Leonardo Haddad Carlos

# CodeWay: simulador interativo 3D voltado para o ensino de programação

Trabalho de Formatura Supervisionado que consiste no desenvolvimento de um aplicativo destinado à exploração de recursos visuais no processo de aprendizagem de programação.

Universidade de São Paulo – USP

Instituto de Matemática e Estatística

Bacharelado em Ciência da Computação

Orientador: Marco Dimas Gubitoso

Coorientador: Guilherme Fernandes Otranto

Brasil

2015

Leonardo Haddad Carlos
CodeWay: simulador interativo 3D voltado para o ensino de programação

Orientador: Marco Dimas Gubitoso
Coorientador: Guilherme Fernandes Otranto

TCC (Graduação) – Universidade de São Paulo – USP
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação, 2015.

*Ensinar não é transferir conhecimento, mas criar as possibilidades para a sua própria produção ou a sua construção.*
*(Paulo Freire)*

# Resumo

Durante os últimos anos, com a intensificação do avanço tecnológico, novos paradigmas de produtos e serviços surgiram para adaptar os métodos e modelos de negócio já existentes à constante evolução da tecnologia. Hoje em dia, as pessoas podem acessar, de forma praticamente instantânea, uma enorme variedade de utilitários e ferramentas utilizando apenas um smartphone. Essa revolução tecnológica, ainda hoje, abre espaço para uma série de novas aplicações e, para que a criação dessas aplicações não se restrinja aos profissionais de T.I., o ensino de computação acaba ganhando cada vez mais importância no cenário educacional.

Este projeto consiste na criação de um aplicativo multiplataforma que possa ser usado como ferramenta no processo de aprendizagem de programação, utilizando elementos visuais para introduzir a lógica de programação e os conceitos envolvidos na criação de código fonte. Para isso, foi criada uma estrutura de simulação de comportamentos em objetos configuráveis, na qual um objeto configurável pode receber trechos de código executável, escrito pelo próprio usuário em uma linguagem simples, definida especificamente para o projeto.

Para o desenvolvimento do aplicativo, foi utilizada uma engine (motor, em inglês) gráfica chamada Unity3D, que suporta programação de comportamentos em C#, Javascript ou Boo (uma linguagem baseada em Python). Todos os scripts necessários para o desenvolvimento do projeto foram escritos em C#.

Esta monografia foi elaborada para a disciplina MAC0499 - Trabalho de Formatura Supervisionado e é dividida em duas partes:

- Parte Objetiva, que contempla os conceitos e detalhes técnicos importantes para o desenvolvimento do projeto.
- Parte Subjetiva, que trata de questões mais abstratas, como as dificuldades enfrentadas e as disciplinas mais relevantes para o desenvolvimento do projeto.

**Palavras-chaves**: aprendizado digital, programação, elementos visuais, interatividade, Unity3D.

# Sumário

# Lista de ilustrações

# Introdução

Durante a graduação, um aluno do curso de ciência da computação aprende diversos conceitos e práticas fundamentais não apenas para a construção de software eficiente e de qualidade, mas também para a participação em outros processos envolvidos em um projeto de computação, como a análise de requisitos, a definição do metamodelo de desenvolvimento a ser usado, a estruturação das atividades a serem executadas e a criação, manutenção e gerenciamento do banco de dados, entre outros.

No entanto, muitos desses conhecimentos não são imperativos na criação de protótipos. Ou seja, é possível testar ideias na prática através de ferramentas de desenvolvimento de alto nível, sem necessariamente desenvolver um projeto robusto e bem estruturado no primeiro momento. Isso permite, por exemplo, que empreendedores com conhecimentos superficiais em programação realizem experimentos para medir a demanda e a aceitação por um aplicativo antes de investir, de fato, em um projeto completo e bem elaborado, com um time de desenvolvedores dedicados.

## 1 Motivação

Esse novo paradigma atrai novos olhares e pontos de vista para o ensino de programação. O Reino Unido, por exemplo, incluiu recentemente a programação na lista de unidades curriculares obrigatórias em todas as escolas, do ensino primário ao secundário, segundo reportagem de Dredge (2014), servindo como inspiração para o desenvolvimento deste trabalho, assim como as iniciativas da Code.org (`https://code.org/`) e o Projeto MIT App Inventor (M.I.T. (2015)).

Por outro lado, a iniciação tecnológica dos jovens, cada vez mais precoce, abre espaço para mudanças nos processos de aprendizagem que, ainda nos dias de hoje, contam majoritariamente com aulas puramente expositivas e material didático baseado apenas nos livros de papel. Para engajar eficientemente os jovens nascidos no século XXI, é necessário adentrar no universo virtual ao qual eles são tão ligados, unindo a gama de ferramentas disponibilizadas pela tecnologia com processos pedagógicos já consolidados.

Nesse âmbito, surgem os programas pedagógicos baseados em livros-aplicativos que, através de ambientes virtuais, permitem que o usuário pratique e aprimore seus conhecimentos através de ferramentas digitais, interativas e convidativas. Essa nova perspectiva educacional, já mencionada por Saldaña (2015), aliada à crescente demanda por produtos e serviços digitais, motivou a construção de uma plataforma digital que seja capaz de apoiar o ensino de programação de jovens e crianças.

# 2 Objetivo

O objetivo deste projeto é explorar essa demanda tecnológica do setor educacional, criando uma ferramenta interativa que permita que o usuário crie diferentes scripts (escritos em uma linguagem própria do projeto) e simule o comportamento de diferentes objetos programáveis sob o controle destes scripts.

Por ser destinado ao ensino de computação, esse aplicativo permitirá que o usuário crie seus scripts através de comandos textuais ou de elementos visuais (associados aos comandos textuais e configurados de forma a apresentar o código do script como um fluxograma) podendo, inclusive, alternar livremente entre estes dois nodos de criação e edição de scripts durante a execução do aplicativo.

Nos capítulos que seguem, a estrutura interna e a organização de conceitos relacionados ao aplicativo serão explicados em maiores detalhes, permitindo que o leitor compreenda melhor as decisões e implementações realizadas durante o desenvolvimento.

# Parte I

# Parte Objetiva

# 1 Código Fonte e Propriedade Intelectual

Este capítulo visa esclarecer questões relacionadas à propriedade intelectual dos elementos que compõem o projeto.

