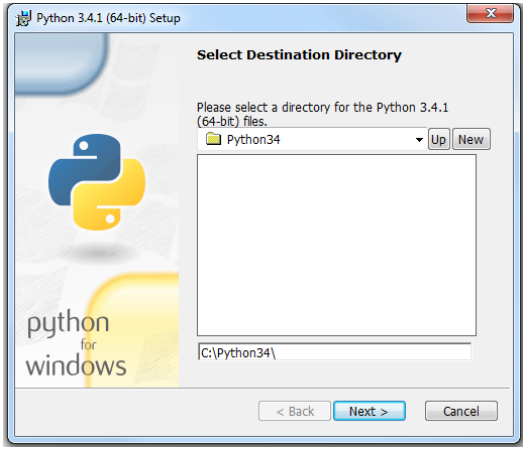
4 – Na tela seguinte, Figura 81, há a opção de instalar o Python para todos os usuários do computador, ou só para si. Escolha a opção desejada e clique em Next.

Figura 81 – Processo de instalação do Python 3.4.1 II



5 – Na Figura 82 podemos escolher o diretório onde instalar o Python. Escolha outro diretório ou deixe no padrão; clique em Next.

Figura 82 – Processo de instalação do Python 3.4.1 III

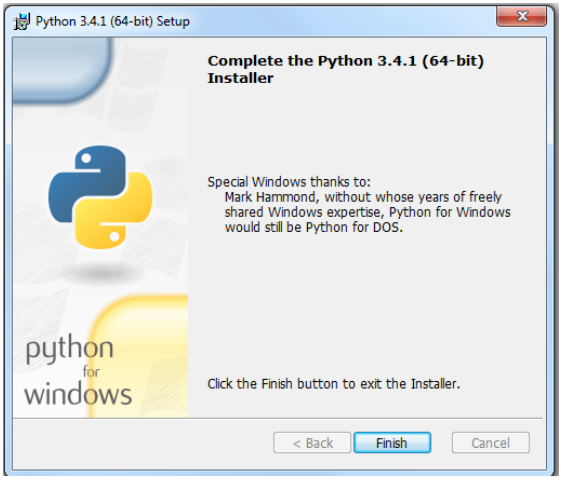


6 – Na Figura 83 você pode escolher os componentes que deseja instalar. Se não souber bem quais componentes deseja, deixe o padrão que já vem selecionado. Clique em Next.

Figura 83 – Processo de instalação do Python 3.4.1 IV

7 – Para concluir a instalação, Figura 84, clique em Finish.

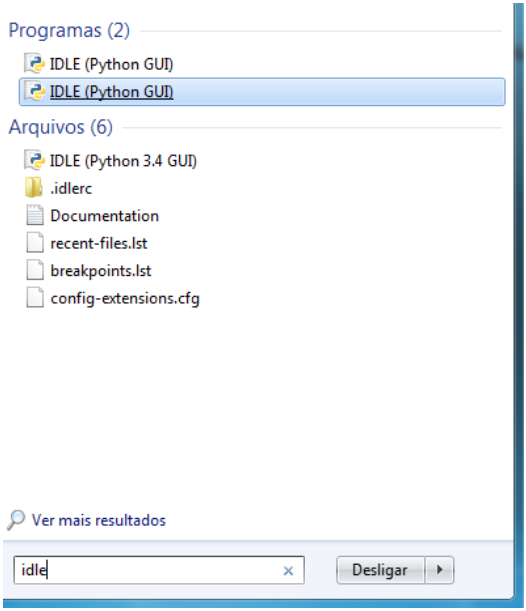
Figura 84 – Processo instalação Python 3.4.1 V



PRIMEIROS PASSOS

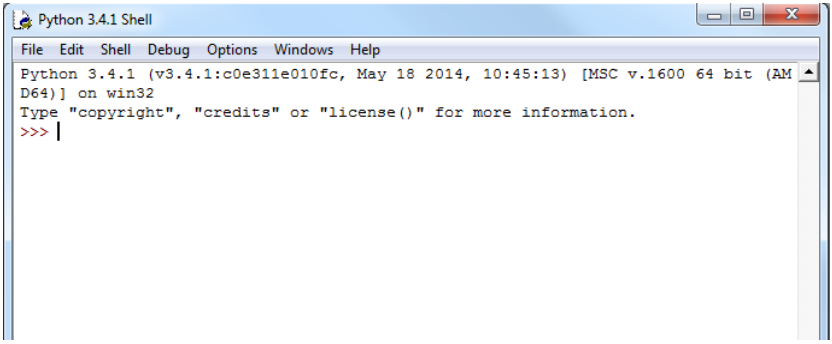
Finalizada a instalação, vamos ver como desenvolver programas e executá-los. Vamos primeiro encontrar o ambiente do Python: no menu iniciar, no campo de pesquisa digite idle, e clique para pesquisar (ver Figura 85):

Figura 85 – Pesquisa por IDLE



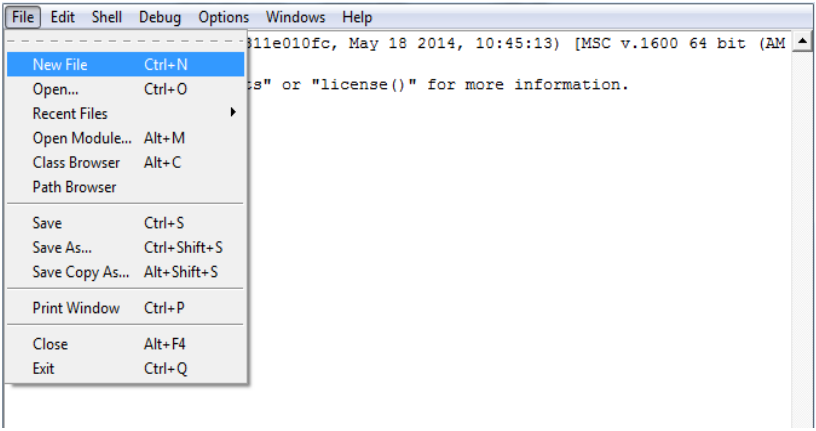
Como resultado, aparecerá na seção programas, um programa chamado IDLE (Python GUI). Clique sobre ele e a tela do `shell` do Python abrirá. A versão do Python aparece na barra de título da janela do aplicativo (ver Figura 86), neste caso, "`Python 3.4.1 Shell`". Trabalhando com mais de uma versão do Python, esse detalhe é importante para diferenciá-las.

Figura 86 – Shell do Python



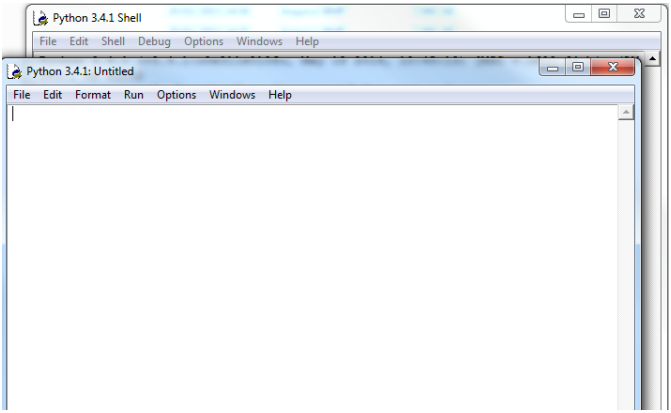
Para escrever um código em Python, é necessário abrir um novo arquivo Python. Abra o menu File, como mostrado na Figura 87:

Figura 87 – Abrindo novo arquivo



Em seguida clique em New File. Será exibida a tela da Figura 88:

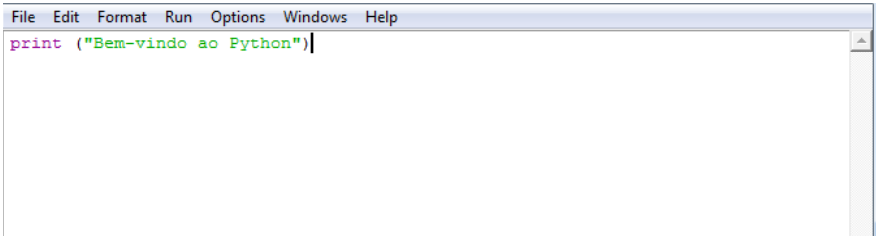
Figura 88 – Tela de um novo programa, em branco



Será aberto um arquivo em branco. Nele será inserido o código dos programas em Python. Vamos fazer um programa para exibir uma saudação; digite o seguinte código[25]: `print ("Bem-vindo ao Python")`, como mostrado na Figura 89. Esse comando faz o programa exibir na tela a mensagem "`Bem-vindo ao Python`".

[25] Caso você copie e cole o texto, redigite as aspas; frequentemente os caracteres aspas são tratados de forma diferente de um editor de textos para outro.

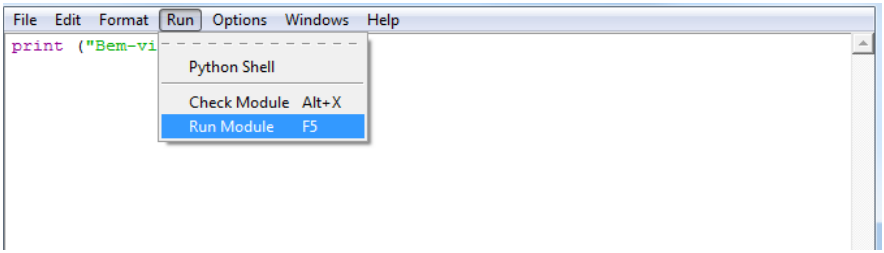
Figura 89 – Comando print



Note que as palavras têm algumas cores, como violeta, preta e verde. A cor violeta é usada para os comandos, a verde para o texto que irá aparecer na tela (menos as aspas) e a preta para as demais partes do código.

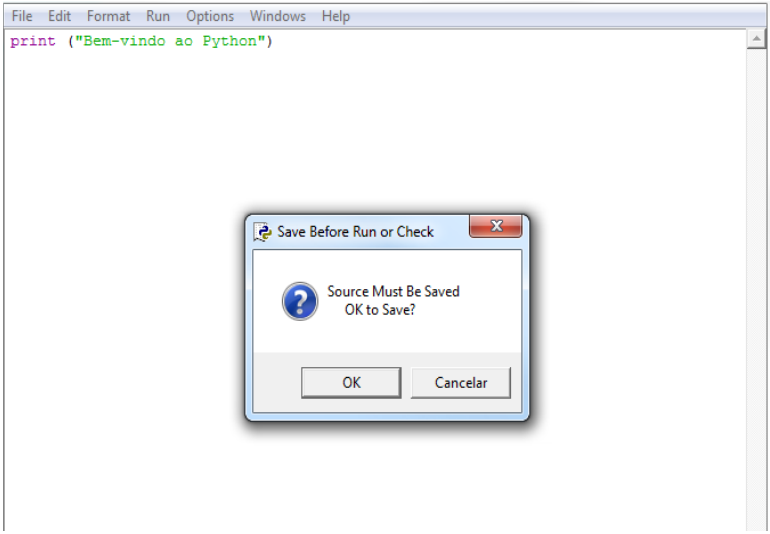
Para executar o código, vá para o menu, clique em "`Run`" e depois em "`Run Module`", como exibido na Figura 90, ou pressione a tecla F5.

Figura 90 – Executando o código



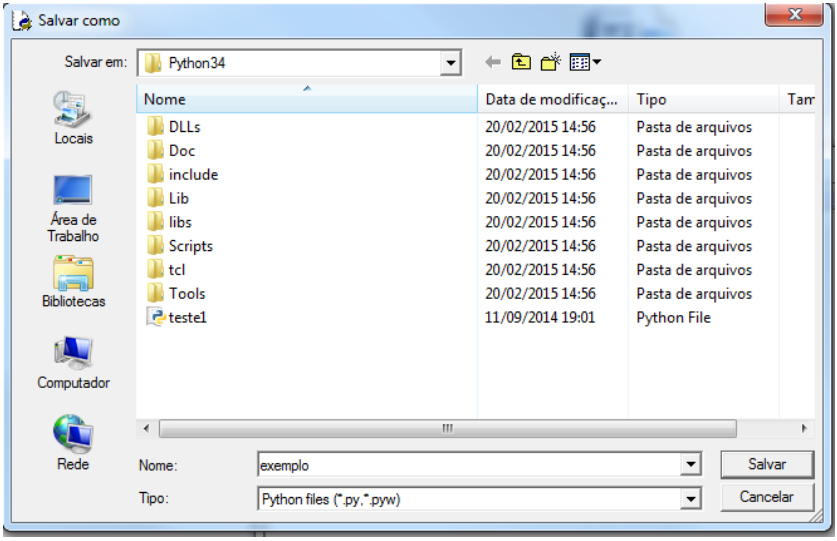
Antes de o programa executar ele precisa ser salvo. Será exibida uma mensagem para confirmação desse procedimento, conforme Figura 91. Clique em Ok para salvar.

Figura 91 - Confirmação para salvar o arquivo.



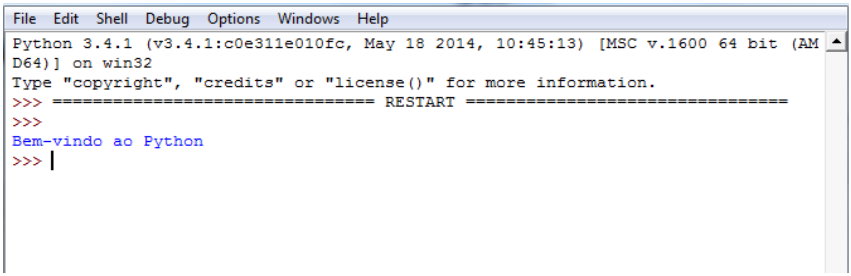
Se confirmar, será perguntado onde deseja salvar o programa e o nome do mesmo, conforme mostrado na Figura 92. Nomeie o programa (pode ser exemplo1), escolha um local para salvá-lo e clique em Salvar.

Figura 92 – Salvando o arquivo



Agora, sim, ele executará. O programa exibirá a mensagem na tela do Shell do Python, de acordo com a Figura 93.

Figura 93 – Mensagem do programa exibida no Shell



Esse é um exemplo de um código bem simples, que só exibe uma mensagem da tela. No próximo código vamos trabalhar com os comandos de entrada e saída.

COMANDOS DE ENTRADA E SAÍDA

Os comandos de entrada e saída são os mais básicos e essenciais comandos para um programa de computador, pois o objetivo dos programas é: receber uma entrada de dados, processá-la e mostrar o resultado desse processamento (saída).

Os `comandos de entrada` recebem e tratam os dados que são passados ao programa pelo usuário. Os `dados de entrada` podem ser: as teclas que o usuário digita, movimentos ou cliques do mouse, dados lidos de um arquivo ou de um banco de dados, informações recebidas por um `scanner` etc. Nos nossos exemplos trabalharemos com entradas no teclado.

Os `comandos de saída` apresentam informações para o usuário. Essas podem ser: informações na tela do computador, dados gravados em um arquivo, páginas impressas etc. No nosso caso, as saídas das informações serão na tela do computador.

O comando utilizado para mostrar uma saída de dados na tela é o `print`. Quando o programa quer mostrar algo para o usuário, ele faz através desse comando. Por exemplo, a saudação `Seja Bem-Vindo ao Programa`, para que nosso programa a escreva na tela, utilizamos o `print`:

Exemplo 1 – Comando print

```
1    print("Seja Bem-Vindo ao Programa")
```

A estrutura desse programa é bem simples: primeiro temos a palavra `print`, depois abrimos parênteses e depois aspas. Tudo que está entre as aspas aparecerá na tela. Não podemos nos esquecer de fechar as aspas e os parênteses, lembrando que o primeiro a abrir (no caso os parênteses) é o último a fechar. Tudo o que for colocado dentro dos parênteses pertence ao comando print.

Para receber as entradas do teclado utilizamos o comando `input`. Esse comando captura o que o usuário digitou no teclado. Porém, somente ler (capturar) o que o usuário digita não é suficiente, pois, quando lemos o que o usuário digita, queremos guardar aquela informação para utilizá-la no programa.

Para guardar as informarções usamos variáveis. Como o próprio nome diz, `variável` é algo cujo valor pode variar. Toda variável tem um nome e um valor. Por exemplo, variável de nome `x`, tem o valor 5, variável `y` tem o valor 1.5, variável nome tem o valor `José`.

O nome das variáveis é o programador que define, sendo possível qualquer nome diferente dos comandos Python. Por exemplo, você não poderá dar o nome de `print` para uma variável, senão o computador achará que você está inserindo um comando de saída.

Em nosso comando de entrada precisaremos de uma variável para receber o que o usuário digitar no teclado. Normalmente, quando se usa esse comando, a instrução anterior

é um comando de saída, para especificar o que o programa quer que o usuário digite. Se quiser que o usuário digite seu nome, é bom antes exibir uma mensagem para perguntar "`Qual é o seu nome?`". Por exemplo:

Exemplo 2 – Comando input

```
1    print("Qual é o seu nome?")
2    seunome = input()
```

Nesse exemplo temos um comando de saída e, logo após, um comando de entrada. Já vimos a estrutura do comando de saída, agora será explicada a estrutura do comando de entrada (`input`).

Antes do comando de entrada temos uma variável de nome `seunome`. O valor dela será o que o usuário digitar no teclado: se ele digitar Pedro, a variável passará a ter o valor `Pedro`. Logo após há um sinal =. Normalmente aprendemos que esse sinal é o `igual`, mas em Python e em várias outras linguagens de programação lemos esse sinal como recebe. Assim, nesse exemplo, a variável seunome recebe o que o usuário digitar. Logo após vem o comando `input`, que tem a função de ler o que é digitado no teclado. Quando o usuário pressiona a tecla `ENTER`, a leitura do teclado é interrompida e tudo o que foi digitado passa para a variável `seunome`.

Há também uma outra forma de escrever esse comando: exibindo uma mensagem dentro dos parênteses. Assim, evita-se o uso do `print`:

Exemplo 3 – Usando input para escrever e ler dados

```
1     seunome = input("Qual é o seu nome?")
```

Esse código faz a mesma coisa que o anterior. A diferença é que o `input` desta vez irá apresentar também uma mensagem na tela, antes que o usuário digite. É muito útil usar o `input` desta forma, pois agiliza a escrita do código e o deixa menor e mais simples.

Com esses conhecimentos já é possível escrever um pequeno código de cadastro, onde serão mais bem detalhadas as funções acima descritas. O primeiro programa é o seguinte:

Programa 1 – Cadastro de Clientes

```
1.    print("Cadastro de Cliente! ")
2.    nome = input("Digite o nome do
      cliente: ")
3.    telefone = input("Digite o telefone de
      contato: ")
4.    cpf = input("Digite o CPF: ")
5.    endereco= input("Digite o endereco: ")
6.    bairro = input("Digite o bairro: ")
7.    cidade= input("Digite a cidade: ")
8.    estado= input("Digite o estado onde
      mora: ")
9.    profissao= input("Digite a profissao:
      ")
```

```
10.  print ("\nCliente: "+nome+"\nTelefone:
     "+telefone+"\nCPF: "+cpf+"\nEndereço:"
11.  +endereco+"\nBairro:
     "+bairro+"\nCidade:
     "+cidade+"\nEstado:"
12.  +estado+"\nProfissão: "+profissao)
```

O código acima pergunta os dados de um cliente, guarda os valores digitados em variáveis e ao final exibe todos os dados digitados. Para facilitar a localização, o código acima foi todo numerado, os números colocados à esquerda das linhas, porém esses não fazem parte do código. O código funciona da seguinte maneira:

A linha 1 informa do que trata o programa: o comando print irá escrever "`Cadastro de Cliente`" na primeira linha da tela.

Logo após é escrita outra linha na tela: `Digite o nome do cliente:`. Essa frase é escrita pelo comando `input` (linha 2), que aguarda o usuário digitar algo. Quando o usuário digita alguma coisa e pressiona a tecla `ENTER`, esse comando transfere tudo que foi lido do teclado (ou seja, as teclas que o usuário digitou) para a variável `nome`. Assim, o conteúdo dessa variável passa a ser o que o usuário digitou. Após isso, toda vez que se desejar saber o que o usuário digitou, basta buscar na variável `nome`. É feito o mesmo processo para `telefone`, `CPF`, `endereço`, `bairro`, `cidade`, `estado`, `profissão` (linhas 3 a 9).

As linhas 10, 11 e 12 fazem com que o programa mostre tudo o que foi digitado até o momento, com o comando `print`.

As três linhas acima são na verdade um comando só; foi quebrado em três linhas porque é muito grande. É importante lembrar que tudo que está entre os parênteses pertence ao comando `print`.

Para começar, o comando escreve `Cliente`: na tela. Porém, note que no exemplo acima vem escrito "`\nCliente: `"; os caracteres `\n` não aparecerão na tela, pois fazem parte da sintaxe do `print`: servem para instruir o programa a pular uma linha na tela. No Python, a cada vez que você usa um comando `print` a mensagem na tela é exibida na próxima linha. Como utilizamos apenas um comando, o programa iria escrever tudo em uma única linha. Entretanto, como queremos todos os dados de forma organizada, uma linha abaixo da outra, e com apenas um comando, utilizamos o `\n`; com isso será saltada uma linha antes da exibição de cada linha. Observe também que, para escrever `Cliente: `, colocamos tudo entre aspas, escrevendo "`\nCliente: `", inclusive o espaço em branco, pois trata-se de um texto que queremos que seja escrito exatamente dessa forma.

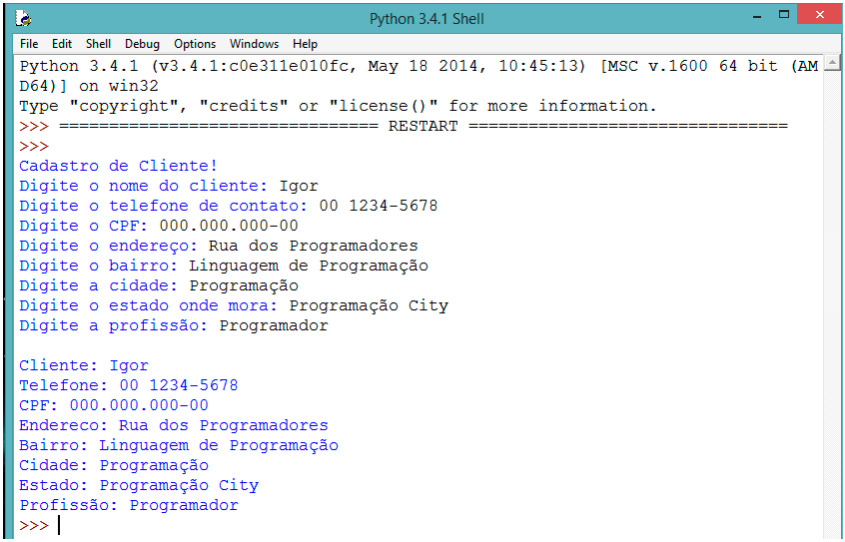
No comando, após "`\nCliente: `", há um sinal +. Esse sinal, nesse contexto, não tem nenhuma ligação com a operação matemática `soma`, já que nesse exemplo trabalhamos com caracteres, que não podem ser somados. O sinal + aqui significa apenas que, depois de `Cliente:`, haverá mais caracteres a serem escritos pelo mesmo comando `print`. Nesse caso, o sinal + poderia ser substituído por uma vírgula, tendo o mesmo efeito.

Assim o programa escreverá `Cliente: ` e mais alguma coisa. Esse "`mais alguma coisa`" que vem após o sinal + é a variável `nome`. Note que nome não é escrito entre aspas,

porque o objetivo não é exibir a palavra "`nome`", mas sim o conteúdo da variável `nome`. Como o que o usuário digitou ficou armazenado na variável `nome`, o comando `print`, ao exibir o valor da mesma, mostrará exatamente o que foi digitado anteriormente. Por exemplo, se o usuário digitou `Pedro`, esta primeira linha aparecerá assim: `Cliente: Pedro`.

Após `nome` há outro sinal de mais, significando que há mais coisa a escrever na tela. O programa pula uma linha por causa do `\n` e escreve `Telefone:` e logo à frente põe o conteúdo da variável `telefone`. É feito o mesmo processo para `CPF`, `endereço`, `bairro`, `cidade`, `estado` e `profissão`. A saída dos dados dependerá do que o usuário digitar, mas a aparência ficará como mostra a Figura 94:

Figura 94 – Cadastro de clientes



COMANDOS DE DECISÃO

Os comandos descritos no item anterior são muito úteis, pois permitem que o programa e o usuário interajam. Porém entrada e saída de dados são tarefas muito simples, que o próprio usuário consegue fazer sozinho. Normalmente espera-se que um programa faça algum tipo de processamento desses dados e mostre o resultado do processamento. Para isso, são usados vários outros tipos de comandos. Um tipo importante abrange os comandos de decisão. Como o próprio nome diz, estes servem para que o programa tome decisões, ou seja, de acordo com uma condição, o programa fará uma ou outra determinada ação.

