UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma Ferramenta para Definição de Níveis para Jogos 2D por meio de Curvas de Terreno no Unity

---

Autor: Leandro Henrique Freitas de Aguiar (lhfa@cin.ufpe.br)
Orientador: Geber Lisboa Ramalho (glr@cin.ufpe.br)
Co-orientador: Marco Túlio Caraciolo Ferreira Albuquerque (tulio@manifestogames.com.br)

Recife – Pernambuco
Julho de 2017

Leandro Henrique Freitas de Aguiar

Uma Ferramenta para Definição de Níveis para Jogos 2D por meio de Curvas de Terreno no Unity

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Pernambuco – UFPE, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal de Pernambuco
Centro de Informática

Recife – Pernambuco
Julho de 2017

Leandro Henrique Freitas de Aguiar

Uma Ferramenta para Definição de Níveis para Jogos 2D por meio de Curvas de Terreno no Unity

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Pernambuco – UFPE, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Orientador: _____

Dr. Geber Lisboa Ramalho (UFPE – CIn)

Avaliador: _____

Dr. Giordano Ribeiro Eulalio Cabral (UFPE – CIn)

Recife – Pernambuco
Julho de 2017

“Vai Dar Tudo Certo!”
— Velho Sábio

Agradecimentos

Gostaria de agradecer a minha família que me aguentou nessa jornada até agora. Mãe, Pai e Laiza, Obrigado!

Gostaria também de agradecer aos meus amigos do colégio que nunca deixaram de acreditar em mim. Luiz, Pedro e Bruno, Obrigado!

Não posso esquecer de agradecer a todos aqueles que não fazem parte das duas patotas de cima mas que de alguma forma me ajudaram a chegar até aqui. Lívia, Juliana, Carlos, Geber, Túlio, Vicente, Leonardo, Ícaro, Átila, Amanda, Paula, Marconi, Pedro, Vinícius, Délio, Eric, Obrigado!

Um pouco de jabá nunca faz mal, então agradeço ao Google, por que sem ele esse trabalho não existiria.

E finalmente, agradeço a mim mesmo, por nunca desistir...

Resumo

Uma Ferramenta para Definição de Níveis para Jogos 2D por meio de Curvas de Terreno no Unity

Durante o desenvolvimento de jogos eletrônicos a utilização de ferramentas de game design facilita o trabalho do game designer e aumenta a sua produtividade. Entretanto, há situações em que as ferramentas disponíveis não são adequadas para as necessidades de um jogo ou não tem integração com o ambiente de desenvolvimento do jogo. O objetivo deste trabalho foi desenvolver uma ferramenta de criação de níveis para jogos 2D que seja integrada ao motor de jogos Unity e que permite ao game designer definir os estágio por meio de curvas de terreno. Para validar a sua eficácia a ferramenta foi integrada a um jogo em desenvolvimento da Manifesto Games e as opiniões do game designer coletadas por meio de um entrevista. Ao final deste processo a ferramenta conseguiu cumprir todos os objetivos propostos.

Palavras-chave: Game Design, Editor de Nível, Curvas 2D, Unity

Abstract

A Tool for Level Definition for 2D Game by means of Terrain Curves on Unity

During the development of an electronic game the usage of game design tools eases the work of the game designer and increases his productivity. However, there are situations when the tools available are not adequate for the needs of a game or are not integrated with the development environment of the game. The objective of the work was to develop a level creation tool for 2D games that is integrated into the Unity game engine and that allows the game designer to define the stages by means of terrain curves. To validate its effectiveness the tool was integrated into a game being developed by Manifesto Games and the opinions of the game designer were collected by means of an interview. At the end of this process the tool was able to fulfill all proposed objectives.

Keywords: Game Design, Level Editor, Curvas 2D, Unity

Lista de Figuras

Figura 2.1 – Flow, Relação entre Habilidade e Dificuldade 18

Figura 4.1 – A interface principal do Unity Editor 25

Figura 5.1 – Uma cena comum ao jogar o Fiat Fantasy Ride 30

Figura 5.2 – Gino 31

Figura 5.3 – Mágico 31

Figura 5.4 – Cientista 31

Figura 5.5 – Macaco 31

Figura 5.6 – Vovó 31

Figura 5.7 – Biscoito 31

Figura 5.8 – Torcedor 32

Figura 5.9 – Sereia 32

Figura 5.10 – Arraia 32

Figura 5.11 – Caubói 32

Figura 5.12 – Balão 32

Figura 5.13 – Fantasma 32

Figura 5.14 – O primeiro mundo do Fiat Fantasy Ride 34

Figura 5.15 – O segundo mundo do Fiat Fantasy Ride 34

Figura 5.16 – O terceiro mundo do Fiat Fantasy Ride 35

Sumário

Agradecimentos **6**

Resumo **8**

Abstract **10**

Lista de Figuras **11**

Sumário **12**

1 – Introdução **14**

1.1 – Motivação 14

1.2 – Objetivo 15

1.3 – Abordagem 15

1.4 – Estrutura do Documento 15

2 – Problemas do Design de Níveis em Jogos **17**

2.1 – Satisfação do Jogador 17

2.1.1 – Estado de flow 17

2.2 – Ferramentas de Game Design 18

3 – Estado da Arte **20**

3.1 – O Tanagra 20

3.2 – O Tiled 20

3.3 – O Bonsai 21

4 – Solução Proposta **23**

4.1 – O Unity 23

4.1.1 – Modelagem dos Jogos 23

4.1.2 – O Unity Editor 24

4.2 – O Projeto Desenvolvido 26

4.2.1 – Editor de Mundo 27

4.2.2 – Editor de Nível 27

4.2.3 – Editor de Terreno 27

5 – Estudo de Caso: O Fiat Fantasy Ride **29**

5.1 – Elementos do Jogo 29

5.2 – Os Mundos 33

6 – Validação **36**

6.1 – Integração ao Fiat Fantasy Ride 36

6.2 – Opinião do Game Designer 36

6.3 – Resultado 36

7 – Conclusão **38**

7.1 – Contribuições 38

7.2 – Limitações 38

7.3 – Trabalhos Futuros 38

Referências **40**

1 – Introdução

A conclusão do curso de Ciência da Computação encerra uma formação teórica e prática em técnicas de computação e a habilitação do acadêmico à expansão desse conhecimento científico, fornecendo fundamentos para solucionar os mais diversos problemas na área de programação e desenvolvimento de softwares.

Neste contexto, a criação de jogos eletrônicos envolve problemas de várias áreas de conhecimento e a interação com profissionais de diferentes áreas de habilidades, como desenhistas, animadores, designers, etc.

