Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Bacharelado em Ciência da Computação

TEd2D: Um editor para criação de cenários de jogos 2D com Unity

Isaac Newton da Silva Beserra

Natal-RN

Junho de 2015

Isaac Newton da Silva Beserra

TEd2D: Um editor para criação de cenários de jogos 2D com Unity

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Orientador

Prof. Dr. Charles Andryê Galvão Madeira

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Informática e Matemática Aplicada – DIMAp

Natal-RN

Junho de 2015

Monografia de Graduação sob o título *TEd2D: Um editor para criação de cenários de jogos 2D com Unity* apresentada por Isaac Newton da Silva Beserra e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Prof. Dr. Charles Andryê Galvão Madeira
Orientador
Instituto Metrópole Digital
Universidade Federal do Rio Grande do Norte

Prof. Dr. Rogério Junior Correia Tavares
Co-orientador
Departamento de Artes
Universidade Federal do Rio Grande do Norte

Prof. Dr. André Maurício Cunha Campos
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte

Prof. Dr. Eduardo Henrique da Silva Aranha
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte

Natal-RN, Dezessete de Junho de 2015

Dedico este trabalho aos melhores pais do mundo, Miguel e Maria Aparecida.

Agradecimentos

Primeiramente a minha Esposa Ana Monielly, que me incentivou a fazer uma graduação. Acredito que sem ela eu não teria tido essa iniciativa.

Minha família: pai, mãe e irmãos que sempre me apoiaram, em todos os momentos durantes esses 4 anos.

Um agradecimento mais que especial para um dos membros da banca examinadora, André Maurício, que com toda certeza fez uma enorme diferença no meu curso.

A todos os novos amigos que fiz durante todo o percurso na universidade, em especial: Jean, Emerson, Wendell, Marcos, Gil, Gabriel, Victor, Caio, Juliano (Pontinho), Lucas, Carine e Rômulo.

Ao atual tutor do PET, Umberto Costa, por sua paciência, compreensão e calma, principalmente nesse último período. Assim como também a todos que fazem parte desse grupo, no qual permaneci durante todo o curso.

Ao meu Orientador, Charles, e Co-Orientador, Roger, principalmente por suas disponibilidades para ajudar na construção do trabalho.

Não sabendo que era impossível, foi lá e fez.

Mark Twain

TEd2D: Um editor para criação de cenários de jogos 2D com Unity

Autor: Isaac Newton da Silva Beserra

Orientador: Prof. Dr. Charles Andryê Galvão Madeira

Resumo

Motores de desenvolvimento de jogos têm ganhado muito espaço nestes últimos anos, tanto na indústria quanto na academia, ajudando a facilitar de forma considerável o desenvolvimento destas aplicações. Unity é um exemplo de motor que está tendo grande sucesso atualmente no mercado, tendo forçado uma mudança nos modelos de negócios de diversos motores tradicionais. Ele possui diversas vantagens por disponibilizar uma grande quantidade de elementos prontos que podem ser facilmente reutilizados, tornando o desenvolvimento de jogos muito mais rápido. Apesar de todas as suas vantagens, o Unity não possui um editor de cenários muito intuitivo. Ele deixa os usuários livres para criarem as suas próprias cenas, mas não os auxiliam na etapa de posicionamento dos diversos objetos no mundo a ser simulado. Muitas vezes isso deixa o trabalho exaustivo, principalmente no caso de cenários grandes e complexos que necessitam de um tempo bastante considerável para conseguir obter um bom alinhamento dos objetos no mundo. O presente trabalho propõe um editor de cenas para auxiliar a construção de cenários 2D com Unity através de um processo de distribuição facilitada dos objetos em um cenário, permitindo assim alavancar bastante a produtividade. Este editor foi experimentado e validado com diversos desenvolvedores e não-desenvolvedores de software no processo de criação de cenários para jogos em 2D. Os resultados obtidos são bastante promissores pois permitem a criação de cenários muito mais rapidamente.

Palavras-chave: Motores de jogos, Editor de cenários, Unity

TEd2D: A scene editor for building 2D games with Unity

Author: Isaac Newton da Silva Beserra

Advisor: Prof. Dr. Charles Andryê Galvão Madeira

Abstract

Game engines have gained notoriety in the last years, both in industry and academy, due to make the development of games easier. Unity is a game engine that is currently having great success in the market, promoting a change in the business models of many traditional game engines. It has several advantages by providing a large amount of prefabs that can be easily reused, making the development of games faster. Despite of these advantages, Unity does not have an intuitive scene editor. It leaves free users to create their own scenes, but not assist in positioning step of the various objects in the world to be simulated. This results in an exhaustive and difficult work, especially for large and complex scenarios that require a huge time for a good alignment of objects in the world. This work proposes a scene editor to assist the development of 2D games in order to increase productivity.. This editor has been experimented and validated with several software developers and non-developers in the process of building scenarios for 2D games. The results are very promisingsince they demonstrate that 2D games can be developped more quickly with TEd2D.

Keywords: Game Engine, Scene Editor, Unity

Lista de figuras

1 Tela do editor de cenas padrão do Unity . . . p. 19

2 Tela do editor de cenas padrão do Unity com um objeto posicionado no cenário . . . p. 19

3 Barra de atalhos para customização . . . p. 20

4 Janela na qual valores de posição, rotação e escala podem ser modificados manualmente . . . p. 20

5 Prefabs para a criação de um ponte . . . p. 22

6 View de Assets onde todos os elementos são organizados em uma hierarquia p. 23

7 Exemplo de alguns componentes que podem ser adicionados em um prefab p. 24

8 Componente Transform . . . p. 25

9 Componente RigidBody 2D . . . p. 25

10 Componente Sprite Renderer . . . p. 26

11 Componentes Box e Circle Collider 2D . . . p. 26

12 Componente Animator . . . p. 26

13 Representação da máquina de estados na interface do Unity . . . p. 27

14 Personagem em estado Idle (Parado) . . . p. 27

15 Personagem em estado GoingUp (Saltando) . . . p. 28

16 Personagem em estado Falling (Caindo) . . . p. 28

17 Personagem em estado Running (Correndo) . . . p. 28

18 Interface para a criação de scripts no Unity . . . p. 29

19 Quando um novo script é criado, os métodos Start e Update são criados automaticamente . . . p. 30

20 Prefab Plataforma . . . . p. 32
21 Cena vazia . . . . p. 33
22 Cenário com 1 prefab . . . . p. 34
23 Cenário com cópias dos prefabs . . . . p. 35
24 Cenário com vários elementos . . . . p. 36
25 Grid . . . . p. 40
26 Menu de itens padrão do Unity . . . . p. 42
27 Menus de itens do Unity com a adição da criação dos TileSets . . . . . p. 42
28 Inspector GUI . . . . p. 43
29 Inspector . . . . p. 43
30 TileSets . . . . p. 45
31 Editor Final . . . . p. 45
32 Importando o editor . . . . p. 46
33 Importando o editor . . . . p. 47
34 Elementos do TEd2D . . . . p. 47
35 Criando um TileSet . . . . p. 48
36 Criando um TileSet . . . . p. 48
37 Caminho para criar o TEd2D . . . . p. 49
38 TEd2D sem nenhum TileSet Selecionado . . . . p. 49
39 Plataforma simples criada com o TEd2D . . . . p. 50
40 Imagem do cenário modelo utilizado para os experimentos. O cenário contém prefabs de plataformas, moedas, itens, etc. . . . . p. 53
41 Gráfico comparativo do tempo gasto, em segundos, para construção do cenário modelo com o TEd2D e sem o TEd2D. . . . . p. 54
42 Lista de questões com possibilidade de respostas Sim ou Não. . . . . . p. 54
43 Lista de questões com possibilidade de respostas: concordo Totalmente, Concordo, Não sei/Indiferente, Discordo e Discordo Totalmente . . . . p. 55

44 Lista de questões com possibilidade de respostas Sim ou Não. . . . . . p. 55

45 Lista de questões com possibilidade de respostas Sim, Não ou Depende do jogo. . . . . p. 55

46 Cena do jogo Mathmare utilizado na disciplina . . . . p. 58

47 Exemplo de tela do jogo TryZ . . . . p. 59

48 Exemplo de tela contendo parte do cenário de uma fase do Enigma . . p. 59

Lista de tabelas

1 Comparação entre editores de cenários para Unity . . . . p. 38

2 Tabela de comparação . . . . p. 51

3 Tempo gasto na seguinte ordem de execução da tarefa: sem TEd2D e com TEd2D . . . . p. 53

4 Tempo gasto na seguinte ordem de execução da tarefa: com TEd2D e Sem TEd2D . . . . p. 53

Sumário

1 Introdução p. 14

1.1 Motivação . . . p. 15

1.2 Objetivo do trabalho . . . p. 16

1.3 Organização do trabalho . . . p. 16

2 Referencial teórico p. 17

2.1 Unity . . . p. 17

2.1.1 Editor de cenários . . . p. 18

2.1.2 Classe Editor . . . p. 20

2.1.2.1 Variáveis . . . p. 20

2.1.2.2 Funções Públicas . . . p. 21

2.1.2.3 Funções estáticas . . . p. 21

2.1.2.4 Mensagens . . . p. 22

2.1.3 Prefabs . . . p. 22

2.1.3.1 Criação de *Prefabs* em tempo de execução . . . p. 24

2.1.3.2 Criação do *Prefab Player* . . . p. 25

2.1.3.3 Player Controller (Script) . . . p. 29

2.1.3.4 Mudança de estado das animações . . . p. 31

2.1.3.5 Criação de outros tipos de Prefabs: Plataforma, Moedas, Personagens, etc. . . . p. 32

2.2 Construindo um cenário com o editor do Unity . . . p. 33

2.3 Trabalhos relacionados . . . p. 36

3 Editor TEd2D p. 39

3.1 Concepção do TEd2D . . . . p. 39

3.1.1 Grid . . . . p. 39

3.1.2 Grid Editor . . . . p. 40

3.1.3 TileSet . . . . p. 41

3.1.4 Inspector GUI . . . . p. 43

3.1.5 Seletor de Prefabs . . . . p. 44

3.2 Utilizando o TEd2D . . . . p. 46

3.2.1 Passo 1 - Importando o Package . . . . p. 46

3.2.2 Passo 2 - Criando os TileSets . . . . p. 47

3.2.3 Passo 3 - Criando o editor . . . . p. 49

3.2.4 Passo 4 - Criando uma Cena com o TEd2D . . . . p. 50

4 Metodologia Experimental p. 52

4.1 Construção de cenários simples . . . . p. 52

4.2 Construção de jogos . . . . p. 56

4.2.1 Mathmare . . . . p. 56

4.2.2 TryZ . . . . p. 58

4.2.3 Enigma . . . . p. 59

5 Considerações finais p. 60

Referências p. 61

1 Introdução

A área de desenvolvimento de jogos vem evoluindo muito nos últimos anos. Este fato ocorre principalmente pela grande variedade de motores de jogos existentes atualmente. Os motores de jogos e ferramentas RAD (Rapid Application Development) são peças fundamentais para o desenvolvimento de jogos, pois permitem adicionar recursos sofisticados a eles, tornando-os mais interessantes, com maior realismo, melhor jogabilidade, e, por consequência, com um maior apelo comercial e competitividade junto ao mercado. (BITTENCOURT, 2006)

Cada motor possui características distintas, e.g. renderização de gráficos em 2D e/ou 3D, motor de modelagem física ou apenas para detecção de colisão, suporte a animação, gerenciamento de efeitos e trilhas sonoras, gerenciamento de arquivos, memória ou linha de execução e suporte a uma linguagem de script (GREGORY, 2009). Cada característica tem um papel fundamental no processo de desenvolvimento. Alguns motores deixam a desejar em algumas delas, fazendo com que os desenvolvedores procurem formas alternativas para suprir suas necessidades. Por esta razão, em muitos casos os desenvolvedores precisam fazer uso de plugins ou frameworks desenvolvidos por terceiros.