Em Python, usa-se o comando `if` para a tomada de decisões. Em português significa `se`, e funciona da seguinte forma: se determinada condição for atendida faça estas ações. A estrutura do `if` é a seguinte:

Exemplo 4 - Estrutura simples de comandos de decisão

```
1    if(condição):
2        comando1
3        comando2
4    comando3
```

Observe que no trecho abaixo do `if` as linhas `comando1` e `comando2` têm um deslocamento à direita; `comando3`, não. Isso acontece porque a indentação (organização do código) em Python tem um papel muito

importante: a partir da mesma é que se sabe quais comandos pertencem ao `if` (`comando1` e `comando2`) e os que não pertencem (`comando3`). Dessa forma mantém-se o código organizado e mais compreensível.

Uma condição, como colocada na sintaxe do `if`, é uma comparação. O resultado da comparação pode ser verdadeiro ou falso. Há várias formas de comparar, e para isso são usados alguns `operadores` (ver Tabela 4 - Operadores):

Tabela 4 - Operadores

Operador	**Significado**
==	Igual
!=	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menos ou igual

Há ainda como juntar duas condições num mesmo `if`. Isso é feito através do operador lógico e, escrito como `&&`, e do operador lógico `ou` escrito como `||`. Dessa maneira pode-se fazer com que o programa só faça determinada ação se ele satisfizer uma condição ou várias outras ao mesmo tempo.

Uma condição pode ser escrita da seguinte forma:

Exemplo 5 – Condição simples

```
1     if numero>0:
2         print("O número é positivo!")
```

Nesse caso estamos comparando se um determinado número, anteriormente informado, é maior do que 0. Se for, o programa exibe a mensagem "`O número é positivo`". Caso o número informado não seja maior do que 0, o programa não executa o comando `print`, já que não atende à condição (o resultado da comparação é `falso`), e nada será impresso. Assim, o comando `print` faz parte do `if` e só se chega a ele quando o resultado da comparação do `if` for `verdadeiro`.

Pode-se unir duas condições com os operadores `&&` e `||` acima explicados:

Exemplo 6 – Condição composta

```
1     if numero > 0  && numero < 10:
2         print("O número está entre 0 e
      10!")
```

Nesse código há duas condições: se o número é maior que 0 e se o número é menor do que 10. Se o número testado atender às duas condições, o programa exibe a mensagem; caso não atenda às duas condições, o comando `print` não é executado.

Até agora vimos o funcionamento do `if` quando a condição é atendida; porém, pode-se também programar alguma ação para quando a condição não seja atendida. Para isso utilizamos a extensão `else`, que significa senão. Se a condição for verdadeira, o programa realizará uma determinada ação; se não for, será realizada outra ação.

Exemplo 7 – Comando else

```
1    if numero > 0:
2        print("O número é positivo!")
3    else:
4        print("O número não é positivo!")
```

Nesse exemplo, a condição é `número maior que 0`. Se o número atender à condição, o programa exibirá a mensagem "`O numero é positivo`". Caso não atenda, o programa mostrará a mensagem "`O número não é positivo`". Note que programa sempre executará uma, e somente uma, das ações.

Em alguns casos será necessário especificar melhor as condições para realizar uma determinada ação: se uma condição não for atendida, pode-se verificar outra, depois outra, e outra, sucessivamente, executando o `else` somente depois de várias condições examinadas. Para esses testes sucessivos podemos usar o `else if`. O `else if` pode ser escrito também de forma simplificada: `elif`.

Exemplo 8 – Estrutura completa com comandos de decisão

```
1    if condição:
2        ação
3    elif condição:
4        ação
5    else:
6        ação
```

Essa é a estrutura do comando de decisão `if`, que pode ser estendido com `else if` (`elif`) e com `else`. O exemplo a seguir testa se um número é positivo, nulo ou negativo. É importante lembrar que, além ou no lugar da instrução `print`, poderia ser colocado um bloco com várias outras instruções.

Exemplo 9 – Comando elif

```
1    if numero > 0:
2        print("O número é positivo!")
3    elif numero == 0:
4        print("O número é nulo!")
5    else:
6        print("O número é negativo!")
```

A primeira condição do exemplo é `numero > 0`. Se atender a essa condição, o número é positivo; se não atender, não podemos afirmar que seja negativo, pois pode ser igual a zero. Assim, temos que testar outra condição: se o número é

igual a 0. Se não for igual a 0, não é necessário outro teste, pois ele só poderá ser negativo.

APLICATIVO PAR OU ÍMPAR

Esse aplicativo utilizará os conceitos até aqui ensinados. Porém, o mesmo incluirá operações aritiméticas. Os `operadores aritméticos` (conforme Tabela 5 - Operadores aritméticos) em Python são:

Tabela 5 - Operadores aritméticos

Operador	Significado
+	Soma
-	Subtração
*	Multiplicação
/	Divisão (resultado exato)
//	Divisão (resultado inteiro)
**	Potenciação
%	Módulo (resto da divisão)

O aplicativo recebe um número digitado e diz se o número é par ou ímpar.

Programa 2 – Par ou ímpar

```
1    print ("Par ou Impar \n\n")
2    x = int (input("Digite um numero: "))
3    if x % 2 == 0:
4        print (x," é par!")
5    else:
6        print (x," é impar!")
```

Na linha 1, o programa exibe uma mensagem inicial informando o objetivo do programa. A seguir, salta duas linhas devido ao `\n\n`.

Na linha 2, o programa exibe uma mensagem solicitando que o usuário digite um número. Note que o comando input está diferente do primeiro aplicativo porque desta vez queremos que o usuário digite um número, e não um caractere. O comando `input` lê os valores do teclado como caracteres; para mudar isso, temos que converter o valor digitado para outro tipo de dado, um número inteiro. Para fazer essa conversão usamos `int()` e dentro dos parênteses colocamos o dado a ser convertido para o tipo `int`. Assim a variável x receberá um número e será do tipo inteiro.

Na linha 3, a condição testada pelo comando `if` é se o número é par. Para saber se um número é par precisamos dividi-lo por 2 e o resto deve dar 0. Para saber o resto de uma divisão, temos que usar o símbolo %. Essa operação `x % 2` retorna o resto da divisão; por exemplo: `5 % 2` é igual a 1, ou seja: dividindo 5 por 2, o `quociente` é 2 e o `resto` é 1. Então o

comando pode ser entendido desta forma: se o número x digitado pelo usuário, dividido por 2, der resto 0, então realize as ações que vêm depois dos dois pontos.

Após digitar os dois pontos, pressione a tecla `ENTER`. A linha abaixo aparece levemente deslocada para a direita. Isso significa que esta ação pertence à condição acima.

Atendida a condição do `if`, o programa exibirá uma mensagem (linha 4), escrevendo o número que foi digitado pelo usuário, seguida pela mensagem `é par`. Por exemplo, se o usuário digitar 4, o programa mostrará a mensagem "`4 é par`".

Caso não atenda à condição do `if`, o programa executará outra ação, correspondente ao `else` (linha 5). Lembre-se de digitar os dois pontos e depois pressionar `ENTER` para que a linha abaixo fique deslocada para a direita.

Esta é a ação do `else`: caso o número, ao ser dividido por 2, se der resto 1, o programa exibirá a mensagem dizendo que o número é ímpar (linha 6).

Após executar o programa, ele retornará um resultado semelhante às Figura 95 e Figura 96 abaixo:

Figura 95 – Testando aplicativo com número ímpar

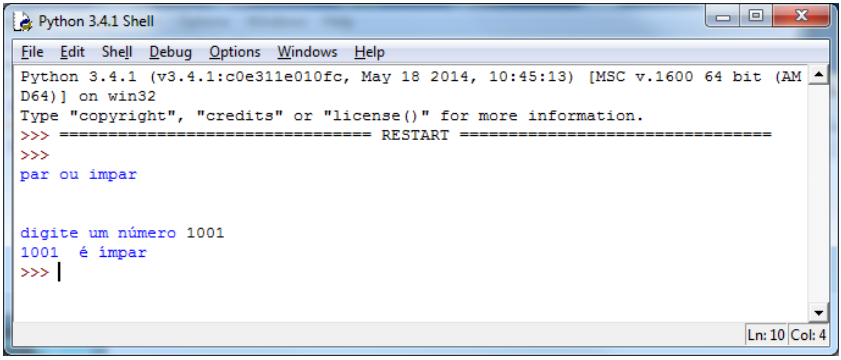
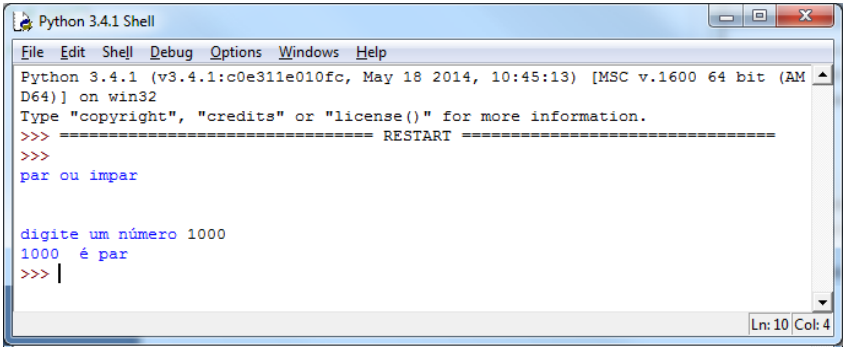


Figura 96 – Testando aplicativo com número par



COMANDOS DE REPETIÇÃO

Para cada ação do programa temos que digitar um comando correspondente. Mas há situações em que é necessário executar os mesmos comandos várias vezes. Por exemplo, imagine se, para exibir mil números, fosse necessário escrever mil comandos `print()`. Para evitar isso, as linguagens de programação têm instruções que permitem que o programa repita um trecho de código várias vezes. A linguagem Python tem dois comandos de repetição: `for` e `while`. Com eles é possível repetir um grupo de instruções (que costumamos chamar `laço`) um determinado número de vezes ou até que ocorra uma condição específica.

Para repetir os comandos uma certa quantidade de vezes, é necessário uma variável para contar quantas vezes o código foi repetido. Por exemplo, para repetir 20 vezes, podemos usar a variável `i`. Para que a coisa funcione, a variável `i` tem de ser inicializada com o valor 0 antes de entrar no laço e, dentro do laço, deve ser incrementada em `1`; a condição dirá que o laço deve ser interrompido quando `i` atingir 20.

Outra forma de estabelecer o momento de parada do comando de repetição é a utilização de uma condição; por exemplo, até o usuário digitar algo. Neste caso, não precisamos contar o número de vezes, mas também precisamos de uma variável para controlar a condição: ela irá armazenar o que o usuário digitar. Por exemplo: podemos dizer ao programa para repetir o grupo de instruções enquanto a variável `palavra` for diferente de “`parar`”; quando o usuário digitar “`parar`”, o programa para de repetir aquele trecho.

Quando sabemos ou temos como saber quantas vezes queremos repetir um trecho de código, normalmente usamos o comando `for`. Para utilizar este comando é necessária uma variável que irá contar quantas vezes ocorreu a repetição. Junto com o `for` utilizamos o comando `range()`. Este comando determina um intervalo, tendo três parâmetros: valor inicial, valor limite final e valor a ser somado a cada repetição. A estrutura do comando é a seguinte:

Exemplo 10 – estrutura do comando for com range()

```
1    for variável in range(parâmetros):
2        comando
3        comando
4        comando
```

Quando utilizamos apenas um argumento no comando range, esse valor será o limite final. Por exemplo: `range(10)` indica um intervalo de 0 a 9; o intervalo terá 10 elementos: `[0,1,2,3,4,5,6,7,8,9]`; o valor 10 não está no conjunto porque o comando irá listar valores até números menores que o número definido (10 é o limite); neste caso, o comando `range` inicia o intervalo com o número 0. Veja o exemplo:

Exemplo 11 – Comando range com apenas um parâmetro

```
1    for i in range(10):
2        print("O valor da variável i é: ",
     i)
```

Neste caso, a variável `i` assume os valores dentro do intervalo [0,1,2,3,4,5,6,7,8,9]. No primeiro momento `i` assume o valor 0 e o comando `for` executa o comando `print` mostrando a mensagem que o valor da variável `i` é 0. Logo após a variável `i` assume o próximo valor que é 1 e o `for` executa o `print` novamente. Isso acontecerá até a variável `i` chegar ao último valor do intervalo, o 9. Como o intervalo tem 10 elementos, o programa executará o trecho 10 vezes, 10 mensagens de saída do `print`. Note que quando é passado somente um elemento ao range, o número de repetições coincide com o número informado.

Quando passamos dois parâmetros para o `range`, o primeiro parâmetro é o primeiro número do intervalo e o segundo é o limite final. Por exemplo, `range(1,10)`: o intervalo começará em 1 e irá de 1 em 1 até 9: [1,2,3,4,5,6,7,8,9] e terá 9 elementos. Isso é mostrado no exemplo abaixo:

Exemplo 12 – Comando range com dois parâmetros

```
1    for i in range(1,10):
2        print("O valor da variável i é: ",
     i)
```

Se utilizarmos três parâmetros, a ordem dos parâmetros será: valor inicial, valor limite e valor a somar por ciclo. Por exemplo, `range(0,11,2)`: o intervalo começará em 0 e irá até 11, de 2 em 2: [0,2,4,6,8,10], contendo 6 elementos; o terceiro parâmetro soma 2 a cada ciclo à variável de controle. Observe o exemplo:

Exemplo 13 – Comando range com três parâmetros

```
1     for i in range(0,11,2)
2         print("O valor da variável i é: ",
      i)
```

Se não for possível determinar quantas repetições serão necessárias, utilizamos o comando `while` (que significa enquanto). Para este comando devemos informar a condição necessária para ele executar, por exemplo, enquanto o usuário digitar algo diferente de "`sair`". Dessa forma, não há como saber quantas vezes o laço `while` será repetido, pois depende do que o usuário digitar. O comando `while` é estruturado desta forma:

Exemplo 14 – Estrutura do comando while

```
1     while condicao:
2         comando
3         comando
4         Comando
```

Agora veja exemplo prático da utilização do comando while:

Exemplo 15 – Comando while

```
1    numero = 0
2    while numero <= 10:
3        print("Digite um numero maior que
     10: ")
4        input(numero)
```

Esse programa pede ao usuário que digite um número maior que 10. Enquanto o usuário não digitar um número maior que 10, o programa solicita novamente a digitação do número. A variável numero é iniciada com 0 para garantir que o laço de repetição `while` execute na primeira vez. Após isso, o `print()` exibe a mensagem "`digite um número maior que 10`" e o `input()` faz a leitura do que o usuário digitar. O programa volta a fazer o teste da condição do `while` com o número digitado.

Quando o usuário digitar um número maior que 10, a condição `numero <= 10` se tornará falsa e então o programa sairá do laço.

APLICATIVO SOMA DE NÚMEROS

O programa a seguir exemplifica a utilização do `for` utilizando duas variáveis no intervalo do range. Esse aplicativo permite ao usuário efetuar a soma de um intervalo de números, por exemplo, de 0 até 100; para isso, ele deverá digitar o menor e o maior número do intervalo.

Programa 3 – Soma de números

```
1     print("Soma de números!") #escreve na
      tela "Soma de números"
2     primeiro_numero = int(input("Digite o
      número a partir do qual deseja
somar: "))
3     ultimo_numero = int(input("Digite o
      último número até onde deseja
somar: "))
4     soma=0
5     for i in
      range(primeiro_numero,ultimo_numero+1):
6         soma = soma + i
7     print("O resultado da soma é: ", soma)
8     #exibe uma mensagem com o valor da
      variável soma
```

Primeiramente, há uma mensagem inicial informando do que o programa se trata: "`soma de números`". Note que há um sinal de # na primeira linha, ele indica que se inicia um comentário no código. O comentário é uma parte do código que não realiza nenhuma ação no código, sua função é apenas informativa, ajudando na compreensão do programa. Ele tem coloração vermelha.

Nas linhas 2 e 3, solicita-se ao usuário que ele digite o primeiro número da série, e depois o último número da série. Os comandos `input()` estão dentro de `int()`. Isso ocorre porque eles precisam ser convertidos para o tipo `int` (inteiro). Observe que as linhas foram quebradas, pois não couberam na página. No editor de Python não se deve quebrar essas linhas.

Para realizar a soma, é necessário armazenar o valor acumulado em uma variável soma (linha 4). Seu valor inicialmente deve ser zerado, pois não somamos nenhum número ainda.

Nas linhas 5 e 6 é que acontece a soma dos números. Usamos um comando de repetição `for` para que o programa vá somando os números consecutivos e acumulando na variável `soma`. O comando `for` precisa de um contador, a variável `i`, para saber onde começa e onde termina. O comando `range` foi usado com dois parâmetros (`primeiro_numero` e `ultimo_numero+1`): o primeiro é o valor inicial que será atribuído à variável `i` e o segundo é até onde o valor da variável `i` chegará. Queremos somar desde o `primeiro_numero` até o `ultimo_numero`, inclusive. Por isso, o segundo parâmetro é `ultimo_numero+1`, isso garante que o programa irá somar também o `ultimo_numero` informado pelo usuário. Como não informamos o terceiro parâmetro do `range`, a cada repetição `i` é incrementado em 1.

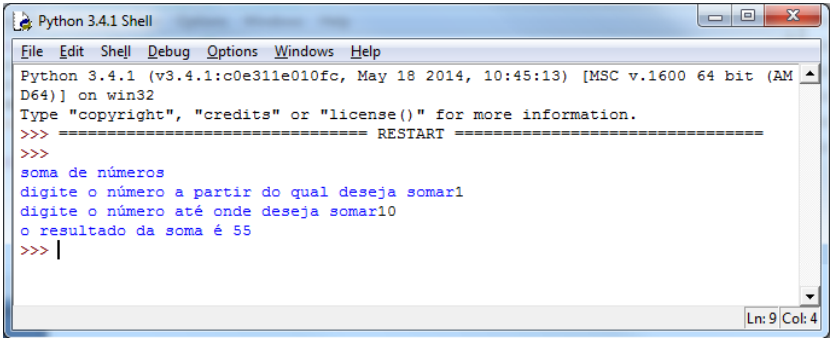
Logo após os dois pontos vem a ação do comando `for`, `somar i` na variável `soma`. Como `i` é incrementada em 1 a cada ciclo, se somarmos o seu valor, teremos a soma da série. A variável soma é responsável por armazenar os valores somados; assim, soma recebe ela mesma mais a variável `i`. O valor de soma que aparece no lado direito do = é o seu valor anterior; o programa resolve as operações que estão descritas no lado direito do = e depois atribui esse resultado à variável que está do lado esquerdo. Assim, é somado o valor anterior de soma com `i` e atribuído o novo valor a `soma`; com isso, terminada aquela instrução, `soma` terá um novo valor.

Por exemplo se o usuário digitar 1 e 3. O valor de i primeiramente será 1 e soma receberá ela mesma (0) mais i (1); assim `soma` passará a ter o valor 1. No segundo ciclo, `i` aumenta em 1 e passa a ter valor 2; assim `soma` recebe ela mesma (1) mais `i` (2); com isso `soma` assume o valor 3. No terceiro ciclo, `i` passa a valer 3, `soma` recebe ela mesma (3) mais `i` (3), com isso `soma` assume o valor 6. Assim a soma dos números de 1 a 3 resulta em 6.

Para finalizar, o programa exibe uma mensagem informando o resultado.

Após executar o programa, o resultado será semelhante ao mostrado na Figura 97.

Figura 97 – Aplicativo soma



LISTA

A linguagem Python possui vários tipos básicos de dados. Eles podem ser inteiros, como 1, 2 e 3, caracteres com a, b e c, ou números com casas decimais, como 1.5, 2.2 e 2.5. Porém variável armazena somente um valor por vez, embora esse valor possa ser mudado sempre que receber um novo valor. Entretanto, às vezes é necessário armazenar uma sequência de valores. Para isso existe a lista de dados, que também pode ser chamada de `array`.

Uma lista é um tipo de dado que suporta vários valores; uma variável do tipo lista é composta por vários elementos, cada um com um valor. Cada seção da lista tem um índice, começando por 0 (zero). O número total de elementos de uma lista pode ser definido ou indefinido. A declaração da lista é feita da mesma forma que se declara uma variável, nomeando-a e atribuindo seu valor; a diferença é o uso de colchetes para informar os valores. Veja um exemplo:

Exemplo 16 – Inciando lista

```
1     lista = [1,2,3]
```

Nesse exemplo, a variável `lista` recebe três valores: 1, 2 e 3. O valor 1 se encontra em `lista[0]`, o valor 2 em `lista[1]` e o valor 3 em `lista[2]`.

Também é possível definir uma lista vazia. Nesse caso a lista não recebe nenhum valor. Veja o exemplo abaixo:

Exemplo 17 – Lista vazia

```
1     lista = []
```

Para acrescentar valores a uma lista vazia é necessário usar o comando `append`, passando o valor entre parênteses:

Exemplo 18 – Acrescentado um valor a lista

```
1     lista.append(5)
```

Assim o valor 5 passa a ocupar a posição 0 da lista. Cada vez que se usa o comando `append()` a lista passa a ter mais um elemento e o valor passa ocupar a última posição da lista. Para remover um elemento da lista pode-se utilizar o comando `remove`, da mesma forma que o anterior:

Exemplo 19 – Removendo um valor da lista

```
1     lista2.remove(5)
```

Dessa forma, o elemento 5 é removido da lista (se houver mais de um elemento com o valor 5, apenas o primeiro é removido). Esse método remove um valor da lista. Porém também se pode remover o valor de uma posição, por exemplo, remover o valor da primeira da lista, através do comando `del`:

Exemplo 20 – Removendo o valor de uma posição da lista

```
1     del lista[0]
```

Cada vez que um elemento é acrescentado ou retirado da lista, altera-se o tamanho da mesma. Para saber o tamanho da lista é utilizado o comando `len()`:

Exemplo 21 - Tamanho da lista

```
1     len(lista)
```

Esse comando retorna o número de elementos da lista e é muito útil ao percorrer uma lista com algum comando de repetição.

APLICATIVO CADASTRO DE FRUTAS

Com todos os comandos aprendidos até aqui, é possível fazer um programa um pouco mais complexo, envolvendo comandos de entrada e saída, decisão, repetição e listas. O aplicativo que será apresentado abaixo será de cadastro e busca de frutas.

Programa 4 – Cadastro de frutas

```
1     print("      Cadastro de frutas
      ")#exibe a mensagem
```

```
2    print("Cadastre as frutas que desejar,
     e para concluir, digite sair")
3    i=0#a variável recebe 0, e servirá
     como contadora no laço for
4    palavra=""#a variável palavra recebe
     conteúdo vazio
5    frutas=[]
6    while(palavra!="sair"):
7        print("Digite o nome da fruta da
     seção: ", i)#exibe mensagem mostrando
     o valor de i
8        palavra=input()#lê o que o usuário
     digita e atribui à variável
9        if(palavra!="sair"):#verifica se a
     palavra é diferente de sair
10           frutas.append(palavra)#se for,
     a coloca na lista
11       i=i+1#o valor da variável i é
     incrementado
12   for i in range(len(frutas)):
13       print("seção",i,"-->",frutas[i])
14   print("\n\nProcurar produtos")#exibe
     mensagem
15   palavra_procurada=input("Digite a
     fruta que deseja procurar ") #realiza
     leitura
16   for i in range(len(frutas)):#percorre
     a lista
```

```
17          if(palavra_procurada==frutas[i]):
18              print("a
      fruta",frutas[i],"está na seção",
      i)#exibe mensagem
```

O aplicativo tem o objetivo de fazer um cadastro de frutas, organizando-as em seções.

Na linha 5 é criada uma lista vazia. A lista irá conter os nomes de frutas que serão digitados pelo usuário, por isso é criada vazia.

Na linha 6 há um comando de repetição `while` e uma condição. Dessa forma, enquanto a variável `palavra`, que o usuário irá digitar, for diferente de "`sair`", o programa repetirá as linhas dentro do laço do `while`. Como a variável palavra começa vazia, quando chegar ao comando `while` pela primeira vez, a condição será verdadeira, pois vazio é diferente de "`sair`". Logo após é pedido para o usuário digitar o nome da fruta da seção `i`.

Nas linhas 9 e 10, que também pertencem ao `while`, há uma outra condição que verifica se a palavra que o usuário digitou é diferente de "`sair`", pois esta palavra não deve ser armazenada. Se for diferente, o programa adiciona a variável palavra ao final da lista. Por exemplo, se a pessoa digitar "`morango`" e a lista já contiver ["`banana`", "`melacia`"], nesta ordem, a lista ficará: ["`banana`", "`melancia`", "`morango`"].

A 11 linha aumenta em 1 a variável i, para que, no `print` (linha 7), o usuário possa saber em qual seção será inserida a palavra digitada por ele.