## 1.1 Estrutura do Sistema

Toda a estrutura do projeto, desde o sistema de gerenciamento dos conteúdos (linguagem, objetos programáveis e ambientes) até o sistema de criação e execução de código fonte, está publicada com código aberto no endereço https://github.com/leeohaddad/CodeWay/, sob os termos da MPL 2.0 (Mozilla Public License Version 2.0), que está disponível entre os arquivos do código fonte.

## 1.2 Elementos Gráficos

Como o foco de desenvolvimento do projeto era nas funcionalidades e não na elaboração de elementos gráficos, muitos dos recursos (imagens e modelos tridimensionais) utilizados no projeto são conteúdos produzidos por terceiros e disponibilizados gratuitamente em bancos de imagens ou na própria Unity Asset Store, loja virtual da Unity3D destinada ao compartilhamento de conteúdos, pagos ou gratuitos.

Em cada diretório do projeto onde há conteúdo não produzido pelo autor deste trabalho, existe um arquivo de texto que contém as informações sobre propriedade (autor e link de obtenção) de todos os conteúdos do diretório que não foram produzidos neste projeto.

# 2 O Motor: Unity3D

A Unity3D, engine de jogos usada para o desenvolvimento do projeto, possui foco em portabilidade, permitindo a entrega em diversas plataformas. A engine se encontra atualmente na versão 5.2.3, que possui suporte para compilação de aplicativos para Windows, OS X, Linux, Unity Webplayer, Android, Nintendo 3DS, iOS, BlackBerry 10, Windows Phone 8, Tizen, WebGL, PlayStation 3, PlayStation 4, PlayStation Vita, Wii U, Xbox 360, Xbox One, TV Android , Samsung Smart TV, Oculus Rift e Gear VR, totalizando 21 plataformas disponíveis para realização de builds dos projetos.

A seguir, estão descritos os elementos computacionais definidos pela estrutura da Unity3D que são mais relevantes para o entendimento da estrutura do projeto CodeWay:

## 2.1 Scenes

Cenas (scenes) são ambientes isolados entre si que guardam os objetos e componentes de cada parte independente do software, de forma que cada cena possui seus próprios objetos e scripts associados.

No desenvolvimento de jogos, por exemplo, as diferentes cenas do projeto costumam corresponder às diferentes fases do jogo e às telas indepentes, como a tela inicial e a tela de configurações.

Naturalmente, isso não é uma regra imposta. É possível desenvolver todas as telas e fases de um aplicativo ou jogo utilizando uma única cena.

## 2.2 GameObjects

GameObjects são os objetos que constituem as cenas. Cada GameObject possui atributos de espaço (posição, rotação e escala) e uma lista de Componentes, que permitem configurar o comportamento e a função do objeto.

O GameObject é a estrutura fundamental de objetos da Unity3D, podendo se apresentar nas mais diversas formas, desde uma câmera ou um elemento de UI (textos e botões, por exemplo) até objetos tridimensionais complexos.

## 2.3 Components

Componentes (components) são scripts MonoBehaviour (escritos em C#, Javascript ou Boo) que podem ser anexados a diferentes GameObjects, criando uma gama de possibilidades de configurações e comportamentos dentro de um mesmo objeto.

Ao adicionar um componente que possui variáveis públicas a um GameObject, é possível associar valores e referências iniciais para essas variáveis através do inspetor de objetos. Essas variáveis públicas configuradas previamente, assim como as associações feitas em tempo de execução, são responsáveis por estabelecer as relações entre os diferentes GameObjects da cena e seus componentes.

Nem todos os componentes usados em um projeto são scripts configurados pelo desenvolvedor. Muitos componentes já estão previamente definidos pela Unity para atender a demandas específicas. Alguns exemplos são o componente Câmera, os componentes de iluminação (como a Luz Direcional e a Luz Pontual, entre outras), os componentes de detecção de colisão e os componentes de controle de animação.

## 2.4 Inspector

O inspetor de objetos (inspector) é uma janela que mostra os componentes e propriedades do objeto selecionado na cena, permitindo que essas propriedades sejam observadas e alteradas através dos valores das variáveis públicas desses componentes. Além disso, a adição e a remoção dos componentes também é feita através do inspetor de objetos.

## 2.5 Animations e Animator Controllers

A Unity, como já foi mencionado, disponibiliza alguns componentes previamente configurados para serem usados nos projetos. Alguns desses componentes são voltados para o controle de animações.

Os principais componentes de animação utilizados neste projeto foram o Animation, sistema de animações legado que permite e execução de animações através de um mecanismo simples, direto e intuitivo, e o Animator Controller, sistema de animações mais robusto que permite a criação de máquinas de estados para definir as animações a serem executadas, além de permitir alguns outros controles sobre a execução dessas animações.

# 3 Entidades Computacionais do Aplicativo

Para congifurar corretamente um simulador de comportamentos rubusto e eficiente, foi necessário definir e padronizar alguns elementos computacionais que foram usados no projeto e que estão listados abaixo:

## 3.1 Ambiente de Execução

Os ambientes de execução são modelos tridimensionais que desempenham a função de cenário para a execução de comportamentos, ajudando na contextualização do simulador. Além do cenário, o ambiente de execução é composto por toda a lógica de simulação do sistema.

Atualmente, o único ambiente configurado no projeto é a floresta (ilustrada na Figura 1), uma vez que o foco do desenvolvimento desta primeira etapa foi a definição e consolidação das estruturas lógicas utilizadas para definir os comportamentos, e não a criação de conteúdos em massa.