Diversos motores existem atualmente. Alguns exemplos deles são os seguintes: Torque Game Engine[1], Unity[2], Blender[3], CryEngine[4] e Unreal Engine[5]. Dentre estes motores, o Unity é um exemplo que merece destaque. Criado pela Unity Technologies, ele é um dos motores para criação de jogos mais utilizados na atualidade (JORDAO, 2013).

Em relação ao seu modelo de interface gráfica, Unity é similar ao Blender, Virtools ou Torque Game Engine. Ele permite desenvolver jogos nas mais variadas plataformas, o que fez com que ganhasse bastante espaço no mercado nos últimos anos. Existe duas versões deste motor, uma gratuita para desenvolvimento amador e uma paga (Unity Pro)

[1]http://torque3d.org/
[2]http://unity3d.com
[3]https://www.blender.org/
[4]http://www.crytek.com/cryengine
[5]https://www.unrealengine.com/

para desenvolvimento profissional.

O Unity possui mecânicas pré-programadas e animações que podem ser adicionadas muito facilmente. Para as animações, utiliza-se uma máquina de estados que indica exatamente qual animação deve ser exibida de acordo com cada determinado estado. Vários componentes podem ser adicionados para deixar o objeto com propriedades físicas: massa, gravidade, velocidade, etc. Para a criação de cenários, Unity deixa o usuário livre, podendo posicionar os objetos de jogos em qualquer que seja a localização da cena. No entanto, esse "deixar livre" pode se tornar um problema se o jogo em questão a ser construído for composto de grandes cenários que precisam ser montados manualmente.

1.1 Motivação

O Unity é um ferramenta para a criação de jogos que conta com diversas vantagens. Ele possui vários recursos que podem agilizar bastante a produção de jogos, tanto em 2D quanto em 3D. Mesmo com todas as vantagens, o Unity deixa a desejar em seu editor de cenários, onde na construção de uma cena, o desenvolvedor fica livre para organizar os objetos nela. Para cada objeto adicionado na cena, é preciso fazer um alinhamento de sua posição no espaço. No caso de jogos em 2D, é preciso alinhar os valores de X e Y para que os objetos fiquem no local adequado. Em cenas com centenas de objetos, um árduo trabalho é necessário para alinhar cada um deles, fazendo com que seja necessário buscar alguma forma automatizada de alinhamento e posicionamento.

Por esta razão, muitos desenvolvedores utilizam formas alternativas de posicionamento de objetos em cena. Uma delas é o posicionamento através da execução de scripts, onde os objetos são dispostos por linhas de comandos, podendo assim serem posicionados adequadamente. O problema dessa técnica é que não é possível visualizar a cena no momento da edição, mas apenas após a execução do procedimento.

Por outro lado, alguns motores dão suporte a um editor de cenas que permite os usuários criarem cenários de maneira facilitada, tornando assim o processo mais rápido e eficiente. Alguns exemplos destes motores são o RPG Maker[6] e o GameMaker[7].

Apesar do Unity não possuir um editor de fácil uso como é o caso do RPG Maker e do GameMaker, ele possui uma biblioteca especial chamada Editor, que permite ao usuário alterar a própria interface do ambiente de criação de cenas do Unity, adicionando,

[6] http://www.rpgmakerweb.com/

[7] https://www.yoyogames.com/studio

modificando ou removendo elementos. Isto significa que, através desta biblioteca, o Unity disponibiliza ferramentas para que o usuário possa criar o seu próprio editor de cenários.

1.2 Objetivo do trabalho

O objetivo deste trabalho consiste em reduzir a limitação do Unity para criar cenários em jogos 2D, para isso, será feito um editor de cenários chamado Ted2D (Tile Editor 2D), que será utilizado para facilitar o posicionamento e a organização dos objetos em ambientes de jogos em 2D. Para isto, serão adicionados novos elementos gráficos na interface de edição do Unity que permitirá ao desenvolvedor selecionar os objetos disponíveis e em seguida adicioná-los de forma automatizada nas cenas. Para cada elemento novo adicionado, sua posição no espaço será alinhada com os demais objetos, fazendo com que não exista a necessidade de controlar manualmente o posicionamento. Dessa forma, a limitação do Unity em criar cenários será reduzida, fazendo com que o desenvolvedor não perca muito tempo na construção dos cenários do jogo. Para avaliar a eficácia desta solução, experimentos serão realizados com usuários fazendo uso da solução proposta e sem fazer uso dela no contexto da construção de cenários de jogos em 2D.

1.3 Organização do trabalho

O trabalho está organizado da seguinte forma: O capítulo 2 apresenta o referencial teórico necessário para esse trabalho. O capítulo 3 introduz o editor de cenas proposto neste trabalho, chamado TEd2D, descrevendo cada uma de suas camadas e seu modo de uso. Em seguida, no Capítulo 4, serão apresentados os experimentos que foram realizados para validar o TEd2D, além da avaliação dos resultados obtidos através destes experimentos. O último capítulo concluirá o documento e apresentará as possibilidades de trabalhos futuros.

2 Referencial teórico

2.1 Unity

Motor de jogos digitais (ou Game Engine) é um software composto por um conjunto de bibliotecas que abstrai o desenvolvimento de jogos, permitindo facilitar consideravelmente esta tarefa. Algumas das funcionalidades que podem ser encontradas em uma Game Engine são, por exemplo, modelagem física, detecção de colisão, motor gráfico para a renderização de gráficos em 2D e/ou 3D, sistema de automatização de personagens, gerenciamento de memória e de arquivos, dentre outros (GREGORY, 2009)

O Unity é um motor de jogos digitais que apresenta uma estrutura organizacional de projeto bem elaborada. Cada recurso utilizado em um projeto pode ser decomposto em uma hierarquia, deixando a navegação mais simplificada. O desenvolvimento de jogos com Unity pode ser feito usando duas linguagens de programação: C# e JavaScript.

O Unity possui vários recursos que agilizam o processo de desenvolvimento, tanto recursos internos quanto externos. Recursos internos são os recursos primitivos, que o próprio motor disponibiliza. Recursos externos são elementos que podem ser adicionados no projeto como um incremento. O Unity permite o uso de recursos já criados por outros desenvolvedores, o que permite agilizar algumas etapas no processo de desenvolvimento. A maioria desses recursos pode ser encontrada na própria loja oficial do Unity (Asset Store)[1]. Essa loja possui vários recursos e projetos prontos, que podem ser baixados e reutilizados em outros projetos, assim como também para o aprendizado. Na Asset Store tanto encontramos elementos gratuitos quanto elementos pagos mais bem elaborados.

Além de elementos prontos, o Unity disponibiliza ferramentas de aprendizado[2] para o desenvolvedor. Dentre estas ferramentas, existem vários tutoriais e toda a documentação necessária para o desenvolvedor utilizar classes do Unity em seus projetos.

[1] https://www.assetstore.unity3d.com/
[2] (http://unity3d.com/learn

O Unity também permite o desenvolvimento de jogos digitais para diversas plataformas. Este é um dos motivos que fez com que a ferramenta ganhasse bem mais espaço no mercado. Com ele, podemos exportar nossos projetos para as seguintes plataformas: iOS, Android, BlackBerry, Windows Phone ou Windows, além de diversos consoles e handhelds existentes. Não é necessário nenhuma programação extra, apenas a reconstrução do projeto para a plataforma-alvo selecionada.

2.1.1 Editor de cenários

Um editor de cenários é uma ferramenta onde podemos criar as cenas dos jogos. Uma cena é o local onde estarão dispostos todos os objetos de jogo, formando o cenário onde o jogador poderá interagir. Cada motor de jogo possui seu próprio editor de cenários padrão. Alguns motores oferecem suporte a um editor mais amigável e de fácil entendimento. Outros simplesmente deixam o desenvolvedor livre para posicionar e organizar os objetos em cena.

Uma das grandes dificuldades existentes quando fazemos uso do Unity, está relacionada à criação de cenas para jogos. Quando se trata de cenas simples, a tarefa não é muito difícil, pois basta arrastar os elementos e organizá-los da forma desejada. Mas quando é preciso criar cenas grandes e mais bem trabalhadas, a tarefa começa a ficar mais complicada.

O Unity, como todo motor de jogo, possui seu editor de cenários padrão. É um editor simples e de fácil uso, pois ele permite que você posicione e organize seus objetos em cenas apenas arrastando-os. Isso é fácil de ser feito, mas a medida que as cenas vão se estendendo, vai ficando cada vez mais complicado gerenciar tudo isso. Cada elemento precisa ser alinhado manualmente, já que o Unity não alinha os objetos automaticamente. Isso torna o trabalho de organização da cena muito demorado e exaustivo.

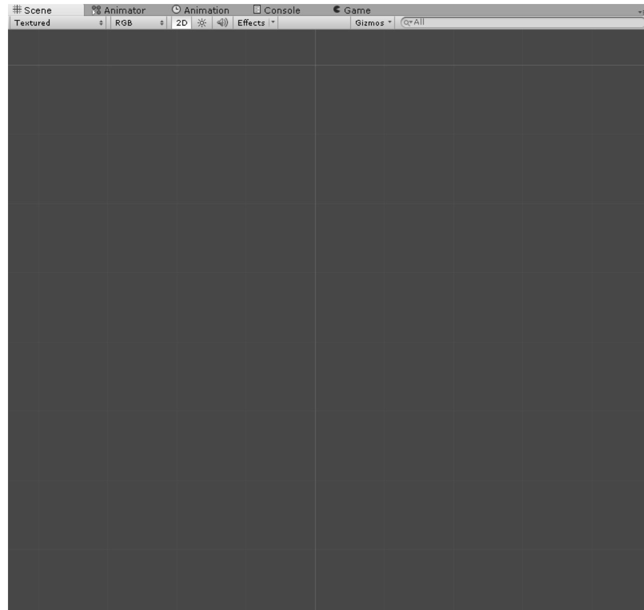

Figura 1: Tela do editor de cenas padrão do Unity

Na figura 1 podemos ver o editor de cenários padrão do Unity. Para criar um cena, basta apenas arrastar os elementos e posicioná-los da maneira que quisermos. Os elementos podem ser sprites, mídia, animações, dentre outros. Percebe-se claramente que não existe nenhum recurso gráfico para auxiliar o desenvolvedor, mas apenas uma cena vazia onde os objetos devem ser posicionados.