Nas linhas 12 e 13 utilizamos o `for` e novamente a variável `i` como contadora. A variável *i* terá seu valor incrementado até chegar ao tamanho da lista. A função `len(frutas)` retorna o tamanho da lista: quantos elementos existem nela. Assim o programa irá repetir o `print` o número de vezes correspondente à quantidade de elementos.

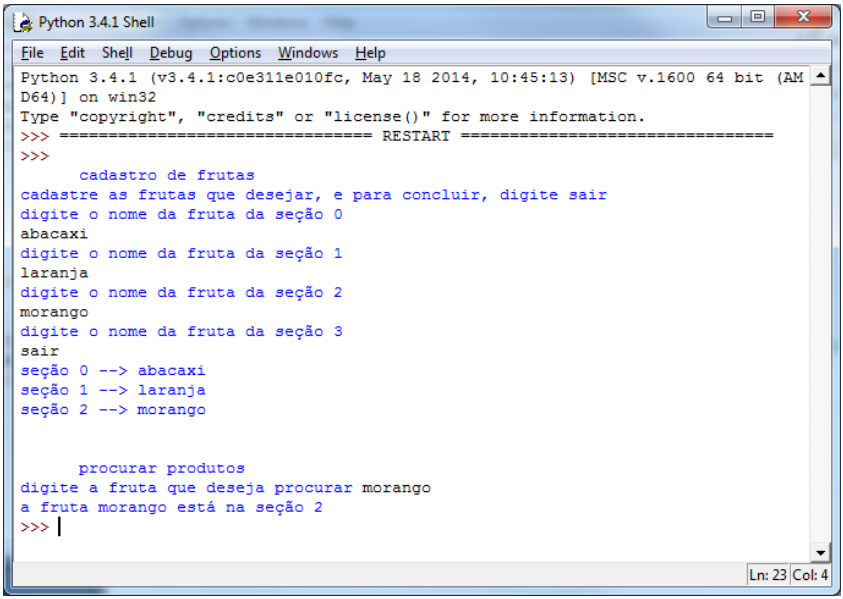
Para cada repetição o programa exibe uma mensagem informando o número da seção `i` e o nome da fruta que está na lista `frutas`. O nome da lista é seguido de colchetes contendo a variável `i` dentro. Como uma lista é uma variável com vários valores, não é possível exibir todos de uma vez, por isso cada elemento da lista recebe um índice, sequencialmente, começando por 0. Assim, a cada ciclo de repetição do `for` é exibida uma palavra que foi cadastrada pelo usuário.

O aplicativo dá a opção de procurar por uma palavra cadastrada (linha 15). Ele salta duas linhas (`\n\n`) e exibe a mensagem "`Procurar produtos`". Logo após solicita ao usuário que digite a palavra que deseja procurar.

Na linha 16 percorre-se toda a lista com o `for` e a variável `i`, indo de 0 até o tamanho total da lista dado pela função `len(frutas)`. Dentro desse laço há um comando `if` (linha 17) que verifica se o valor da variável `palavra_procurada`, que foi informada pelo usuário, corresponde à palavra que está na lista no índice `i`. Se a condição do `if` for atendida, é informado o nome da fruta e a posição dela dentro da lista (linha 18). Como o `for` continua a ser repetido mesmo se a palavra for achada, se a `palavra_procurada` estiver repetida na lista, ela será listada outras vezes.

Executando o programa, o resultado exibido será semelhante ao da Figura 98.

Figura 98 – Cadastro de frutas



PYGAME

O desenvolvimento de jogos eletrônicos se assemelha ao desenvolvimento de programas comuns. Porém, os jogos têm algumas características próprias que precisam ser tratadas pelo desenvolvedor, como imagens, animações, sons, colisões, entre outras. Além disso, outra diferença de um programa comum é que eles executam continuamente: normalmente possuem um

laço de repetição principal que é repetido sem a necessidade de o usuário interferir no programa.

Uma parte delicada que precisa ser programada é a gráfica: é necessário posicionar imagens e fazer com que elas se movimentem. Ao lidar com o movimento de várias imagens, pode haver colisão entre elas. Quando há colisão entre personagens ou com obstáculos, é preciso identificá-los e programar o impacto que isso gera no jogo. Para facilitar o tratamento da parte gráfica dos `games` há muitos ambientes de programação específicos. Alguns são chamados de `engine` (ou motores de jogo), que abstraem a programação da parte gráfica do jogo e oferecem várias funções, como identificação de colisão, entre outras.

Para a linguagem Python, temos uma biblioteca chamada pygame desenvolvida especificamente para auxiliar no desenvolvimento de jogos. Dentre as suas características principais estão: suporte a fontes, imagens, sons, `sprites`, detecção de colisões, eventos etc. Assim, a biblioteca tem vários comandos pré-programados para serem utilizados, o que torna o desenvolvimento mais rápido e fácil.

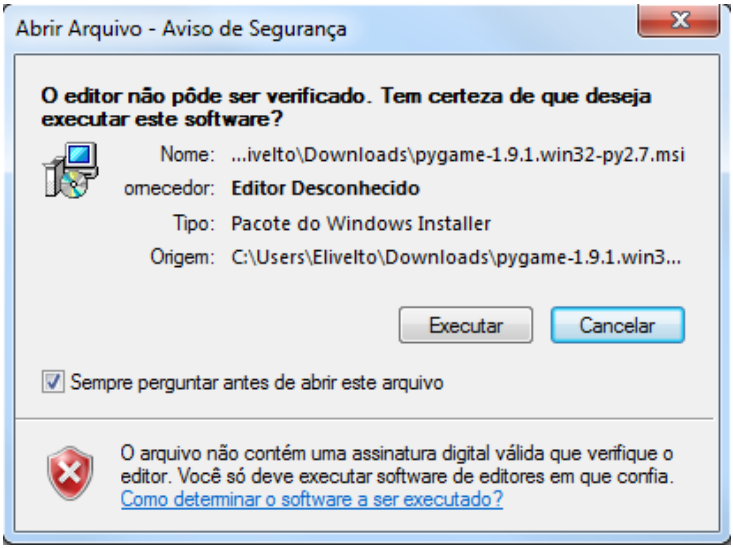
INSTALANDO O PYGAME

Para utilizar a biblioteca `pygame` é necessário primeiro instalar a versão compatível do Python. No momento a última versão que suporta a biblioteca é a `2.7.8`, que pode ser baixada em `https://www.python.org/downloads/release/python-278/`. A instalação no Windows é feita da mesma forma que a outra versão, descrita no capítulo anterior.

Após instalar o Python 2.7.8, é necessário instalar a biblioteca pygame, disponível em: `http://pygame.org/ftp/pygame-1.9.1.win32-py2.7.msi`. A instalação para Windows é feita desta forma:

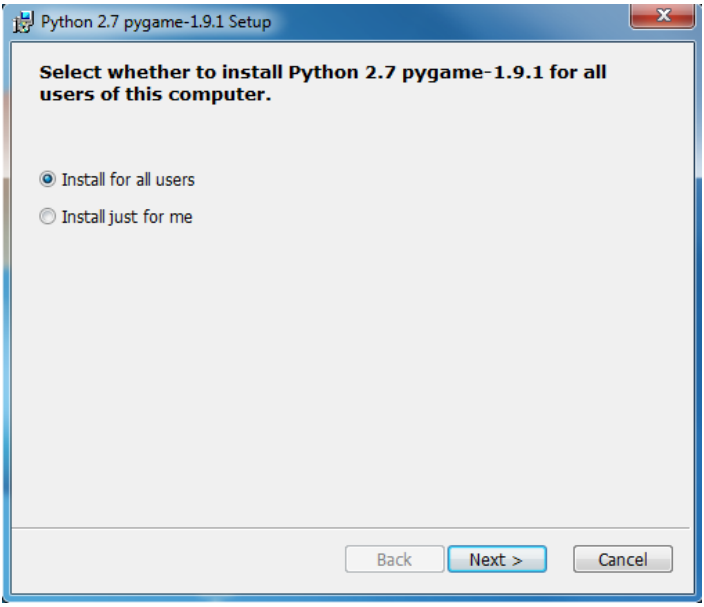
1. Abra o arquivo baixado e clique em executar, como mostra a Figura 99.

Figura 99 – Processo de instalação do Pygame I



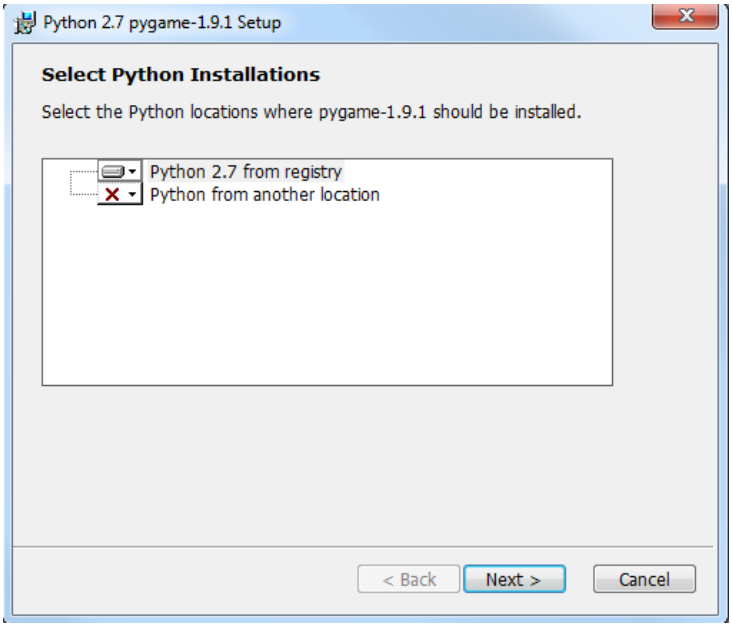
2. Como mostra a Figura 100, escolha para quais usuários vai instalar e clique em `next`.

Figura 100 – Processo de instalação do Pygame II



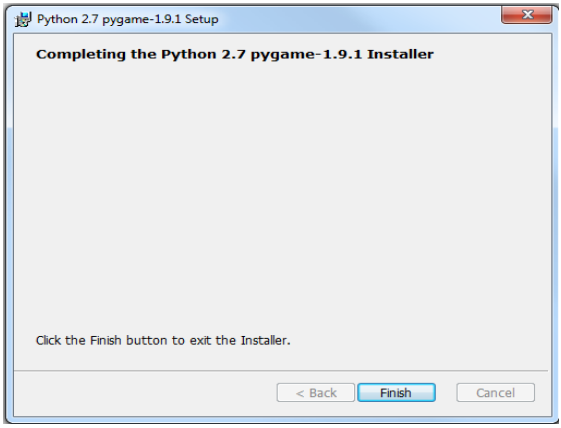
3. Selecione onde deseja instalar e clique em `next`, assim como mostra a Figura 101.

Figura 101 – Processo de instalação do Pygame III



4. Assim como a Figura 102, clique em `Finish`.

Figura 102 – Processo de instalação do Pygame IV



TELA

A primeira coisa que temos que definir ao começar a produção de um jogo é a tela. Nela são inseridos todos os elementos gráficos do jogo com os quais o jogador consegue interagir e visualizar a saída de dados do programa. Uma tela possui duas dimensões: horizontal e vertical. Ao começar escrever um jogo em pygame é necessário informar esses dois valores. No exemplo abaixo é mostrado como iniciar uma tela em pygame.

Exemplo 22 – Tela

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()
4    tela = pygame.display.set_mode([600,
     400])
5    continuar = True
6    while continuar == True:
7        pygame.display.update()
8    pygame.quit()
```

As duas primeiras linhas importam as bibliotecas `pygame`, `sys` e `pygame.locals`, que contêm várias constantes e funções que serão muito utilizadas nos jogos.

Após isso, iniciamos o pygame com o comando `pygame.init()` e criamos uma variável chamada `tela`, do tipo `display`, pois recebe `pygame.display`. Este comando tem a função de instanciar uma nova janela ou tela. Esse objeto tem um método, ou seja, uma função em que passamos alguns valores para alterar algumas de suas propriedades. O método é `set_mode()` e, dentro dos parênteses, passamos os valores `[600,400]`, que correspondem à largura e à altura da tela respectivamente; se forem passados valores diferentes, o tamanho da janela será alterado.

Em seguida, criamos uma variável chamada `continuar`, atribuindo-lhe o valor `True` (verdadeiro). Os jogos em pygame têm um laço de repetição principal e,

enquanto ele estiver ativo, o jogo continua executando. Nosso laço é um comando `While` que executa enquanto a variável `continuar` for verdadeira. Abaixo do `While` temos um só comando pertencente a ele, `pygame.display.update()`, cuja função é atualizar as imagens da tela.

Quando o programa sair do laço de repetição `While` (quando a variável continuar for `falsa`), chegaremos ao comando `pygame.quit()`, que encerra o jogo. Porém, no jogo acima, a variável continuar não é alterada dentro do `While`; assim, a instrução `pygame.quit()` não é executada, portanto, mesmo se clicando no botão `fechar` da janela, o jogo não finalizará. Para fechar, clique no botão `Fechar` do `Shell` do pygame.

Uma tela tem que ter a opção de fechar, senão gera uma sensação incômoda para o usuário. O clique na opção de fechar da janela é o que chamamos de `evento`. O conceito de `evento` será explicado mais à frente, por ora vamos somente programar a reação ao evento de fechar a janela do aplicativo. O programa com essa alteração fica assim:

Exemplo 23 – Tela com evento de sair

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init() #inicia o modulo pygame
4    tela = pygame.display.set_mode([600,
     400]) #define o tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
```

```
6     while continuar == True: #laço de
      repetição
7         for evento in pygame.event.get():
      #captura todos os eventos
8             if evento.type == pygame.QUIT:
      #verifica se é o evento de fechar o
      programa
9                 continuar = False
10        pygame.display.update()#atualiza o
      display
11    pygame.quit()#encerra o jogo
```

O programa é o mesmo anteriormente apresentado, porém permitindo ao usuário fechar a tela. A diferença está nas linhas 7, 8 e 9, dentro do `while`:

Na linha 7 temos um comando de repetição `for` que percorre todos os eventos do pygame que estão ocorrendo no momento, com o comando `pygame.event.get()`. A variável `evento` recebe todos esses eventos, um de cada vez. Na linha 8, com o comando `if`, o evento é comparado: se for do tipo `pygame.QUIT` (que detecta que o usuário clicou no botão `Fechar` da tela), então será executada a linha `continuar = False` (`continuar` recebe `falso`). Com a variável `continuar` valendo `False`, o programa sai do laço `While` e desce para o comando `pygame.quit()`, que termina o jogo.

CORES

Nos programas apresentados acima, temos apenas uma janela preta aparecendo. Um jogo normalmente traz uma interface gráfica bem colorida. Para conseguir tal efeito é necessário aprender a trabalhar com cores.

As cores estão no formato RGB (`Red`, `Green` e `Blue`), ou seja, são formadas pelas combinações de vermelho, verde e azul. O tom de cada uma dessas cores varia de 0 a 255. O vermelho é composto por (255,0,0): 255 de vermelho, 0 de verde e 0 de azul. Variando esses três valores, podemos definir todas as cores possíveis. O programa a seguir mostra como mudar a cor da tela:

Exemplo 24 - Cores

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia o modulo pygame
4    tela = pygame.display.set_mode([600,
     400])#define o tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    branco = (255,255,255)#cor branca
7    azul = (0,0,255) #cor azul
8    verde = (0,255,0) #cor verde
9    vermelho = (255,0,0) #cor vermelha
```

```
10  while continuar == True: #laço de repetição
11      for evento in pygame.event.get():#captura todos os eventos
12          if evento.type == pygame.QUIT: #verifica se é o evento de fechar programa
13              continuar = False
14      tela.fill(branco)#define a cor da tela como branca
15      pygame.display.update()#atualiza o display
16  pygame.quit()#encerra o programa
```

O programa acima é o mesmo anteriormente apresentado, porém a cor de fundo da tela é branca. Ele também inicializa algumas cores que poderão ser utilizadas.

Nas linhas 6, 7, 8 e 9 criamos quatro variáveis do tipo cor, às quais demos nomes de cores. Declarar essas variáveis não muda as cores da tela; isto só ocorrerá quando as usarmos dentro dos métodos que usam cores. À primeira demos o nome de `branco` e lhe atribuimos três valores 255 (o valor máximo de cada cor, já que a cor branca é a união de todas as cores). Para as outras variáveis (`azul`, `verde` e `vermelho`) atribuimos 255 para a sua posição (dentro do RGB) e 0 para as demais.

Nesse programa as linhas que alteram a cor de fundo da tela são a 14 e 15. Os comandos dessas linhas encontram-se

dentro do laço principal `While`. A variável `tela` tem um método `fill`(), ao qual deve ser passada uma cor como parâmetro. O parâmetro pode ser uma variável do tipo cor, como fizemos, `tela.fill(branco)`; ou pode ser um valor de uma cor: `tela.fill((255,255,255))`; o resultado de ambos será o mesmo. Após informar a cor de fundo, temos que pedir ao `pygame` que atualize a imagem da tela, o que é feito com o comando `pygame.display.update()`.

Como exercício, tente mudar para outra cor de fundo do programa para ver a diferença. Utilize por exemplo `tela.fill(azul)`, ou `tela.fill(verde)`, ou `tela.fill(vermelho)`. Você também pode definir outras variáveis do tipo cor (alterando os três valores entre 0 e 255), e depois passá-las para o método `tela.fill`(). Você encontra a tabela de cores RGB em vários `sites` da Internet.

IMAGENS

Trabalhar com cores torna o jogo mais agradável visualmente, porém um jogo precisa de mais elementos gráficos, como imagens. A biblioteca `pygame` permite carregar imagens e colocá-las no jogo. Pegue a imagem de uma bola no formato `png` com dimensões menores do que as da tela (para caber sem problemas) e coloque-a na mesma pasta em que se encontra o programa. Esse arquivo deve receber o nome bola.png. O código a seguir utiliza essa imagem no jogo:

Exemplo 25 – Imagens

```
1   import pygame, sys
2   from pygame.locals import *
3   pygame.init()#inicia o módulo pygame
4   tela = pygame.display.set_mode([600,
    400])#define o tamanho da janela
5   continuar = True #variável que
    controla o laço de repetição
6   branco = (255,255,255)#cor branca
7   imagem =
    pygame.image.load("bola.png")#carrega
    a imagem
8   posicao = (100,100) #coordenadas
9   while continuar == True: #laço de
    repetição
10      for evento in
    pygame.event.get():#captura todos os
    eventos
11          if evento.type == pygame.QUIT:
    #verifica se é o evento de fechar
12              continuar = False
13      tela.fill(branco) #pinta a tela
    com a cor branca
14      tela.blit(imagem, posicao) #coloca
    a imagem na tela
15      pygame.display.update()#atualiza o
    display
```

```
16    pygame.quit()#encerra o programa
```

Esse programa carrega o arquivo com a imagem `bola.png`. É importante colocar o nome do arquivo corretamente, senão o programa não o encontra. Essa imagem é colocada na posição (100,100) e com o fundo branco.

Nessas 6, 7 e 8 iniciamos a parte gráfica do jogo. Primeiramente criamos uma variável do tipo cor chamada `branco` que será utilizada como pano de fundo. Na linha seguinte criamos uma variável do tipo imagem, e com o nome `imagem`, que irá receber a imagem desejada, carregada pelo comando `pygame.image.load()`. Dentro dos parênteses informamos o nome do arquivo de imagem entre aspas. Caso o arquivo esteja em outro diretório, informamos o `path` (caminho) e o nome do arquivo em uma única `string`.

Para definir onde a imagem irá se posicionar na tela criamos uma variável chamada `posição`, que contém duas coordenadas, a primeira horizontal e a segunda vertical. O ponto (0,0) encontra-se no canto superior esquerdo da tela; a medida `horizontal` aumenta quando o ponto se desloca para a direita; a medida vertical aumenta quando o ponto se desloca para `baixo`. Assim, o ponto (100,100) encontra-se 100 pixels à direita e 100 pixels abaixo do canto superior esquerdo da tela.

Até aqui, porém, apenas inicializamos as variáveis; elas ainda não fazem parte do laço principal. Para inserir a imagem na tela, usamos as linhas 13, 14 e 15.

A linha 13 define o pano de fundo: o comando `tela.fill`(branco) é responsável por isso. É importante colocar na tela primeiro o pano de fundo e depois a imagem;

caso contrário, a imagem ficará invisível, debaixo do pano de fundo. Na linha 14 o comando `tela.blit()` monta a imagem a ser colocada na tela. Ele possui dois parâmetros a serem informados: a imagem a ser carregada e a posição onde deve ser colocada. Como já carregamos a imagem na variável `imagem` e definimos a posição na variável `posicao`, passamos essas duas variáveis como parâmetros: `tela.blit(imagem, posicao)`. Para que a cor de fundo e a imagem sejam mostradas, atualizamos a tela com o comando `pygame.display.update()`.

Podemos também colocar uma imagem como pano de fundo em vez de simplesmente uma cor. O código a seguir exemplifica isso.

Exemplo 26 – Pano de fundo

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia o modulo pygame
4    tela = pygame.display.set_mode([600,
     400]) #define o tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    pano_de_fundo =
     pygame.image.load("fundo.jpg")#carrega
     a imagem de fundo
7    imagem = pygame.image.load("bola.png")
     #carrega a imagem da bola
8    posicao_fundo = (0,0) #coordenadas da
     imagem de fundo
```

```
9   posicao_imagem = (100,100) #coordenadas da bola
10  while continuar == True: #laço de repetição
11      for evento in pygame.event.get():#captura todos os eventos
12          if evento.type == pygame.QUIT: #verifica se é o evento de sair
13              continuar = False
14      tela.blit(pano_de_fundo, posicao_fundo)#coloca a imagem de fundo na tela
15      tela.blit(imagem, posicao_imagem) #coloca a imagem da bola na tela
16      pygame.display.update()#atualiza o display
17  pygame.quit()#encerra o programa
```

Nesse programa substituímos o fundo branco por uma imagem. É desejável que a imagem de fundo tenha as mesmas dimensões que a tela. O programa é feito da mesma forma que o anterior:

Carregamos as imagens em duas variáveis e definimos as posições que devem ocupar na tela. Como são duas imagens, precisamos de duas localizações. A primeira é a do pano de fundo (linha 8), que deve iniciar no ponto (0,0) para cobrir toda a tela, pois, em qualquer outro ponto, parte da tela ficará sem a

imagem. Depois colocamos a imagem da bola no ponto (100,100).

Montamos a imagem do pano de fundo primeiro (linha 14), depois montamos a segunda imagem (linha 15), de forma que o fundo não encubra a imagem. Para mostrar a imagem na tela, atualizamos o `display` (linha 16).

TEXTO

Além de cores e imagens, muitos jogos apresentam textos com mensagens para os usuários. Eles podem dar instruções, mostrar algum estado do jogo ou informar quando o usuário perde ou ganha. Vamos ver um exemplo com o seguinte código:

Exemplo 27 – Texto

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia o modulo pygame
4    tela = pygame.display.set_mode([600,
     400])        #define o tamanho da
     janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    branco = (255,255,255) #cor branca
7    azul=(0,0,255) #cor azul
```

```
8     fonte =
      pygame.font.Font("freesansbold.ttf",
      20) #define tipo e tamanho da fonte
9     texto = fonte.render("Olá, bem-vindo
      ao Pygame", True,azul)   #texto e cor
      da fonte
10    posicao = (100,100) #posição do texto
11    while continuar == True: #laço de
      repetição
12        for evento in
      pygame.event.get():#captura todos os
      eventos
13            if evento.type == pygame.QUIT:
      #verifica se é o evento de fechar
14                continuar = False
15        tela.fill(branco) #pinta a tela
      com a cor branca
16        tela.blit(texto,posicao) #colocao
      texto na tela
17        pygame.display.update()#atualiza o
      display
18    pygame.quit()#encerra o programa
```

Esse programa exibe a mensagem "`Olá, bem-vindo ao pygame`" na tela. É necessário definir a fonte, o tamanho e a cor da letra, além da posição do texto na tela.

Com o comando `pygame.font.Font()` na linha 8, definimos uma variável do tipo fonte e com o nome `fonte`,

passando como parâmetros o nome e o tamanho da fonte; foi utilizada a fonte `freesanbold` com tamanho 20. Após isso, utilizamos o comando método `render()` na linha 9, para editar a mensagem; esse método tem três parâmetros: o primeiro é a mensagem que deve ser passada, entre aspas; o segundo deve ser deixado como `True` por padrão; o terceiro é a cor da letra. O texto precisa posicionar-se na tela, então definimos uma variável `posicao` para isso na linha 10.

No laço de repetição principal: definimos a cor de fundo como branco, depois montamos a imagem do texto posicionando-a no lugar definido com o método `blit()` na linha 16, e finalmente pedimos que o `pygame` atualize a tela através do comando `pygame.display.update()`.

EVENTOS

Um jogo é composto por elementos gráficos e pela lógica de programação; porém, sua principal característica é a interação constante com o usuário. Numa animação ou em um filme, a pessoa apenas acompanha o conteúdo, porém tudo já está definido e não pode ser modificado. No jogo o usuário tem a oportunidade de interferir de diversas formas, tomar decisões e mudar caminhos e o resultado. É o jogador que define o andamento, ele faz parte da história do game.

O que determina a interação com o usuário são os eventos. Um evento é uma ação que o jogador realiza durante o jogo, por exemplo, um clique do mouse. Os principais eventos vêm do teclado e do mouse. Os eventos de mouse normalmente são os cliques em seus botões, ou a mudança da posição do

ponteiro na tela, muito usada em jogos de acertar algum alvo com auxílio de uma mira. Os eventos de teclado são disparados quando se pressiona ou quando se solta alguma tecla; como um teclado tem várias teclas, podem ser programadas muitas ações diferentes, pois o evento detectado para cada tecla pode prever uma ação diferente.

EVENTO CLIQUE DO MOUSE

Normalmente o mouse possui três botões: o esquerdo, central (rolagem) e o direito. O seguinte código captura o evento do clique desses botões:

Exemplo 28 – Clique do mouse

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia o modulo pygame
4    tela = pygame.display.set_mode([600,
     400]) #define o tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    branco = (255,255,255)#cor branca
7    verde = (0,255,0) #cor verde
8    azul = (0,0,255) #cor azul
9    laranja = (255,127,0) #cor laranja
```

```
10  cor = branco #define cor inicial
11  while continuar == True: #laço de repetição
12      if pygame.mouse.get_pressed()[0] == True: #botão esquerdo pressionado
13          cor = verde #muda cor para verde
14      if pygame.mouse.get_pressed()[1] == True:#botão central pressionado
15          cor = azul #muda cor para azul
16      if pygame.mouse.get_pressed()[2] == True: #botão direito pressionado
17          cor = laranja #muda cor para laranja
18      for evento in pygame.event.get():#captura todos os eventos
19          if evento.type == pygame.QUIT: #verifica se é o evento de sair
20              continuar = False
21      tela.fill(cor)#pinta a tela da cor da variável cor
22      pygame.display.update()#atualiza o display
23  pygame.quit()#encerra o programa
```

Esse programa começa com o pano de fundo branco; se o usuário clicar com o botão esquerdo, a tela fica verde; se for o botão central a cor muda para azul; se for o botão direito, muda para laranja.