Devido à crescente competição no mercado de jogos e a pressão para elevar os níveis de produtividade no setor, torna-se necessário o desenvolvimento de técnicas mais avançadas e eficientes que assegurem a produção de conteúdo de qualidade nos prazos estabelecidos, seja para realizar o lançamento ou entregar o produto final ao cliente (RABIN, 2010).

1.1 – Motivação

No desenvolvimento de um jogo eletrônico, o game designer tem a função de gerar no jogo as condições ideais que garantam a plena satisfação do jogador ao jogá-lo. Para realizar determinada tarefa, é essencial criar conteúdo atraente, por meio da utilização de ferramentas de auxílio bem projetadas que permitam ao game designer visualizar e simultaneamente fazer modificações no mundo que ele está criando (ROUSE, 2000).

Entretanto, há situações em que as ferramentas disponíveis não se adequam às necessidades de um jogo ou então elas simplesmente não tem integração com o ambiente de desenvolvimento escolhido para criar o jogo. Nestes casos, o game designer acaba recorrendo a técnicas menos eficientes apenas pelo fato de já funcionarem no ambiente escolhido.

Dentre os casos de ausência de ferramentas apropriadas, observou-se que embora existam vários editores focados em criar níveis para jogos 2D, não existem editores de níveis que permitam definir o estágio utilizando curvas de terreno a fim de se obter uma sensação maior de fluidez no nível. Além disso, poucas ferramentas oferecem integração com o motor de jogos *Unity* (UNITY3D, [s.d.]).

1.2 – Objetivo

Neste contexto, o objetivo deste trabalho é criar uma nova ferramenta de criação de níveis para jogos 2D que permite ao game designer realizar a definição do estágio por meio de curvas de terreno. As curvas de terreno deverão ser configuráveis de maneira que o resultado final parece natural aos olhos do jogador, aumentando a sua satisfação ao jogar. E por fim esta ferramenta precisa funcionar em conjunto com o motor de jogos Unity para poder agilizar o processo de criação.

1.3 – Abordagem

Através da API de extensão da plataforma *Unity* foram criados assets personalizados que guardam as configurações dos níveis e de suas respectivas curvas de terreno. Estes assets são manipulados por meio de interfaces específicas criadas para permitir que o game designer possa visualizar, editar e testar os níveis criados de maneira fácil.

Após a sua implementação, a ferramenta foi integrada a um jogo durante a sua produção para se descobrir como ela se comporta em um ambiente de desenvolvimento real e colher a opinião do game designer responsável sobre como ela alterou a sua maneira de trabalhar e quais foram os benefícios trazidos por sua utilização.

1.4 – Estrutura do Documento

Além deste capítulo de introdução, este documento possui a seguinte estrutura:

Capítulo 2 – Problemas do Design de Níveis em Jogos: aborda a satisfação do jogador e a utilização de ferramentas de game design;

Capítulo 3 – Estado da Arte: apresenta ferramentas de design de níveis e extensões ao Unity;

Capítulo 4 – Projeto Desenvolvido: explica o motor de jogos *Unity* e detalha o projeto desenvolvido;

Capítulo 5 – Estudo de Caso: O Fiat Fantasy Ride: apresenta o jogo utilizado para testar a ferramenta desenvolvida;

Capítulo 6 – Validação: apresenta o resultado da integração da ferramenta ao Fiat Fantasy Ride, a opinião do game designer e o resultado obtido pelo projeto;

Capítulo 7 – Conclusão: expõe as considerações finais e possíveis trabalhos futuros.

2 – Problemas do Design de Níveis em Jogos

Este capítulo explica a satisfação do jogador, sua importância na criação de conteúdo para um jogo e como a utilização de ferramentas de game design pode ajudar neste processo.

2.1 – Satisfação do Jogador

Assim como um bom livro, um jogo precisa manter o interesse do jogador. Um único nível ruim pode se tornar no motivo de abandono de um jogo, especialmente se for um dos níveis iniciais. Por isso é importante que o game designer tenha sempre em mente a satisfação do jogador, se perguntar "este nível é divertido?" (RYAN, 1999).

Satisfação é o sentimento de bem estar que o jogador deve sentir ao jogar o jogo. É esse sentimento que vai fazer com que ele tenha interesse a voltar a jogar. Por isso é um dos fatores mais importantes que o game designer deve ter em mente ao projetar o jogo, e consequentemente os seus níveis.

Para realizar tal tarefa é preciso entender como manter o jogador satisfeito durante as sessões de jogo. Essa satisfação comumente virá quando o jogador completa objetivos dentro do jogo, sejam eles designados pelo próprio jogo ou determinados pelo jogador, que pode estar procurando um desafio maior enquanto joga.

2.1.1 – Estado de flow

Segundo a teoria de *flow* de Mihaly Csikszentmihalyi (1990), um indivíduo possui um conjunto de objetivos que definem o seu eu e qualquer estado mental que entre em conflito com esses objetivos é chamado de entropia psíquica (CARL, 1994).

A entropia psíquica se manifesta como medo, tédio, ansiedade, etc, e depende dos tipos de objetivos com o qual ela entrou em conflito. Quando se elimina a entropia psíquica do consciente entra-se em um estado de experiência ótima ou estado de *flow*. É nesse estado que indivíduo experimenta sensações como felicidade, satisfação, prazer, etc (CARL, 1994).

Assim, indivíduo está sempre a procura de se manter no estado de *flow* o máximo possível. Para conseguir tal feito enquanto se está jogando, deve haver um equilíbrio entre a dificuldade dos desafios encontrados pelo jogador e a sua habilidade naquele momento.

Se o desafio for difícil demais, o jogador ficará ansioso. Por outro lado, se ele for fácil demais, o jogador ficará entediado (CARL, 1994). Essa relação pode ser visualizada na (Figura 2.1).