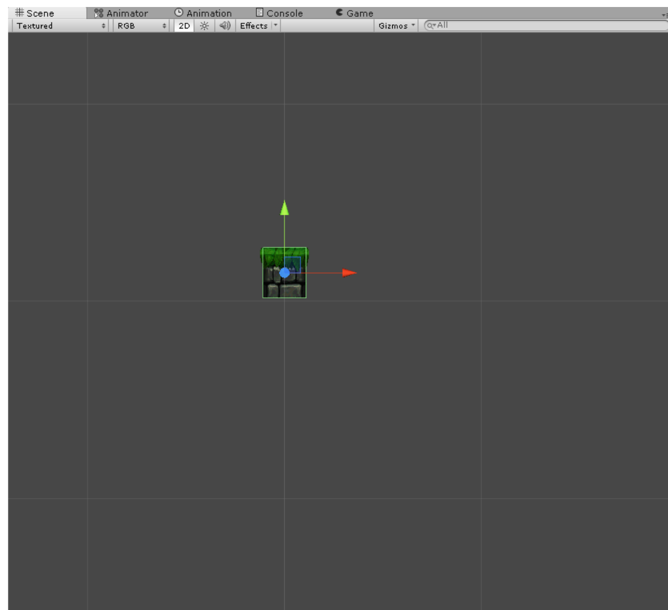

Figura 2: Tela do editor de cenas padrão do Unity com um objeto posicionado no cenário

Na figura 2, podemos ver um objeto posicionado em cena. Depois de adicionado, podemos modificar sua posição, escala ou rotação. Isso pode ser feito de duas formas:

usando os ícones de atalhos de customização (ver figura 3) ou modificando manualmente os valores no espaço do elemento (ver figura 4).

Figura 3: Barra de atalhos para customização

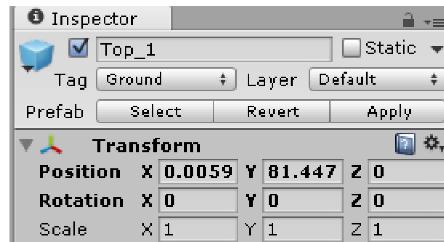

Figura 4: Janela na qual valores de posição, rotação e escala podem ser modificados manualmente

Se quisermos algo preciso, o ideal é alterar manualmente os valores no espaço dos elementos. Isso não é difícil de ser feito, porém, a medida que os cenários crescem, fica cada vez mais difícil organizá-los.

2.1.2 Classe Editor

A classe *Editor* do Unity permite derivar editores personalizados para seus objetos. Cada objeto possui sua própria interface *Inspector*, onde ficará as informações de cada objeto. Fazendo uso da classe *Editor*, é possível personalizar da maneira que for preciso a interface *Inspector*.

A classe *Editor* possui diversos métodos que podem ser usados para fazer alterações na interface do Unity. Abaixo será listado os elementos principais dessa classe. Os elementos em **negrito** são os que foram utilizados nesse trabalho.

2.1.2.1 Variáveis

- serializedObject: Um SerializedObject representa o objeto ou objetos que estão sendo inspecionados.
- **target: O objeto que está sendo inspecionado.**
- targets: matriz de todo o objeto que está sendo inspecionado.

2.1.2.2 Funções Públicas

- DrawDefaultInspector: Desenhe o inspector embutido.
- DrawHeader: Chame esta função para desenhar o cabeçalho do editor.
- DrawPreview: O primeiro ponto de entrada para pré-visualização do desenho.
- GetInfoString: Implementar este método para mostrar informações de objetos em cima da pré-visualização do mesmo.
- GetPreviewTitle: Substituir este método se você quiser mudar o rótulo de área de visualização.
- HasPreviewGUI: Substituir esse método nas subclasses se você implementar OnPreviewGUI.
- **OnInspectorGUI: Implementar esta função para fazer um inspector personalizado.**
- OnInteractivePreviewGUI: Implementar para criar o seu próprio um pré-visualização personalizada.
- OnPreviewGUI: Implementar para criar o seu uma pré-visualização personalizada para a área de visualização do inspector.
- OnPreviewSettings: Substituir este método se você quer mostrar controles personalizados no cabeçalho da pré-visualização.
- RenderStaticPreview: Substituir este método se você quiser tornar uma visualização que mostra estática.
- **Repaint: repinta quaisquer inspetores mostrado no Editor.**
- UseDefaultMargins: Substituir esse método nas subclasses para retornar false se você não quer que as margens padrão.

2.1.2.3 Funções estáticas

- CreateCachedEditor: Retorna o editor anterior para um objeto alvo ou objetos alvo. A função retorna se o editor já está rastreando os objetos, ou Destrói o editor anterior e cria um novo.
- **CreateEditor: Faça um editor personalizado para objeto ou objetos alvo.**

2.1.2.4 Mensagens

- **OnSceneGUI: Permite trabalhar com eventos durante o uso do Editor.**

2.1.3 Prefabs

Para criar um cena completa no Unity, primeiramente temos que entender o conceito de Prefabs, pois eles são utilizados como elementos principais para a criação de cenas neste motor.

Prefabs são objetos pré-fabricados que podem ser utilizados em qualquer posição de um cenário de jogo. Os prefabs podem ser adicionados manualmente em cena ou através de scripts. A figura 5 ilustra alguns exemplos de prefabs, onde a *Bridge* constitui um prefab composto, enquanto os demais são prefabs simples.

Figura 5: Prefabs para a criação de um ponte

Para criar um Prefab no Unity é bastante simples, pois é necessário apenas criar um novo objeto de jogo e arrastá-lo para a View de Assets (ver figura 6). Na view de assets, é possível criar diversas pastas para organizar o projeto. O ideal é criar pastas específicas para cada tipo de elemento novo. Com um prefab criado, é possível criar cópias dele em cena, apenas arrastando-o da View de Assets para a cena.

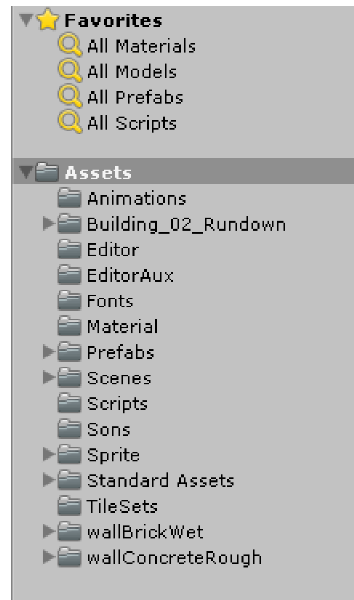

Figura 6: View de Assets onde todos os elementos são organizados em uma hierarquia

Cada Prefab pode ser customizado de forma diferente. Por exemplo, variando a sua posição, tamanho e direcionamento no espaço. Além disso, atributos que forem públicos, como velocidade, vida, entre outros, também podem ser customizados. Isso Depende de como a classe controladora do objeto foi criada.

A criação de Prefabs torna o desenvolvimento rápido, fazendo com que não seja preciso recriar o mesmo objeto várias vezes. Para desenvolver um jogo, é preciso criar diversos prefabs que podem ser reaproveitados em cada nova cena.

A figura 7 mostra exemplos de atributos que podem ser modificados para tornar um prefab diferente dos outros.

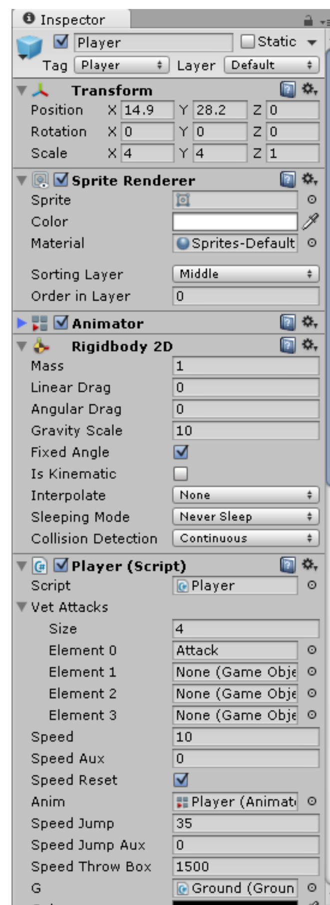

Figura 7: Exemplo de alguns componentes que podem ser adicionados em um prefab

2.1.3.1 Criação de *Prefabs* em tempo de execução

Prefabs podem ser criados em tempo de execução, o que se caracteriza por ser bastante útil para criar objetos que só serão instanciados em momentos definidos, como por exemplo o lançamento de um projétil.

Em diversos jogos, o uso de tiros é comum, tanto por inimigos como pelo personagem principal. Com a criação do prefab Tiro (shoot) é possível fazer com que objetos do jogo, ou até mesmo o jogador principal, possa instanciar tiros. Para isso, é necessário utilizar um comando de instância de objetos no Unity, que é o *Instanciate(<Object>)*. Este comando cria uma nova instância de um objeto que pode ser um prefab e ter alguns de seus atributos modificados no momento da criação. Com isso, objetos criados podem assumir diversas posições em um cenário de jogo.

2.1.3.2 Criação do *Prefab Player*

Para qualquer gênero de jogo, o prefab do personagem principal é o de maior importância no projeto. Nele estarão contidos todos os componentes necessários para o jogador controlá-lo. Todo objeto em jogo possui um ou mais componentes. Por *default*, todo objeto possui o componente *Transform*. Este componente é responsável pelas informações de posição, escala e rotação do objeto em jogo. A figura 8 mostra os elementos contidos no componente Transform. Cada um desses elementos pode ser modificado para que o objeto possa assumir diferentes valores no espaço tais como posição, rotação e escala.

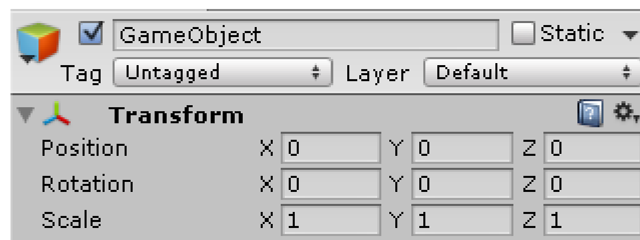

Figura 8: Componente Transform

O *prefab Player* possui o componente *Transform* por padrão, mas ele também é composto por diversos outros componentes, cada um deles com um propósito específico. Dentre eles estão os componentes *RigidBody 2D*, *Sprite Renderer*, *Animator*, *Player Control (Script)*, *Player Monster Controller (Script)*, *Player Hack Mechanics (Script)*, *Box Collider 2D* e *Circle Collider 2D*, conforme descritos abaixo..

- *RigidBody 2D:* Responsável por controlar a parte física dos objetos. Uma vez adicionado, o objeto passará a ter massa, atrito, gravidade, detecção de colisão, entre outros. Todos esses atributos podem ser modificados na própria interface do Unity (ver figura 9).

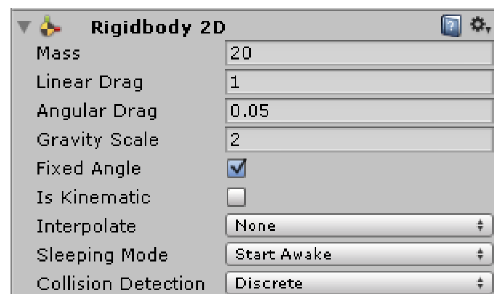

Figura 9: Componente RigidBody 2D

- *Sprite Renderer:* Responsável pela renderização do objeto. O componente Sprite

Renderer pode ser visto na figura 10.