No primeiro momento, inicializamos as variáveis do tipo cor: `branco`, `azul`, `verde` e `laranja`. Após isso criamos outra variável, de nome `cor`, que inicialmente recebe a cor branca. De acordo com os cliques do usuário, essa variável alterará seu valor, recebendo as outras cores.

O comando responsável por capturar os eventos do mouse é `pygame.mouse.get_pressed()`. Esse comando devolve uma resposta com três valores booleanos (verdadeiro ou falso). Se o botão esquerdo for pressionado, a resposta é (`True, False, False`), ou seja, verdadeiro para o primeiro valor e falso para os dois restantes. Para saber se um botão específico está clicado, usamos o comando dessa forma: `pygame.mouse.get_pressed()[n]`; entre os colchetes colocamos o número do botão que queremos saber se foi clicado: 0: esquerdo, 1: meio e 2: direito. No código acima, dentro do laço principal temos três comandos perguntando se aquele botão foi clicado: se foi o esquerdo (linha 12), a variável cor recebe o valor da cor verde (linha 13); se foi o do meio (linha 14), o valor será azul (linha 15); se foi o da direita (linha 16), o valor será laranja (linha 17).

Na linha 21, é atualizada a cor da tela para o valor atual da variável cor, e a linha 22 implementa essas mudanças.

EVENTO POSIÇÃO DO MOUSE

Outro evento importante é a posição do mouse. Com ele podemos identificar onde o jogador está com o mouse na tela. O jogo pode ter áreas onde pode clicar e áreas onde não pode. Há ainda a possibilidade de movimentar o personagem com o mouse.

O comando para capturar a posição do mouse na tela é `pygame.mouse.get_pos()`. O código a seguir mostra um exemplo:

Exemplo 29 – Posição do mouse

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init() #inicia modulo pygame
4    tela = pygame.display.set_mode([600,
     400])#define tamanho da janela
5    continuar = True #variável que
     controla fluxo de repetição
6    branco = (255,255,255) #cor branca
7    verde = (0,255,0) #cor verde
8    azul = (0,0,255) #cor azul
9    cor = branco #cor inicia com branco
10   while continuar == True: #laço de
     repetição
```

```
11      posicao_mouse = pygame.mouse.get_pos()#pega posição do mouse
12      if(posicao_mouse[0] <= 300): #verifica posição do mouse
13          cor = azul #muda cor para azul
14      if(posicao_mouse[0] > 300): #verifica posição do mouse
15          cor = verde #muda cor para verde
16      for evento in pygame.event.get(): #captura todos os eventos
17          if evento.type == pygame.QUIT: #verifica se é o evento de fechar
18              continuar = False
19      tela.fill(cor) #pinta a tela numa determinada cor
20      pygame.display.update()#atualiza o display
21  pygame.quit()#encerra o programa
```

Nesse programa, quando o mouse está na parte esquerda da tela, o fundo fica azul; quando passamos para o outro lado a cor muda para verde. Primeiramente definimos as cores branca, verde e azul. Logo após inserimos uma variável chamada `cor` e atribuímos o valor da cor branca a ela.

O comando na linha 11, `pygame.mouse.get_pos()` retorna a posição do mouse em

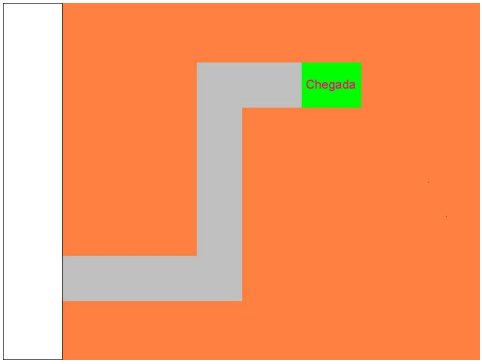
duas coordenadas: horizontal e vertical. Dessa forma inserimos uma variável chamada `posicao_mouse` para receber essas coordenadas. Essa variável é um vetor de duas posições. Para acessar a posição horizontal do mouse utilizamos `posicao_mouse[0]` e para acessar a posição vertical usamos `posição_mouse[1]`.

Na linha 12 temos uma condição: se a posição horizontal do mouse (`posicao_mouse[0]`) for menor ou igual a trezentos, a cor do fundo recebe azul (linha 13). Como definimos a largura da tela em 600 pixels na linha `tela = pygame.display.set_mode([600, 400])`, toda posição inferior a 300 faz parte da metade esquerda da tela. Da mesma forma, quando a posição do mouse for maior que 300 (linha 14), na metade direita da tela, a cor de fundo passará a ser verde (linha 15). No final do programa estabelecemos a cor do fundo com o comando `tela.fill()` e atualizamos a tela com `pygame.display.update()`.

JOGO 1 – MOUSE GAME

Vamos fazer agora um exemplo mais completo, desenvolvendo um jogo com o nome de `Mouse Game`. O objetivo dele é passar a bola por um caminho sem tocar as laterais. Primeiramente temos que desenhar o pano de fundo com o caminho que o jogador deve percorrer. A Figura 103 ilustra um caminho que o jogador poderá percorrer.

Figura 103 - Pista do jogo



A imagem é composta por um retângulo branco na lateral esquerda da tela, formado por quatro pontos cujas coordenadas são (0,0), (100,0), (0,600) e (100,600). O caminho a ser percorrido é formado por três retângulos cinzas. O primeiro é formado pelos pontos (100, 425), (400, 425), (100, 500) e (400, 500); o segundo, pelos pontos (325, 425), (400, 425), (325, 175) e (400, 175); o terceiro, pelos pontos (325, 175), (500, 175), (325, 100) e (500, 175). Na chegada há um retângulo verde formado pelos pontos (500, 100), (600, 100), (500, 175) e (600, 175); dentro desse retângulo está escrito a palavra "`Chegada`" em laranja; quando o jogador atinge este retângulo ele vence o jogo. O restante da imagem está na cor laranja; se o jogador tocar essa região, ele perde. Essa imagem pode ser desenhada com um aplicativo de edição de imagens. Grave o desenho com o nome `pista.jpg`.

Após desenhar o pano de fundo, temos que pegar a imagem de uma bola na Internet ou desenhar uma. É importante redimensioná-la para o tamanho 50 por 50 pixels, pois a largura do caminho é de 75 pixels; assim, passar a bola pelo caminho

não se torna nem muito difícil e nem muito fácil. A Figura 104 nos mostra um exemplo de bola para utilizarmos no nosso projeto. Grave a imagem com o nome `bola.png`.

Figura 104 – Bola

Fonte : Wikimedia (Wikimedia.org)

Essas duas imagens devem estar na mesma pasta do arquivo do jogo, para facilitar o carregamento das imagens.

O código do jogo é o seguinte:

Jogo 1 – Mouse game

```
1    import pygame , sys #Importação das
     bibliotecas
2    from pygame.locals import *
3    pygame.init()#inicia o módulo pygame
4    azul = (0,0,255) #inicia uma variável
     com a cor azul
5    tela =
     pygame.display.set_mode((800,600))
     #define as dimensões da tela
```

```
6    fonte =
     pygame.font.Font("freesansbold.ttf",20
     ) #define a fonte a ser utilizada
7    texto = fonte.render("Chegue até o
     final sem tocar na região
     Laranja!",True,azul)
8    pista = pygame.image.load('pista.jpg')
     #carrega a imagem da pista
9    bola = pygame.image.load('bola.png')
     #carrega a imagem da bola
10   continuar = True #variável que
     controla primeiro laço de repetição
11   exibir_resultado = True #variável que
     controla o segundo laço de repetição
12   while continuar == True: #laço de
     repetição
13       tela.blit(pista, (0,0)) #coloca a
     imagem da pista na tela
14       tela.blit(texto, (100,0)) #coloca
     o texto com as instruções na tela
15       posicao =
     pygame.mouse.get_pos()#pega posição do
     mouse
16       posicao_horizontal = posicao[0]
     #recebe posição horizontal do mouse
17       posicao_vertical = posicao[1]
     #recebe posição vertical do mouse
18
     tela.blit(bola,(posicao_horizontal,pos
```

```
    icao_vertical)) #coloca a bola na
    tela, na posição do mouse
19      if(posicao_horizontal > 500 and
    posicao_horizontal <= 600 and
20      posicao_vertical  > 100 and
    posicao_vertical <=175): #verifica se
    chegou ao final da pista
21          texto_resultado = "Parabéns!!!
    você ganhou!" #informa que o jogador
    venceu
22          continuar = False
23      elif(posicao_horizontal <= 50 or
24      (posicao_horizontal > 100-50 and
    posicao_horizontal <= 400-50 and
25       posicao_vertical > 425 and
    posicao_vertical <= 500 - 50)or
26      (posicao_horizontal > 325 and
    posicao_horizontal <= 400-50 and
27       posicao_vertical > 175-50 and
    posicao_vertical <=425)or
28      (posicao_horizontal > 325 and
    posicao_horizontal <= 500 and
29       posicao_vertical > 100 and
    posicao_vertical <= 175-50)):
    #verifica se a bola está dentro do
    percurso ou da zona permitida
30          continuar = True #o laço
    continua executando
```

```
31      else: #caso a bola esteja fora da zona permitida, ou seja, na região laranja
32          texto_resultado = "Você perdeu!" #informa ao jogador que ele perdeu
33          continuar = False
34      for evento in pygame.event.get():#captura todos os eventos
35          if evento.type == QUIT: #verifica se é o evento de sair
36              continuar = False #o laço principal não executará mais
37              exibir_resultado = False #o programa não executa segundo laço
38      pygame.display.update() #atualiza display
39  while exibir_resultado == True: #segundo laço de repetição
40      texto2 = fonte.render(texto_resultado,True,azul) #monta o texto que informa qual foi resultado do jogador
41      tela.blit(texto2,(100,100)) #coloca o texto com o resultado na tela
```

```
42          for event in
    pygame.event.get():#captura todos os
    eventos
43              if event.type == QUIT:
    #verifica se é o evento de sair
44                  exibir_resultado=False #o
    segundo laço de repetição não executa
    mais
45          pygame.display.update()#atualiza o
    display
46      pygame.quit()#encerra o programa
```

Nesse jogo, trabalhamos com imagens, textos e eventos de mouse. O programa tem também dois laços de repetição: o primeiro é o principal, no qual acontece a ação do game, e o segundo é para exibir o resultado do jogo.

A tela do jogo terá 800 pixels de largura e 600 de altura (linha 5). Será utilizada a fonte `freesansbold` de tamanho 20 (linha 6). Depois montamos a mensagem na variável `texto`, acessando o método `render` da fonte, passamos a mensagem "`Chegue até o final sem tocar a região laranja`" para orientar o jogador e definimos a cor da mensagem como azul (linha 7).

A imagem com o pano de fundo que contém a pista é carregada na variável `pista` (linha 8); a imagem da bola é carregada na variável `bola` (linha 9).

A variável `continuar` (linha 10) controla a repetição do laço em que acontecem as ações do jogo; ela é iniciada como `True` e, enquanto não receber `False`, o laço principal irá

executar. A variável `exibir_resultado` (linha 11) controla a repetição do segundo laço, utilizado apenas para mostrar o resultado ao jogador, se ele perdeu ou se ganhou. Essa variável também inicia como verdadeira, pois quando saímos do laço principal, vamos direto ao segundo laço.

O laço principal `while` (linha 12) executará enquanto a variável `continuar` for verdadeira. Todos os comandos ndentados à direita e anteriores ao segundo `while` pertencem a esse laço. Nas duas linhas seguintes (13 e 14) são montadas as imagens da pista e do texto. O comando `blit()` põe a imagem carregada da pista na posição (0,0), início da tela. Por cima do plano de fundo é carregada a imagem do texto na posição (100,0).

Para capturar a posição do mouse usamos o comando `pygame.mouse.get_pos()` e a colocamos na variável posicao (linha 15). Essa variável possui dois valores: a posição horizontal `posicao[0]` e a posição vertical `posicao[1]`. Criamos uma variável para guardar a posição horizontal, `posicao_horizontal` (linha 16), e outra para a vertical, `posicao_vertical` (linha 17).

Como temos as localizações horizontal e vertical do mouse e a imagem da bola carregada, já podemos montar a imagem da bola na tela (linha 18). A bola poderá ser movimentada pelo jogador, ficando sempre onde o mouse se encontrar na tela.

O jogador pode movimentar a bola, porém existem áreas da tela em que não pode tocar. Assim, temos que estabelecer condições para a movimentação da bola e os seus efeitos no jogo. Temos a primeira condição (linhas 19 e 20): quando a bola atinge a área verde, a chegada. Para estar nesse local, o mouse

deve ter sua posição horizontal maior que 500 e menor ou igual a 600, e a posição vertical deve ser maior que 100 e menor ou igual a 175 (veja as coordenadas usadas no desenho do retângulo verde – Figura 108). Se o mouse estiver nessa área, a variável `texto_resultado` recebe o texto "`Parabéns!!! Você ganhou`" (linha 21) e essa mensagem será exibida por um comando no segundo laço de repetição. A variável `continuar` passa a ser `falsa` (linha 22), para que o programa saia do laço principal.

Se a primeira condição não for atendida, o jogador não venceu; então temos que verificar se ele está no espaço permitido: o retângulo branco ou algum dos três retângulos cinzas. Na linha 23 é verificado se ele está no retângulo branco. Toda região com eixo horizontal inferior a 100 pixels é branca. Como a bola tem a largura de 50 pixels, então retirando 50 de 100, temos 50 pixels. Dessa forma, a bola pode percorrer toda região branca.

As linhas 24 e 25 testam se bola está dentro do primeiro retângulo cinza. Se a bola estiver entre 50 e 350 pixels horizontalmente e entre 425 e 450 pixels verticalmente, então estará dentro do retângulo. Sempre que descontamos 50 pixels é por causa do tamanho da bola, pois as coordenadas da bola se referem ao canto superior esquerdo dela. As linhas 26 e 27 testam se bola está dentro do segundo retângulo cinza e as linhas 28 e 29 testam se bola está dentro do terceiro retângulo cinza. Confira as coordenadas citadas em todas as condições citadas deste `elif` com o desenho feito na Figura 108.

Satisfazendo as condições (a bola se encontra na zona branca ou em algum retângulo cinza), a variável `continuar` permanece verdadeira (linha 30): o laço será repetido, permitindo continuar a movimentar a bola.

Nas condições anteriores, testadas no `if`, o jogador vencia; no `elif`, continuava jogando. Porém, se não satisfizer nenhuma delas significa que tocou em uma região laranja. Por isso, no comando `else` (linha 31) o jogador perde o jogo: exibimos a mensagem `você perdeu` (linha 32) e a variável `continuar` se torna `falsa` (linha 33) para que o programa saia do laço principal.

O usuário também tem a possibilidade de fechar o jogo antes que vença ou perca. Se houver um evento QUIT (linha 35), interrompemos o laço principal, atribuindo falso para a variável `continuar` (linha 36) e, para que o programa não exiba nenhum resultado no laço seguinte, atribuímos `False` para a variável `exibir_resultado` (linha 37).

Na linha 38 o display é atualizado. Se isso não for feito, as imagens montadas (com a última posição da bola) não serão exibidas.

Na linha 39 há o segundo laço de repetição. Só se entra nele depois que o primeiro for interrompido. Ele repete a imagem da tela para que o usuário veja o resultado; sem o laço, o programa mostraria o resultado rapidamente e fecharia. Dentro do `while`, é montado o resultado na variável `texto2`: passamos a variável `texto_resultado` para o método `render()` da fonte (linha 40), pois essa variável recebeu o texto "`Parabéns!!! Você ganhou!`" se o jogador chegou à região verde, ou "`Você perdeu`" se tocou na região laranja. O texto é exibido em azul e a imagem do texto é montada na posição (100,100).

Novamente é necessário capturar o evento "`fechar janela`" como foi feito no laço anterior. Quando o usuário fechar a janela, definimos a variável que controla o segundo

laço `exibir_resultado` como `False` (linha 44). Não podemos usar a variável `continuar`, pois ela entra nesse laço valendo `False`. Também é necessário atualizar o `display` na instrução seguinte, pois isso foi feito somente no primeiro `while`.

Quando as variáveis continuar e exibir_resultado são falsas é que se chega à linha 46, que encerra o jogo. Após executar o jogo o resultado será conforme a Figura 105:

Figura 105 – Jogo Mouse Game

EVENTOS DE TECLADO

Além dos eventos de mouse, importantes também são os eventos de teclado. O mouse tem apenas três botões, enquanto o teclado tem várias teclas. A variedade de ações que isso pode dar a um jogo é grande. Os principais eventos de teclado são: `KEYDOWN` e `KEYUP`: o primeiro ocorre quando pressionamos uma tecla e o segundo quando a soltamos. Também podemos verificar se a tecla está pressionada. Para verificar quando o usuário pressiona uma tecla, é necessário capturar todos os eventos com o comando `pygame.event.get()`, que já utilizamos. Vamos ver um exemplo com o seguinte código:

Exemplo 30 – Evento pressionar tecla

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia o modulo pygame
4    tela = pygame.display.set_mode([800,
     600]) #define o tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    branco = (255,255,255) #cor branca
7    bola = pygame.image.load("bola.png")
     #carrega imagem
8    posicao_horizontal = 0
9    posicao_vertical = 0
```

```
10  while continuar == True: #laço de repetição
11      for evento in pygame.event.get():#captura todos os eventos
12          if evento.type == pygame.QUIT: #verifica se é o evento de fechar
13              continuar = False
14          if evento.type == pygame.KEYDOWN: #tipo do evento:pressionar tecla
15              if evento.key == K_UP: #se tecla pressinada é seta para cima
16                  posicao_vertical = 0
17              elif evento.key == K_DOWN: #se tecla pressionada é seta para baixo
18                  posicao_vertical = 550
19              elif evento.key == K_RIGHT: #se tecla pressionada é seta para direita
20                  posicao_horizontal = 750
21              elif evento.key == K_LEFT: #se tecla pressionada é seta para esquerda
22                  posicao_horizontal = 0
23      tela.fill(branco)#pinta a tela com a cor branca
```

```
24
      tela.blit(bola,(posicao_horizontal,pos
      icao_vertical)) #colaca a imagem na
      tela
25        pygame.display.update() #atualiza
      o display
26    pygame.quit()#encerra o programa
```

Esse programa movimenta a bola, posicionando-a nos cantos da tela.

Carregamos a imagem de uma bola (dimensões 50 x 50 pixels) na linha 7. A seguir criamos duas variáveis para a posição da bola na tela (`posicao_horizontal` e `posicao_vertical`) e definimos ambas iniciando em zero para que a imagem da bola comece no canto superior esquerdo da tela (linhas 8 e 9).

Na linha 11 capturamos todos os eventos ocorridos. O primeiro evento que verificamos é se o usuário clicou para fechar a janela do jogo (linhas 12 e 13). As linhas 14 a 22 verificam o uso das teclas de setas. A linha 14 verifica se alguma tecla foi pressionada: se o evento é do tipo `pygame.KEYDOWWN`. Se for, é necessário identificar qual tecla disparou esse evento. Isso é feito através do comando `evento.key`, que informa a tecla. Os nomes das teclas normalmente têm "`K_`" no início e depois o seu nome em inglês. Por exemplo a letra "`a`" é representada por "`K_a`", a tecla espaço é representada por "`K_SPACE`"; a lista com todas as teclas pode ser encontrada no site oficial (`http://www.pygame.org/docs/ref/key.html`).

No nosso código, se houver uma tecla pressionada, verificamos primeiramente, na linha 15, se é a seta para cima (`K_UP`); se for, atribuímos 0 para a variável `posicao_vertical`(linha 16), para que a bola fique na parte de cima da tela. Se for a seta para baixo (`K_DOWN`), a bola irá ficar na parte inferior da tela (linhas 17 e 18): a variável `posicao_vertical` recebe 550 (esse valor corresponde à altura total da tela, 600 pixels, menos a altura da bola, 50 pixels). Se a tecla for seta para a direita (`K_RIGHT`), a bola se posicionará no lado direito da tela (linhas 19 e 20): a variável `posicao_horizontal` receberá 750 (largura da tela menos a largura da bola). Se a tecla for seta para a esquerda (`K_LEFT`), a variável `posicao_horizontal` receberá 0: a bola ficará no lado esquerdo da tela (linhas 21 e 22).

Nas linhas 23 a 26, aplicamos branco para a cor de fundo, montamos a imagem da bola, posicionando-a de acordo com as variáveis `posicao_horizontal` e `posicao_vertical` e atualizamos o display para que as imagens sejam exibidas.