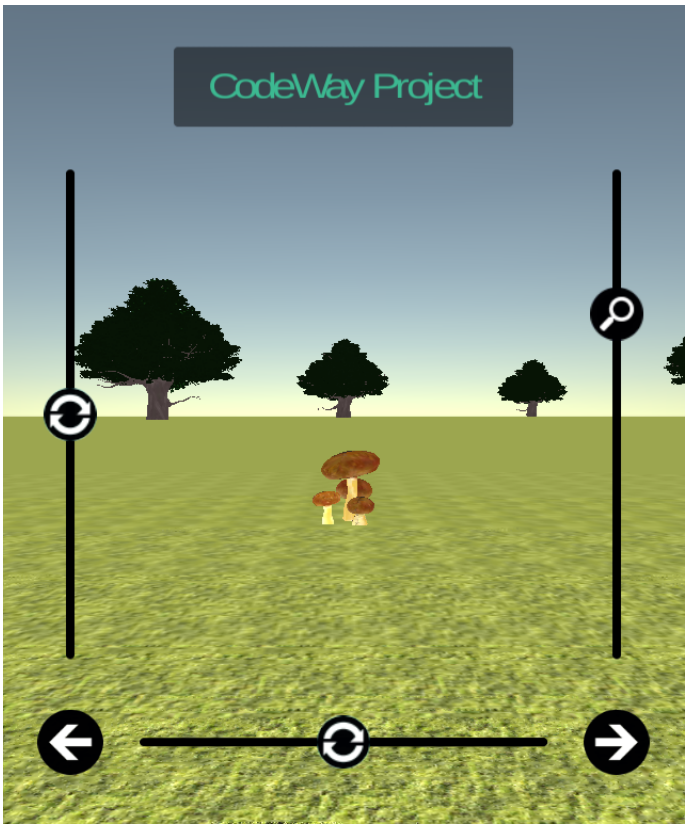


Figura 1: Ambiente de Execução: Floresta

## 3.2 Objeto Programável

Os objetos programáveis são elementos tridimensionais (compostos por malhas, materiais, texturas e animações) capazes de se associar a códigos fonte para simular os comportamentos descritos nestes códigos.

O Dragão Bruce, ilustrado na Figura 2, é um dos objetos programáveis disponíveis na primeira versão do projeto.

Figura 2: Objeto Programável: Dragão Bruce

## 3.3 Biblioteca

Bibliotecas são conjuntos de comandos divididos por áreas de atuação. Atualmente, o sistema possui apenas duas bibliotecas, sendo uma de movimentação e a outra de batalha. No entando, a estrutura dos comandos permite que novos comportamentos e bibliotecas sejam adicionados ao projeto sem muito trabalho.

## 3.4 Comando

Comandos são ações independentes que podem ser executadas pelos objetos programáveis. Cada comando está associado a uma palavra (o comando textual) e a um bloco de ação, cuja textura ilustra a ação associada ao comando.

## 3.5 Bloco de Ação

Objeto tridimensional que representa um comando específico dentro do fluxograma do modo visual de edição de código fonte.

## 3.6 Código Fonte

Códigos fonte são trechos de texto que representam sequências de comandos. Os códigos fonte podem ser salvos ou restaurados pelo usuário a qualquer momento. Uma vez salvos, podem ser injetados em diferentes objetos programáveis para que seja possível simular seus comportamentos.

# 4 Funcionamento do Aplicativo

Este capítulo visa documentar as funcionalidades e a forma de utilização do aplicativo, permitindo que o leitor compreenda o funcionamento de cada módulo do sistema.

## 4.1 Divisão da Tela

A tela do programa, ilustrada na Figura 3, é dividida ao meio, de forma que a metade da esquerda renderiza um dos módulos de edição de código fonte (explicados na seção 2 deste capítulo) enquanto a metade de direita renderiza o ambiente de execução (explicado na seção 3 deste capítulo).

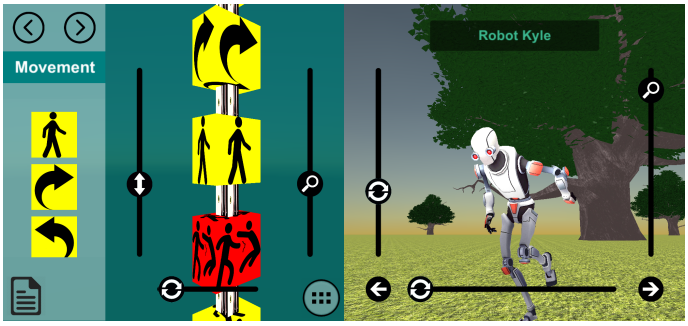


Figura 3: Tela do Aplicativo

## 4.2 Edição de Código Fonte

Os scripts contendo código fonte são o foco deste projeto, já que é através deles que os usuários terão contato com a lógica de programação. Os scripts podem ser construídos através de dois módulos diferentes, o Modo Visual e o Modo Textual, descritos nesta seção.

### 4.2.1 Modo Visual

O modo visual de edição de código fonte consiste em um grande fluxograma de blocos de ação. A ordem dos blocos de ação dentro deste fluxograma determina a ordem em que os comandos serão executados.

Nesta tela, ilustrada na Figura 4, existem alguns sliders (botões arrastáveis) que ajudam a manipular o fluxograma:

- O slider (botão arrastável) localizado à esquerda do fluxograma, com símbolo de setas duplas, permite que o usuário navegue ao longo do fluxograma, centralizando a tela do modo visual nos diferentes blocos de ação que compõem o fluxograma.
- O slider (botão arrastável) localizado embaixo do fluxograma, com símbolo de setas duplas rotativas, permite rotacionar o fluxograma ao longo de seu eixo vertical.
- O slider (botão arrastável) localizado à direita do fluxograma, com símbolo de lupa, permite aumentar o reduzir o zoom sobre o fluxograma, ou seja, aumentar ou reduzir a distância entre o fluxograma e a câmera que está captando sua imagem.

No modo visual, é possível adicionar novos blocos de ação arrastando-os para o fluxograma a partir de um menu lateral que contém uma lista de todos os comandos disponíveis no sistema, divididos por bibliotecas. Em contrapartida, é possível remover comandos através de um sistema de seleção de blocos de ação (que funciona através da detecção de cliques nos blocos de ação do fluxograma) e de um botão de remoção de blocos de ação, representado por uma lixeira (Figura 4).