Figura 10: Componente Sprite Renderer

- *Box Collider e Circle Collider:* Responsáveis por determinar a área de colisão dos objetos. Para haver colisão, um dos objetos precisa conter o componente *RigidBody 2D*. Para tornar o procedimento de tratamento de colisão mais preciso, são usados dois componentes em conjunto. A figura 11 ilustra ambos os componentes.

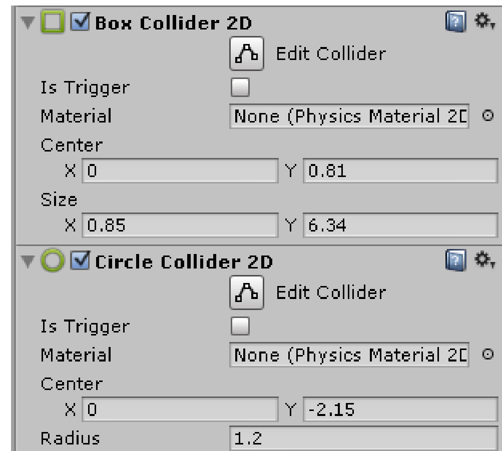

Figura 11: Componentes Box e Circle Collider 2D

- *Animator:* Responsável por realizar a animação do personagem. O componente *Animator* possui uma máquina de estados que controla a transição entre as animações. O *script* controlador é o responsável por enviar os pedidos de transição. A figura 12 ilustra o componente Animator e seus atributos que podem ser configurados.

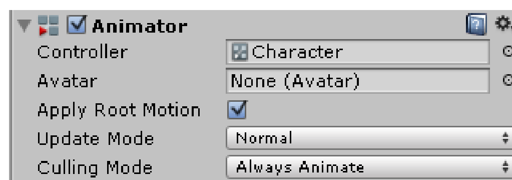

Figura 12: Componente Animator

- *Máquina de estados:* A máquina de estados do *Animator* pode ser vista na Aba *View* do Unity, selecionando Animator. Nesta aba pode-se criar variáveis para controlar

a transição entre as animações, e acessá-las via *script* para modificá-las. A figura 13 mostra como é a representação da máquina de estados na interface do Unity.

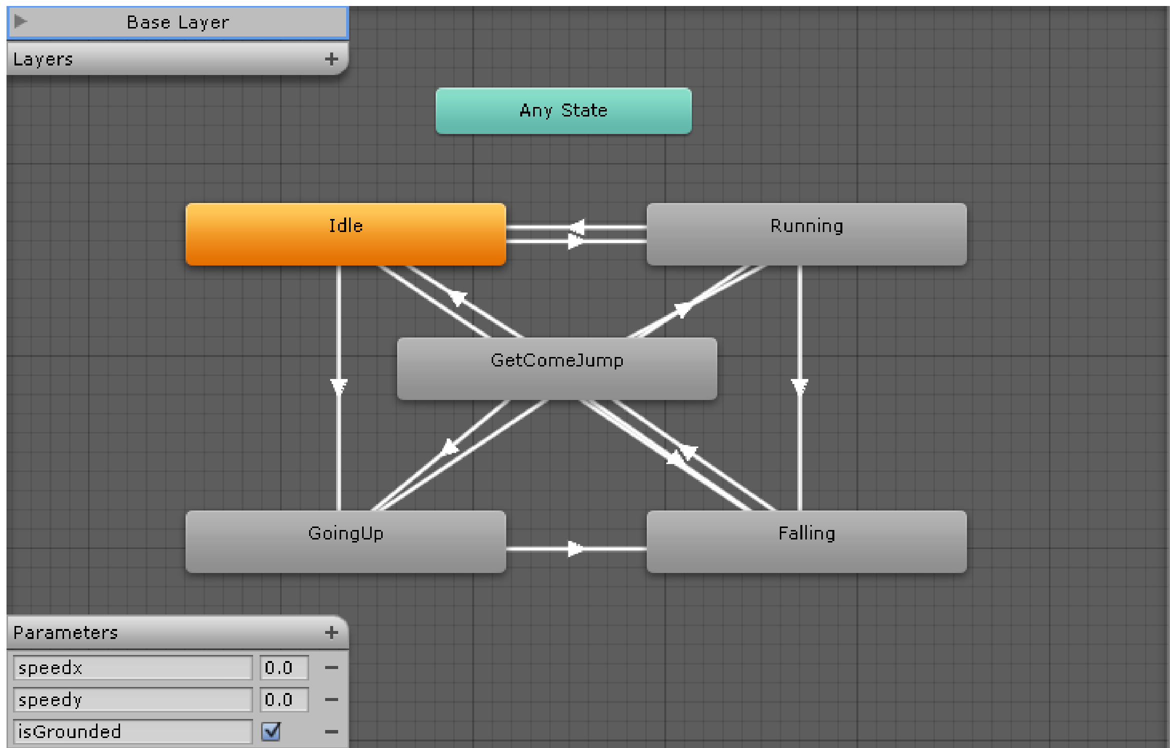

Figura 13: Representação da máquina de estados na interface do Unity

Cada um desses estados consiste em um tipo de animação de um objeto. As figuras 14 até 17 exemplificam animações do prefab de um personagem de um jogo em 2D.

Figura 14: Personagem em estado Idle (Parado)

Figura 15: Personagem em estado GoingUp (Saltando)

Figura 16: Personagem em estado Falling (Caindo)

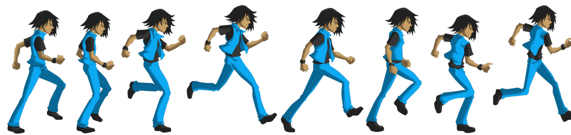

Figura 17: Personagem em estado Running (Correndo)

2.1.3.3 Player Controller (Script)

O componente principal do *Player* é responsável por definir os controles de jogo, como por exemplo: *Mover* e *Saltar*. Para a criação de um novo *Script*, basta clicar com o botão direito na View Asset e escolher Create/Script (ver figura 18). Em seguida basta abrir o script com um editor qualquer. Apesar do Unity disponibilizar o seu editor padrão, muitos usuários preferem editar seus scripts com outros editores.

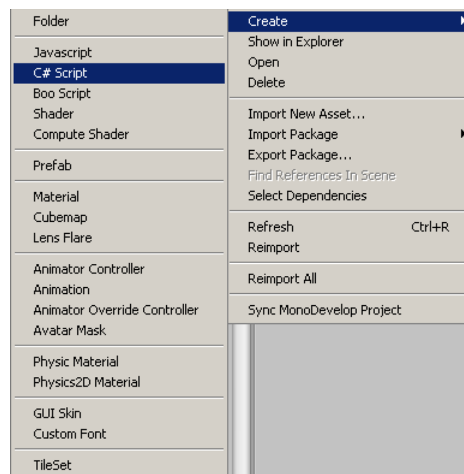

Figura 18: Interface para a criação de scripts no Unity

Por padrão são criados dois métodos: *Start* e *Update* (ver figura 19). O método *Start* é usado para inicializar o objeto vinculando ao script, sendo executado apenas uma única vez no momento da sua criação. Já o método *Update* é usado para atualização do objeto vinculado ao script, sendo executado a cada frame. Ele é chamado a cada tick do jogo, e é nele onde se encontram os comandos de ações do jogador.

No *script* podemos declarar variáveis públicas e privadas. Quando uma variável é declarada de forma pública, outras classes podem ter acesso a esses valores, como também essas variáveis estarão visíveis na interface do Unity e podem ser configuradas manualmente. Já as variáveis privadas, ficam limitadas a apenas serem usadas dentro da classe.

Um exemplo de variável customizada é a velocidade do *Player* que pode ser modificada na própria interface. Isso se torna útil para encontrarmos o valor correto da velocidade do personagem.