Para utilizar o evento de soltar a tecla, o código é bem parecido com esse; veja o exemplo no seguir:

Exemplo 31 – Evento soltar tecla

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init() #inicia o módulo pygame
4    tela = pygame.display.set_mode([800,
     600])#define o tamanho da janela
```

```
5    continuar = True #variável que controla o fluxo de repetição
6    branco = (255,255,255) #cor branca
7    azul = (0,0,255) #cor azul
8    verde = (0,255,0) #cor verde
9    cor = branco #define a cor inicialmente como branca
10   while continuar == True: #laço de repetição
11       for evento in pygame.event.get():#captura todos os eventos
12           if evento.type == pygame.QUIT: #verifica se é o evento de sair
13               continuar = False
14           if evento.type == pygame.KEYDOWN: #se é o evento pressionar tecla
15               if evento.key == K_SPACE: #se tecla é barra de espaço
16                   cor = azul #muda cor para azul
17           if evento.type == pygame.KEYUP: #se é o evento de soltar tecla
18               if evento.key == K_SPACE: #se tecla é barra de espaço
```

```
19                  cor = verde #muda cor
    para verde
20      tela.fill(cor)  #pinta a tela na
    cor da variável no momento
21      pygame.display.update()  #atualiza
    o display
22  pygame.quit()#encerra o programa
```

Nesse programa, quando a tecla espaço é pressionada (`KEYDOWN`), o pano de fundo muda para a cor azul; quando ela é solta, a cor muda para verde (`KEYUP`).

Primeiramente definimos as cores que serão utilizadas: branco, azul e verde, linhas 6 a 8. A variável `cor` inicia com branco (linha 9).

Na linha 11 capturamos todos os eventos ocorridos. O primeiro evento que tratamos é se o usuário clicou para fechar a janela do jogo (linhas 11 a 13). As linhas 14 a 19 verificam o uso da tecla de espaço.

Se o evento for do tipo tecla pressionada (`KEYDOWN`) e se a tecla for espaço (`K_SPACE`), a variável `cor` passa a ser azul (linhas 14 a 16).

Se o evento for do tipo `soltar a tecla` (`KEYUP`) e se a tecla for espaço (`K_SPACE`), atribuímos verde à variável `cor` (linhas 17 a 19).

A cor da variável `cor` é aplicada ao pano de fundo da tela pelo método `fill`(); em seguida, atualiza-se o `display`.

Esses eventos `KEYDOWN` e `KEYUP` são disparados somente no momento em que o usuário pressiona ou solta uma

tecla. Se a pessoa pressioná-la e segurá-la, o programa fará a ação programada somente no momento em que pressionar a tecla; se a tecla for mantida pressionada, o programa não entenderá como um novo evento `KEYDOWN`.

Para verificar se uma tecla está pressionada, usamos o comando `pygame.key.get_pressed()`, que informa quais teclas estão pressionadas. Normalmente usamos uma variável para receber a informação das teclas pressionadas e perguntamos se são as teclas que queremos. Um exemplo desse emprego é mostrado no seguinte código:

Exemplo 32 – Evento tecla pressionada

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init() #inicia o modulo pygame
4    tela = pygame.display.set_mode([800,
     600]) #define o tamanho da janela
5    continuar = True # variável que
     controla o fluxo de repetição
6    branco = (255,255,255) #cor branca
7    bola =
     pygame.image.load("bola.png")#carrega
     imagem
8    posicao_horizontal = 0
9    posicao_vertical = 0
10   while continuar == True: #laço de
     repetição
```

```
11      for evento in
pygame.event.get():#captura todos os
eventos
12          if evento.type == pygame.QUIT:
#verifica se é o evento de sair
13              continuar = False
14      tecla_pressionada =
pygame.key.get_pressed()#recebe a
tecla pressionada
15      if tecla_pressionada[pygame.K_UP]
== True:#se a tecla for seta para cima
16          posicao_vertical =
posicao_vertical-1 #diminui em 1 a
posição vertical
17      if
tecla_pressionada[pygame.K_DOWN] ==
True: #se a tecla for seta para baixo
18          posicao_vertical =
posicao_vertical+1 #aumenta em 1 a
posição horizontal
19      if
tecla_pressionada[pygame.K_RIGHT] ==
True: #se a tecla for seta para
direita
20          posicao_horizontal =
posicao_horizontal+1 #soma mais 1 a
posição horizontal
21      if
tecla_pressionada[pygame.K_LEFT] ==
```

```
    True: #se a tecla for seta para
    esquerda
22              posicao_horizontal =
    posicao_horizontal-1 #subtrai 1 da
    posição horizontal
23          tela.fill(branco)#pinta a tela com
    a cor branca
24
    tela.blit(bola,(posicao_horizontal,pos
    icao_vertical)) #coloca a imagem em
    sua respectiva posição
25          pygame.display.update()#atualiza o
    display
26      pygame.quit()#encerra o programa
```

Nesse programa podemos movimentar a imagem de uma bola na tela utilizando as setas do teclado.

Primeiramente carregamos a imagem da bola e definimos as variáveis de posicionamento (linhas 7 a 9), da mesma forma do Exemplo 30.

Na linha 14 criamos uma variável chamada `tecla_pressionada` para receber as teclas que estão pressionadas.

Nas linhas 15 a 22 comparamos se as teclas pressionadas são as setas. Na linha 15 verifica-se se a seta para cima `K_UP` está pressionada; se retornar verdadeiro, a variável `posição_vertical` é diminuída de 1 (linha 16); assim a imagem subirá. Se a seta para baixo `K_DOWN` estiver pressionada (linha 17) incrementamos a `posição_vertical`

em 1 (linha 18), fazendo com que a imagem desça. Aplicamos o mesmo procedimento para incrementar a variável `posicao_horizontal` com a seta para direita `K_RIGHT` (linhas 19 a 20) e decrementá-la com a seta para a esquerda `K_LEFT` (linhas 21 a 22), para movimentar a bola para a direita e para a esquerda.

JOGO 2 – QUIZ PYGAME

O programa mostrado a seguir que exemplifica os eventos de teclado é um jogo de perguntas e respostas, utilizando também vários recursos de fontes. Veja o código:

Jogo 2 – Quiz Pygame

```
1    import pygame , sys
2    from pygame.locals import *
3    pygame.init() #inicia o módulo pygame
4    azul = (0,0,255) #cor zul
5    branco = (255,255,255) #cor branca
6    tela =
     pygame.display.set_mode((800,600))
     #define tamanho da janela
7    fonte =
     pygame.font.Font("freesansbold.ttf",15
     ) #define fonte a ser utilizada
8    instrucao = "Quiz Pygame: Pressione a
     letra da resposta correta" #instrução
```

```
9    pergunta1 = "Qual é a capital do
     Brasil? a)Brasília b)Rio de Janeiro
     c)São Paulo d)Vitória e)Santos"
     #primeira pergunta
10   pergunta2 = "Qual é a capital da
     França? a)Rio de Janeiro b)Paris
     c)Tóquio d)Madri e)Lisboa"
     #segunda pergunta
11   pergunta3 = "Qual a capital da China?
     a)Xangai b)Nova York c)Berlim d)Pequim
     e)Rio de Janeiro"
     #terceira pergunta
12   resultado = "" #mensagem que informa
     resultado (inicia vazia)
13   perguntas =
     [pergunta1,pergunta2,pergunta3] #lista
     que contém as perguntas
14   respostas =
     [pygame.K_a,pygame.K_b,pygame.K_d]
     #lista que contém as teclas das
     respostas das perguntas
15   numero_pergunta = 0 #variável que
     controla qual pergunta será exibida
16   continuar = True #variável que
     controla o fluxo de repetição
17   exibir_resultado = False #variável que
     controla o segundo fluxo de repetição
18   while continuar == True: #laço de
     repetição
```

```
19      tela.fill(branco) #pinta a tela
com a cor branca
20      texto_inicial =
fonte.render(instrucao,True,azul)
#define texto de instrução
21      tela.blit(texto_inicial, (50,50))
#mostra texto com as instruções na
tela
22      texto =
fonte.render(perguntas[numero_pergunta
],True,azul) #define qual pergunta da
lista será exibida e forma o texto
23      tela.blit(texto,(0,100)) #mostra a
pergunta na tela
24      for evento in
pygame.event.get():#captura todos os
eventos
25          if evento.type == QUIT:
#verifica se é o evento de sair
26              continuar = False #não
executará mais o primeiro laço de
repetição
27              exibir_resultado = False
#não executará mais o segundo laço
28          if evento.type ==
pygame.KEYDOWN: #verifica se é o
evento pressionar tecla
29              if evento.key ==
respostas[numero_pergunta]: #verifica
```

```
se tecla pressionada corresponde à
resposta da pergunta
30                  if numero_pergunta <
2: #se não for a última pergunta da
lista
31                      numero_pergunta+=1
#passa para a próxima pergunta
32                  else: #se for a última
pergunta da lista
33                      continuar = False
#pára de executar primeiro laço
34                      exibir_resultado =
True #habilita segundo laço
35                      resultado =
"Parabéns! Você venceu!"     #mensagem
do resultado
36              else: #se a tecla
pressionada não corresponder à
resposta correta
37                  continuar = False
#para de executar primeiro laço
38                  exibir_resultado =
True #habilita segundo laço
39                  resultado = "Resposta
errada, você perdeu!" #mensagem do
resultado
40      pygame.display.update()#atualiza
display
```

```
41    while exibir_resultado ==
      True:#segundo laço de repetição
42        for evento in pygame.event.get():
      #captura todos os eventos
43            if evento.type ==
      QUIT:#verifica se é o evento de sair
44                exibir_resultado = False
      #não executará mais o laço
45        texto =
      fonte.render(resultado,True,azul)
      #monta o texto com o resultado
46        tela.blit(texto,(100,200)) #exibe
      o texto na tela
47        pygame.display.update()#atualiza
      display
48    pygame.quit()#encerra o programa
```

Nesse jogo são apresentadas perguntas de países e capitais. São três perguntas em sequência; se o usuário errar alguma, perde; se acertar todas, vence. Elas apresentam três opções, letras `a`, `b`, `c`: o jogador precisa pressionar uma dessas no teclado.

Nas linhas 8 a 11 são definidos os textos que serão usados: a instrução para o jogador e as três perguntas com suas possíveis respostas. As variáveis: `pergunta1`, `pergunta2` e `pergunta3` formarão uma lista (ou `array`) posteriormente. A variável resultado (linha 12) será utilizada para exibir o texto informando se o usuário ganhou ou perdeu; a princípio ela recebe "`vazio`".

Como os jogos executam em laços de repetições, trabalhar com listas é vantajoso. Dessa forma, de acordo com a ação, é necessário somente mudar o índice da lista, ao invés de trocar de variável. Para as perguntas é inserida uma lista chamada `perguntas` (linha 13), que agrupará as variáveis das três perguntas. Da mesma forma, é inserida uma lista para as respostas com o nome das teclas das respostas corretas (linha 14). A resposta da primeira pergunta é o primeiro elemento da lista respostas e assim sucessivamente. Uma lista precisa de um índice; para isso é criada a variável `numero_pergunta`; a primeira posição de uma lista é a zero, por isso `numero_pergunta` é inicializada com 0 (linha 15).

No laço de repetição principal (linha 18), o texto com as instruções é montado na variável `texto_inicial` (linha 20), utilizando a fonte (que foi definida na linha 7) na cor azul. Na linha 21, esse texto será desenhado na tela na posição (50,50). É feito o mesmo para a primeira pergunta: é montada a variável `texto` com a pergunta a ser feita e são utilizadas a mesma fonte e a mesma cor (linhas 22 e 23). A pergunta, retirada da variável lista `perguntas[]` é a primeira porque o índice (a variável `numero_pergunta`) está valendo 0.

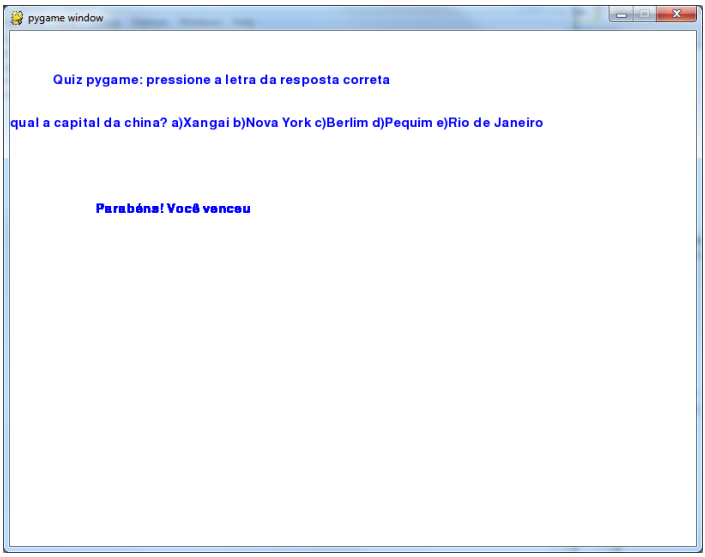
A linha 28 verifica a ocorrência do evento pressionar de uma tecla (`KEYDOWN`), se ocorrer a linhs 29 compara a tecla (`evento.key`) pressionada com a lista de respostas; se for a mesma apontada pelo índice `numero_pergunta`, é preciso ainda verificar se é a última pergunta (linha 30). Como o índice da terceira pergunta é 2, se `numero_pergunta` é menor que 2 (não é a última pergunta), então o programa acrescenta 1 à variável `numero_pergunta` (linha 31); com isso, na próxima repetição virá a pergunta seguinte da lista. Se o índice `numero_pergunta` for igual a 2 (no `else`), é a última

pergunta e já se pode exibir o resultado da partida (linha 32): a variável `continuar` recebe falso (linha 33) para sair do laço principal; a variável `exibir_resultado` recebe verdadeiro (linha 34) para entrar no segundo laço e a variável `resultado` recebe "`Parabéns! Você venceu`" (linha 35). Se a tecla pressionada (`evento.key`) não corresponder à resposta correta (linha 36), a variável `continuar` recebe falso (linha 37) para sair do laço principal; a variável `exibir_resultado` recebe verdadeiro para entrar no segundo laço (linha 38) e a variável `resultado` recebe "`Resposta errada, você perdeu`" (linha 39).

Nas linhas 41 a 47 ocorre o segundo laço de repetição, que mostra o resultado do jogo até fechar o programa. Já vimos trecho semelhante em programas anteriores. Ele executa enquanto a variável `exibir_resultado` for verdadeira. A frase da variável resultado (informando se ganhou ou perdeu) é `renderizada` na cor azul e atribuída à variável `texto` (linha 45), que é montada na posição (100,200) da tela (linha 46).

Ao executar o jogo, o resultado final será semelhante ao mostrado na Figura 106.

Figura 106 – Quiz Pygame



FUNÇÃO DE TEMPO E ANIMAÇÃO

Até aqui o laço de repetição dos programas executava indefinidamente, várias vezes por segundo. Porém não exercíamos controle sobre isso. Isso significa que a velocidade de um objeto pode variar de computador para computador, de acordo com o hardware que executa o programa.

O pygame permite controlar a quantidade de vezes que o programa executa por segundo. É necessário definir uma variável para controlar o tempo (por exemplo, `variável_de_tempo`) e ela deve receber o comando `pygame.time.Clock()`. O método `tick()` dessa variável

define a quantidade de vezes que o programa deve executar por segundo, por exemplo, `variável_de_tempo.tick(30)`.

Veja o exemplo abaixo:

Exemplo 33 – Tempo

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia o módulo pygame
4    tela = pygame.display.set_mode([800,
     600]) #define o tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    branco=(255,255,255) #cor branca
7    bola=pygame.image.load("bola.png")#car
     rega imagem
8    posicao_horizontal = 0
9    posicao_vertical = 0
10   movimento_horizontal = 1 #direção do
     movimento
11   relogio = pygame.time.Clock()#controle
     de tempo
12   while continuar == True: #laço de
     repetição
13       for evento in
     pygame.event.get():#captura todos os
     eventos
```

```
14          if evento.type ==
    pygame.QUIT:#verifica se é o evento de
    sair
15              continuar = False
16      if posicao_horizontal >=
    750:#verifica posição horizontal
17          movimento_horizontal=-1 #valor
    da direção
18      elif posicao_horizontal <= 0:
    #verifica posição horizontal
19          movimento_horizontal = 1
    #valor da direção
20      posicao_horizontal =
    posicao_horizontal+movimento_horizonta
    l #soma a posição horizontal com o
    valor da direção
21      tela.fill(branco) #pinta a tela
    com a cor branca
22      tela.blit(bola,
    (posicao_horizontal,
    posicao_vertical)) #coloca imagem na
    tela na posição corrente
23      pygame.display.update()#atualiza
    display
24      relogio.tick(30) #define a
    quantidade de quadros por segundo
25  pygame.quit()#encerra o programa
```

Esse programa movimenta uma bola de um canto a outro, deslocando um pixel por vez, fazendo isso trinta vezes por segundo.

A imagem da bola é carregada e são criadas duas variáveis para a posição da bola, ambas recebendo 0 para que a bola inicie no canto superior esquerdo da tela (linhas 7 a 9). A bola deve ter o tamanho 50 por 50 pixels.

Na linha 10 inserimos uma variável para deslocar a bola chamada `movimento_horizontal`, que poderá ter o valor 1 ou -1. Quando somada à variável `posicao_horizontal`, deslocará a bola para a esquerda ou para a direita. Logo após é inserida uma variável de tempo chamada `relógio` (linha 11).

Se o valor da variável `posicao_horizontal` chegar a 750 (que é a largura da tela menos a largura da bola), a variável `movimento_horizontal` recebe -1 para que a bola volte para a esquerda (linhas 16 e 17). Mas se a posição da bola chegar a zero, a variável `movimento_horizontal` recebe 1 para que a bola vá para a direita (linhas 18 e 19). Dessa forma a bola ficará indo e voltando na tela, sem sair dela.

Na linha 20, a variável `posição_horizontal` (iniciada com 0) pode aumentar ou diminuir, de acordo com o valor de `movimento_horizontal`; esta variável `movimento_horizontal` inicia com o valor 1 para que `posicao_horizontal` comece aumentando, levando a bola para a direita. Quando `posição_horizontal` chegar a 750, a variável `movimento_horizontal` passa a ser negativa, fazendo com que a bola volte para a esquerda.

Na linha 24 utilizamos a variável de tempo chamada `relogio` e passamos 30 para o seu método `tick()`; isso fará o laço de repetição principal executar 30 vezes por segundo. Se

esse valor for aumentado, será possível perceber que a bola aumenta sua velocidade na tela.

ANIMAÇÕES

Até o momento, movimentamos os objetos na tela. Os objetos, porém, não eram animados, não tinham movimentos próprios. As animações são muito comuns em jogos e podemos fazê-las também com o pygame. Como no cinema, as animações são a exibição de várias imagens consecutivas em um curto intervalo de tempo. A visão humana consegue perceber 30 imagens por segundo; alguns jogos, porém, utilizam 60 ou mais imagens por segundo.

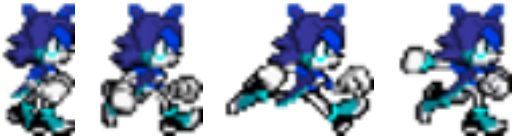
Para o próximo programa, serão necessárias imagens consecutivas. Veja um exemplo na Figura 107:

Figura 107 - Imagens consecutivas

Fonte : Wikimedia (Wikimedia.org)

Essas imagens mostram o movimento de um personagem correndo. Com um editor de imagens conseguimos separar essa imagem em 4 outras (ver Figura 108), cada uma contendo um momento do movimento do personagem. Grave-as com os nomes `atleta1`, `atleta2`, `atleta3`, `atleta4`.

Figura 108 – Imagens consecutivas II



Fonte : Wikimedia (Wikimedia.org)

Alternando essas imagens rapidamente, podemos criar a impressão de que o personagem está correndo. O código para isso é mostrado a seguir.

Exemplo 34 – animação

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia o módulo pygame
4    tela = pygame.display.set_mode([800,
     600]) #define o tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    branco = (255,255,255) #cor branca
```

```
7     atleta1 =
      pygame.image.load("atleta1.png")#carre
      ga primeira imagem
8     atleta2 =
      pygame.image.load("atleta2.png")#carre
      ga segunda imagem
9     atleta3 =
      pygame.image.load("atleta3.png")#carre
      ga terceira imagem
10    atleta4 =
      pygame.image.load("atleta4.png")#carre
      ga quarta imagem
11    posicao_horizontal = 0
12    posicao_vertical = 150
13    relogio = pygame.time.Clock()#controle
      de tempo
14    animacao =
      [atleta1,atleta2,atleta3,atleta4]
      #lista com as quatro imagens
15    contador = 0 #variável contadora
16    while continuar == True: #laço de
      repetição
17        for evento in
      pygame.event.get():#captura todos os
      eventos
18            if evento.type == pygame.QUIT:
      #verifica se é o evento de sair
19                continuar = False
```

```
20      if(posicao_horizontal < 700): #verifica posição horizontal
21          posicao_horizontal+=2 #acrescenta 2 à posição horizontal
22      if(contador < 3): #se contador menor que 3(último índice da lista)
23          contador+=1 #acrecenta 1 ao contador
24      else:
25          contador=0 #reinicia o contador em zero(primeiro índice da lista)
26      tela.fill(branco) #pinta a tela de branco
27      tela.blit(animacao[contador],(posicao_horizontal,posicao_vertical)) #coloca a imagem do personagem na tela
28      pygame.display.update() #atualiza o display
29      relogio.tick(30) #define a quantidade de quadros por segundo (fps)
30  pygame.quit()#encerra o programa
```

Nesse programa, o personagem percorre a tela horizontalmente, até chegar ao ponto 700.

São carregadas as 4 imagens consecutivas do personagem correndo (Figura 113), em quatro variáveis diferentes (linhas 7 a 10).

Definimos a posição horizontal com 0 (linha 11) para que o personagem comece no lado esquerdo da tela, e a posição vertical com 150 (linha 12). Iniciamos também uma variável de tempo chamada `relogio` (linha 13).

Definimos um vetor (ou lista), que recebe as 4 imagens (linha 14). Cada imagem corresponde a uma posição do vetor; a primeira está na posição 0 e a última na posição 3. Também é definido um índice (`contador`), que trocará de valor a cada repetição do laço (linha 15); essa variável controlará qual imagem deve ser mostrada no momento.

Enquanto a posição do personagem for inferior a 700, desloca-se o mesmo 2 pixels a direita (linhas 20 e 21).

As linhas 22 a 25 fazem com que as imagens fiquem alternando. Se a variável `contador` for menor que 3, acrescentamos 1 a ela; senão, ela reinicia com 0. A variável começa em 0, na primeira repetição muda para 1, depois para 2, depois para 3; depois volta para zero, iniciando o ciclo novamente. Esses números correspondem às posições das imagens no vetor animacao.

A linha 27 monta a imagem do personagem, pegando a imagem da lista, de acordo com o valor de `contador`. Por exemplo, se `contador = 1`, então carregamos a imagem de índice 1 do vetor, que corresponde à segunda imagem chamada "`atleta2`". A primeira imagem corresponde ao índice 0. Dessa forma, a cada instante é montada uma imagem diferente. Quanto mais imagens consecutivas, melhor será a definição do movimento.

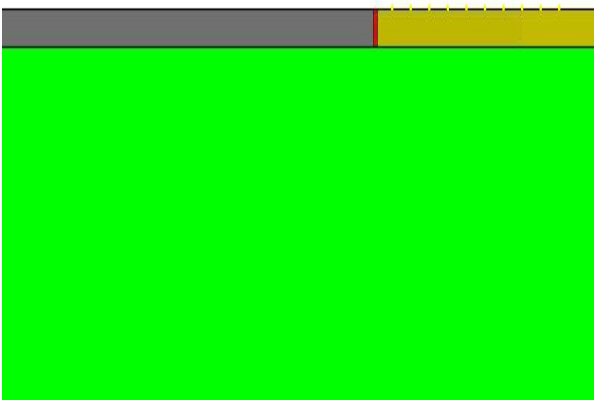
Na linha 29 é estabelecido que o programa execute 30 vezes por segundo.

JOGO 3 – SALTO EM DISTÂNCIA

O código a seguir exemplifica o uso de movimentos. O aplicativo é um jogo em que o personagem deve saltar o máximo que puder antes de chegar à linha vermelha. Para atingir o melhor salto deve-se saltar com o ângulo mais próximo possível de 45 graus e chegar próximo à linha vermelha, sem tocá-la. Para esse jogo serão utilizadas as mesmas imagens do exemplo anterior, porém cada imagem deverá ser redimensionada para 50 pixels de altura, e a largura deve ficar por volta de 25 ou 30 pixels.