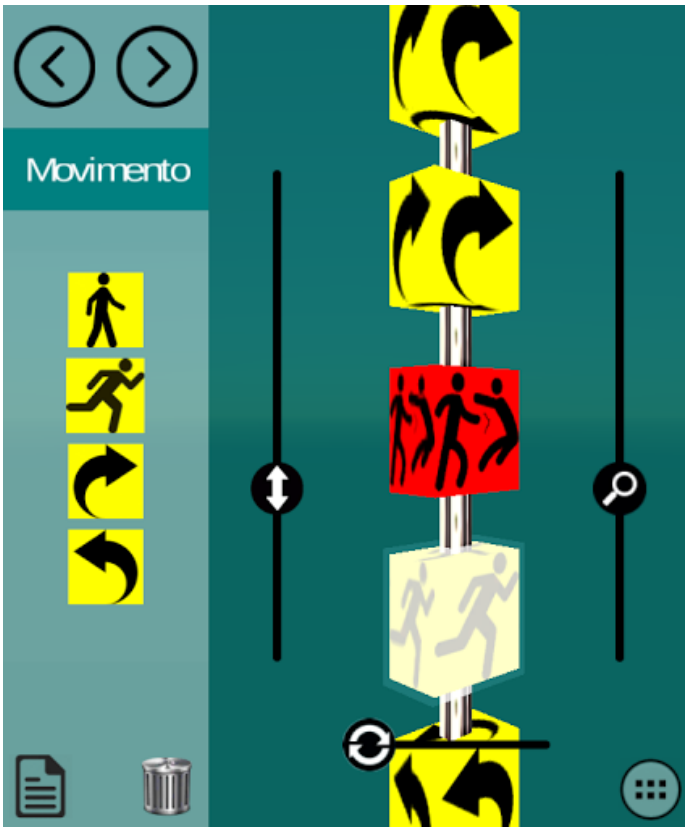


Figura 4: Modo Visual de Edição de Código Fonte

### 4.2.2 Modo Textual

O modo textual de edição de código fonte consiste em uma grande caixa de texto onde o usuário pode escrever seu código fonte livremente. Os comandos são escritos em 'lower camel case' (exemplo: turnRight) e separados por ponto e vírgula.

Para a primeira versão do software, não foram implementadas ferramentas de produtividade de edição de código, como 'code completion' (reduzir a necessidade de digitação através de sugestões de comandos), para priorizar a construção da base do sistema. No entanto, tais mudanças fazem parte da lista de implementações futuras do aplicativo, visto que a praticidade na digitação dos comandos tende a melhorar consideravelmente a experiência do usuário.

### 4.2.3 Transição e Compilação

É possível alternar livremente entre os dois modos de edição de código fonte através do botão localizado no canto inferior esquerdo da tela, e o sistema se encarregará de fazer a transição entre os dois modos, convertendo o fluxograma de blocos de ação para texto ou vice-versa.

Devido à liberdade dada ao usuário no modo de edição textual, o sistema compila o código para verificar se há erros antes de alternar para o modo visual. Caso haja erros (como um comando escrito da forma errada, por exemplo), o sistema mostra uma mensagem de erro na tela identificando o erro e impede que o modo de edição seja alterado. Na transição inversa, no entanto, essa verificação não é necessária, já que o modo de edição visual funciona de forma mais fechada, que garante que nenhum tipo de lixo seja inserido no fluxograma.

## 4.3 Ambiente de Execução

Como já foi mencionado, o ambiente de execução é responsável pelos processos envolvidos na simulação dos scripts. É através das funcionalidades associadas a ele que o usuário poderá instanciar objetos programáveis, atribuir comportamentos a essas instâncias (associação de scripts) e, finalmente, iniciar a simulação, executando todos os códigos fonte presentes nas intâncias de objetos programáveis contidas no ambiente. Para dar ao usuário um feedback visual apropriado dos comportamentos configurados nos scripts, o sistema utiliza as animações e implementações associadas aos comandos que compõem esses scripts. Esses elementos e funcionalidades estão descritos nos itens a seguir.

### 4.3.1 Controles de Ambiente

O ambiente de execução comporta os objetos tridimensionais, como é possível verificar na Figura 3. Sendo assim, a imagem a ser exibida neste módulo depende das propriedades de espaço (posição, rotação e escala) de um objeto de câmera, ou seja, um GameObject que possui o componente de câmera. Os controles de ambiente são um conjunto de botões que foram construídos através do sistema de UI da Unity e que permitem que o usuário controle a movimentação da câmera, para que ele possa assistir à simulação a partir de diversas perspectivas diferentes.

A câmera que capta a imagem do ambiente de simulação possui um sistema de movimentação, configurado através de scripts C#, que faz com que ela siga uma determinada instância de objeto programável. A instância a ser seguida é definida através de um sistema de seleção de objetos, controlado pelas setas laterais do menu de controles de ambiente.

Além de escolher o objeto a ser seguido pela câmera, o usuário também pode escolher a posição da câmera relativa ao objeto seguido. Ou seja, a câmera manterá sempre a mesma distância e o mesmo ângulo de inclinação em relação ao objeto, e esses valores podem ser configurados pelo usuário através de três sliders (botões arrastáveis) que ficam situados no módulo do Ambiente de Execução:

- O slider (botão arrastável) localizado no canto esquerdo do Ambiente de Execução, com símbolo de setas duplas rotativas, permite que o usuário rotacione a câmera ao longo do eixo horizontal.
- O slider (botão arrastável) localizado no canto inferior do Ambiente de Execução, com símbolo de setas duplas rotativas, permite que o usuário rotacione a câmera ao longo do eixo vertical.
- O slider (botão arrastável) localizado no canto direito do Ambiente de Execução, com símbolo de lupa, permite que o usuário controle o zoom, ou seja, controle a distância entre a câmera e o objeto selecionado.

### 4.3.2 Instanciação de Objetos Programáveis

Para que o usuário possa escolher os objetos programáveis que atuarão no ambiente de execução, foi criado um sistema de instanciação de objetos, que mostra todos os objetos programáveis disponíveis e permite que o usuário escolha um deles para ser instanciado na cena.

Não há limites para instanciação de objetos. Ou seja, o usuário pode instanciar quantos objetos ele quiser. Isso é útil nos casos em que o usuário deseje ver diferentes instâncias dos agentes tridimensionais executando scripts simultaneamente.

### 4.3.3 Associação de Comportamentos

Uma vez instanciados, os objetos podem receber scripts, que definirão seu comportamento no momento da simulação.

Para associar um comportamento a uma instância de objeto programável, o usuário precisa salvar o código contendo o comportamento, selecionar a instância de objeto programável desejada através dos controles de ambiente e, por fim, selecionar a opção de associação de comportamento no menu de opções do ambiente de execução.