No método *Update*, a entrada do teclado é utilizada para recuperar o comando solicitado pelo jogador. Se o comando for mover para a direita, o atributo velocidade do componente RigidBody2D é acrescido positivamente no eixo X.

```
using UnityEngine;
using System.Collections;

public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Figura 19: Quando um novo script é criado, os métodos Start e Update são criados automaticamente

O componente *RididBody2D* pode ser acessado diretamente dentro da classe. Alguns componentes no Unity podem ser acessados diretamente sem o uso de nenhum comando especial. Já outros, precisam ser acessados através de comandos específicos.

Com o comando Mover para a direita ativado, podemos acessar o vetor velocidade do componente *RigidBody2D* e modificá-lo da maneira desejada. Desta forma, um novo vetor é alocado com a componente X modificada. O Valor dessa modificação pode ser controlado por uma variável.

O movimento para a esquerda é feito de maneira análoga, porém temos que ter cuidado com a animação nesse ponto. O *player* precisa olhar para o lado oposto sempre que mudar de direção. Isso é feito modificando o componente *Transform* sempre que a direção for alterada. A mudança deve ocorrer através da inversão do sinal positivo para negativo ou vice-versa.

Essa mudança de escala pode ser verificada utilizando a velocidade do *player* no eixo X. Sempre que a velocidade for positiva, a escala também é positiva, caso contrário, a escala também passará a ser negativa.

Um outro comando importante é o salto. Por exemplo, em um jogo em 2D de plataforma, na maioria das vezes o personagem pode saltar entre as plataformas existentes no cenário. Neste caso, é necessário utilizar mais um vez o componente *Rigidbody2D* para efetuar o salto. Assim, uma força é aplicada no eixo Y do player. O comando para a aplicação dessa força é o *rigidbody.AddForce(Vector2)*, onde *Vector2* é o vetor força que será aplicado. Este vetor pode definido nas declarações de variáveis, e se o deixarmos público,

seremos capazes de customizá-lo na própria interface do *Unity*.

No entanto, existe uma particularidade no salto. Sempre que for pressionado o botão referente ao salto, a força será aplicada. Isso faz com que o jogador possa ficar repetindo o salto sem parar. Para resolver esse problema, uma vez que o salto somente deve ser executado quando o personagem estiver no chão, um outro componente chamado Ground é criado.

O objeto *Ground* envia uma mensagem para o *player* notificando-o sobre a possibilidade de executar o salto. Para isso, um *BoxCollider* é usado para detectar sua colisão com o solo. Caso ele esteja colidindo, o personagem poderá saltar.

2.1.3.4 Mudança de estado das animações

Para alguns comandos executados pelo jogador, existe a necessidade de uma alteração na animação de alguns objetos. Por exemplo, nas animações do personagem principal tais como, movimentação, salto, ou ficar parado. A animação ficar parado acontece sempre que o personagem se encontra no solo e com velocidade nula em ambos os eixos. A mudança de estado para *correndo* acontece sempre que a velocidade em Y é igual a zero e em X diferente de zero. O estado saltando acontecerá quando a velocidade em Y for positiva e o estado caindo quando a mesma for negativa.

Para efetuar essas mudanças de estado, temos que acessar o componente *Animator* via *Script* e alterar os valores das variáveis criadas nas transições, fazendo com que o estado seja modificado de acordo com estas variáveis.

O acesso ao componente *Animator* é feito diferentemente do *RigidBody2D*. As duas formas de acesso possíveis são as seguintes: declarar um *Animator* público e vinculá-lo ao Player ou utilizar o método *GetComponente<Animator>()*.

Depois do componente estar vinculado ao objeto, é possível acessá-lo via *script* e alterar os valores das variáveis de mudança de estado. Segue exemplo de mudança de valores de transição:

$$animator.SetFloat("speedY", velocity.y); \tag{2.1}$$

Os valores dos atributos tanto podem ser recuperados quanto modificados. Existe comandos específicos para recuperá-los ou modificá-los. O editor do *Unity* apresenta os possíveis métodos utilizados e o seus respectivos parâmetros.

O comando mostrado em 2.1 altera o valor da variável *speedY* para o valor da velocidade em Y do objeto. Caso esta velocidade seja não nula, o estado será mudado de acordo com a implementação existente na máquina de estados.

2.1.3.5 Criação de outros tipos de Prefabs: Plataforma, Moedas, Personagens, etc.

Os principais elementos para um jogo em 2D são: plataformas, personagens, itens, moedas, obstáculos, entre outros. Uma vez que estes elementos são criados, é possível fazer uso deles para montar uma cena com o editor de cenários padrão do Unity.

A criação do prefab Plataforma é bem simples. Ela pode ser feita criando um GameObject com a imagem referente ao tipo de plataforma escolhido e adicionando o componente BoxCollider2D. plataformas simples, basta seguir esse procedimento. No entanto, um problema existirá quando houver a necessidade de criar plataformas maiores e mais complexas.

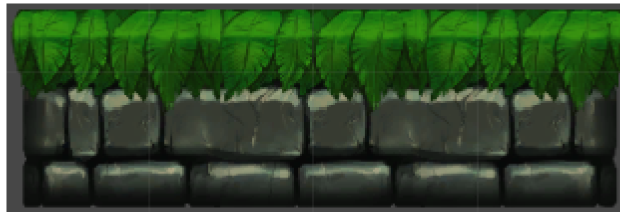

Figura 20: Prefab Plataforma

A figura 20 mostra um exemplo de plataforma simples criada. Para isto, foram utilizados 3 sprites diferentes, um para a parte esquerda da figura, outro para o centro e outro para a direita. Pode-se perceber que o sprite que é utilizado no centro é repetido duas vezes. O centro pode ter diversas repetições para a criação de uma plataforma de tamanho maior. Podemos criar vários prefabs com tamanhos diferentes para facilitar na criação dos cenários.

O prefab da Plataforma não possui nenhum script associado a ele, pois a única funcionalidade dele é possuir um componente para detectar colisões.

O prefab da Moeda já possui alguns componentes a mais que o da plataforma. Ele tem um Animator para sua animação, um Cicle Collider para detecção de colisão e um script que enviará uma notificação de incremento caso a moeda for coletada.

Os personagens são um pouco mais complexos do que as plataformas e as moedas. Cada personagem pode conter seu conjunto de atributos específicos. Eles têm que executar

movimentos repetidos entre as plataformas, andando de uma margem até a outra. Para isso, cada personagem possui dois componentes que delimitam o caminho a ser percorrido. Assim, cada um pode efetuar percursos diferentes apenas alterando a posição deste componente.