Além disso, a pista terá que ser desenhada, veja a Figura 109:

Figura 109 - Pista de atletismo



Essa pista é composta por um retângulo cinza que mede 500 pixels de largura e 20 pixels de altura, seguido por um pequeno retângulo vermelho medindo 5x20 pixels, que é a linha limite; após isso há um retângulo amarelo. Abaixo da pista há uma grande área verde. A imagem deve ter as mesmas dimensões da tela. Essa imagem tem 800 pixels de largura e 600 pixels de comprimento. O código do jogo é o seguinte:

Jogo 3 – Salto em distância

```
1    import pygame, sys, math
2    from pygame.locals import *
3    pygame.init() #inicia o módulo pygame
4    tela = pygame.display.set_mode([800,
     600]) #define tamanho da janela
5    continuar = True #variável que
     controla o fluxo de repetição
6    laranja = (255,127,0) #cor laranja
7    atleta1 =
     pygame.image.load("atleta1.png")
     #primeira imagem do personagem
8    atleta2 =
     pygame.image.load("atleta2.png")
     #segunda imagem do personagem
9    atleta3 =
     pygame.image.load("atleta3.png")
     #terceira imagem do persongem
10   atleta4 =
     pygame.image.load("atleta4.png")
     #quarta imagem do personagem
```

```
11  pista                                          =
    pygame.image.load("atletismo.jpg")#ima
    gem da pista
12  fonte                                          =
    pygame.font.Font("freesansbold.ttf",20
    )#define a fonte
13  pulou = False #informa se o personagem
    já saltou
14  posicao_horizontal = 0 #posição
    horizontal
15  posicao_vertical = 150 #posição
    vertical
16  angulo = 0 #ângulo do salto do
    personagem
17  distancia = 0 #distância do salto
18  barra = pygame.Surface((1, 50)) #barra
    que indica a quantidade de graus
19  barra.fill(laranja) #define a cor da
    barra como laranja
20  relogio = pygame.time.Clock()#controle
    de tempo
21  animacao =
    [atleta1,atleta2,atleta3,atleta4]
    #lista com as imagens do personagem
22  tamanho_barra = 0 #tamanho da barra
23  contador = 0 #índice da lista de
    imagens
24  distancia_total = 0 #distancia total
    do salto
```

```
25  ponto_de_salto = 0 #local de início do
    salto
26  while continuar == True: #laço de
    repetição
27      tela.blit(pista,(0,0)) #coloca a
    imagem da pista na tela
28      for evento in
    pygame.event.get():#captura todos os
    eventos
29          if evento.type == pygame.QUIT:
    #verifica se é o evento de sair
30              continuar = False
31          if evento.type == pygame.KEYUP
    and pulou == False : #se soltou tecla
    e ainda não saltou
32              if evento.key == K_SPACE:
    #se a tecla é a barra de espaço
33                  pulou = True #indica
    que o personagem iniciou o salto
34                  ponto_de_salto =
    posicao_horizontal #captura o local do
    salto
35      tecla_pressionada =
    pygame.key.get_pressed() #captura
    tecla pressionada
36      if
    tecla_pressionada[pygame.K_SPACE] and
    pulou == False: #verifica se tecla é
```

```
    barra de espaço e se o personagem
    ainda não saltou
37          if angulo < 90: #se a barra
    indicativa de graus ainda não estiver
    cheia
38              angulo+=1 #acrescenta um
    ao ângulo
39      if pulou == False: #se não pulou
40          posicao_horizontal+=3 #desloca
    a posição do personagem para a direita
41          if(contador < 3): #se não for
    a última posição da lista de imagens
42              contador+=1 #passa-se para
    a imagem seguinte
43          else: #se for a última imagem
44              contador = 0 #reinicia na
    primeira posição da lista
45      else: #se já pulou
46          if ponto_de_salto < 475:
    #verifica se pulou antes da linha
    vermelha
47              radianos = math.pi *
    angulo / 180 #transforma graus em
    radianos
48              distancia=(9 * 9) *
    (math.sin(2*radianos)) /9.8 #fórmula
    da distância do salto
49              distancia_da_linha =
    ((500.0-(ponto_de_salto+25)) /25.0)
```

```
    #distância do ponto de salto até a
    linha vermelha (distância perdida)
50              salto = distancia -
    distancia_da_linha #distância do
    salto, descontando a distância perdida
51              if salto < 0 : #se o salto
    não chegou à linha vermelha
52                  salto = 0 #como não há
    salto negativo, salto recebe 0
53              texto2 =
    fonte.render("Você saltou %.2f
    metros!" % salto,True,laranja) #texto
    informando a distância do salto
54              tela.blit(texto2,(400,50))
    #exibe o texto na tela
55              distancia_total =
    ponto_de_salto + (distancia*25)
    #distância em pixels
56              if posicao_horizontal <
    distancia_total-(distancia*25/2): #o
    personagem irá subir até atingir a
    metade da distância do salto
57                  posicao_horizontal+=3
    #desloca a posição para a direita
58                  posicao_vertical-=1
    #desloca a posição para cima
59              elif posicao_horizontal <
    distancia_total: #se passar da metade
    do salto, o personagem irá caindo até
    atingir a distância total do salto
```

```
60                  posicao_horizontal+=3
#desloca a posição para a direita
61                  posicao_vertical+=1
#desloca a posição para baixo
62          else: #se pulou após a linha
vermelha
63              texto2 =
fonte.render("Passou da linha, salto
inválido!",True,laranja) #mensagem
informando que o salto foi inválido
64              tela.blit(texto2,(400,50))
#exibe texto na tela
65
tela.blit(animacao[contador],(posicao_
horizontal,posicao_vertical)) #exibe a
imagem corrente do personagem na tela
66      barra = pygame.Surface((angulo,
25)) #preenche a barra que indica a
quantidade de graus do salto
67      barra.fill(laranja) #pinta a barra
de laranja
68      tela.blit(barra,(300,250)) #mostra
barra na tela
69      texto =
fonte.render(str(angulo),True,laranja)
#monta o texto informando a quantidade
de graus
70      tela.blit(texto,(400,250)) #mostra
o texto ao lado da barra
```

```
71          pygame.display.update() #atualiza
        o display
72          relogio.tick(60) #60 quadros por
        segundo
73      pygame.quit() #encerra o programa
```

Esse código é dividido em dois momentos: antes e depois de iniciar o salto. Antes de iniciar o salto, é exibida a animação do personagem correndo. Após iniciar o salto, é calculada a distância do salto e é exibida a animação do salto. Também é exibido o texto informando qual distância o jogador saltou.

Como no exemplo anterior, são carregadas as 4 imagens consecutivas do personagem correndo e também a imagem da pista, para ser utilizada como pano de fundo (linhas 7 a 11).

Criamos a variável chamada `pulou`, para saber se o personagem já iniciou o salto ou não (linha 13). Também é utilizada uma variável para receber o ângulo do salto (linha 16) e outra para a distância do mesmo (linha 17).

Na linha 18 é criada uma variável chamada `barra`, que recebe, com o comando `pygame.Surface(1,50)`, um retângulo de 1 pixel de largura e 50 de altura; com o método `fill()`, a barra fica com a cor laranja (linha 19).

A variável `tamanho_barra` (linha 22) é usada para determinar o tamanho da barra que indica a quntidade de graus do salto, a variável `contador` (linha 23) é o índice da lista de imagens `animacao` (definida na linha 21), a variável `distancia_total` (linha 24) é utilizada para o cálculo da distância total que o personagem atinge na pista e a variável

`ponto_de_salto` (linha 25) é usada para marcar o ponto onde o personagem inicia o salto.

Dentro do laço principal, se ocorrer o evento `pygame.KEYUP` (soltar a tecla) e a variável pulou for falsa (linha 31), verificamos qual tecla disparou o evento. Se foi `K_SPACE` (a tecla espaço), atribuímos verdadeiro (`True`) para a variável pulou e pegamos a localização do salto, atribuindo o valor da variável `posicao_horizontal` à variável `ponto_de_salto` (linhas 32 a 34). Quando o jogador solta a tecla espaço o atleta inicia o salto. O atleta pode saltar somente se a variável `pulou` for falsa; se ele já saltou, não pode saltar novamente na mesma tentativa.

É necessário também verificar se a tecla espaço está pressionada antes de ser solta. Se estiver e a variável `pulou` for falsa, aumentamos o ângulo em um grau (linhas 36 a 38). O ângulo do salto pode ir de 0 a 90 graus, portanto há uma condição (`ângulo < 90`) para não deixar o ângulo passar desse valor. Nesse caso também é importante que a variável `pulou` seja falsa; se o jogador já pulou, não há como mudar o ângulo do salto.

Se a variável `pulou` for falsa (linha 39), o atleta ainda não pulou, ainda está correndo. Então, ele se desloca em direção ao local do salto (para a direita), aumentando a `posicao_horizontal` em 3 pixels (linha 40). Quanto maior for o acréscimo, maior a velocidade do personagem na animação.

Como no exemplo anterior, o controle das imagens da animação é feito com a variável `contador`. Essa variável faz referência ao índice da imagem do personagem a ser exibida a cada momento (linhas 41 a 44).

Se a variável pulou não for falsa (linha 45), o personagem já iniciou o salto. Se ele não passou da linha vermelha (linha 46), o que não é permitido, então é necessário calcular a distância do salto. A função do pygame não calcula o `seno` em graus, mas em radianos; precisamos converter os graus em radianos, multiplicando o ângulo pelo número PI e dividindo por 180 (linha 47). A fórmula na linha 48 calcula a distância do salto: velocidade elevada ao quadrado vezes o seno de duas vezes o radiano, tudo dividido pela força da gravidade. Definimos: velocidade: 9 metros por segundo; seno: a função `math.sin`(); força da gravidade: 9,8 metros por segundo. Calculada a distância do salto, ainda assim não se tem o valor real do salto, é necessário saber qual a distância que o atleta perdeu antes da linha vermelha. Para o cálculo da distância da linha, supomos que a pista meça 20 metros (`= 500 pixels`), assim cada 25 pixels é 1 metro. Subtraímos de 500 o valor de `ponto_de_salto` mais a largura do atleta (25 pixels); assim temos o valor em pixels da distância. Porém, para os cálculos, é necessário converter esse valor em metros, o que é feito dividindo-se por 25 (linha 49).

Dessa forma, o valor real do `salto` é calculado subtraindo `distancia_da_linha` da `distancia` do salto (linha 50).

Se o personagem pular muito antes da linha, não conseguirá nem passar da linha: seu salto é negativo; nesse caso, atribuímos valor zero ao salto (linhas 51 e 52).

As linhas 53 e 54 são responsáveis pela divulgação do resultado. A mensagem "`Você saltou %.2f metros`" é renderizada e colocada em `texto2`, na cor laranja. No lugar de `%.2f` é exibido o valor da variável `salto` e o `.2f` faz com que somente duas casas decimais após a vírgula sejam

mostradas. O texto é montado pelo comando `blit()` na posição `(400,50)`.

Na linha 55 é calculada a distância total percorrida pelo atleta em pixels, somando-se o ponto de salto mais a distância do salto convertida para pixels (multiplicada por 25).

As linhas 56 a 61 programam o salto do personagem. Isso é feito da seguinte forma: se a posição horizontal for menor que o ponto central do salto do atleta (`distancia_total-(distancia*25/2)`), então acrescentamos 3 pixels à posição horizontal e subtraímos 1 da posição vertical, para que ele vá para a direita e suba. Porém, se ele já passou da metade do salto (ponto mais alto do salto) e ainda não atingiu a distância total do salto, continuamos acrescentando 3 pixels horizontalmente para que ele vá para a direita e acrescentamos 1 pixel à posição vertical, para que ele desça no salto.

Porém, se o atleta, para iniciar o salto, já tiver passado do ponto 475 (passou da linha vermelha), seu salto não será considerado (linha 62): a variável `texto2` é renderizada com a seguinte mensagem "`Passou da linha, salto inválido`", e a cor da letra é definida como laranja (linha 63). Na linha 64, a imagem do texto é montada na posição (400,50).

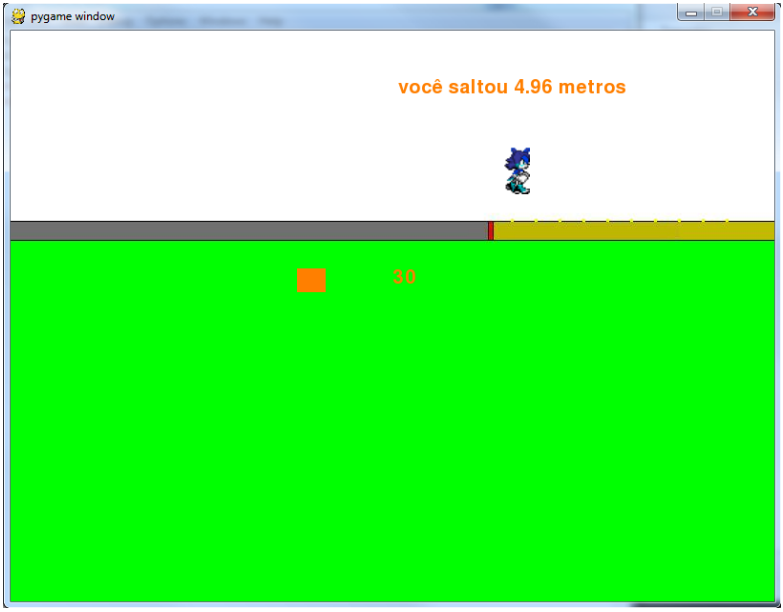
É desenhado um retângulo na tela com a largura em pixels de acordo com o valor da variável `angulo` e de altura 25 pixels (linha 66). Por exemplo, se o valor variável `angulo` for 45, então será desenhado um retângulo de 45x25 pixels. A cor de fundo da barra é definida com laranja e em seguida é montada essa imagem (linha 67).

De acordo com o tempo em que a pessoa pressiona a tecla espaço, o valor do ângulo do salto aumenta (ver Figura 110). Assim, é montado um texto "`str(angulo)`" na cor

laranja, para que o usuário saiba em que valor o ângulo está. O comando `str()` serve para converter um número em texto (linha 69). Logo após a imagem do texto é montada (linha 70), com o método `blit`(), na posição (400,250).

Para que as imagens sejam exibidas na tela, atualizamos o display. Definimos o número de vezes que o programa executa por segundo em 60.

Figura 110 – Jogo salto em distância



SPRITES

Um game normalmente é formado por cena e personagens. A cena, ou fundo, normalmente se compõe de objetos que não influenciam diretamente na ação do jogo. Os personagens têm ação dentro do jogo: podem ser movimentar, responder a comandos do jogador ou mesmo causar reações em outros personagens ou objetos. Em pygame, podemos representar os personagens como objetos. Cada objeto tem suas características e ações (em programação chamamos de `propriedades` e `métodos`). Antes de inserir um objeto, é preciso definir a sua `classe`. Na classe são definidas as características e ações que ele terá. Algumas de suas características podem ser: imagem, localização, tamanho, cor etc. Algumas de suas ações ou métodos podem ser: andar, correr, saltar etc. Pode haver vários objetos da mesma classe; porém, um objeto pertence a uma única classe; não pode haver um objeto que pertença a mais de uma classe.

Em pygame todos os objetos são do tipo `sprite`. Essa classe tem características básicas: imagem, coordenadas, tamanho etc., e necessita de um método inicial chamado `__init()__`; nesse método, todas as características do objeto são definidas. Dentro dos parênteses pode haver uma lista de parâmetros ou não; quando existirem, os parâmetros têm de ser passados na mesma ordem.

Os objetos `sprite` normalmente pertencem a um ou mais grupos. Os grupos na verdade são uma classe do pygame. Eles agrupam objetos do tipo sprite e contêm vários métodos para trabalhar com os mesmos. Por isso é comum a inserção de um sprite em um grupo. Os grupos são instanciados pelo

comando `pygame.sprite.Group()`. Nos grupos, os objetos podem ser inseridos pelo método `add()`, com o objeto sendo passado como parâmetro, ou removidos com o método `remove()`. Outro método deste é o `draw()`, que desenha todos os `sprites` do grupo na tela, sendo necessária a passagem da tela como parâmetro.

No exemplo abaixo, será mostrado um programa no qual podemos movimentar uma bola pelas setas do teclado. O programa é o mesmo do `exemplo 32`, porém neste a bola terá uma classe própria para ela. Veja o programa:

Exemplo 35 – Classes

```
1    import pygame, sys
2    from pygame.locals import *
3    class      Bola(pygame.sprite.Sprite):
     #classe bola herdando da classe Sprite
4        def __init__(self): #construtor da
     classe
5
     pygame.sprite.Sprite.__init__(self)
     #inicia classe Sprite
6            self.image                    =
     pygame.image.load("bola.png") #carrega
     imagem da classe
7            self.rect                     =
     self.image.get_rect()#inicializa    as
     coordenadas
8    pygame.init() #inicia modulo pygame
```

```
9   todos_sprites = pygame.sprite.Group()#grupo de sprites
10  tela = pygame.display.set_mode([800, 600]) #define tamanho da janela
11  bola = Bola() #objeto do tipo bola
12  todos_sprites.add(bola) #adiciona objeto bola ao grupo de sprites
13  continuar = True #variável que controla o laço de repetição
14  branco = (255,255,255) #cor branca
15  relogio = pygame.time.Clock()#controle do tempo
16  while continuar == True: #laço de repetição
17      for evento in pygame.event.get():#captura todos os eventos
18          if evento.type == pygame.QUIT: #verifica se é o evento de sair
19              continuar = False
20      tecla_pressionada = pygame.key.get_pressed()#captura tecla pressionada
21      if tecla_pressionada[pygame.K_UP] == True:#se tecla for seta para cima
22          bola.rect.y-=1 #desloca objeto para cima
```

```
23      if tecla_pressionada[pygame.K_DOWN] == True: #se tecla for seta para baixo
24          bola.rect.y+=1  #desloca bola para baixo
25      if tecla_pressionada[pygame.K_RIGHT] == True:#se tecla for seta para direita
26          bola.rect.x+=1 #desloca objeto para direita
27      if tecla_pressionada[pygame.K_LEFT] == True:#se tecla for seta para a esquerda
28          bola.rect.x-=1 #desloca objeto para esquerda
29      tela.fill(branco)  #pinta a tela com a cor branca
30      todos_sprites.draw(tela)  #desenha os objetos do grupo na tela
31      pygame.display.update()#atualiza o display
32      relogio.tick(30)  #30 quadros por segundo
33  pygame.quit()#encerra o programa
```

Notamos nesse programa que o modulo do pygame `pygame.init()` não segue logo após as importações. Antes disso há a definição da classe `Bola`.

Uma classe é definida pela palavra `class`, em seguida o nome da classe e, entre os parênteses, `pygame.sprite`.Sprite, que indica que a classe herda características da classe básica de `sprites` do pygame, como os atributos `image`, `rect`, entre outros (linha 3).

Na linha 4 temos o método inicial da classe. Em Python, é utilizada a palavra `def` para métodos ou funções. O método inicial é chamado de `__init__()`, sendo também o construtor da classe. Quando um objeto da classe é instanciado, o construtor executa automaticamente. É necessário passar para esse método a palavra `self`, que indica que se passa a própria classe como parâmetro. Outros parâmetros são opcionais e nesse exemplo não foram utilizados.

A seguir, na linha 5, é necessário iniciar a classe básica do `pygame`, o que é feito chamando o método inicial do pygame: `__init__()`, e passar como parâmetro a própria classe(`self`).

Iniciamos a atribuição das características básicas da classe: para isto usamos `self` (que indica que é a própria classe), um ponto e o nome da propriedade. A primeira que definimos é image: carregamos a imagem pelo comando `pygame.image.load()` e passamos o nome da imagem que é `bola.png` (linha 6). A propriedade seguinte `rect` define o tamanho da imagem e é formada por quatro valores: as coordenadas x e y, o comprimento `width` e a altura `height`. Esse atributo também define o retângulo de colisão de um `sprite` com outro. O comando `get_rect()`, retorna o atributo `rect` da imagem (linha 7). Dessa forma é possível definir os quatros pontos que formam o retângulo da imagem.

Alguns métodos, como o `get_rect()`, não necessitam de nenhum parâmetro.

Logo após é iniciado o método principal do pygame: `init`(). Depois é criada uma variável do tipo grupo chamado `todos_sprites` com o comando `pygame.sprite.Group()` na linha 9. Normalmente, há um grupo que contém todos os `sprites` do jogo. Um objeto pode estar em mais de um grupo. É comum agrupar objetos com características semelhantes, por exemplo, grupo de veículos, grupo de personagens, grupo de oponentes etc.

Na linha 11, um objeto bola é criado: para isso atribuímos a classe `Bola` a uma variável à qual demos o nome `bola`. O nome do objeto se chama `bola` e ele recebe `Bola()`, ou seja, se torna do tipo `Bola` (a classe que foi definida anteriormente). Observe que o nome da classe `Bola` foi escrito com maiúscula e o da variável bola foi escrito com minúscula: são nomes diferentes. A seguir, esse objeto é adicionado ao grupo `todos_sprites`, pelo método `add`(), passando no parâmetro o nome do objeto, `bola` (linha 12).

Após isso é feita a programação das ações da bola. Usamos a variável `tecla_pressionada` para capturar a tecla que foi pressionada (linha 20). Dependendo de qual foi a seta, vamos alterar o atributo `rect` do objeto `bola`; já falamos anteriormente que `rect` tem quatro atributos, entre os quais `x` e `y`. Se foi pressionada `seta para cima` (linha 21), subtraímos 1 do atributo `y` de `rect` (a coordenada vertical do objeto) na linha 22; se foi `seta para baixo` (linha 23), aumentamos `y` em 1 (linha 24). Se foi pressionada `seta para a direita` (linha 25), somamos 1 ao atributo `x` de `rect` (a

coordenada horizontal do objeto) na linha 26; se foi a `seta para a esquerda` (linha 27), diminuímos em 1 (linha 28).

Depois de programadas as ações da bola, definimos o fundo como branco e os `sprites` são desenhados na tela. O grupo `todos_sprites` tem um método chamado `draw`; para ele passamos o nome da tela que estamos utilizando, pois esse método é responsável por desenhar todos os sprites pertencentes ao grupo na tela (linha 30). A seguir o display é atualizado e o número de ciclos por segundo é definido como 30.

DETECÇÃO DE COLISÃO

No exemplo acima foi definida só uma classe e só um objeto na tela, porém, pode haver vários objetos da mesma classe e de classes diferentes. Cada objeto tem comportamento independente e, por isso, os objetos podem colidir entre si. Quando um objeto colide com outro, normalmente reage mudando seu comportamento. Por exemplo, quando uma bola colide com uma parede, muda a direção. Cada colisão dever ter uma ação programada, caso contrário, um objeto simplesmente passará por cima do outro.

A primeira coisa a se fazer ao programar uma colisão é identificá-la. A biblioteca pygame oferece comandos prontos de identificação de colisão. Cada objeto possui o atributo `rect`, o retângulo ocupado pela imagem, que define os cantos das imagens. Os comandos de colisão do pygame verificam se algum ponto de um objeto está dentro do retângulo de outro objeto. Essa detecção de colisão é simples, porém bastante útil.

Há duas formas de identificar colisões: por objeto e por grupo. Para verificar a colisão por objeto, usamos `colliderect()`, passando como parâmetro o objeto que se deseja verificar se há colisão, por exemplo: `objeto1.rect.colliderect(objeto2)`. Para verificar se há colisão com um grupo, usamos `spritecollide()`, passando como parâmetros o objeto, o grupo de objetos com o qual ele pode colidir e o valor `False`: esse comando retorna uma lista contendo os objetos com os quais o objeto colidiu. O exemplo a seguir mostra o uso de ambos:

Exemplo 36 – Colisão

```
1   import pygame, sys
2   from pygame.locals import *
3   vermelho = (255,0,0) #cor
    vermelha
4   class Bola(pygame.sprite.Sprite):
    #classe Bola, herdando de Sprite
5       def __init__(self):
    #construtor da classe
6
    pygame.sprite.Sprite.__init__(sel
    f) #inicia classe Sprite
7           self.image=
    pygame.image.load("bola.png")
    #carrega imagem da classe
8           self.rect =
    self.image.get_rect()#inicializa
    as coordenadas
```

```
9     class
      Obstaculo(pygame.sprite.Sprite):#
      classe Obstaculo, herdando de
      Sprite
10        def
      __init__(self,posicao_horizontal,
      posicao_vertical): #construtor
11
      pygame.sprite.Sprite.__init__(sel
      f) #inicia Sprite
12            self.image =
      pygame.Surface([50,50])#define
      imagem como quadrado
13            self.rect =
      self.image.get_rect()#inicializa
      as coordenadas
14
      self.image.fill(vermelho)#pinta o
      retângulo de vermelho
15            self.rect.x =
      posicao_horizontal #define a
      posição horizontal
16            self.rect.y =
      posicao_vertical #define a
      posição vertical
17    pygame.init()#inicia módulo
      pygame
18    todos_sprites =
      pygame.sprite.Group()#grupo geral
      de sprites
```

```
19  lista_obstaculos =
    pygame.sprite.Group()#grupo de
    obstáculos
20  tela =
    pygame.display.set_mode([800,
    600]) #define tamanho da janela
21  bola = Bola()#instancia objeto da
    Classe Bola
22  todos_sprites.add(bola) #adiciona
    objeto bola ao grupo geral de
    sprites
23  obstaculo1 = Obstaculo(200,100)
    #instancia objeto da classe
    Obstaculo
24  todos_sprites.add(obstaculo1)
    #adiciona objeto ao grupo geral
    de sprites
25  lista_obstaculos.add(obstaculo1)#
    adiciona objeto ao grupo de
    obstáculos
26  obstaculo2 =
    Obstaculo(300,200)#instancia
    outro objeto da classe Obstaculo
27  todos_sprites.add(obstaculo2)
    #adiciona objeto ao grupo geral
    de sprites
28  lista_obstaculos.add(obstaculo2)#
    adiciona objeto ao grupo de
    obstáculos
```

```
29    continuar = True #variável que
      controla o fluxo de repetição
30    branco = (255,255,255) #cor
      branca
31    relogio =
      pygame.time.Clock()#controle de
      tempo
32    colidiu = False #variável usada
      para informar se o objeto bola
      colidiu
33    while continuar == True: #laço de
      repetição
34        tela.fill(branco) #pinta a
      tela com a cor branca
35        for evento in
      pygame.event.get(): #captura
      todos os eventos
36            if evento.type ==
      pygame.QUIT: #verifica se é o
      evento de sair
37                continuar = False
38        lista_colisao =
      pygame.sprite.spritecollide(bola,
      lista_obstaculos, False)
      #armazena os objetos com os quais
      a bola colide, em uma lista
39        for obstaculo in
      lista_colisao: #procura obstaculo
      na lista de colisão
```

```
40              colidiu = True
    #encontrando, muda variável
    colidiu para verdadeiro
41          if(colidiu == False): #se não
    colidiu:
42              tecla_pressionada =
    pygame.key.get_pressed() #captura
    tecla pressionada
43              if
    tecla_pressionada[pygame.K_UP] ==
    True: #se for seta para cima
44                  bola.rect.y-=1
    #desloca a bola para cima
45              if
    tecla_pressionada[pygame.K_DOWN]
    == True: #se for seta para baixo
46                  bola.rect.y+=1
    #desloca bola para baixo
47              if
    tecla_pressionada[pygame.K_RIGHT]
    == True:#se é seta para direita
48                  bola.rect.x+=1
    #desloca bola para a direita
49              if
    tecla_pressionada[pygame.K_LEFT]
    == True: #se é seta para esquerda
50                  bola.rect.x-=1
    #desloca bola para esquerda
```

```
51          else: #se colidiu é
        verdadeiro
52              fonte =
        pygame.font.Font("freesansbold.tt
        f",20)      #define fonte
53              texto =
        fonte.render("Você
        colidiu!",True,vermelho) #define
        mensagem
54
        tela.blit(texto,(300,300))#mostra
        o texto na tela
55
        todos_sprites.draw(tela)#desenha
        todos os objetos na tela
56
        pygame.display.update()#atualiza
        o display
57          relogio.tick(30) #30 quadros
        por segundo
58      pygame.quit()#encerra o programa
```

Nesse programa há uma bola (variável bola) e dois quadrados na tela (`obstaculo1` e `obstaculo2`). O jogador pode movimentar a bola e, se colidir com algum dos quadrados, o programa exibe uma mensagem dizendo que colidiu. Nele há duas classes: Bola, com um objeto `bola` e `Obstaculo` (para os quadrados), com dois objetos: `obstaculo1` e `obstaculo2`.