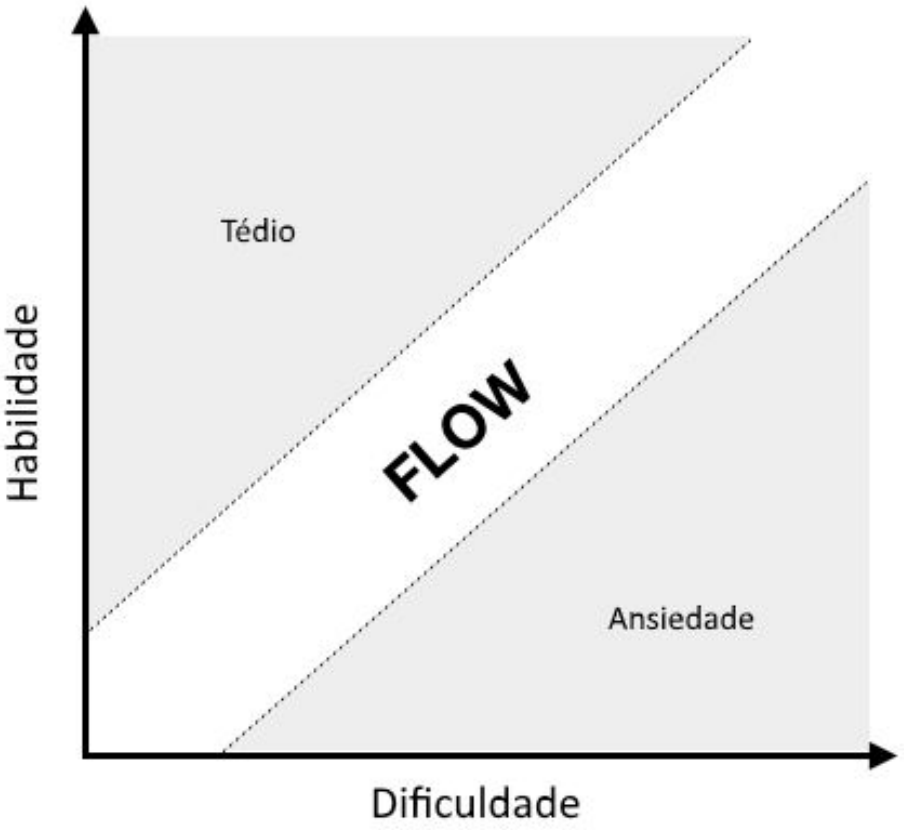

Figura 2.1 – Flow, Relação entre Habilidade e Dificuldade

Portanto, quanto mais compatível for o desafio for com a habilidade, maior será a sua satisfação. Por este motivo é sempre importante apresentar ao jogador um nível de dificuldade compatível, para evitar o caso em que ele fique frustrado e decida abandonar o jogo (CARL, 1994). E é trabalho do game designer garantir um desafio adequado ao jogador.

2.2 – Ferramentas de Game Design

Para poder transformar este conhecimento de como funciona a satisfação do jogador em níveis de qualidade, o game designer precisa de ferramentas que lhe permitam expressar aquilo que ele acredita que vá levar o jogador a encontrar essa satisfação. Na ausência de ferramentas satisfatórias, o game designer precisa recorrer a meios mais

ineficientes de produção e que acabam se tornando mais complicados para se ver o resultado final de suas alterações.

Por exemplo, é muito comum a utilização de planilhas ou arquivos *XML* e *JSON* para realizar o balanceamento de valores de jogo. O problema dessa abordagem é que nem sempre é possível ter noção completa de quais efeitos as alterações terão até que se tenha uma compilação do jogo com os valores atualizados para se testar. Até que isso aconteça já se perdeu um bom tempo de desenvolvimento que poderia ter sido aproveitado para outra tarefa.

O objetivo mais importante de uma ferramenta de criação de mundo é permitir que o game designer visualize o mundo que ele está criando ao mesmo tempo em que ele realiza mudanças no mesmo (ROUSE, 2000). A possibilidade utilizar de tal tipo de ferramenta traz consigo melhorias no processo de criação de conteúdo, pois quanto mais rapidamente e facilmente o game designer puder criar e alterar os níveis criados maior será a quantidade de tempo que ele pode passar testando estes mesmos níveis para garantir a sua qualidade.

A ferramenta também deve ter algum tipo de integração com o ambiente de desenvolvimento do jogo, caso contrário não será possível levar os benefício de utilizá-la até o jogo. Quando esta integração não está disponível ou não funciona, a tarefa de implementá-la e mantê-la recai sobre um ou mais programadores do jogo, sendo necessário desviar atenção do desenvolvimento de funcionalidades do jogo para resolver tal tarefa.

3 – Estado da Arte

Este capítulo apresenta algumas ferramentas de design de níveis e extensões ao Unity e para cada uma delas é feita uma breve análise de como elas se relacionam com o projeto desenvolvido para este trabalho.

3.1 – O *Tanagra*

O *Tanagra (SMITH; WHITEHEAD; MATEAS, 2010)* é uma ferramenta de level design protótipo para jogos de plataforma 2D no estilo de *Super Mario World* ou *Sonic the Hedgehog*. Para representar os níveis foi escolhida uma estrutura de batidas, como em músicas, que permite que o nível como um todo tenha um ritmo de jogo bem definido.

Cada batida tem um tempo de duração e um conjunto de blocos de construção associados a ela. Sua função principal é garantir que a sua geometria tenha um comprimento adequado, baseado na distância máxima que o jogador pode conseguir alcançar dentro da duração da batida. Esta distância máxima é calculada a partir de um modelo físico que define a velocidade máxima e a altura de pulo máxima do avatar do jogador.

Os blocos de construção associados a uma batida são responsáveis por definir sua geometria. Estes blocos são definidos baseado em quais ações o jogador deve desempenhar ao encontrá-lo, como por exemplo correr ao longo de uma plataforma ou pular para matar um inimigo. A geometria de cada bloco também pode ser editada pelo game designer.

Embora o *Tanagra* auxilie no problema de criar níveis atraentes para o jogador, a sua arquitetura focada em jogos de plataforma 2D não serve para solucionar o problema de criar níveis com curvas que necessitam de uma gama grande de variações para se obter uma boa sensação de fluidez no nível.

3.2 – O *Tiled*

O *Tiled* (MAPEDITOR, [s.d.]) é um programa que permite ao game designer criar mapas 2D baseados em uma grade de blocos de construção menores, os *tiles* (de onde vem o nome da ferramenta). Os mapas criados podem ser exportados em formato padrão como XML ou JSON, o que facilita bastante a sua importação em outras ferramentas.

Diversos motores de jogos, como *Phaser* (PHASER, [s.d.]), *Cocos2D* (COSOS2D, [s.d.]) e até o próprio *Unity*, possuem a capacidade de importar mapas exportados pelo *Tiled*, seja utilizando de uma funcionalidade nativa do motor ou por meio de um *plugin*.

O editor permite também a inclusão de metadados nos mapas criados, a separação por camadas de mapas e a adição de objetos que não são afixados à grade mas posicionados livremente, o que amplia bastante a possibilidade de uso dos mapas 2D para além de simples descritores visuais.