Os demais prefabs que forem necessários ao jogo que estiver sendo desenvolvido podem ser criados de forma similar a estes. A seguir será demonstrado como estes prefabs podem ser usados para criar cenários utilizando o editor do Unity.

2.2 Construindo um cenário com o editor do Unity

Para construir um cenário completo com o Unity, é preciso ter todos os prefabs prontos. Depois disso, basta adicioná-los em cena, através de um procedimento que pode ser feito manualmente ou através de script.

A figura 21 mostra uma cena vazia do Unity onde serão organizados todos os prefabs para formar um cenário. Para construir um cenário, é preciso colocar cada objeto em seu devido lugar.

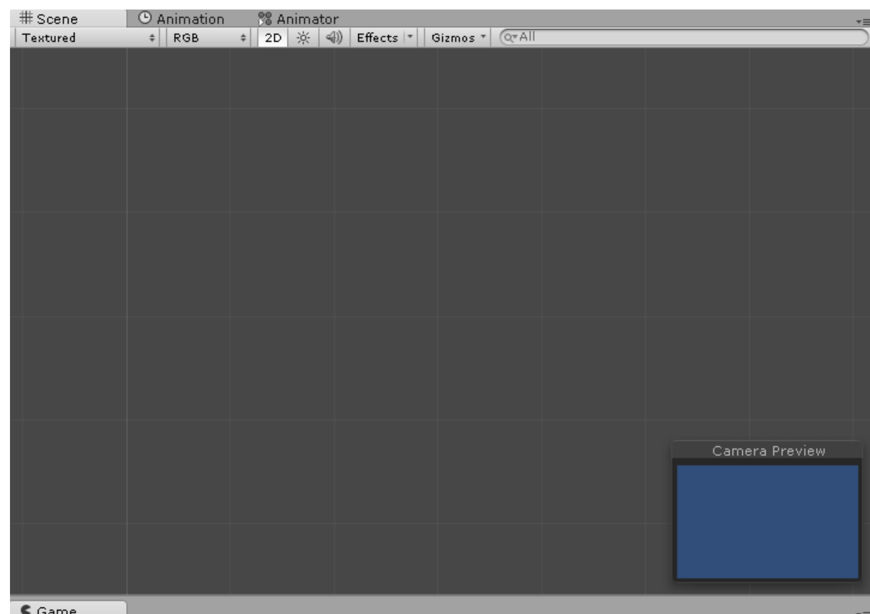

Figura 21: Cena vazia

A figura 22 mostra a adição de um prefab de plataforma. Para construir o piso é preciso fazer cópias desse prefab, posicionando-as e alinhando-as uma a uma manualmente.

Na Figura 22 existem duas telas. A tela superior é o editor de cenários, onde os objetos serão inseridos para formar um cenário. A tela inferior é uma pré-visualização do cenário

formado, onde é mostrado o que a câmera visualiza. Os objetos que estiverem fora do campo de visão só serão visualizados caso a câmera entre em movimento e alcance suas posições.

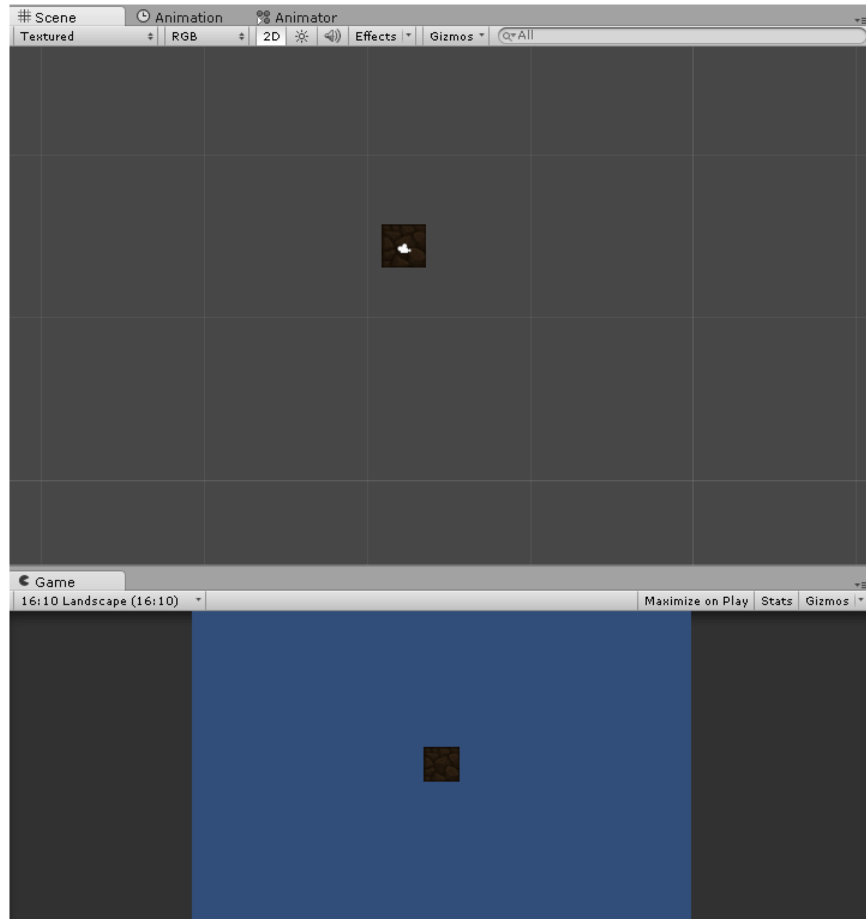

Figura 22: Cenário com 1 prefab

A figura 23 apresenta um cenário contendo várias cópias do mesmo prefab. Isso permite formar uma plataforma onde o personagem irá caminhar.

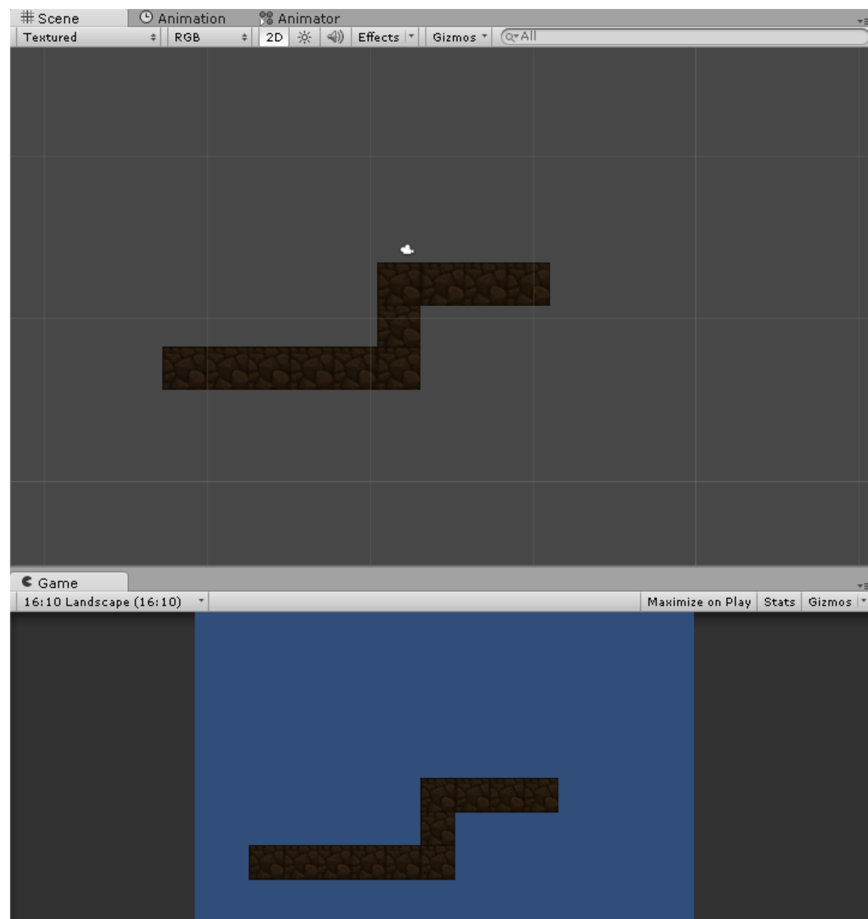

Figura 23: Cenário com cópias dos prefabs

A figura 24 mostra um cenário com vários elementos. Para adicionar outros elementos como personagens, moedas, obstáculos, tudo pode ser feito da mesma forma. Este processo de criação de cenários com o editor padrão do Unity leva bastante tempo para efetuar todo o alinhamento e posicionamento de cada um dos objetos. O problema acontece porque cada um desses prefabs tem que ser inserido manualmente no cenário. Cada um deve ser alinhado para o cenário não ficar com falhas. Posicionar cada um deles em seu devido lugar é uma tarefa enfadonha.

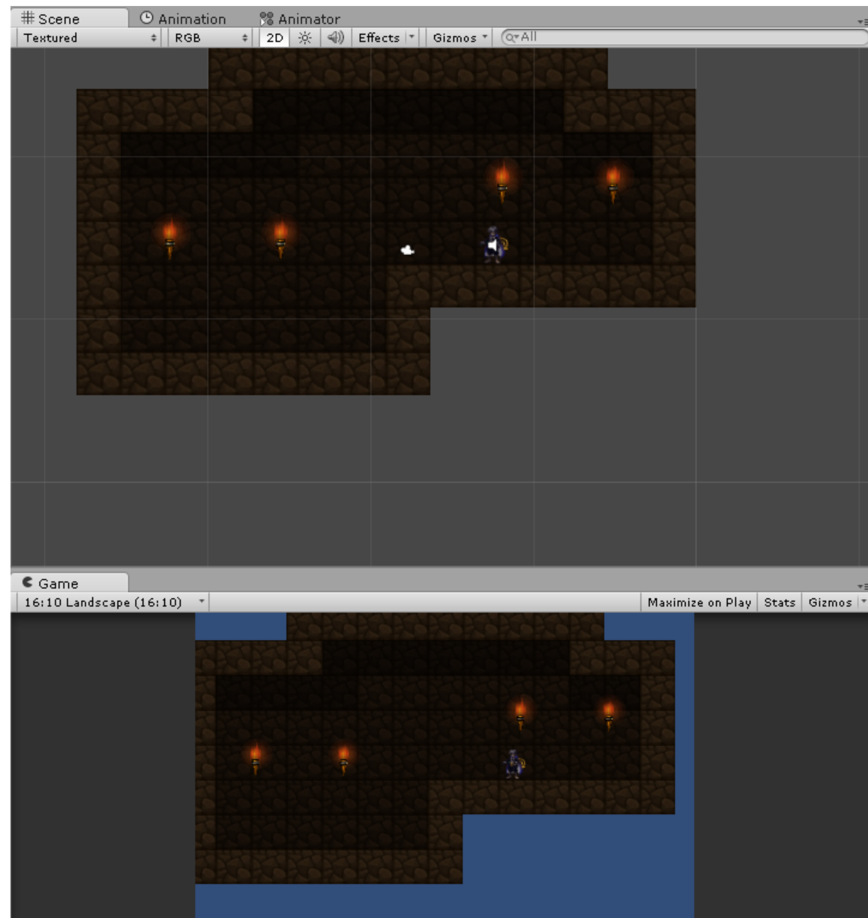

Figura 24: Cenário com vários elementos

2.3 Trabalhos relacionados

Existem diversos editores de cenários para Unity. A maioria deles voltado para a criação de cenários em 3D. Alguns dos editores mais conhecidos para Unity são logo abaixo.

TileEditor (JOHNSTON, 2013) é um editor usado para a criação de jogos em 3D. Ele possui diversas funcionalidades para facilitar o desenvolvimento dos cenários, como por exemplo o auto-alinhamento dos prefabs. No entanto, o TileEditor deixa um pouco a desejar em sua pré-visualização dos prefabs. Os prefabs a serem escolhidos para o cenário são divididos apenas por nome, o que dificulta a sua localização. Talvez para um projeto pequeno isso não seja um problema, mas para um projeto onde existe centenas ou milhares de prefabs fica totalmente inviável localizar novos elementos. Além disso, o TileEditor é pago.

proTile Map Editor[1] é bem mais completo. Diferente do TileEditor, ele possui uma pré-visualização dos prefabs que estão disponíveis, o que torna a usabilidade muito melhor. Ele também foi desenvolvido para dar suporte a criação de cenários em jogos 3D. O proTile Map Editor, assim como o TileEditor, é pago. Na página inicial do site oficial do proTile Map Editor pode ser encontrado um vídeo de demostração do seu uso.

Outro editor, que foi publicado por Zerofield[2], igual ao primeio editor citado, também se chama Tile Editor. Diferentes dos anteriores, ele é voltado a criação de jogos em 2D. A maior desvantagem dele é não fazer uso de prefabs para criar os cenários. Sendo assim, não é possível montar um cenário completo, mas apenas a parte estática do mesmo pode ser construída. Qualquer objeto que possua funcionalidade terá que ser inserido no cenário manualmente utilizando o editor padrão do Unity. As funcionalidades necessárias podem ser adicionadas nos objetos depois de posicioná-los em cena, o que passa a ser uma tarefa bem mais complicada. O editor é pago e pode ser encontrado na loja do Unity.

Um outro editor para jogos em 2D é um feito foi desenvolvido (BRANICKI, 2013). Diferente do anterior, ele faz uso dos prefabs, assim, eliminando o problemas de construir apenas os cenários estatísticos. Infelizmente, ele tem a mesma falha que o TileEditor, apresentado anteriormente. Ele não disponibiliza uma pré-visualização dos prefabs existentes, mostrando apenas os nomes de cada um deles, o que deixa difícil a organização caso o projeto seja grande. A vantagem deste editor é que ele é gratuito.

Existem outros editores, mas todos eles possuem características semelhantes aos que foram citados. A maioria dos editores são de certa forma similares, mas cada um possui características únicas, que podem ser úteis para alguns projetos e outros não. O trabalho aqui proposto, usará as principais funcionalidades de cada um destes, para tentar fazer um editor ideal para a criação de jogos em 2D.

Uma pesquisa mais detalhada sobre editores de cenários pode ser feita na Asset Store por Tile Editor, lá é possível encontrar vários editores pagos.

Após um estudo comparativo entre diversos editores existentes, foram constatadas lacunas conforme indicadas na tabela 1. Por este motivo, foi decidido desenvolver o TEd2D para eliminar algumas destas lacunas e facilitar a criação de cenários em jogos 2D.

Este trabalho propõe o desenvolvimento de um editor de cenários para o Unity que tem como objetivo facilitar a criação de cenários em jogos 2D. Este editor, chamado TEd2D, tem uma interface onde podemos escolher os objetos que queremos disponibilizar em cena

[1] http://protilemapeditor.com/
[2] https://www.assetstore.unity3d.com/en/#!/publisher/9316

e uma forma de adicioná-los a mesma de maneira rápida e eficiente, sem a necessidade de alinharmos manualmente cada um deles.

Existe também softwares que auxiliam a criação de cenários 2D, um exemplo de software para esse proposito é o Tiled[3], com ele você pode criar seus cenários fora do Unity e importá-lo para dentro do projeto. O problema dessa técnica é que os cenários criados não fazer uso dos prefabs, e você só poderá criar a parte estática do mesmo.

A tabela 1 mostra algumas das características existentes nos editores estudados.

Tabela 1: Comparação entre editores de cenários para Unity

Características	TileEditor	proTile	Zerofield	Branicki
Uso de Prefabs	Sim	Sim	Não	Sim