### 4.3.4 Execução da Simulação

A execução da simulação ocorre de uma forma muito simples, em que as instâncias de objetos programáveis interpretam seus scripts associados e executam os comandos contidos neles. Para cada comando, o objeto programável executa a ação relacionada ao comando e toca a animação correspondente. Cada comando dura exatamente 1 segundo, o que facilita a identificação do comando que está sendo executado em cada momento da simulação.

## 4.4 Menu de Opções

O botão de menu, localizado no canto inferior direito da tela de edição de código (Figura 4), permite que o usuário acesse o menu de opções, que dá acesso a diversas funcionalidades do sistema, listadas nesta seção.

### 4.4.1 Código Fonte

Submenu responsável pelo gerenciamento da persistência dos scripts criados pelo usuário.

#### 4.4.1.1 Abrir Código

Permite selecionar um script existente e carregá-lo no ambiente de edição.

#### 4.4.1.2 Salvar Código

Permite criar ou sobrescrever um script, salvando dentro dele os comandos que estão atualmente configurados no ambiente de edição de código fonte.

### 4.4.2 Ambiente de Execução

Submenu que dá acesso às funcionalidades do ambiente de execução.

#### 4.4.2.1 Executar

Inicia a simulação, fazendo com que os objetos programáveis executem seus respectivos códigos fonte.

#### 4.4.2.2 Instanciar

Abre o menu de instanciação, que permite que o usuário escolha um objeto programável para instanciar no ambiente de execução.

#### 4.4.2.3 Definir Comportamento

Mostra para o usuário uma lista com os scripts salvos e permite que escolha um desses scripts para injetar no objeto selecionado.

#### 4.4.2.4 Reiniciar Objeto Selecionado

Transfere o objeto selecionado para a posição origem do ambiente de execução, reiniciando suas propriedades de espaço.

#### 4.4.2.5 Reiniciar Objetos

Transfere todos os objetos instanciados no ambiente de execução para a posição origem do ambiente, reiniciando suas propriedades de espaço.

### 4.4.3 Configurações

Submenu que dá acesso a alguns ajustes de configuração do aplicativo.

#### 4.4.3.1 Idiomas

Permite que o usuário altere o idioma do aplicativo, podendo escolher entre português, inglês e espanhol.

#### 4.4.3.2 Tamanho do Texto

Permite que o usuário altere o tamanho de alguns textos do aplicativo, podendo escolher entre pequeno, médio, grande e gigante.

### 4.4.4 Sair

Encerra o aplicativo.

# 5 Estrutura Interna do Projeto

Este capítulo visa documentar a arquitetura do projeto, possibilitando que o leitor compreenda os padrões de registro de conteúdos dentro da hierarquia, assim como o funcionamento interno dos scripts que compõem o sistema de simulação.

## 5.1 As Cenas do Projeto

A divisão de telas em formas de cenas é muito útil para a construção de jogos, onde são feitas poucas transições entre as telas e onde não há uma troca constante de informação entre essas telas.

Como o aplicativo desenvolvido tem um caráter mais dinâmico em relação à troca de telas, o sistema foi montado através de uma outra abordagem, gerenciando a visibilidade de GameObjects e Câmeras para alternar entre as diferentes telas. Sendo assim, o aplicativo possui apenas duas cenas.

Uma das cenas (SplashScreen.unity) funciona como tela de SplashScreen, uma tela que mostra o banner do software por alguns segundos antes de inicializar o aplicativo. Essa cena de SplashScreen é um recurso disponibilizado gratuitamente na internet e não foi desenvolvido pelo autor deste trabalho. Foi necessário usar esse recurso para construir uma SplashScreen porque a versão da Unity utilizada nesse projeto foi a versão gratuita (Unity for Indies), e o recurso de SplashScreen desenvolvido pela própria Unity Technologies está disponível apenas na versão paga.

A outra cena (CodeWay.unity) é a cena que, de fato, guarda toda a estrutura do projeto. É nessa cena que todas as funcionalidades serão acessadas, desde a criação de código até a instanciação de objetos e execução da simulação. Todos os outros elementos do projeto, que serão citados ao longo deste capítulo, estão configurados nessa cena.

## 5.2 O Sistema de Câmeras e de Divisão da Janela

Para organizar eficientemente a disposição das telas dentro do aplicativo, foi criado um sistema unificado de divisão da janela entre as diferentes câmeras, chamado de CameraSplitScreen e localizado na raiz da hierarquia da cena principal do projeto.

Como já foi mencionado nesse trabalho, as câmeras desempenham um papel fundamental na visualização dos objetos, captando a imagem de uma determinada porção do mundo virtual tridimensional, a partir de uma determinada perspectiva.

O sistema de câmeras desenvolvido para este projeto permite configurar diferentes câmeras para ocuparem diferentes porções da janela do software (ou da tela do dispositivo, para o caso de dispositivos mobiles ou para outros casos de execução em tela cheia). O sistema de câmeras permite que o desenvolvedor configure facilmente esses parâmetros, registrando as câmeras (ou seja, diferentes visualizações) e atribuindo pesos que definem a importância dessas câmeras, ou seja, a proporção da tela que será ocupada pelas imagens captadas por cada uma elas. A versão atual do aplicativo, por exemplo, está configurada em uma visão 1-1, na qual metade da tela é ocupada pelo sistema de edição de código enquanto a outra metade é ocupada pelo ambiente de execução, já que ambas as telas possuem o mesmo peso (peso 1).

Além disso, cada câmera pode ter um sistema de controle de movimentação independente, para ajudar o usuário a definir a porção do mundo tridimensional que deseja ver, tal como a perspectiva dessa visualização. Esse sistema de controle específico é formado por uma série de scripts (que se encontram na pasta Camera Control, mesma pasta onde está o CameraSplitScreen) e é utilizado, por exemplo, pelos botões arrastáveis que configuram a rotação e o zoom das câmeras do ambiente de execução e do modo visual de edição de código fonte.

## 5.3 As Bibliotecas de Comandos da Linguagem

Dentro da hierarquia da cena principal do projeto, existe um GameObject responsável por gerenciar os comandos e bibliotecas existentes na linguagem que é usada pelo aplicativo.