No entanto, uma grade de *tiles* 2D não consegue resolver o problema proposto. Enquanto jogos arcade, como *Pac-Man*, ou jogos de plataforma 2D, como *Metroid*, que utilizem blocos retangulares de cenário ou *RPGs* clássicos, como *Chrono Trigger*, poderiam ter seu processo de criação de níveis facilitado por um mapa de *tiles*, basta o cenário ou mapa do jogo não se encaixar em uma grade para que a ferramenta se torne irrelevante. Como o projeto deste trabalho necessitava da criação de níveis com uma grande variação de formatos curvos, o *Tiled* é uma solução inadequada para incluir no processo de criação.

3.3 – O *Bonsai*

Assim como o projeto desenvolvido para este trabalho, o *Bonsai* (CARDOSO, 2013) é uma extensão para o *Unity* e seu editor. Ele permite que o desenvolvimento da lógica do jogo seja feita de maneira visual, sem que seja necessário qualquer conhecimento das linguagens de programação disponíveis no plataforma. Para poder modelar a lógica de maneira visual foi escolhida uma estrutura baseada em árvores de comportamento, devido a sua facilidade de representar os conceitos de lógica equivalentes a aqueles encontrados em linguagens de programação.

A sua execução dentro de um jogo acontece por meio do *Bonsai Component*, cuja função é armazenar e executar uma árvore de comportamento. Embora cada componente só tenha uma única árvore associada, múltiplos componentes podem ser adicionados ao mesmo *GameObject* permitindo que ele herde a lógica de várias árvores de comportamento diferentes.

Cada árvore de comportamento é composta por nós. Estes nós são independentes um do outro e possuem funções específicas, tais como o *Sequence*, que executa os seus nós filhos em sequência, e o *Translate*, que aplica uma transformação de translação em um objeto. Para poder manipular valores externos à árvore, como aqueles definidos em

componentes nativos do *Unity*, se utiliza nós herdados do tipo *Box*, que permite a armazenagem e manipulação de um valor aos seus nós filhos.

Para realizar a construção e edição destas árvores foi criado um conjunto de janelas adicionais ao *Unity*. A janela principal, também chamada de *Bonsai*, exibe os nós da árvore em uma lista hierárquica e permite sua manipulação de maneira similar a uma árvore de pastas. Ao selecionar um dos nós, as opções deste nó aparecem listadas no inspetor do *Bonsai Component* da árvore para que possam ser editados.

Embora seja uma ferramenta flexível e tenha feito uma excelente utilização da plataforma extensível do *Unity*, o *Bonsai* acaba sendo apenas uma alternativa para a produção de lógica de jogo. Como ele não oferece suporte a criação de ferramentas não serve para resolver o nosso problema.

Como foi visto, nenhuma das soluções encontradas consegue solucionar o problema proposto completamente. Tanto o *Tanagra* quanto o *Tiled* auxiliam na criação de níveis, mas não tem flexibilidade suficiente para tratar estágios com curvas mais complexas, além do fato de que o *Tanagra* não possui nenhuma integração com o Unity.

Enquanto isso, o Bonsai já funciona dentro do Unity, mas não é uma ferramenta de criação de níveis, e sim uma ferramenta de programação visual que permite evitar a dependência de um programador para implementar as mecânicas do jogo. Por estes motivos fica evidente a necessidade de se criar uma nova solução para tratar do problema.

4 – Solução Proposta

Este capítulo explica o funcionamento e os conceitos relacionados na criação de jogos utilizando o *Unity*, bem como funciona sua API de extensão. Em seguida, há o detalhamento do projeto desenvolvido.

4.1 – O Unity

O *Unity* (UNITY3D, [s.d.]), ou *Unity3d* como também é conhecido, é um motor de jogos multi-plataforma desenvolvido pela *Unity Technologies.* O desenvolvimento de jogos neste motor é feito com o *Unity Editor*, um programa para os sistemas operacionais *Windows* e *macOS*. Este editor possui funcionalidades focadas nos mais variados ramos do desenvolvimento de jogos, como programação, animação, level design, etc.

O seu ambiente é capaz de processar os principais formatos de *assets* utilizados no desenvolvimento moderno de jogos, o que permite a sua adoção por diferentes times e empresas com os mais variados processos de produção. Além disso, a sua capacidade de exportar compilações para todas as plataformas de jogos significativas do mercado a partir de um único projeto permite produzir mais rápido e por um menor custo.

O código no *Unity* roda em cima da plataforma .NET e pode ser escrito em uma das três linguagens de programação disponíveis: C# e UnityScript, um dialeto de ECMAScript criado pela própria *Unity Technologies*. Dentro do *Unity Editor* o código é separado em dois tipos: runtime e editor. O código de runtime define o funcionamento do jogo. Já o código de editor tem acesso à API de extensão do *Unity Editor* e permite definir processos e lógicas de desenvolvimento personalizados para o projeto.

4.1.1 – Modelagem dos Jogos

Jogos criados no *Unity* são estruturados baseados na arquitetura entidade componente. As cenas são compostas por *GameObjects*, que podem ser organizados de maneira hierárquica, criando assim uma árvore de objetos. Cada *GameObject* possui uma lista de *Components* que encapsulam os dados e a lógica necessários para o processamento de quaisquer sistemas do jogo, sendo assim utilizados para adicionar comportamentos aos objetos.

Na instalação padrão existe uma gama de componentes diferentes disponíveis para uso imediato. Por exemplo, o componente *MeshRenderer* associa um modelo 3D, criado em uma ferramenta de modelagem separada, a um *Material* que define como esse modelo será renderizado na tela.

Entre os tipos de componentes disponíveis existe o *MonoBehaviour*, que permite ao desenvolvedor criar componentes personalizados. Estes componentes possuem um ciclo de vida bem definido dentro do jogo, permitindo executar código em vários momentos do loop de jogo.

Em contrapartida ao *MonoBehaviour* que habilita a criação de componentes, também existe o tipo *ScriptableObject* que habilita a criação de assets personalizados. Estes assets permitem que dados de configuração do projeto sejam guardados no formato nativo do *Unity*. É muito comum utilizar o *ScriptableObject* para definir os objetos que o game designer utilizará para modelar e balancear o jogo.

4.1.2 – O Unity Editor

O *Unity Editor* é o ambiente de desenvolvimento de jogos com *Unity*. Nele é possível criar e editar cenas, organizar e configurar *assets* e exportar compilações para as várias plataformas a que o motor dá suporte. As janelas principais da sua interface podem ser vistas na (Figura 4.1) e são as seguintes:

- **Scene**: mostra os objetos presentes na cena atual em um ambiente 3D;
- **Hierarchy**: lista todos os objetos da cena atual de forma hierárquica e permite a sua edição desta hierarquia;
- **Project**: lista o conteúdo completo do projeto. Os scripts que compõem o código, os *assets* utilizados no jogo e quaisquer outros os arquivos aparecem nesta janela;
- **Game**: apresenta uma prévia da renderização da cena a partir de suas câmeras. Também é utilizada para testar o jogo dentro do próprio editor;
- **Inspector**: permite ao usuário editar as configurações de quaisquer objetos que estejam selecionados no momento, sejam eles presente na cena, pela janela *Hierarchy*, ou no projeto, pela janela *Project*. Isso inclui mostrar todos os componentes associados a um *GameObject* que esteja selecionado.