Pré-Visualização	Não	Sim	Sim	Não
Ajuste de Visualização	Não	Sim	Não	Não
Gratuito	Não	Não	Não	Sim
Auto-Ajustar	Sim	Sim	Não	Sim
Grid (On/Off)	Sim	Não	Sim	Sim
Color Grid	Não	Não	Não	Sim
Divisão de TileSets	Sim	Sim	Sim	Sim
Cenários 3D	Sim	Sim	Não	Não

[3]http://www.mapeditor.org/

3 Editor TEd2D

Talvez para alguns jogos não seja preciso um editor de cenários pois nem todos os jogos possuem cenários grandes. Mas para um grande número de jogos de plataforma, montar um cenário requer muita atenção.

Para podermos conceber o editor de cenários TEd2D a fim de contribuirmos neste contexto, a interface do *Unity* precisa ser alterada. Isso é possível utilizando uma biblioteca chamada *UnityEditor*. Ela nos permite alterar a interface do *Unity*, adicionando, removendo ou alterando elementos gráficos existentes nele. O desenvolvimento desta interface será guiado pelas metas de usabilidade definidas por (ROGERS; PREECE, 2013)

3.1 Concepção do TEd2D

A concepção de um editor de cenários necessita que diversas tarefas sejam executadas. A seguir, será mostrado o passo-a-passo de como o editor TEd2D foi desenvolvido.

3.1.1 Grid

Para o TEd2D se tornar mais intuitivo, foi projetada uma Grid para dividir a cena e facilitar o posicionamento dos objetos. Essa Grid permite que o desenvolvedor possa saber exatamente onde o objeto será posicionado. Ela não faz o uso da biblioteca *UnityEditor* e tem um sistema de auto-ajuste, uma vez que cada prefab possui dimensões diferentes e, consequentemente, precisa de espaços de tamanhos diferenciados no cenário.

Utilizando uma função no Unity que nos permite desenhar linhas na tela, linhas verticais e horizontais foram desenhadas para obter a formação de uma Grid.

Através do evento *OnDrawGizmos* o método *Gizmos.draw* do Unity é usado para desenhar cada uma das linhas necessárias. Para desenhar as linhas na vertical e na horizontal, foram utilizados dois laços (*for*) (estrutura de repetição usada em linguagens de

programação). A condição de parada destes laços é controlada por dois valores referentes a altura e a largura da grid. Esses valores poderão ser customizados, e assim, permitir o uso de *Prefabs* de todos os tamanhos. Ou seja, de acordo com o prefeb utilizado, os valores de altura e largura da Grid são modificados.

Uma outra variável é usada para definir as cores das linhas. Isso será útil dependendo da cor do cenários que estamos construindo. Por exemplo, caso as linhas usadas possuírem a cor branca e o background do cenário também seja desta mesma cor, ficará difícil das linhas serem visualizadas. Por isso , um controle de variação da cor foi inserido na interface do editor para permitir ser alterado da melhor forma.

Com as variáveis necessárias, a *Grid* pode ser desenhada na tela. No entanto, apenas com a visualização da Grid não podemos fazer muita coisa. Por isso uma outra classe é responsável pelo controle do alinhamento para permitir alinhar cada Prefab exatamente no centro de cada bloco da *Grid*.

A figura 25 mostra o resultado obtido depois da Grid criada. Ela ainda não possui funcionalidade, pois apenas desenha as linhas na tela do editor do Unity.

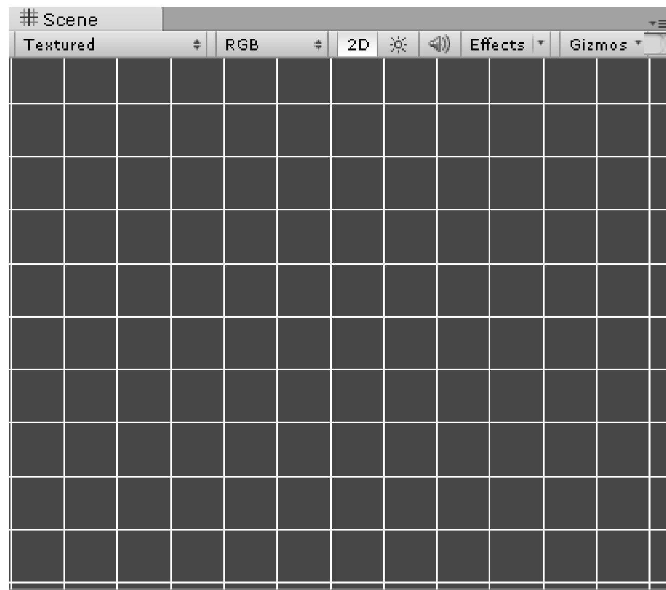

Figura 25: Grid

3.1.2 Grid Editor

Uma vez a *Grid* criada, os objetos precisam que ser instanciados em cada espaço. Para isso, um novo *Script* fará o controle dos posicionamentos. O Script *GridEditor* será responsável por criar os objetos na cena apenas selecionando-o e clicando em um local da

Grid. Ele é o script principal de todo o editor de cenários. O evento *OnSceneGUI* fará esse controle, sendo chamado quando estivermos interagindo com o editor de cenários. Já que este evento ocorre durante a interação do usuário com o editor, é possível inserir comandos de instanciação de objetos durante essa interação. Isso faz com que os prefabs possam ser criados durante o uso do editor.

O *GridEditor* é a classe principal do editor de cenários. Ela fará uso da classe *Grid* para adquirir os valores ideal para posicionar os prefabs. Na sua implementação, apenas uma única variável do tipo Grid será usada para o armazenamento dos valores necessários para a criação dos objetos em seus devidos lugares.

Lembrando que a Classe *GridEditor* tem que fazer uso da biblioteca *UnityEditor*, pois ela modificará os elementos gráficos na interface do Unity. No evento *OnEnable*, que é ativado logo no início da sua execução, é alocado o valor da variável grid. A Grid que estará sendo mostrada no Editor será atribuída a essa variável.

Agora podemos controlar os valores da *Grid* dentro da classe *GridEditor*, e com isso fazer uso dos seus valores para instanciar os objetos de acordo com cada bloco. Dentro da classe GridEditor,, também são inseridos os comandos de execução. Estes comandos serão utilizados para selecionar os objetos na nova interface e colocá-los em cena. Um exemplo de uso detalhado será mostrado mais a frente quando o editor estiver sido totalmente apresentado.

O próximo passo consiste em criar o método *createTileSet*, além de uma classe para representar o *tileSet*. O GridEditor fará uso da classe de TileSet para controlar os objetos que poderão ser criados com o editor. O interessante em criar tileSets é que podemos dividir os elementos que serão criados em Tiles diferentes, e assim facilitar a busca por novos elementos.

3.1.3 TileSet

TileSet é uma classe responsável pelo armazenamento dos objetos/prefabs que podem ser criados. Ela é uma classe simples, porém essencial para o funcionamento de todo o sistema. A sua composição consiste apenas em um vetor de objetos de tamanho customizável para podemos inserir o número de elementos que desejarmos em seu conjunto de objetos.

O método *createTileSet* é responsável por modificar a interface do Unity adicionando um novo elemento no menu de itens. O menu de itens é o mesmo utilizado para criar os

scripts (ver figura 26). Nele, aparecerá uma nova opção para permitir a criação de um novo TileSet. Para isso é utilizada a instrução: *[MenuItem("Assets/Create/TileSet")]*, que adicionará um elemento para esse caminho ficando da forma mostrada na figura 27.

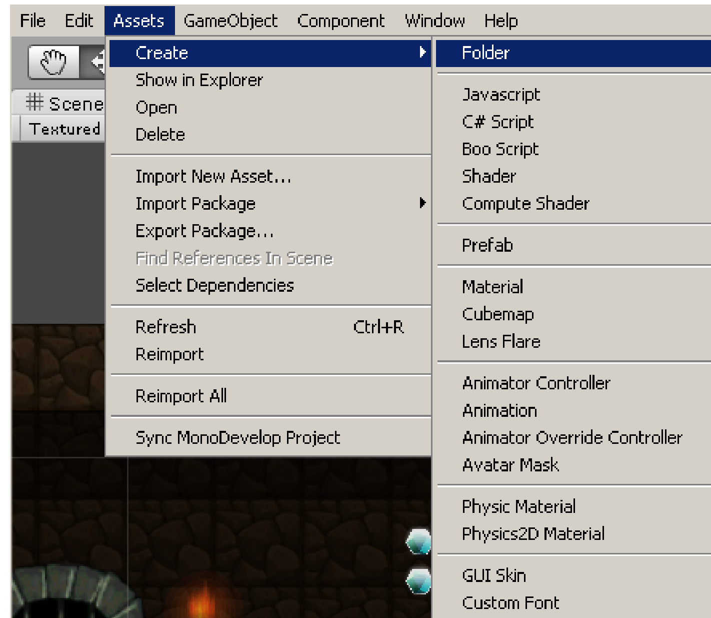

Figura 26: Menu de itens padrão do Unity

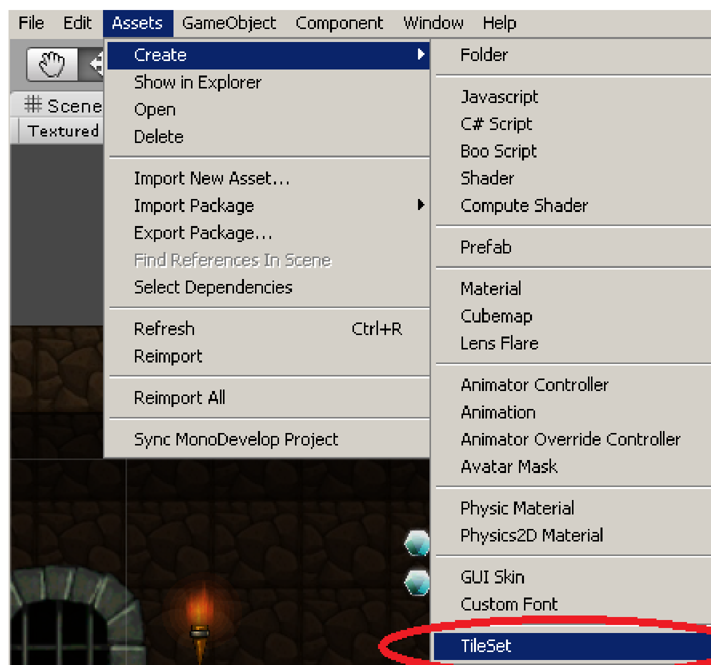

Figura 27: Menus de itens do Unity com a adição da criação dos TileSets

Vários *TileSets* poderão ser criados para facilitar o uso do TEd2D. Por exemplo, um *TileSet* específico para os itens e outro para as plataformas. O emprego de apenas um