A primeira classe a ser definida é a classe `Bola`. Ela é a mesma do exemplo anterior (linhas 4 a 8).

A segunda classe definida é `Obstaculo`, que também herda as características de `pygame.sprite.Sprite` (linha 9). O método `__init__` dessa classe é definido tendo como parâmetros: a própria classe (`self`) mais `posicao_horizontal` e `posicao_vertical` (linha 10). Esses parâmetros terão que ser passados quando um objeto `Obstaculo` for criado: eles definirão a posição do obstáculo na tela. Após isso definimos a propriedade image do obstáculo: um quadrado de 50 x 50, com o comando `pygame.Surface([50,50])` na linha 12.

Após isso, nas linhas 13 e 14, são definidos o tamanho da imagem (que é um retângulo) e sua cor (vermelha). As coordenadas `x` e `y` da propriedade `rect` recebem os valores de `posicao_horizontal` (linha 15) e `posicao_vertical` (linha 16) que forem passados como parâmetros na criação do objeto; dessa forma, a posição do objeto é definida ao instanciar um objeto.

Definimos o grupo principal chamado `todos_sprites` (linha 18); nesse grupo serão adicionados todos os `sprites`. Um segundo grupo é definido chamado `lista_obstáculos` (linha 19), que abrangerá somente os objetos da classe `Obstaculo`.

O objeto bola é instanciado (linha 20): a variável bola passa a ser do tipo `Bola` e esse objeto é adicionado ao grupo `todos_sprites` (linha 21), pelo método `add()`.

São criados dois objetos da classe `Obstaculo`. O primeiro, `obstaculo1`, foi criado passando como parâmetros

(200, 100), que correspondem às suas posições horizontal e vertical respectivamente (linha 23); o objeto é adicionado ao grupo `todos_sprites` (linha 24) e também à `lista_obstaculos` (linha 25). O segundo objeto, `obstaculo2`, terá um posicionamento diferente (300, 200) e é adicionado aos mesmos grupos (linhas 26 a 28).

Na linha 38 é feita a detecção de colisão. A variável `lista_colisao` recebe os objetos com os quais a bola colidiu. É utilizado o comando `pygame.sprite.Spritecollide()` passando como atributos a variável `bola`, o grupo `lista_obstaculos` e o valor `False`. Em seguida o comando for percorre a `lista_colisao` (linha 39); se houver algum obstáculo na lista, a variável `colidiu` passa a ser verdadeira (linha 40).

Se a variável colidiu for falsa (linha 41), o objeto bola pode receber as ações do jogador: para isso são capturadas as teclas pressionadas e, de acordo com elas, o objeto é movimentado para umas das quatro direções (linhas 42 a 50), como visto no exemplo anterior.

Se a variável colidiu não for falsa (linha 51), então é definida a variável `fonte` (com a fonte `freesansbold.ttf` e tamanho 20) e montado o texto: "`Você colidiu`" na cor vermelha, para ser exibido na tela na posição (300,300), nas linhas 52 a 54.

Todos os objetos do grupo `todos_sprites` são desenhados na tela pelo comando `draw()`, na linha 55. Como todos os objetos pertencem a esse grupo, todos serão desenhados. A seguir, o `display` é atualizado e o número de ciclos por segundo é definido como 30.

SONS E MÚSICA

A interface gráfica é muito importante num jogo, mas normalmente, além de uma boa aparência, os jogos também emitem sons. Isso aumenta a quantidade de sentidos utilizados pelo jogador, induzindo maior interatividade, colaborando para a aprovação do jogo pelo usuário. A biblioteca pygame apresenta comandos para o carregamento de música e sons e suporta formatos de música como `.wav`, `.ogg`, `.mp3` etc. Além disso, permite que se controle o estado das músicas e sons durante o jogo, com comandos de executar, parar, pausar etc.

É comum nos jogos uma música executar continuamente como som de fundo. O comando que permite carregar músicas é `pygame.mixer.music.load()`, passando como parâmetro o nome do arquivo. No primeiro exemplo será demonstrado como carregar uma música (arquivo `musica.wav`) e fazê-la executar e pausar no jogo; o arquivo de áudio deve estar na mesma pasta do programa. O código do programa é o seguinte:

Exemplo 37 – Música

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init()#inicia módulo pygame
4    branco = (255,255,255) #cor branca
5    tela = pygame.display.set_mode([800,
     600]) #define o tamanho da janela
```

```
6   continuar = True #variável que controla o fluxo de repetição
7   relogio = pygame.time.Clock()#controle de tempo
8   pygame.mixer.music.load("musica.wav")#carrega arquivo de aúdio
9   pygame.mixer.music.play(-1)#define a quantidade de vezes que o áudio irá executar
10  #o valor -1 é usado para executar indeterminadas vezes
11  while continuar == True: #laço de repetição
12      for evento in pygame.event.get():#captura todos os eventos
13          if evento.type == pygame.QUIT: #verifica se é o evento de sair
14              continuar = False
15      if evento.type == pygame.KEYDOWN: #verifica se evento é pressionar tecla
16          if evento.key == pygame.K_LEFT: #se tecla é seta para esquerda
17
    pygame.mixer.music.pause()#pausa o áudio
```

```
18          if evento.key == pygame.K_RIGHT: #se tecla é seta para direita
19
pygame.mixer.music.unpause()#volta a executar o áudio a partir momento em que foi pausado
20          if evento.key == pygame.K_DOWN: #se tecla é seta para baixo
21
pygame.mixer.music.stop()#para o áudio
22          if evento.key == pygame.K_UP: #se tecla é seta para cima
23              pygame.mixer.music.play(-1) #volta a executar o áudio do início
24      tela.fill(branco) #pinta a tela com a cor branca
25      pygame.display.update()#atualiza o display
26      relogio.tick(30) #30 quadros por segundo
27  pygame.quit()#encerra o programa
```

No programa é carregada uma música que executa continuamente, porém o usuário pode pausá-la, continuar de onde pausou, pará-la ou executá-la de novo.

Na linha 8 é carregado o arquivo `musica.wav`. Na linha 9, usamos `pygame.mixer.music.play()`, cuja função é iniciar a música; o valor do parâmetro passado, se positivo, diz quantas vezes a música deve executar; passamos -1, que indica que deve tocar indefinidamente.

Se ocorrer um evento `KEYDOWN` (linha 15), o programa verifica qual foi a tecla pressionada. Se foi seta para a esquerda (`K_LEFT`), executa o comando `pygame.mixer.music.pause()`, que faz a música pausar (linha 17).

Se for utilizado a opção de seta para a direita (`K_RIGHT`), será executado o comando `pygame.mixer.music.unpause()`, que retoma a execução da música do ponto em que foi interrompida (linha 19).

Se foi seta para baixo (`K_DOWN`), executa o comando `pygame.mixer.music.stop()`, que para a música. Existe uma diferença deste para o `pause`: com `stop`, a música só pode ser retomada do início; com `pause`, pode ser retomada do ponto em que foi interrompida (linha 21).

Se foi seta para cima (`K_UP`), executa o comando `pygame.mixer.music.play(-1)`, que recomeça a execução de forma contínua (linha 23).

Além de uma música de fundo, normalmente há outros sons que ocorrem durante o jogo, provocados por algum evento, como ao colidir com algo, ao saltar etc. Normalmente esses sons são mais curtos que as músicas e não se repetem continuamente. Para provocar esses sons existe o comando `pygame.mixer.Sound()`, que também pede o nome do

arquivo de áudio como parâmetro. O exemplo a seguir mostra como carregar e reproduzir esses sons.

Exemplo 38 – Sons

```
1    import pygame, sys
2    from pygame.locals import *
3    pygame.init() #inicia módulo pygame
4    branco=(255,255,255)    #define    cor
     branca
5    tela  =  pygame.display.set_mode([600,
     400])#define tamanho da janela
6    continuar   =   True   #variável   que
     controla o fluxo de repetição
7    relogio=pygame.time.Clock()#controle
     de tempo
8    som1=pygame.mixer.Sound("som1.wav")#ca
     rrega primeiro som
9    som2=pygame.mixer.Sound("som2.wav")#ca
     rrega Segundo som
10   som3=pygame.mixer.Sound("som3.wav")#ca
     rrega terceiro som
11   som4=pygame.mixer.Sound("som4.wav")#ca
     rrega quarto som
12   while   continuar==True:   #laço   de
     repetição
13       for            evento            in
     pygame.event.get():#captura  todos  os
     eventos
```

```
14          if                evento.type==
    pygame.QUIT:#verifica se é o evento de
    fechar
15              continuar=False
16      if    evento.type==pygame.KEYDOWN:
    #evento pressionar teclar
17          if  evento.key==pygame.K_LEFT:
    #se tecla é seta para esquerda
18              som1.play()#reproduz     o
    primeiro som
19          if evento.key==pygame.K_RIGHT:
    #se tecla é seta para a direita
20              som2.play()#reproduz
    Segundo som
21          if  evento.key==pygame.K_DOWN:
    #se tecla é seta para baixo
22              som3.play()#reproduz
    terceiro som
23          if    evento.key==pygame.K_UP:
    #se tecla é seta para cima
24              som4.play()      #reproduz
    quarto som
25      tela.fill(branco)  #pinta  a  tela
    com a cor branca
26      pygame.display.update()#atualiza o
    display
27      relogio.tick(30)  #30  quadros  por
    segundo
```

```
28    pygame.quit()#encerra o programa
```

Nesse programa são carregados quatro sons diferentes; de acordo com a tecla que o usuário pressionar, é reproduzido um deles.

São criadas 4 variáveis, `som1`, `som2`, `som3` e `som4`; cada uma recebe o comando `pygame.mixer.Sound()`, tendo como parâmetro o nome do seu respectivo arquivo de áudio (linhas 8 a 11).

No laço principal verifica-se se o evento foi o pressionar de tecla (`KEYDOWN`); se verdadeiro, verifica-se qual foi a tecla pressionada. Se foi a seta para a esquerda, o primeiro som é reproduzido (linha 18), executando o método `play()` da variável `som1`. Se for pressionada seta para direita, é reproduzido o segundo som (linha 20); se for seta para baixo é reproduzido o terceiro (linha 22); se for seta para cima, é reproduzido o quarto (linha 24).

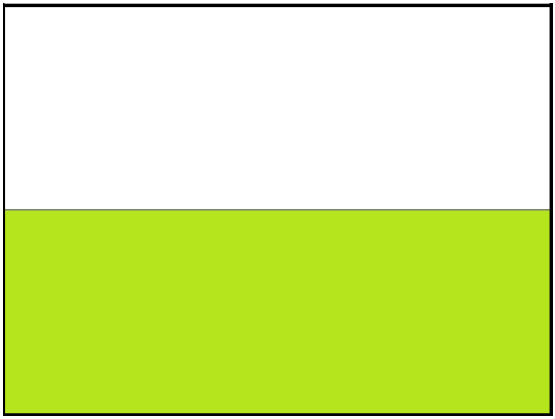
JOGO SUPERPYGAME

Utilizando `sprites`, detecção de colisão e sons, pode-se fazer jogos bem completos. O jogo que será programado apresenta como personagem principal o mesmo atleta do salto em distância, porém agora tendo que passar pelos obstáculos e adversários para vencer o jogo.

Para esse jogo serão necessárias algumas imagens. A primeira é a pista, que tem o tamanho de 800x600 pixels, sendo

formada por um retângulo branco e outro verde, ambos do mesmo tamanho, como mostrado na Figura 111:

Figura 111 – Pista do Jogo



Também serão utilizadas imagens de chamas de fogo, com o tamanho de 20x20 pixels, como mostrado na Figura 112:

Figura 112 – Fogo

Fonte : Wikimedia (Wikimedia.org)

Outra imagem necessária é da estrela (tamanho de 20x20 pixels), mostrada na Figura 113.

Figura 113 - Estrela

Fonte: Wikimedia (Wikimedia.org)

Também é necessário ter o desenho de uma pequena bola (ou outro objeto), para que o personagem possa lançá-la em seu adversário. Usaremos a seguinte bola, com diâmetro de 10 pixels, mostrada na Figura 114:

Figura 114 - Bola

Fonte: Wikimedia (Wikimedia.org)

Outra imagem é a do adversário do personagem principal, que pode ser desenhado ou buscado na Internet. Usaremos uma imagem de tamanho 20x20 pixels, como mostrada na Figura 115:

Figura 115 – Oponente

Fonte: Wikimedia (Wikimedia.org)

As imagens para o atleta (`atleta1`, `atleta2`, `atleta3` e `atleta 4`) são as mesmas já utilizadas no aplicativo salto em distância. Lembre-se de colocar todas as imagens na pasta do programa. Veja o código do jogo:

Jogo 4 - Super Pygame

```
1    import pygame, sys
2    from pygame.locals import *
3    pista                                    =
     pygame.image.load("pista_jogo.jpg")#ca
     rrega imagem da pista
4    personagem1                              =
     pygame.image.load("atleta1.png")#prime
     ira imagem do personagem
5    personagem2                              =
     pygame.image.load("atleta2.png")#segun
     da imagem do personagem
6    personagem3                              =
     pygame.image.load("atleta3.png")#terce
     ira imagem do personagem
```

```
7   personagem4 = pygame.image.load("atleta4.png")#quarta imagem do personagem
8   estrela = pygame.image.load("estrela.png")#carrega imagem da estrela
9   imagem_bola = pygame.image.load("bola.png")#carrega imagem da bola
10  fogo = pygame.image.load("fogo.jpg")#carrega imagem da fogueira
11  oponente = pygame.image.load("oponente.png")#carrega imagem do oponente
12  animacao = [personagem1, personagem1,personagem2,personagem2, personagem3,personagem3,personagem4,personagem4] #lista com imagens do personagem
13  class Personagem(pygame.sprite.Sprite): #classe Personagem
14      nivel = 1 #indica o nível de habilitade do personagem
15      indice_imagem = 0 #índice usado na animação do personagem
16      velocidade = 0 #velocidade do personagem
```

```
17          direcao_salto = 0 #direção do
    salto
18          def __init__(self): #construtor da
    classe
19
    pygame.sprite.Sprite.__init__(self)
    #inicia classe Sprite
20              self.image =
    animacao[self.indice_imagem] #imagem
    da classe
21              self.rect =
    self.image.get_rect()#coordenadas da
    imagem
22              self.rect.y = 250 #posiciona
    personagem no solo da pista
23          def update(self): #método
    atualizar
24              self.rect.x+= self.velocidade
    #movimenta de acordo com a velocidade
25              if self.rect.y <= 210:
    #verifica se está no ponto máximo do
    salto
26                  self.direcao_salto = 2
    #direciona salto para baixo
27              elif self.rect.y >= 250 and
    self.direcao_salto > 0:       #se
    chegou ao solo
28                  self.direcao_salto = 0 #o
    personagem para de descer
```

```
29
    self.rect.y+=self.direcao_salto
    #define a altura de acordo com o salto
30          if(self.direcao_salto==0 and
    self.velocidade != 0): #se personagem
    está no solo e em movimento
31              if(self.indice_imagem <
    7): #verifica se não é o último índice
    da lista
32                  self.indice_imagem+=1
    #passa-se para a proxima imagem da
    lista
33              else: #se for  última
    imagem da lista
34                  self.indice_imagem = 0
    #reinicia índice da lista em zero
35          self.image =
    animacao[self.indice_imagem] #atribui
    imagem da lista à classe
36  class Estrela(pygame.sprite.Sprite):
    #classe Estrela
37      def __init__(self): #construtor da
    classe
38
    pygame.sprite.Sprite.__init__(self)
    #inicia classe Sprite
39          self.image = estrela #a imagem
    da classe recebe a imagem da estrela
```

```
40          self.rect =
    self.image.get_rect()#define
    coordenadas da imagem
41          self.rect.x = 75 #posiciona
    imagem horizontalmente
42          self.rect.y = 200 #posiciona
    imagem verticalmente
43  class Bola (pygame.sprite.Sprite):
    #classe Bola
44      def
    __init__(self,posicao_horizontal,posic
    ao_vertical): #construtor da classe
45
    pygame.sprite.Sprite.__init__(self)#in
    icia classe Sprite
46          self.image = imagem_bola
    #imagem da classe recebe imagem da
    bola
47          self.rect =
    self.image.get_rect()#define
    coordenadas da imagem
48          self.rect.x =
    posicao_horizontal #recebe
    posicionamento horizontal
49          self.rect.y = posicao_vertical
    #recebe posicionamento vertical
50      def update(self): #método
    atualizar
```

```
51              self.rect.x+=5 #desloca-se
    para a direita
52      class Obstaculo(pygame.sprite.Sprite):
    #classe Obstaculo
53          def
    __init__(self,posicao_horizontal,posic
    ao_vertical): #construtor da classe
54
    pygame.sprite.Sprite.__init__(self)#in
    icia classe Sprite
55              self.image = fogo #imagem da
    classe recebe a imagem da fogueira
56              self.rect =
    self.image.get_rect()#define
    coordenadas da imagem
57              self.rect.x =
    posicao_horizontal #recebe
    posiocionamento horizontal
58              self.rect.y = posicao_vertical
    #recebe poscionamento vertical
59      class
    Adversario(pygame.sprite.Sprite):
    #Classe Obstáculo
60          movimento = 1 #direção do
    movimento
61          def __init__(self): #construtor da
    classe
```

```
62
    pygame.sprite.Sprite.__init__(self)
    #inicia classe Sprite
63          self.image = oponente #imagem
    da classe recebe imagem do oponente
64          self.rect =
    self.image.get_rect() #define
    coordenadas da imagem
65          self.rect.x = 220
    #posicionamento horizontal
66          self.rect.y = 280
    #posiocionamento vertical
67      def update(self): #método
    atualizar
68          if(self.rect.x <= 220): #se
    chegar a primeira fogueira
69              self.movimento=1 #muda
    direção para direita
70          if(self.rect.x >= 480): #se
    chegar a segunda fogueira
71              self.movimento=-1 #muda
    direção para esquerda
72          self.rect.x+=self.movimento
    #movimenta o oponente de acordo com a
    direção
73  class
    Obstaculo_voador(pygame.sprite.Sprite)
    :#classe Obstaculo voador
```

```
74          movimento = 1 #direção do
    movimento
75          def
    __init__(self,posicao_horizontal,posic
    ao_vertical): #construtor da classe
76
    pygame.sprite.Sprite.__init__(self)
    #inicia classe Sprite
77              self.image =
    pygame.Surface([20,20]) #define imagem
    como um quadrado
78              self.rect =
    self.image.get_rect()#define as
    coordenadas da imagem
79              self.image.fill(vermelho)
    #pinta quadrado de vermelho
80              self.rect.x =
    posicao_horizontal#recebe
    posicionamento horizontal
81              self.rect.y = posicao_vertical
    #recebe posionamento vertical
82          def update(self): #método
    atualizar
83              if(self.rect.y <= 170): #se
    chegar a altura máxima
84                  self.movimento = 1 #muda
    movimento para descida
85              if(self.rect.y >= 280): #se
    chegar a altura mínima
```

```
86                  self.movimento=-1 #muda
    movimento para subida
87              self.rect.y+=self.movimento
    #movimenta de acordo com a direção
88  pygame.init() #inicia módulo pygame
89  som_lancamento =
    pygame.mixer.Sound("lancamento.wav")#s
    om de lançar bolinhas
90  som_salto =
    pygame.mixer.Sound("salto.wav")
    #carrega som do salto
91  som_explosao =
    pygame.mixer.Sound("explosao.wav")#som
    de explosão das bolinhas
92  todos_sprites =
    pygame.sprite.Group()#grupo geral de
    Sprites
93  lista_obstaculos =
    pygame.sprite.Group()#grupo de
    obstáculos
94  lista_bolas =
    pygame.sprite.Group()#grupo de bolas
95  tela = pygame.display.set_mode([800,
    600]) #define tamanho da janela
96  vermelho = (255,0,0) #cor vermelha
97  fonte =
    pygame.font.Font("freesansbold.ttf",20
    )#define a fonte
```

```
98  personagem = Personagem()#instancia
    Objeto da classe Personagem
99  todos_sprites.add(personagem)
    #adiciona personagem ao grupo geral de
    Sprites
100 estrela = Estrela()#instancia objeto
    da classe Estrela
101 todos_sprites.add(estrela)#adiciona
    estrela ao grupo geral de Sprites
102 obstaculo1 =
    Obstaculo(200,280)#instancia objeto da
    classe Obstaculo
103 todos_sprites.add(obstaculo1)
    #adiciona obstáculo ao grupo geral de
    Sprites
104 lista_obstaculos.add(obstaculo1)#adici
    ona obstáculo ao grupo de Obstáculos
105 obstaculo2 =
    Obstaculo(500,280)#instancia objeto da
    classe Obstaculo
106 todos_sprites.add(obstaculo2)#adiciona
    obstáculo ao grupo geral de Sprites
107 lista_obstaculos.add(obstaculo2)
    #adiciona obstáculo ao grupo de
    Obstáculos
108 adversario = Adversario()#instancia
    objeto da classe adversário
```

```
109 todos_sprites.add(adversario)
    #adiciona adversário ao grupo geral de
    Sprites
110 lista_obstaculos.add(adversario)
    #adiciona adversário ao grupo de
    Obstáculos
111 voador1 =
    Obstaculo_voador(600,220)#instancia
    objeto da classe Obstaculo_voador
112 todos_sprites.add(voador1) #adiciona
    obstáculo voador ao grupo geral de
    Sprites
113 lista_obstaculos.add(voador1)
    #adiciona obstáculo voador ao grupo de
    Obstáculos
114 voador2 = Obstaculo_voador(620,280)
    #instancia objeto da classe
    Obstaculo_voador
115 todos_sprites.add(voador2) #adiciona
    obstáculo voador ao grupo geral de
    Sprites
116 lista_obstaculos.add(voador2)
    #adiciona obstáculo voador ao grupo de
    Obstáculos
117 continuar = True #variável que
    controla o fluxo de repetição
118 relogio = pygame.time.Clock()#controle
    de tempo
119 colidiu = False #controle de colisão
```

```
120 while continuar == True: #laço de repetição
121     tela.blit(pista,(0,0)) #monta imagem da pista
122     for evento in pygame.event.get():#captura todos os eventos
123         if evento.type == pygame.QUIT: #verifica se é o evento de sair
124             continuar = False
125     tecla_pressionada = pygame.key.get_pressed()#captura tecla pressionada
126     if(personagem.rect.x >= 700): #verifica se personagem chegou ao final da pista
127         texto = fonte.render("Você venceu!",True,vermelho) #monta mensagem
128         tela.blit(texto,(400,50)) #monta texto na tela
129     else: #se personagem ainda não chegou ao final da pista
130         if(colidiu == False): #verifica se personagem colidiu
131             if tecla_pressionada[pygame.K_UP]: #se seta para cima está pressionada
```

```
132                     if(personagem.rect.y
>= 250): #se persongem está no solo
133
personagem.direcao_salto=-1 #define
direção do salto (subida)
134                         som_salto.play()
#reproduz som do salto
135                 if
tecla_pressionada[pygame.K_RIGHT]: #se
seta para direita pressionada
136                     personagem.velocidade
= 2 #velocidade de 2 pixels para
direita
137                 elif
tecla_pressionada[pygame.K_LEFT]:#seta
para esquerda pressionada
138
personagem.velocidade=-2 #velocidade
de 2 pixels para esquerda
139                 elif personagem.rect.y ==
250: #se não pressionar teclas de
movimento e estiver no solo
140                     personagem.velocidade
= 0 #muda velocidade para zero
(parado)
141                 if
tecla_pressionada[pygame.K_SPACE]: #se
pressionar barra de espaço
142                     if personagem.nivel ==
2: #se o nível do personagem é dois
```