Este objeto, cujo nome é EvoLang, tem uma estrutura extremamente simples, na qual cada biblioteca da linguagem é representada por um GameObject que, além de ser filho do GameObject EvoLang dentro da hierarquia, possui um componente (script) EvoLangLib.

Analogamente, cada comando de uma biblioteca é representado por um GameObject que, além de ser filho do GameObject relativo a essa biblioteca dentro da hierarquia, possui um componente (script) EvoLangCmd.

O componente EvoLangCmd, por sua vez, possui um parâmetro de texto que define o texto associado ao comando (para o modo textual de edição de código fonte) e a aparência (material, textura e cor) do bloco de ação associado ao comando (para o modo visual de edição de código fonte).

Para adicionar uma nova biblioteca ou um novo comando para a linguagem, basta adicionar novos GameObjects nesta hierarquia, preenchendo corretamente os parâmetros e seguindo o padrão definido nos comandos já configurados.

## 5.4 O Ambiente de Execução

Dentro da hierarquia da cena principal do projeto, existe um GameObject destinado ao ambiente de execução, o PlaySpace. O GameObject PlaySpace possui o componente (script) PlaySpace, que gerencia todas as funcionalidades relacionadas ao ambiente de execução. O componente PlaySpace possui uma lista pública onde são registrados todos os objetos programáveis disponíveis no aplicativo. Esses objetos programáveis, assim como o objeto tridimensional que representa o ambiente em si, encontram-se dentro da hierarquia do GameObject PlaySpace.

### 5.4.1 Os Objetos Programáveis

Um objeto programável é um GameObject que possui o componente (script) ProgrammableObject, além dos elementos listados a seguir:

- Alguma representação tridimencional (malha) dentro de sua hierarquia.
- Uma variável com referência para o componente PlaySpace do ambiente de execução.
- Uma variável opcional contendo posição central da visão da câmera sobre o objeto.
- A lista de referências para implementações das bibliotecas suportadas pelo objeto.

Para adicionar um novo objeto programável ao sistema, basta criar um novo GameObject dentro da hierarquia do ambiente de execução, adicionar o componente ProgrammableObject, congifurá-lo para que atenda todos os requisitos listados acima e registrá-lo na lista de objetos programáveis do componente PlaySpace.

#### 5.4.1.1 As Implementações das Bibliotecas

As implementações das bibliotecas são GameObjects com componentes que estendem a classe EvoLangImplementation. Esses GameObjects, por questões de organização, ficam localizados dentro da hierarquia do objeto programável em questão, sendo filhos do GameObject LibrariesSupported que, por sua vez, é filho do objeto programável em questão dentro da hierarquia.

Os scripts que estendem a classe EvoLangImplementation são scripts que possuem uma função assíncrona configurada para cada comando de uma determinada biblioteca, definindo a consequência efetiva causada no objeto por ação desse comando (exemplo: comando 'walk' movimenta o objeto para frente durante 1 segundo, enquanto o comando

'turnRight' rotaciona o objeto ao longo de seu eixo vertical, completando uma rotação de 90º dentro do intervalo de 1 segundo).

Além disso, as implementações das bibliotecas também possibilitam que o programador configure animações para serem executadas junto com os comandos (exemplo: comando 'walk' aciona a animação que movimenta as pernas dos objetos que possuem pernas), de forma a aumentar o realismo da simulação. Para cada biblioteca implementada (ou seja, para cada GameObject com o componente EvoLangImplementation), é possível associar um componente Animation e um componente Animator Controller. Sempre que um comando é acionado, o objeto programável irá executar as funções dos arquivo de implementação que possuírem o mesmo nome que o comando, além de tocar os clipes de animação que também possuírem o mesmo nome que o comando, estejam eles dentro do componente Animation (sistema legado de animações) ou dentro do componente Animator Controller (sistema completo de animação com uso de máquinas de estado).

### 5.4.2 O Gerenciador de Instâncias

O gerenciador de instâncias é o sistema que possibilita que o usuário defina quais objetos programáveis estarão inseridos no ambiente de execução. O gerenciador de instâncias é um GameObject que possui o componente (script) InstancesManager e que está localizado na raiz da hierarquia da cena principal do projeto.

O componente do gerenciador de instâncias possui uma referência ao componente PlaySpace, para que as instâncias possam ser devidamente registradas no ambiente de execução. Além disso, o gerenciador de instâncias também possui uma referênia para um local no espaço tridimensional onde ele pode armazenar os modelos de objetos disponíveis, além de usar esse espaço para mostrar os objetos para o usuário no momento da escolha do objeto a ser instanciado.

Quando o usuário escolhe o objeto a ser instanciado, o gerenciador cria uma cópia desse objeto e utiliza a referência ao PlaySpace para levar a nova cópia para o ambiente de execução, mudando sua posição no mundo tridimensional e registrando-a na lista (privada) de instâncias do componente PlaySpace.

## 5.5 O Sistema de Edição de Código

O GameObject CodeCanvas, localizado na raiz da hierarquia da cena principal do projeto, aglomera os principais painéis de elementos UI do projeto. Dentro de sua hierarquia, estão localizados o painel do modo visual de edição de código fonte, o painel do modo textual de edição de código fonte, o botão e o painel associados ao menu contextual

do aplicativo, os painéis associados a persistência de código e o painel utilizado para o sistema de mensagens, descrito em outra seção deste capítulo.

O foco desta seção é descrever o funcionamento dos dois primeiros painéis, que cuidam, respectivamente, do modo visual e do modo textual de edição de código fonte. A alternância dos modos de edição de código fonte é gerenciada por um sistema de abas que foi desenvolvido para este projeto, no qual os botões que realizam a troca de abas são os botões localizados no canto inferior esquerdo da tela de edição de código fonte.

### 5.5.1 Modo Visual

O modo visual possui uma estrutura robusta que permite que o usuário crie diversos scripts diferentes sem digitar uma única palavra sequer, utilizando apenas o mouse para arrastar os comportamentos desejados e para navegar ao longo do fluxograma de blocos de ação. Essa estrutura se baseia em dois GameObjects principais, AvailableBlocks e SourceBlocks, de forma que a estrutura do primeiro GameObject guarda os comandos disponíveis para o usuário, enquanto o segundo GameObject guarda os blocos que compõem o código fonte atualmente descrito pelo fluxograma.