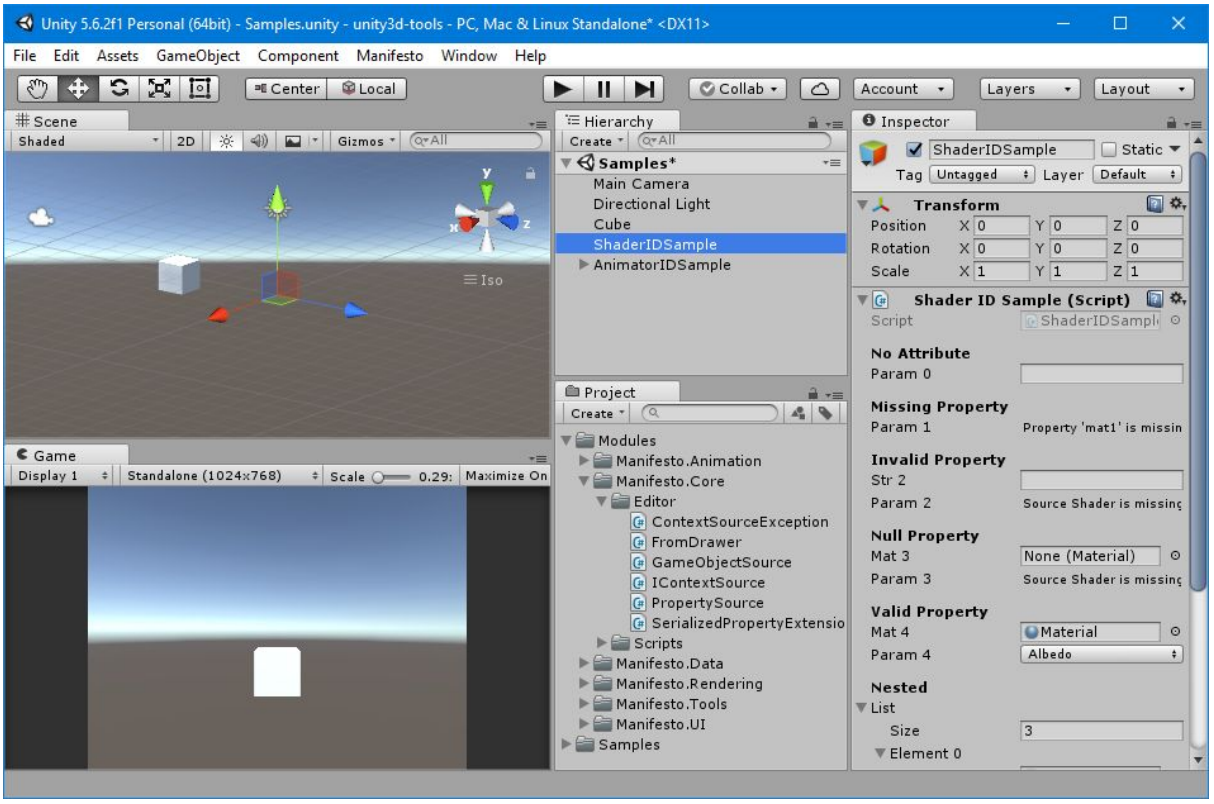

Figura 4.1: A interface principal do Unity Editor

Além da interface nativa do editor, é possível criar novas interfaces utilizando as classes utilitárias da *Immediate Mode GUI* ou *IMGUI*. Nas versões mais antigas do *Unity*, a *IMGUI* era o único sistema disponível na plataforma para se criar as interfaces dos jogos.

Entretanto, com o tempo caiu em desuso por causa das limitações de performance inerentes ao modo imediato de execução. Sistemas alternativos baseados em componentes, como o EZ GUI (ANBSOFT, [s.d.]) e o NGUI (THASAREN, [s.d.]), permitiam a construção da interface diretamente na cena de jogo sem necessidade de se escrever código.

Eventualmente estes sistemas inspiraram a própria *Unity Technologies* a criar um novo sistema de interface próprio baseado em componentes chamado *Unity UI* e integrá-lo diretamente do *Unity*. A partir de então o uso da *IMGUI* ficou restrito a construção de interfaces para o editor.

A *IMGUI* pode ser dividida nas seguintes partes:

- As classes *GUI* e *EditorGUI* possuem funções para desenhar elementos na interface utilizando posições e dimensões absolutas;

- As classes *GUILayout* e *EditorGUILayout* possuem métodos similares a *GUI* e *EditorGUI*, respectivamente, mas sem a necessidade de especificar a posição e dimensões dos elementos, pois elas usam um sistema de layout automático para calcular estes valores;
- As classes *GUIUtility* e *EditorGUIUtility* possuem funções utilitárias.

Os métodos presentes nestas classes representam diversos controles de interface, tais como textos, botões e caixas de seleção. Alguns destes controles permitem que o usuário visualize e manipular valores como textos, números, cores, etc. Também é possível criar novos controles por meio do tratamento manual do loop de eventos. Durante a execução da rotina de interface, a classe *Event* representa o estado atual do loop de eventos, indicando qual é o tipo de evento e suas propriedades. Por exemplo, durante o evento de *Repaint* é feito o envio de comandos para desenhar na tela, enquanto que durante o evento *MouseMove* é feito o tratamento do movimento do mouse.

Existem duas maneiras principais de interação com as interfaces criadas com *IMGUI* no editor. A primeira é a criação de uma nova janela por meio da extensão da classe base *EditorWindow*. Na realidade, todas as janelas do Unity Editor são criadas por meio desta classe, mas suas implementações não estão visíveis na API exposta pelo *Unity*. A segunda maneira se dá por meio da classe base *Editor*, que controla como o *Inspector* apresenta um único componente (subclasse de *MonoBehaviour*) ou asset (subclasse de *ScriptableObject*).

4.2 – O Projeto Desenvolvido

Utilizando as capacidades de extensão do *Unity* foram criados assets e interfaces personalizadas para permitir ao game designer configurar os níveis do jogo. Ao apresentar uma interface mais acessível e de melhor entendimento o game designer pode se tornar mais produtivo durante o processo de desenvolvimento.