```
143                         bola =
    Bola(personagem.rect.x,
    personagem.rect.y+30) #um objeto da
    classe bola é instanciado, e a sua
    posição e de acordo com a do
    personagem
144
    todos_sprites.add(bola)#adiciona bola
    ao grupo geral de sprites
145
    lista_bolas.add(bola) #adiciona bola
    ao grupo de bolas
146
    som_lancamento.play()#reproduz som de
    lançamento da bola
147
    if(personagem.rect.colliderect(estrela
    .rect)): #verifica se personagem
    colide com a estrela
148
    todos_sprites.remove(estrela)#remove a
    estrela do grupo
149                     personagem.nivel=2
    #muda o nível do personagem
150                 lista_colisao =
    pygame.sprite.spritecollide(personagem
    , lista_obstaculos, False) #verifica
    se o personagem colide com algum
    obstáculo
```

```
151                 if len(lista_colisao)>1:
    #verifica se há objetos na lista
152
    todos_sprites.remove(personagem)#se
    encontrar, remove personagem
153                     colidiu = True
    #estabelece que houve colisão
154
    bola_colisao=pygame.sprite.spritecolli
    de(adversario,lista_bolas,False)
    #verifica se o adversário colide com
    alguma bola
155                 for bola in bola_colisao:
    #percorre lista
156                     if
    todos_sprites.has(adversario): #se
    adversário existe no grupo
157
    som_explosao.play()#reproduz som de
    explosão
158
    todos_sprites.remove(adversario)#remov
    e adversário do grupo
159
    todos_sprites.remove(bola)#remove bola
    do grupo
160
    lista_obstaculos.remove(adversario)
    #remove adversário do grupo
```

```
161
    lista_bolas.remove(bola) #remove bola
    do grupo
162
    todos_sprites.update()#executa método
    atualizar dos objetos
163             else: #se colidiu com algum
    obstáculo
164                 texto = fonte.render("Você
    perdeu, pressione "R" para
    reiniciar!",True,vermelho) #monta
    mensagem
165                 tela.blit(texto,(400,50))
    #monta texto na tela
166
    if(tecla_pressionada[pygame.K_r]):#se
    pressionar tecla "r"
167                     personagem.rect.x = 0
    #retorna personagem a posição inicial
168                     personagem.rect.y =
    250 #retorna personagem a posição
    inicial
169
    personagem.direcao_salto = 0 #define
    direção do salto com nula
170
    todos_sprites.add(personagem)#adiciona
    ao grupo geral de Sprites
```

```
171                     for bola in
lista_bolas: #procura bolas na lista
de bolas
172
lista_bolas.remove(bola) #remove as
bolas do grupo
173
todos_sprites.remove(bola)#remove as
bolas do grupo
174                     if(personagem.nivel ==
2): #se personagem é do nível 2
175
todos_sprites.add(estrela)#adiciona a
estrela ao grupo
176                         personagem.nivel=1
#define o nível do personagem como 1
177                     if not
todos_sprites.has(adversario):#se o
adversário foi destruido
178
todos_sprites.add(adversario)
#adiciona adversário ao grupo
179
lista_obstaculos.add(adversario)
#adiciona adversário ao grupo
180                     colidiu = False
#define colidiu como falso
181     todos_sprites.draw(tela) #desenha
todos os objetos na tela
```

```
182         pygame.display.update()#atualiza
    display e implementa as mudanças
183         relogio.tick(30) #30 quadros por
    segundo
184     pygame.quit() #encerra o programa
```

Nesse jogo o objetivo é chegar ao final da pista sem tocar nos obstáculos nem nos adversários. No início da pista há uma estrela que pode ser pega pelo personagem, dando-lhe a habilidade de jogar pequenas bolas que destroem o adversário localizado na parte central da pista. O primeiro obstáculo é uma fogueira, que ele deve pular por cima sem tocá-la, senão perderá o jogo. A seguir, há um adversário em movimento; ele também pode pular por cima, ou, se tiver apanhado a estrela, jogar uma bola nele. Depois há outra fogueira; ele deve proceder da mesma forma que com a primeira. Após isso, há dois blocos que se movimentam para cima e para baixo na pista; ele deve passar por baixo deles para vencer o jogo. O código do jogo é explicado a seguir.

Nas linhas 3 a 11 são carregadas todas as imagens que serão utilizadas no jogo: a pista, as quatro imagens consecutivas do personagem, a estrela, a bola, a chama de fogo e o oponente.

Na linha 12 é montada a lista das imagens que fazem a animação do personagem. A lista recebe duas vezes cada imagem, o que torna a movimentação mais lenta e suave no jogo. Com apenas uma imagem de cada, a impressão é que ele se movimenta muito rápido. O ideal é que haja mais imagens consecutivas diferentes (não apenas quatro), para dar mais realismo ao movimento.

A primeira classe no jogo é Personagem, a do personagem principal, que herda as características de `pygame.sprite.Sprite` (linha 13). Essa classe tem variáveis próprias. A primeira é `nível` (linha 14), usada para atribuir habilidades ao personagem: no nível 1, ele não tem habilidade de jogar bolas no adversário, no nível 2, tem. A variável `indice_imagem` é usada para marcar qual imagem do personagem deve ser exibida naquele momento (linha 15). A variável `velocidade` (linha 16) é usada para movimentar o personagem: quando = 0, o personagem fica parado; quando = 2, se movimenta para frente; quando = -2, para trás. A variável `direcao_salto` (linha 17) define a direção vertical do salto: quando = -1, o personagem sobe; quando = 2, ele desce; quando = 0, permanece na mesma posição vertical.

As linhas 18 a 22 apresentam o método construtor da classe `Personagem`. É chamado o método `pygame.sprite.Sprite.__init__()`. Logo após, o atributo image recebe a imagem da lista de animação (da posição índice_imagem, que de início é 0). O atributo `y` de `rect` recebe 250, que coloca o atleta na posição vertical 250; o atributo `x` fica valendo 0, pois, não tendo sido definido, assume 0 como valor padrão.

As linhas 23 a 35 definem o segundo método da classe, `update`(), cuja função é atualizar os seus atributos. A palavra `def` indica que é um novo método; passa-se como parâmetro um objeto da própria classe (`self`) para que o método tenha acesso a todos os atributos. Na linha 24 há um incremento no atributo `x` de `rect`, acrescentado a `velocidade`. Após isso (linha 25), há um `if` para verificar se o personagem já chegou à altura máxima do salto (210). Como a altura inicial é 250 e a

altura máxima é 210, então se salta 40 pixels de altura: vale ressaltar que, quanto menor a posição vertical, mais próximo da parte superior da tela. Se a condição for verdadeira (`<=210`), o personagem deve descer, assim `direcao_salto` recebe 2 (linha 26). Porém, se ele continuar descendo, vai ficar abaixo da pista; por isso é necessária uma condição de parada, definida no `elif`: se o atributo `y >= a 250` e se `direcao_salto > 0` (linha 27). A primeira verifica se ele está no chão e a segunda se seu movimento é de descida; se forem verdadeiras, a variável `direcao_salto` passa a ser 0, fazendo com que o personagem pare de cair (linha 28). Após essas condições, a variável `direcao_salto` é somada ao atributo `y` de `rect` (linha 29). Como `direcao_salto` pode valer -1, 0 ou 2, é nessa linha que as mudanças na posição vertical são aplicadas.

Nesse método também é feito o controle da animação do personagem (linhas 30 a 35). O `if` seguinte (linha 30) verifica se o atleta está correndo (`velocidade!=0`), e não pulando (`direcao_salto==0`). Nesse jogo ele não pode correr pelo ar, a imagem permanece parada durante o salto; da mesma forma, se o personagem estiver parado, não se pode trocar a sua imagem. Se forem atendidas as duas condições, troca-se a imagem da animação: se o índice `indice_imagem < 7` (como a lista de animação tem 8 imagens, o índice pode ir de 0 a 7), é incrementado em 1; senão volta-se a usar a primeira imagem (`indice_imagem=0`).

Por fim, na linha 35 desse método, o atributo `image` recebe a imagem da lista de animação da posição `indice_imagem`.

As linhas 36 a 42 definem a classe `Estrela`, que dá habilidade especial ao personagem. A classe tem só o método

__init__. Ela recebe a imagem da estrela (contida na variável de mesmo nome). A estrela é posicionada a 75 pixels na horizontal e a 200 pixels na vertical, ficando suspensa no ar, acima do personagem. Essa classe não possui o método update, pois sua posição não muda durante o jogo. O único evento que pode ocorrer com ela é desaparecer, porém essa ação não é provocada por ela mesma, mas, sim, pelo personagem.

Quando o personagem pega a estrela, ele ganha a habilidade de jogar pequenas bolas. Para isso é necessário haver uma classe para elas. A classe Bola tem dois métodos __init__ e update. O método __init__ recebe a própria classe (self), mais a posicao_horizontal e a posicao_vertical (linha 44). Não se pode fixar a posição inicial do objeto bola criado, pois depende da posição do personagem no momento. Por isso, quando uma bola é lançada, são passadas para a classe Bola as posições horizontal e vertical do personagem. A classe recebe a imagem da bola (linha 46). A posição horizontal da classe, x, recebe o valor da variável posicao_horizontal (linha 48) passada como parâmetro, que contém a posição do personagem. Da mesma forma, a posição vertical da classe, y, recebe o valor da variável posicao_vertical que contém a posição do personagem (linha 49).

As linhas 50 e 51 definem o método update da classe Bola, responsável pelo deslocamento da bola. Nela, o atributo x de rect é acrescido de 5: isso faz com que, a cada atualização, ela se desloque 5 pixels para a direita.

Também é definida uma classe para as fogueiras, chamada Obstaculo (linha 52). Essa classe recebe, em seu método inicial: ela própria (self), uma posição horizontal e

uma vertical (linha 53). Ela recebe a imagem da variável `fogo` (linha 55). As posições horizontal e vertical recebem as variáveis `posicao_horizontal` e `posicao_vertical` recebidas como parâmetro (linhas 57 e 58).

As linhas 59 a 72 definem uma classe para o adversário que se movimenta na pista. Ela tem uma única variável própria chamada movimento (linha 60), cuja função é deslocar o objeto na tela: quando o valor é 1, o objeto se desloca para a direita e quando é -1, para a esquerda.

A classe `Adversario` tem dois métodos: `__init__` e `update`. No método `__init__` (linhas 61 a 66) é definida a imagem da classe, que está contida na variável `oponente`, e o posicionamento da imagem.

No método `update()` da classe Adversário é definida a movimentação do objeto na pista. A primeira condição (linha 68) verifica se a sua posição horizontal é menor ou igual a 220 (posição limite para se aproximar da primeira fogueira); se for, o valor da variável `movimento` é definido como 1, para deslocá-lo para a direita (linha 69). Se a sua posição horizontal for maior ou igual a 480 (posição limite para se aproximar da segunda fogueira), o valor de `movimento` passa a ser -1, para deslocá-lo para a esquerda (linhas 70 e 71). Na linha 72, o atributo `x` de `rect` é acrescido do valor do movimento; nessa linha é que ocorre a alteração do valor da posição horizontal.

A última classe do jogo é `Obstaculo_voador` (linhas 73 a 87). Essa é a classe dos blocos que flutuam no final da pista. Ela tem um único atributo, a `variável` movimento (linha 74), utilizada para controlar o movimento vertical do objeto; quando o seu valor é 1, o bloco desce e quando é -1, ele sobe.

A classe tem também dois métodos: `__init__` e `update`. O método `__init__` da classe recebe ela própria (`self`) e as posições horizontal e vertical. O atributo `image` é definido como um quadrado de 20x20 (linha 77). A cor do quadrado é definida como vermelha (linha 79), e a posição é definida de acordo com os dois parâmetros passados para a classe (linhas 80 e 81).

O método `update()` da classe `Obstaculo_voador` é responsável por controlar o movimento do bloco. O primeiro `if` (linha 83) verifica se a posição vertical do objeto é menor ou igual a 170: esta é a altura máxima, portanto o objeto precisa descer; a variável `movimento` então recebe 1 (linha 84). O segundo `if` (linha 85) verifica se a posição vertical do objeto é maior ou igual a 280: ele chegou ao chão, portanto deve subir; a variável `movimento` então recebe -1 (linha 86). Na linha 87, a variável movimento é somada à posição vertical `y`; como ela pode ser 1 ou -1, poderá aumentar ou diminuir a posição vertical do objeto.

Nas linhas 89 a 91 são definidas três variáveis para os `sons`: `som_lancamento`, para o lançamento das bolas, `som_salto`, para o salto e `som_explosao`, para quando o personagem acertar uma bola no oponente.

São criados três grupos com o comando: `todos_sprites` é um grupo geral que contém todos os objetos; `lista_obstaculos` conterá os obstáculos do jogo (esse grupo é útil para verificar quando o personagem colide e perde o jogo); `lista_bolas` conterá as bolas que o personagem dispara (linhas 92 a 94).

É criado um objeto da classe `Personagem` chamado `personagem`, que é adicionado ao grupo `todos_sprites`

pelo método `add()`. Esse é o ator principal do jogo, responsável por realizar a maior parte das ações do jogo (linhas 98 e 99).

É criado um objeto da classe `Estrela` e adicionado ao grupo `todos_sprites` (linhas 100 e 101). A estrela não é um obstáculo ao personagem, mas, sim, um item que adiciona habilidade ao personagem; assim, não é adicionada ao grupo dos obstáculos.

Teremos dois obstáculos da classe `Obstaculo`. A linha 102 cria o primeiro, chamado `obstaculo1`, a primeira chama de fogo do jogo. Os parâmetros 200 e 280 correspondem às posições horizontal e vertical onde o objeto será posicionado. Em seguida o objeto é adicionado aos grupos `todos_sprites e lista_obstaculos` (linhas 103 a 104).

É criado `obstaculo2`, também da classe `Obstaculo` (a segunda chama de fogo do jogo), com os parâmetros 500 x 280, também adicionado aos grupos `todos_sprites` e `lista_obstáculos` (linhas 105 a 107).

Na linha 108 é criado um objeto chamado `adversario` da classe `Adversario` (que é o personagem que se move no centro do cenário), também adicionado aos grupos `todos_sprites` e `lista_obstaculos` (linhas 109 e 110).

Teremos dois obstáculos da classe `Obstaculo_voador`, que são os blocos que flutuam na parte final da pista. O primeiro, chamado voador1, tem os parâmetros 600 e 220. O segundo, chamado de `voador2`, tem os parâmetros 620 e 280. Ambos são adicionados aos grupos

`todos_sprites` e `lista_obstaculos` (linhas 111 a 116).

A primeira coisa a ser verificada (após verificar o evento de sair) é se o usuário venceu: se chegou ao final da pista (`posição horizontal >= 700`) sem tocar em nenhum obstáculo. O `if` (linha 126) verifica a posição horizontal do personagem (`personagem.rect.x >= 700`); se verdadeiro, exibe-se mensagem "`Você venceu`" (linhas 127 e 128).

Se não atingiu a posição 700 (`else`), ainda está jogando (linha 129). O `if` (linha 130) verifica se o personagem bateu em algum obstáculo; se a variável `colidiu` é falsa, podem continuar a acontecer as ações do jogo.

Se não colidiu com nada, o atleta pode realizar suas ações. Nas linhas 131 a 134 é verificada a tecla seta para cima: se foi pressionada, `K_UP`, é disparada a ação do salto. Porém, se o personagem estiver no ar, não pode saltar novamente; assim, verifica-se sua posição vertical `personagem.rect.y`: se for maior ou igual a 250 (linha 132), ele está no chão; atribui-se, então, -1 para `personagem.direcao_salto`, para que ele suba (linha 133). Assim, a cada laço de repetição o personagem tem sua posição vertical subtraída de 1, ficando mais perto da parte de cima da tela (cujo valor é 0). A descida do personagem é controlada pela própria classe do personagem. A linha 134 executa um som quando o personagem salta.

Se for pressionada a tecla seta para a direita, `K_RIGHT` (linha 135), é acionada a ação de correr para frente, atribuindo 2 a `personagem.velocidade`, o que faz o personagem se movimentar 2 pixels à direita a cada repetição do laço (linha 136).

Se o usuário pressionar a tecla seta para esquerda, `K_LEFT` (linha 137), é executada a ação de correr para trás: atribui-se -2 a `personagem.velocidade`, o que faz o personagem se movimentar 2 pixels à esquerda a cada repetição do laço (linha 138).

Se não for pressionada nenhuma das teclas citadas acima e o personagem estiver no chão (linha 139), ele deve ficar parado: basta atribuir 0 a `personagem.velocidade` (linha 140).

Como a tecla `K_SPACE` dispara as bolas para o personagem, precisa ser verificada. O personagem só tem essa habilidade se tiver apanhado a estrela, passando para o nível 2. Se a tecla foi pressionada, verifica-se também se o personagem está no nível 2. Se os dois `ifs` (linhas 141 e 142) forem verdadeiros, é criado um novo objeto do tipo `Bola` na variável `bola` (linha 143) e adicionado aos grupos `todos_sprites e lista_bolas` (linhas 144 e 145) e toca-se um som de lançamento de bola (linha 146). O objeto bola deve ser criado na posição em que está o personagem, por isso passam-se como parâmetros as posições do personagem, horizontal e vertical (que está 30 pixels abaixo).

É necessário também verificar as colisões. Se o personagem colidir com a estrela, verificado pelo método `colliderect`(), seu nível aumenta para 2. O método compara se o retângulo (`rect`) do personagem invadiu o `rect` da estrela; se isso ocorrer, a estrela é removida do grupo `todos_sprites` (linhas 147 a 149).

As linhas 150 e 151 verificam se o personagem colidiu com os obstáculos do jogo. O primeiro comando cria uma lista (a variável `lista_colisao`) com os obstáculos com os quais

o personagem colidiu: o comando `pygame.sprite.spritecollide()` recebe como parâmetros o personagem, o grupo `lista_obstaculos` e um valor `False`. Para verificar se houve colisão, verifica-se o tamanho da `lista_colisao` (linha 151) e, se for maior que zero (o que significa que colidiu), remove o personagem com o comando `todos_sprites.remove`(personagem) (linha 152) e a variável `colidiu` recebe `True` (linha 153), para que o personagem não possa mais realizar ações no jogo.

As linhas 154 e 155 verificam se alguma bola lançada pelo personagem colidiu com o adversário, para eliminá-lo, se isto ocorrer. O mesmo comando `pygame.sprite.spritecollide` é aqui usado com os parâmetros `adversario` e `lista_bolas`, criando a lista chamada `bola_colisao`. O comando `for` percorre a lista `bola_colisao` e, achando algum objeto, deve fazer o som de explosão (linha 157) e remover os objetos que colidiram: `adversario` e `bola` (linhas 158 a 161). Antes dessas ações (linha 156), é feita a verificação se `adversario` ainda existe na lista `todos_sprites`, o que é feito pelo método `has()`.

Se o personagem não chegou ao final da pista e colidiu com algo (linha 163), então ele perde. Quando isso ocorre é exibida uma mensagem na tela: "`Você perdeu, pressione r para reiniciar`" (linhas 164 e 165).

Se o usuário pressionar a tecla r (linha 166) o jogo deve ser reiniciado. Para isso, as situações do início do jogo devem ser restauradas. O personagem é colocado no início da pista, atribuindo 0 para sua posição horizontal (linha 167) e 250 para a vertical (linha 168) e definindo a direção do salto como 0 (linha 169). Ao tocar em algum obstáculo e perder, ele foi removido da

lista de `sprites`, portanto é necessário adicioná-lo novamente ao grupo `todos_sprites` (linha 170).

É necessário também remover todas as bolas que estejam em movimento. Para isso, remove-as dos grupos `lista_bolas e todos_sprites` (linhas 171 a 173).

Se o personagem estiver no nível 2, ele pegou a estrela; assim, ela também foi removida e deve ser inserida de volta. A estrela é então adicionada novamente ao grupo `todos_sprites` e o nível do personagem volta para 1 (linhas 174 a 176).

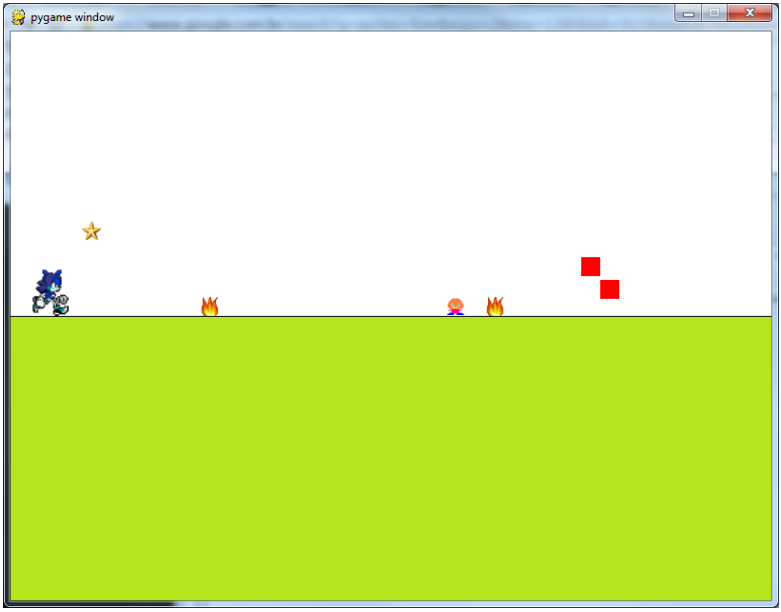
Se o adversário tiver sido destruído, também é necessário colocá-lo de volta ao jogo. O método `has()` verifica se `todos_sprites` contém `adversario` (o operador `not` usado é usado para negar a condição). Portanto, se não houver adversário, adiciona-o aos grupos `todos_sprites` e `lista_obstaculos` (Linhas 177 a 179).

Para que o jogo não reinicie novamente, a variável `colidiu` recebe `False` (linha 180), pois poderia estar valendo `True`, o que estaria encerrando novamente o jogo a ser iniciado.

Os objetos são desenhados na tela pelo método `draw` do grupo `todos_sprites`; as alterações são implementadas pelo comando `pygame.display.update`(), e o número de quadros por segundo é definido como 30.

O resultado desse jogo será semelhante ao apresentado na Figura 116.

Figura 116 – Jogo Superpygame



Referências

O tópico inicial sobre a história do Rasperry Pi e seus fundadores foram tirados dos seguintes sites (acessos feitos durante o mês de junho/2015):

1. https://www.raspberrypi.org
2. https://www.raspberrypi.org/about/
3. https://www.raspberrypi.org/documentation/configuration/raspi-config.md
4. http://www.cl.cam.ac.uk/~rdm34/
5. http://www.emma.cam.ac.uk/contact/fellows/?fellow=239
6. http://www.newelectronics.co.uk/electronics-interviews/fruits-of-his-labours-interview-with-pete-lomas-raspberry-pi-hardware-designer/42132/
7. http://www.ukbusinessangelsassociation.org.uk/about
8. https://www.sdcard.org/downloads/formatter_4/SDFormatter_4e.pdf
9. https://commons.wikimedia.org/wiki/File:Micro_USB_phone_charger.jpg
10. https://commons.wikimedia.org/wiki/File:EthernetCableBlue.jpg
11. https://commons.wikimedia.org/wiki/File:HDMI-Kabel.jpg
12. https://commons.wikimedia.org/wiki/File:Micro-SD-Adapter.jpg
13. Demais tópicos a respeito do funcionamento da Placa Lógica Raspberry Pi e do Software Scratch:

14. GOMES, Anabela; HENRIQUES, Joana; MENDES, José Antônio. *Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores*. 2008. Disponível em <http://etf.educom.pt/index.php/etf/article/view/23/16>.
15. MARJI, Majed. *Aprenda a programar com Scratch*. São Paulo: Novatec Editora, 2014.
16. MONK, Simon. *Programando o Raspberry Pi – Primeiros Passos com Python*. São Paulo: Novatec Editora, 2013.
17. RICHARDS, Matt, WALLACE, Shawn. *Primeiros Passos com o Raspberry Pi*. São Paulo: Novatec Editora, 2013.
18. UPTON, Eben; HALFACREE, Gareth. *Raspberry Pi Manual do Usuário*. São Paulo: Novatec Editora 2013.

Tópicos sobre a história e o funcionamento dos softwares Python e Pygame (acessos feitos durante o mês de junho/2015):

19. http://wiki.python.org.br/
20. http://pygame.org/hifi.html
21. http://www.pygame.org/wiki/about

Aprender a programar pode ser uma atividade simples e divertida se iniciada da forma correta, por isso, escolher a primeira linguagem de programação é muito importante, pois uma sintaxe complexa pode desestimular o aprendizado.

A programação não deve ser vista como algo árduo, mas como uma arte. Através dela é possível construir desde aplicações simples até simulações do mundo real e jogos complexos.

Mais do que propriamente ensinar, o objetivo deste livro é estimular o leitor a gostar de programar. Para isso foram escolhidas ferramentas simples e exemplos didáticos.

Além de fazer entender como funcionam os jogos e programas, este livro torna possível construir os seus próprios projetos.

ISBN 978-85-63406-94-1
9 788563 406941

MINISTÉRIO DA EDUCAÇÃO
BRASIL GOVERNO FEDERAL