#### 5.5.1.1 Blocos de Ação Disponíveis

O GameObject AvailableBlocks, que possui o componente (script) AvailableBlocks, é responsável por disponibilizar ao usuário os comandos registrados no objeto EvoLang, descrito anteriormente neste capítulo.

Os blocos de ação disponíveis são mostrados no menu lateral esquerdo do modo visual de edição de código fonte, sendo divididos por bibliotecas. O sistema mostra o nome da biblioteca sendo mostrada atualmente, além de permitir que o usuário escolha outra biblioteca através das setas presentes em sua UI.

Entre as variáveis públicas deste componente, há uma referência para o texto que mostra o nome da biblioteca sendo mostrada atualmente, além de uma referência para um modelo de bloco de ação que representará graficamente os comandos disponíveis. Por fim, o componente também possui uma lista pública de referências para as bibliotecas disponíveis no projeto, localizadas dentro da hierarquia do objeto EvoLang.

#### 5.5.1.2 Blocos do Fluxograma

O GameObject SourceBlocks, que possui o componente (script) SourceBlocks, é responsável por armazenar e organizar os blocos de ação contidos no fluxograma do código fonte.

O componente possui diversas referências que devem ser configuradas através de suas variáveis públicas. Tais referências estão listadas abaixo:

- O componente SourceCode, presente no campo de texto do modo textual.
- O slider (botão arrastável) responsável pela navegação ao longo do fluxograma.
- O modelo de bloco de ação que representará os comandos dentro do fluxograma.
- O modelo de conexão, que será usado para interligar os comandos do fluxograma.
- O objeto de seleção, que será usado para destacar o objeto selecionado.
- Uma lista de referências para as bibliotecas disponíveis no projeto.

O modelo de bloco de ação obrigatoriamente deve possuir o componente EvoLangCmdCube, que estará consequentemente presente em todos os blocos de ação do fluxograma. A adição de blocos de ação ao fluxograma é gerenciada pelo script InsertBlock e pelo próprio EvoLangCmdCube, que possuem componentes que detectam quando um bloco de ação é arrastado até o fluxograma. A remoção de blocos, por outro lado, é gerenciada pelo componente SourceBlocks através do sistema de seleção de blocos que, por sua vez, funciona através dos cliques do usuário sobre os blocos de ação do fluxograma.

### 5.5.2 Modo Textual

O modo textual é muito mais simples do que o modo visual em termos de estrutura. Possui um objeto de caixa de texto que ocupa a tela de edição inteira, além de um componente SourceCode, que identifica que aquele texto, na verdade, representa um código fonte.

## 5.6 O Menu Contextual

Para tornar a edição do menu contextual uma tarefa simples, foi criado um sistema robusto de menus que gera automaticamente os botões e suas relações, sem demandar trabalho repetitivo por parte do programador.

Esse sistema de menu é gerenciado pelo script ContextMenuOnClick que, ao ser associado, na forma de componente, a um GameObject clicável, faz com que esse GameObject se torne o botão que abre e fecha o menu. Após adicionar o componente ao GameObject, basta preencher uma única vez os atributos que definirão a aparência do botão, além de preencher uma lista de funcionalidades que definirá os botões a serem criados pelo menu contextual.

Para cada elemento configurado na lista de botões do menu contextual, o programador deve definir o nome do botão, a(s) funcionalidade(s) disparada(s) por ele e, finalmente, o botão para o qual ele está associado, caso faça parte de um submenu (exemplo: Inglês, Português e Espanhol formam um submenu associado ao botão Idiomas).

Uma vez configurados os botões do menu contextual e a aparência do botão modelo, basta executar o projeto e o sistema cuidará de criar os botões, criar as relações de submenu entre eles e associar suas respectivas funcionalidades.

## 5.7 O Sistema de Persistência de Código Fonte

O sistema de persistência utiliza o PersistentDataPath (um local no disco, definido para cada plataforma da Unity, onde o aplicativo pode guardar arquivos auxiliares que serão mantidos mesmo após o fim da execução) para fazer a persistência de código fonte, permitindo que o usuário salve um arquivo diferente para cada script criado.

## 5.8 O Sistema de Mensagens

Para que fosse possível mostrar aos usuários mensagens informativas (como a mensagem que avisa que o código fonte foi salvo corretamente) e mensagens de erro (como a mensagem que avisa que existem erros no código fonte), foi criado um sistema de mensagens simples e intuitivo. O sistema de mensagens é gerenciado pelo script DebugMessage e permite que o programador, através das 3 funções estáticas SuccessMessage, WarningMessage e ErrorMessage, configure diversas mensagens diferentes, mantendo o usuário sempre informado sobre os processos que estiverem ocorrendo no aplicativo.

## 5.9 O Sistema de Tradução

O sistema de tradução utilizado no projeto também foi obtido gratuitamente, cedido gentilmente pela empresa EvoBooks para o desenvolvimento deste projeto. Algumas alterações, no entanto, foram feitas neste sistema para adequar seu funcionamento ao aplicativo.

Basicamente, o sistema de tradução funciona através de duas frentes. Em uma frente temos os arquivos de tradução e, na outra, temos os scripts responsáveis pelo funcionamento do sistema.

Os arquivos de tradução são arquivos XML cuja estrutura permite definir palavras-chave (tokens) que identificam as mensagens, e as traduções para essa mensagens. Isso

é feito através de uma estrutura de tags XML (ilustrada na Figura 5) que guarda as informações relativas à tradução de mensagens.

```
<messages>

  <message id="SourceCode">
    <language type="PT">Código Fonte</language>
    <language type="EN">Source Code</language>
    <language type="ES">Código Fuente</language>
  </message>

  <message id="LoadCode">
    <language type="PT">Abrir Código</language>
    <language type="EN">Load Code</language>
    <language type="ES">Cargar el Código</language>
  </message>

  <message id="SaveCode">
    <language type="PT">Salvar Código</language>
    <language type="EN">Save Code</language>
    <language type="ES">Guardar el Código</language>
  </message>

</messages>
```

Figura 5: Exemplo de Arquivo de Tradução

Para o gerenciamento das traduções, são utilizados 3 scripts. O script principal do sistema é o TranslationManager, que não precisa ser instanciado na cena, sendo apenas chamado de forma estática por outros scripts para realizar a ponte entre o aplicativo e os arquivo de tradução. O TranslationManager possui uma função estática GetMessage, que recebe o nome de um arquivo de tradução e uma palavra-chave (token) e devolve o texto traduzido para o idioma que estiver definido no sistema. Os outros scripts utilizados são o TranslationToken, script que deve ser inserido, na forma de componente, em objetos de texto para que os mesmos se tornem tradutíveis, e o ContextMenuTranslationTokenizer, que permite que os botões criados automaticamente pelo menu contextual também sejam tradutíveis.