Para guardar as informações gerais do jogo é utilizado um asset chamado *GameInfo*. Este asset guarda quais mundos são acessíveis enquanto se joga. Caso um mundo seja criado e incluído na compilação do jogo mas não esteja configurado neste asset ele nunca estará acessível. Sua interface foi criada por meio de uma subclasse de *Editor* chamada *GameInfoEditor*, que além de expor a lista de mundos configurados permite a fácil criação de novos mundos.

4.2.1 – Editor de Mundo

A definição de cada mundo é feita por meio de assets chamados *WorldInfo*, onde cada mundo é representado por um único asset. Assim como *GameInfo*, sua interface foi criada por meio da classe *WorldInfoEditor*. Por meio dela é possível visualizar a lista de níveis associados ao mundo, criar novos níveis e configurar todos os valores atribuídos ao mundo. Estes valores são seguintes:

- Os elementos que definem a sua identidade visual. Estes são elementos que o jogador não terá interação direta, tais como textura do terreno, elementos de cenário, etc;
- Os objetos que podem aparecer nos seus níveis. Estes objetos são aqueles com os quais o jogador vai interagir durante o jogo de alguma maneira;
- A quantidade de estrelas necessárias para destravá-lo.

4.2.2 – Editor de Nível

A definição de cada nível é feita por meio de assets chamados *LevelInfo*. Assim como *GameInfo* e *WorldInfo*, sua interface foi criada por meio da classe *LevelInfoEditor*. Nesta interface é possível visualizar uma prévia da curva de terreno associada ao nível e configurar valores relativos ao funcionamento do nível. São eles:

- A condição de vitória. Para o nível ser completado com sucesso e se avançar ao nível seguinte é necessário completar o objetivo;
- A condição para se obter as estrelas. O valor de cada uma das três (3) estrelas é configurado separadamente;
- As posições inicial e final. A sessão de jogo começa com o avatar do jogador posicionado no ponto inicial e termina quando ele alcança o ponto final;
- As posições de ativação de tutoriais. Permite que diferentes tutoriais sejam mostrados ou escondidos do usuário;
- As posições de *spawn* dos objetos juntamente com os pesos de cada um. Durante a execução do jogo os pesos são utilizados para realizar o sorteio ponderado de qual objeto será utilizado.

4.2.3 – Editor de Terreno

Como a criação do terreno tem uma grande importância na definição do nível, seu processo de edição foi separado do editor de nível para permitir que o game designer

pudesse fazê-lo de maneira focada, sem precisar se preocupar com os outros elementos que definem o nível.

Os dados da curva de terreno são guardados em um asset chamado *Curve* e o sua interface é implementada pela classe *CurveEditor*. Há também a classe *CurvePreview*, que funciona como um controle de interface e encapsula a lógica da visualização da curva. Ambas as classes *LevelInfoEditor* e *CurveEditor* utilizam *CurvePreview* para mostrar a curva na interface.

Utilizando os valores configurados em *Curve* é possível calcular a coordenada y de qualquer coordenada x passada à implementação, e com isso obter a lista de pontos que compõem a curva de terreno. Este processo é chamado de geração da curva. Para realizar este cálculo, é utilizada uma lista de *CurveBlocks*. Cada *CurveBlock* representa um pedaço da curva completa por meio de uma série de sub-curvas. Cada sub-curvas é definida por uma função de simples implementação, como seno e cosseno, e por dois parâmetros: *offset* e *multiplier*. Durante a geração, para se obter a coordenada *y* para um determinado *x* do bloco é realizado o somatório do a aplicação de cada sub-curva definida utilizando a seguinte equação:

$$y = \sum_{i=1}^{n} fn_i(x + offset_i) * multiplier_i$$

E finalmente, para evitar a quebra da suavidade ao fazer a transição de um bloco para outro, é feita uma interpolação do ângulo final do bloco anterior até o ângulo inicial do seguinte.

5 – Estudo de Caso: O Fiat Fantasy Ride

Este capítulo apresenta o jogo utilizado para testar a ferramenta desenvolvida. É feita uma descrição de como o jogo é estruturado e de quais elementos compõem o seu game design.

O jogo Fiat Fantasy Ride, desenvolvido pela Manifesto Games para as plataformas Android e iOS, serviu de laboratório de testes para o desenvolvimento deste projeto. Ele tem um público alvo de crianças na faixa etária de 5 a 7 anos e foi disponibilizado ao público no ano de 2014 por meio das lojas online Apple App Store e Google Play. Dentro do jogo, o jogador realiza uma jornada por meio de vários mundos de fantasia, onde ele controla um carro e encontra diversos caroneiros com os quais ele interage durante o percurso. O objetivo é sempre completar o percurso com a maior quantidade possível de moedas e pontos.

O estilo do jogo foi baseado em outro jogo de grande sucesso, o *Tiny Wings* (ILLIGER, 2011), lançado na iOS App Store da Apple em 18 de Fevereiro de 2011 e relançado em versão HD em 12 de Julho de 2012 e escolhido iPhone Game of the Year em 2011 na Europa e em outros países. Neste jogo, o jogador controla um pássaro que possui asas pequenas demais para voar. Sem a possibilidade de levantar vôo sozinho, o jogador precisa utilizar o terreno do mundo 2D para obter velocidade suficiente para voar por algum tempo até voltar ao chão. Os níveis são gerados proceduralmente, o que torna toda sessão de jogo única.

5.1 – Elementos do Jogo

No Fiat Fantasy Ride, os níveis possuem elementos característicos que o jogador pode identificar. Como pode ser visto na (Figura 5.1), são eles:

- O carro;
- O terreno acidentado, definido pela curva de terreno;
- Um caroneiro esperando uma carona;
- As moedas.

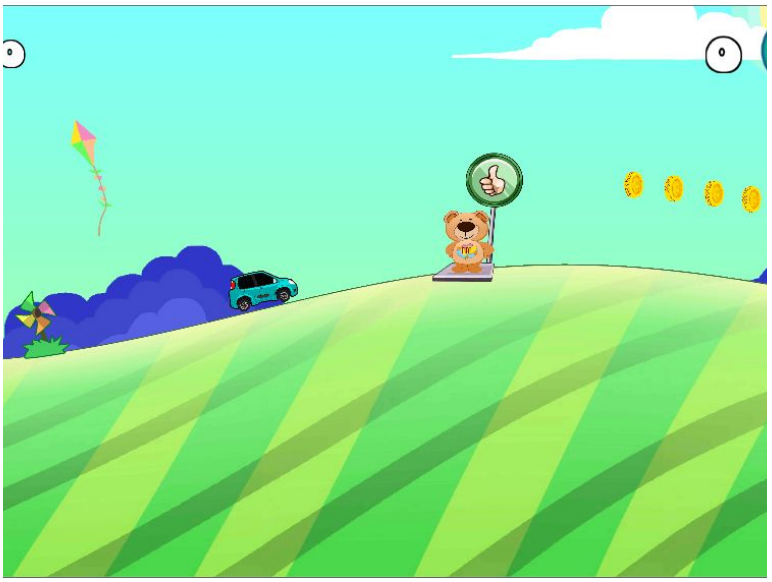

Figura 5.1 – Uma cena comum ao jogar o Fiat Fantasy Ride

Para controlar o carro o jogador interage com a tela do aparelho móvel das seguintes maneiras:

- Pressionar a tela com apenas um dedo aplica força para baixo no carro, fazendo com que ele caia mais depressa. Com isso, o carro ganha velocidade ao descer ladeiras, mas a perde ao subi-las;
- Quando o carro está fora de contato com o solo é possível pressionar a tela com dois dedos para fazer o carro girar no ar, ganhando os pontos que servem para completar objetivos.

Ao longo de cada nível é possível encontrar caroneiros, coletar moedas e ganhar pontos. Os caroneiros são personagens que estão presentes nos níveis do jogo esperando por uma carona para poder continuar suas próprias viagens. Diferentes caroneiros causam diferentes efeitos no carro. Os caroneiros disponíveis e seus respectivos efeitos são os seguintes:

- **Gino (Figura 5.2)**: faz com que o carro pare de responder aos controles do jogador e acelere a uma alta velocidade, percorrendo uma grande distância;
- **Mágico (Figura 5.3)**: cria moedas na frente do carro, que podem ser facilmente coletadas;
- **Cientista (Figura 5.4)**: gera um campo magnético ao redor do carro que atrai moedas próximas facilitando que elas sejam coletadas;

Figura 5.2 – Gino

Figura 5.3 – Mágico

Figura 5.4 – Cientista

- **Macaco (Figura 5.5)**: aumenta o peso do carro, tornando mais difícil subir as ladeiras e faz diminuir o tempo que o carro permanece no ar;
- **Vovó (Figura 5.6)**: impede que o jogador ganhe pontos ao girar o carro no ar;
- **Biscoito (Figura 5.7)**: faz com que o jogador ganhe bônus ao girar o carro no ar;

Figura 5.5 – Macaco

Figura 5.6 – Vovó

Figura 5.7 – Biscoito

- **Torcedor (Figura 5.8)**: pula dentro do carro fazendo com que ele desvie aleatoriamente de seu curso;
- **Sereia (Figura 5.9)**: diminui o efeito da gravidade sobre o carro, fazendo com que alcance alturas mais altas;
- **Arraia (Figura 5.10)**: gruda o carro ao chão, fazendo com que ele perca velocidade;

Figura 5.8 – Torcedor

Figura 5.9 – Sereia

Figura 5.10 – Arraia

- **Caubói (Figura 5.11)**: aumenta a potência do carro, fazendo alcançar rapidamente sua velocidade máxima;
- **Balão (Figura 5.12)**: faz o carro flutuar e perder toda a sua velocidade. O jogador tem que realizar toques repetidos no balão até que ele estoure e o carro fique livre novamente;
- **Fantasma (Figura 5.13)**: faz o jogador perder o controle do carro, que passa a flutuar devagar sem contato com o solo.

Figura 5.11 – Caubói

Figura 5.12 – Balão

Figura 5.13 – Fantasma

No início de cada nível pode ser apresentado um objetivo que precisa ser completado para que o nível seguinte do mesmo mundo seja destravado. Quando presente, o objetivo é um dos seguintes:

- **Limite de tempo:** O nível tem que ser completado antes que o tempo restante se esgote, caso contrário o jogador perde imediatamente;
- **Quantidade mínima de moedas:** Ao chegar ao final do nível o jogador tem que ter

obtido um valor mínimo de moedas para poder continuar;

- **Quantidade mínima de pontos:** Ao chegar ao final do nível o jogador tem que ter obtido pelo valor mínimo de pontos para poder continuar;
- **Quantidade mínima de caronas:** Ao chegar ao final do nível o jogador tem que ter dado carona a pelo valor mínimo de caroneiros para poder continuar;
- **Dar carona a um caroneiro específico:** Ao chegar ao final do nível o jogador tem que ter dado carona ao caroneiro requisitado para poder continuar.

Em certo ponto do nível o jogador encontra uma roda dourada gigante, que indica o final do nível. Ao completar um nível o jogador ganha de zero (0) a três (3) estrelas dependendo do seu desempenho, que são utilizadas para desbloquear novos mundos para serem jogados.

O final do nível também faz surgir uma tela de resumo das conquistas do jogador naquele nível. Este resumo informa quais caroneiros o jogador encontrou durante seu percurso, quantas estrelas ele obteve baseado em sua performance e a quantidade de pontos e moedas adquiridos, juntamente com o total acumulado destes dois últimos valores.

5.2 – Os Mundos

Os níveis do jogo são agrupados em três mundos. Os mundos são compostos por 50 níveis cada e possuem uma identidade própria de forma a apresentar um tema em comum, evidenciado pela arte utilizada, e pelos caroneiros e grupos de moedas que podem ser encontrados nos seus níveis.

O primeiro mundo, (Figura 5.14), aborda a temática de parque de diversões, é composto por um fundo com montanhas-russas e roda-gigante onde é possível encontrar os caroneiros Gino, Mágico, Cientista, Balão, Fantasma e Vovó.

Figura 5.14 – O primeiro mundo do Fiat Fantasy Ride

O segundo mundo, (Figura 5.15), traz a temática subaquática, apresenta um fundo composto por cardumes que passeiam de forma aleatória com a presença de bolhas sobre o terreno onde é possível encontrar os caroneiros Sereia, Arraia, Caubói, Fantasma, Gino, Vovó, Cientista e Torcedor.

Figura 5.15 – O segundo mundo do Fiat Fantasy Ride

O terceiro mundo, (Figura 5.16), apresenta a temática da terra dos doces e evidencia um fundo composto por nuvens de algodão doce com canudos de chocolate e balões de sorvete onde é possível encontrar os caroneiros Biscoito, Caubói, Macaco, Fantasma, Cientista, Gino, Vovó e Torcedor.

Figura 5.16 – O terceiro mundo do Fiat Fantasy Ride

6 – Validação

Este capítulo apresenta o resultado da integração da ferramenta desenvolvida com o projeto do Fiat Fantasy Ride, a opinião do game designer responsável pelo projeto obtida através de uma entrevista e por fim uma ponderação sobre o resultado final da criação da ferramenta.