# 6 Conclusão e Considerações Finais

Os resultados obtidos no final do desenvolvimento foram muito positivos e satisfatórios. A estrutura do sistema, como planejado, está robusta e expansível, apresemtando um formato bem definido para a criação e inclusão de cada tipo de novo conteúdo (expansão da linguagem utilizada no projeto, inclusão de mais objetos programáveis e diversificação de ambientes).

Durante o período de idealização do projeto, no entanto, foi necessário fazer uma redução considerável de escopo em relação à ideia inicial, deixando de lado as funcionalidades de alto nível para focar apenas na consolidação da base do sistema. Isso ajudou a garantir a estrutura robusta, já que, sem essa redução de escopo, não haveria tempo hábil para desenvolver todas as funcionalidades idealizadas, possivelmente comprometendo a entregabilidade e a expansibilidade do sistema.

# Parte II

# Parte Subjetiva

# 7 Desafios e Frustrações

A maior frustração do projeto foi um erro relativamente comum entre programadores, a superestimação do escopo viável. No momento de sua idealização, o projeto tinha uma dimensão muito maior do que no momento em que a proposta efetiva foi entregue. Ao conversar com meu orientador, durante as reuniões, fui alertado de que seria inviável realizar um projeto tão complexo em tão pouco tempo. Tal aviso provou-se legítimo nos últimos meses do desenvolvimento, pois até mesmo o projeto de escopo reduzido mostrou-se muito trabalhoso, deixando claro que a consolidação do projeto idealizado inicialmente dentro do tempo hábil do TCC seria realmente um desafio gigante, praticamente inviável.

# 8 Disciplinas cursadas mais relevantes para o desenvolvimento do projeto

## 8.1 MAC0110: Introdução à Computação

Primeira disciplina de computação cursada, permitiu uma visão geral sobre conceitos de programação, servindo de base para todas as outras disciplinas de computação vistas durante o curso.

## 8.2 MAC0122: Princípios de Desenvolvimento de Algoritmos

Tópicos mais específicos de programação que deram base para o desenvolvimento de algoritmos mais complexos, expandindo a gama de funcionalidades possíveis no desenvolvimento dos comportamentos.

## 8.3 MAC0323: Estrutura de Dados

Estudo de estruturas que trouxeram uma visão mais profunda sobre organização de dados dentro de um software, permitindo a criação de um ambiente robusto e bem definido, onde a expansão de objetos e comportamentos se dá de forma prática e intuitiva.

## 8.4 MAC0211/MAC0242: Laboratório de Programação I e II

Disciplinas que introduziram um olhar mais prático sobre desenvolvimento de software, desviando um pouco o foco (até então puramente teórico) do aprendizado para dar lugar a uma experiência prática sobre desenvolvimento de projetos.

## 8.5 MAC0332: Engenharia de Software

Aprofundamento dos conhecimentos teóricos relacionados a estruturação e gerenciamento de projetos obtidos nas disciplinas de Laboratório de Programação, elevando tais conhecimentos a um nível superior de organização e de dimensão de projetos.

## 8.6 MAC0318: Introdução à Programação de Robôs Móveis

Disciplina que introduziu o paradigma de associação dinâmica de comportamentos configuráveis em objetos programáveis através do uso de comandos conhecidos por tais objetos. Base do sistema utilizado nos objetos programáveis.

## 8.7 MAC0342: Laboratório de Programação Extrema

Aprofundamento da experiência prática de programação e desenvolvimento de projetos obtida nas disciplinas de Laboratório de Programação, elevando essa experiência a um nível superior de dimensão e duração de projetos.

## 8.8 MAC0420: Computação Gráfica

Muito útil para a compreensão dos vários conceitos de CG, que são amplamente utilizados pela ferramenta Unity3D e por quaisquer outras ferramentas de programação de elementos gráficos, tridimensionais ou não.

# 9 Possibilidades Futuras

O projeto foi estruturado de forma a favorecer e estimular sua continuidade. Sua estrutura, robusta e expansível, garante que não seja necessário gastar muito tempo para criar novos comandos ou configurar novos objetos programáveis.

Além disso, existem muitas possibilidades de melhorias e expansões para as funcionalidades do projeto. Entre as possibilidades futuras mais relevantes, podemos citar:

- Expansão de conteúdo: expansão da linguagem e das listas de objetos programáveis e de ambientes disponíveis.
- Criação de contextualizações e desafios solucionáveis através de código-fonte.
- Ferramentas de produtividade no modo textual (exemplo: code completion).
- Melhorias de UX e de interface (exemplo: na inserção de comandos no modo visual.
- Introdução de linguagens de programação reais.
- Apresentação de problemas e algoritmos famosos da área de computação.
- Criação de trilhas pedagógicas que utilizem contextualizações para guiar o processo de aprendizagem de programação.

# Referências

DREDGE, S. *Coding at school: a parent's guide to England's new computing curriculum*. 2014. Notícia do Jornal The Guardian. Disponível em: <http://www.theguardian.com-/technology/2014/sep/04/coding-school-computing-children-programming>. Acesso em: 25.11.2015. Citado na página 8.

M.I.T. *MIT App Inventor*. 2015. Site do App Inventor. Disponível em: <http://ai2-.appinventor.mit.edu/>. Acesso em: 16.10.2015. Citado na página 8.

SALDAñA, P. *Uso de aplicativos para celular ganha força na escola*. 2015. Notícia do Jornal Estadão. Disponível em: <http://educacao.estadao.com.br/noticias/geral,uso-de-aplicativos-para-celular-ganha-forca-na-escola,1749345>. Acesso em: 25.11.2015. Citado na página 8.