6.1 – Integração ao Fiat Fantasy Ride

Como a ferramenta foi criada como uma extensão da plataforma *Unity*, integrá-la a outros projetos dentro do mesmo ambiente é relativamente simples. Basta adicionar o código fonte da extensão ao projeto em questão. A partir daí podem ser implementadas quaisquer lógicas necessárias para integrar a ferramenta ao resto do projeto.

Entretanto, foi necessário fazer algumas alterações por causa da maneira como o Fiat Fantasy Ride lida com os elementos de cenário de cada mundo. Ao invés de utilizar referências diretas aos assets destes elementos foi necessário utilizar referências indiretas por meio do nome dos assets a serem utilizados.

6.2 – Opinião do Game Designer

A fim de validar a efetividade da ferramenta foi realiza uma entrevista com o game designer responsável pelo projeto do Fiat Fantasy Ride para escutar a sua opinião sobre o comportamento da ferramenta durante a criação dos níveis para o jogo.

Nesta entrevista, o game designer avaliou a ferramenta como uma excelente alternativa ao processo de criação por meio de arquivos de texto que estava sendo usado até então, embora que o fato da visualização da curva de terreno estar restrita à janela *Inspector* limitasse a sua capacidade criativa. Ele sugeriu que a visualização também ficasse disponível na janela Scene, além de permitir a alteração direta das posições de tutorial e *spawn* utilizando funções de arrastar e soltar os marcadores na cena.

6.3 – Resultado

Apesar das complicações durante a integração com o projeto, a ferramenta se mostrou capaz de solucionar o problema proposto e permitiu que o game designer a utilizasse no processo de criação dos níveis de maneira mais eficiente que as outras

alternativas existentes, mesmo. Infelizmente o editor de terreno não ficou à altura de sua visão inicial, mas ainda assim se mostrou eficaz durante a definição dos níveis.

7 – Conclusão

Neste capítulo há as considerações finais deste trabalho, no qual analisa-se as contribuições geradas pelo projeto e apresentá-se algumas sugestões acerca de trabalhos futuros na ferramenta criada.

Após esta seção, são apresentadas as referências bibliográficas utilizadas como base para este documento.

7.1 – Contribuições

Embora a definição de curvas 2D sejam uma disciplina amplamente utilizada nos vários ramos da matemática e computação, não havia nenhuma ferramenta que permitisse utilizá-las dentro do ambiente de desenvolvimento do Unity. Esta nova alternativa já foi colocada à prova por meio de um projeto posterior também desenvolvido pela Manifesto Games que criou um jogo com mecânicas muito similares ao Fiat Fantasy Ride, o que permitiu a reutilização da ferramenta criada neste trabalho.

7.2 – Limitações

Devido ao prazo para entrega do projeto do Fiat Fantasy Ride e a necessidade de utilizá-lo para realizar a validação deste projeto não foi possível dedicar muito tempo ao desenvolvimento do editor de terreno.

Um exemplo dos efeitos disso é o método de composição de curvas implementado na ferramenta. A forma padrão de definir curvas em geral é utilizando ferramentas de desenho. É visualmente intuitivo, permite interação com a curva, e se relaciona diretamente com a forma de pensar e definir curvas não-virtualmente. Como game designers têm backgrounds muito variados, a definição de curvas por meio de edição de parâmetros pode se tornar inclusive um empecilho no design do nível para aqueles que têm mais afinidade com trabalho visual.

7.3 – Trabalhos Futuros

A melhoria mais imediata e que traria um ganho de qualidade significativo é a alteração do editor de terreno para funcionar baseado em métodos de definição de curvas

mais robustos, como curvas de bézier ou splines. A adição de tal capacidade vai permitir que o game designer tenha ainda mais controle sobre a curva final gerada.

A implementação da visualização da curva de terreno na janela de Scene também

Outro ponto que pode ser melhorado é o processo inicial de criação de um nível. Ao ser criado o nível está vazio e precisa ser populado do zero para que fique jogável. Ao invés disso deveria ser possível gerar uma versão inicial, criada aleatoriamente baseada em poucos parâmetros, para que o game designer pudesse começar a testar e iterar mais rápido.

Referências

ANBSOFT. **EZ GUI**. Disponível em: <http://www.anbsoft.com/middleware/ezgui/>. Acesso em: 09 jul. 2017.

CARDOSO, Anderson Campos. **Bonsai: um editor de programação gráfica para o motor de jogos Unity 3D**. 66 f. Monografia (Bacharelado) – Departamento de Ciência da Computação, Universidade de Brasília, 2013.

CARL, Walter. **Flow – A Theory of Optimal Experience: History and Critical Evaluation**, 1994. Theories of Communication. Disponível em <https://pdfs.semanticscholar.org/a910/76e9fadb3f2bbd3faaa008e18873ac951f04.pdf>. Acesso em: 09 jul. 2017.

COCOS2D. **Cocos2d**. Disponível em: <http://cocos2d.org/>. Acesso em: 09 jul. 2017.

CSIKSZENTMIHALYI, Mihaly. **Flow**: The Psychology of Optimal Experience. HarperCollins, 1990.

ILLIGER, Andreas. **Tiny Wings**, fev. 2011. Disponível em: <https://itunes.apple.com/br/app/tiny-wings/id417817520>. Acesso em: 09 jul. 2017.

MAPEDITOR. **Tiled**. Disponível em: <http://www.mapeditor.org/>. Acesso em: 09 jul. 2017.

PHASER. **Phaser**. Disponível em: <http://phaser.io/>. Acesso em: 09 jul. 2017.

RABIN, Steven. **Introduction to Game Development:** Second Edition. 2. ed. Course Technology Cengage Learning, 2010.

ROUSE, Richard. **Designing Design Tools**, mar. 2000. Disponível em: <http://www.gamasutra.com/view/feature/131850/designing_design_tools.php>. Acesso em: 09 jul. 2017.

RYAN, Tim. **Beginning Level Design, Part 1**, abr. 1999. Disponível em: <http://www.gamasutra.com/view/feature/131736/beginning_level_design_part_1.php>. Acesso em: 09 jul. 2017.

SMITH, Gillian; WHITEHEAD, Jim; MATEAS, Michael. Tanagra: A Mixed-Initiative Level Design Tool. **Proceedings of the Fifth International Conference on the Foundations of Digital Games**, Monterey, p. 209-216, jun. 2010.

TASHAREN. **NGUI: Next-Gen UI kit**. Disponível em: <http://www.tasharen.com/?page_id=140>. Acesso em: 09 jul. 2017.

UNITY3D. **Unity**. Disponível em: <https://unity3d.com/>. Acesso em: 09 jul. 2017.