TileSet resultaria em muitos objetos aparecendo no editor, tornando assim a criação de cenas mais complicada.

3.1.4 Inspector GUI

A interface que será modificada é a Inspector GUI (ver figura 28). Essa interface é responsável por mostrar as informações de cada objeto. Alguns exemplos de informações contida nela são: Transform, RigidBody2D e Scripts. Nessa interface, será inserido os elementos visuais para facilitar as escolhas dos prefabs.

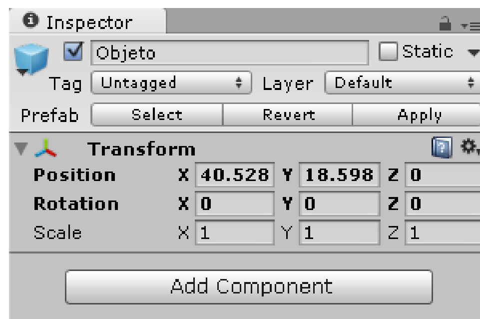

Figura 28: Inspector GUI

O uso do evento *override OnInspectorGUI* elimina todos os elementos da interface InspectorGUI e permite que possa ser adicionados novos elementos na sua interface. Primeiramente será adicionado dois novos elementos, o *TileSet*, que será usado para selecionar o TileSet e o *Prefab* que mostrará qual prefab foi selecionado.

Para estes dois elementos serem adicionados será preciso fazer uso da Classe *EditorGUILayout*, e instanciar um *ObjectField* para a variável *TilePrefab* e outro pra o *TileSet*.

A figura 29 mostra o resultado obtido depois das modificações.

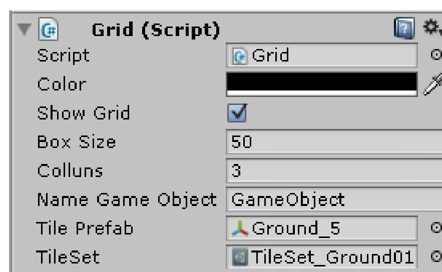

Figura 29: Inspector

Os seis primeiros elementos da Figura 29 são configurados automaticamente com o uso da instrução *base.OnInspectorGUI*. Essa instrução permite mostrar as variáveis que

foram deixadas públicas. Elas também podem ser customizadas para permitir alterações na interface do editor que está sendo criado. Isso faz com que o editor fique flexível para algumas mudanças. Por exemplo, se quisermos visualizar a Grid, bastará apenas selecionar o elemento Show Grid.

Os dois últimos elementos da Figura 29 são os que foram criados com o *EditorGUILayout*. Isso permite que o usuário possa modificar o *TileSet* que ele estiver usando. Caso ele tenha vários TileSets, será possível selecioná-los através dessa opção.

3.1.5 Seletor de Prefabs

O seletor de *prefabs* irá mostrar todos os objetos que estão contidos no *TileSet* selecionado. Ele mostrará estes objetos divididos em blocos com o seu Sprite principal desenhado. Isso facilitará a escolha de cada *prefab*.

O TileSet que estiver selecionado poderá possuir zero ou mais Prefabs. Caso o TileSet possua um ou mais prefabs, um botão será criado utilizando o comando *GUILayout.Button* para cada um deles. Para cada um dos botões criados na interface do Inspector, o Sprite do prefab selecionado é renderizado. Isso é feito para facilitar a escolha do prefab em questão. A altura e largura desses botões poderão ser customizadas na própria interface no campo Box Size. Caso o TileSet contenha muitos prefabs, talvez alguns deles não apareçam na interface. Mas com um ajuste do tamanho dos botões, essa questão da visualização dos mesmos poderá ser regularizada.

Os botões serão alinhados em linhas e colunas para facilitar a visualização. O número de colunas também pode ser alterado na interface através do campo Colluns.

Um botão extra foi criado para auxiliar a criação de um Game Object vazio. Caso seja necessário criar algum prefab composto, esse botão poderá ser útil.

A figura 30 apresenta exemplos de 3 interfaces com diferentes TileSets. Cada um desses botões consiste em um elemento que pode ser escolhido e inserido em cena através de um clique.

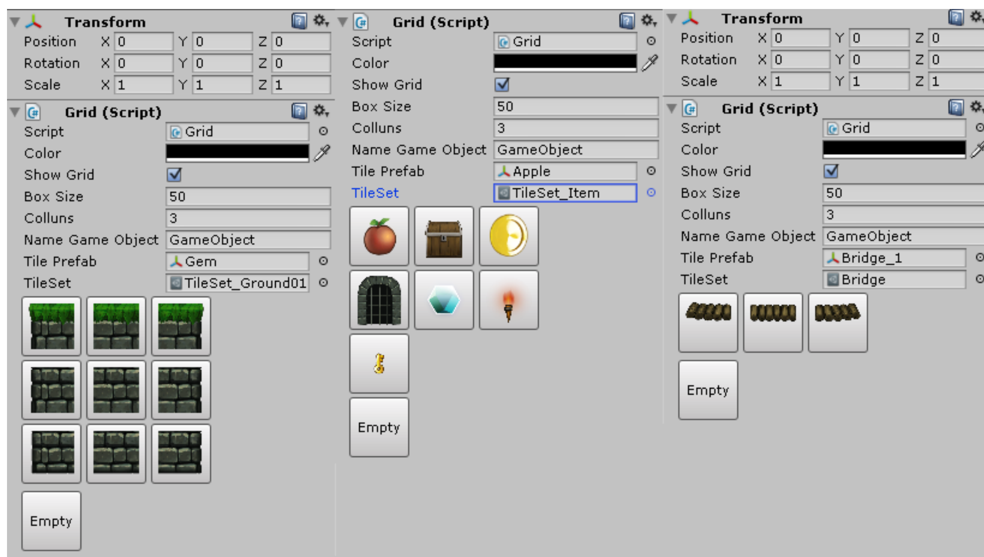

Figura 30: TileSets

Para finalizar a interface do editor de cenários foi adicionado um Label e um Box para mostrar qual Prefab está selecionado a cada momento. O Label contém o texto "Tile Selected: Nome do Prefab". A figura 31 mostra a interface completa do editor.

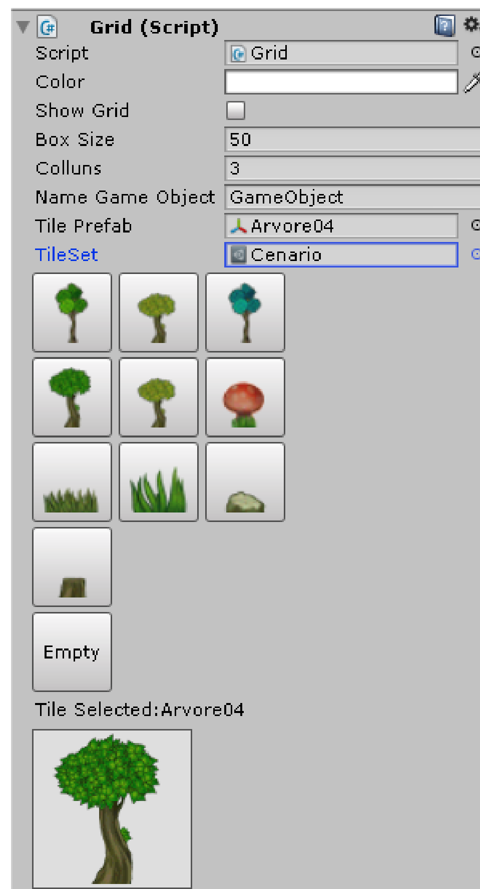

Figura 31: Editor Final

3.2 Utilizando o TEd2D

Nesta etapa, o TEd2D está pronto e será usado para criar um cenário simples fazendo uso dos prefabs. Será mostrado abaixo o passo-a-passo para utilizar o TEd2D.

3.2.1 Passo 1 - Importando o Package

A primeira atividade a ser efetuada é importar o editor para o projeto. É possível importá-lo utilizando a interface do Unity na aba Assets/Import Package/Custom Package...(Ver figura 32). Em seguida, selecionar o Package do editor que pode ser encontrado atráves do link: `http://migre.me/q9yo0`.

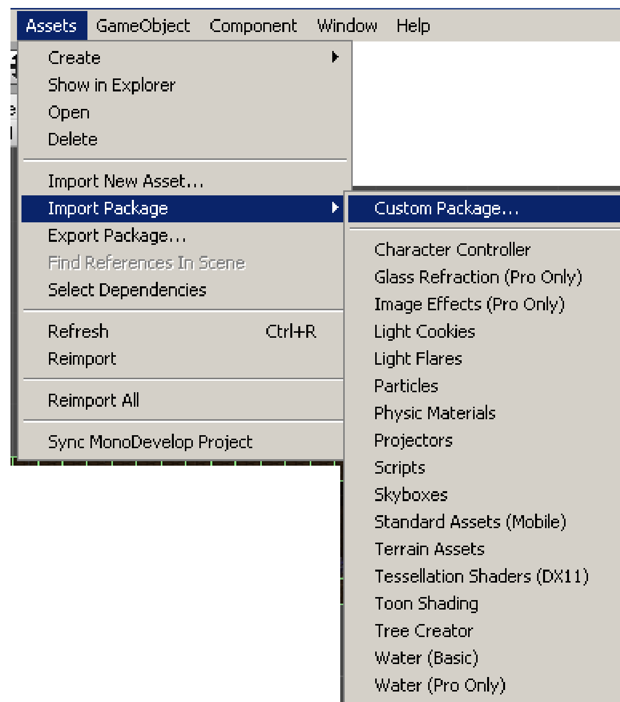

Figura 32: Importando o editor

Quando o Package for selecionado, uma nova aba contendo os elementos do editor será mostrada (ver figura 33). Ele contém alguns Sprites de exemplo, que podem ser eliminados caso não sejam necessários ao projeto. Para isso, basta desmarcar a opção referentes aos sprites.

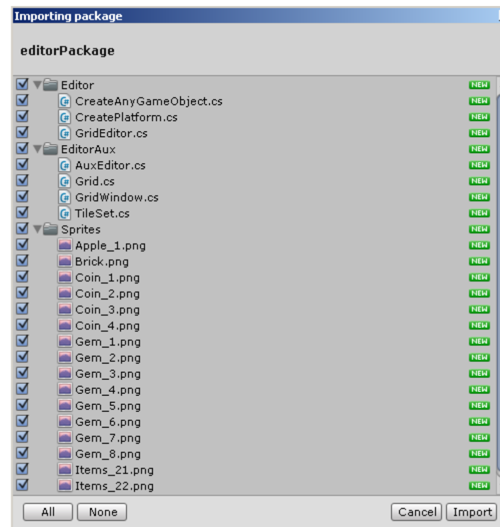

Figura 33: Importando o editor

A figura 34 mostra as pastas já presentes no projeto contendo o TEd2D. A partir deste ponto, o editor faz parte do projeto e pode ser utilizado. Mas antes de usar, você precisa entender como utilizar cada elemento.

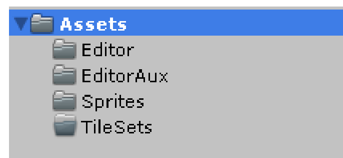

Figura 34: Elementos do TEd2D

Para usar o TEd2D é preciso ter os prefabs a serem inseridos no projeto. Não necessariamente todos os prefabs precisam estar prontos pois poderemos ir adicionando novos no decorrer do projeto.

3.2.2 Passo 2 - Criando os TileSets

Depois de alguns prefabs criados é preciso criar os TileSets para armazena cada um dos prefabs. Lembrando que vários TileSets podem ser criados para melhor organizar o projeto.

Para criar um TileSet, existe uma pasta vazia chamada TileSets. Precisamos selecionar essa pasta e dentro dela clicar com o botão direito do mouse, seguindo o caminho create/TileSet (ver 35).

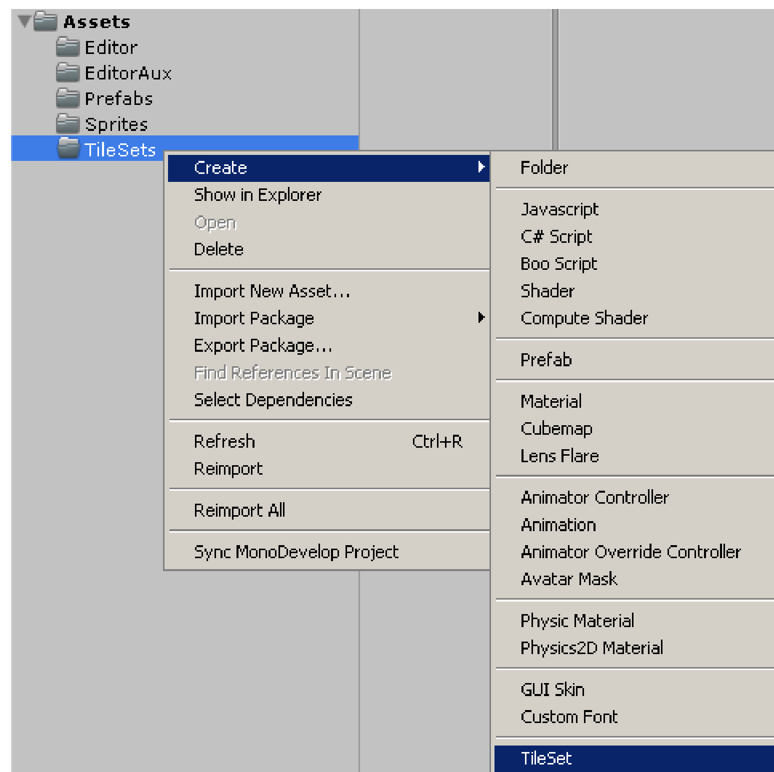

Figura 35: Criando um TileSet

Dentro da pasta TileSets, será criado um novo TileSet que poderá ser renomeado para ficar mais intuitivo. Por exemplo, se este for um TileSet que armazenará itens, pode ser renomeado para Tile_Item. Mas isso fica a escolha do usuário.

Selecionando o TileSet podemos alterar a quantidade de valores de prefabs que ele armazena (ver figura 36). Agora, basta arrastar um prefab qualquer para um dos elementos e preencher o TileSet com os elementos que deseja armazenar neles para depois fazer uso no Editor.

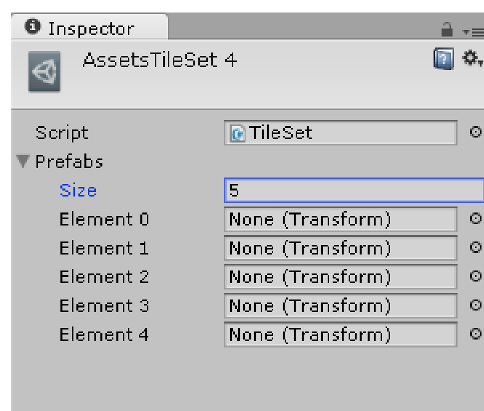

Figura 36: Criando um TileSet

3.2.3 Passo 3 - Criando o editor

Depois de criar os TileSets necessários, o editor precisa ser instanciado. Para isso, precisamos ir ao Menu GameObject/Create Editor (ver figura 37). Um novo objeto de jogo será criado com as propriedades de edição de cena contido nele. Podemos então selecionar este objeto e observar a sua barra Inspector (ver figura 38).

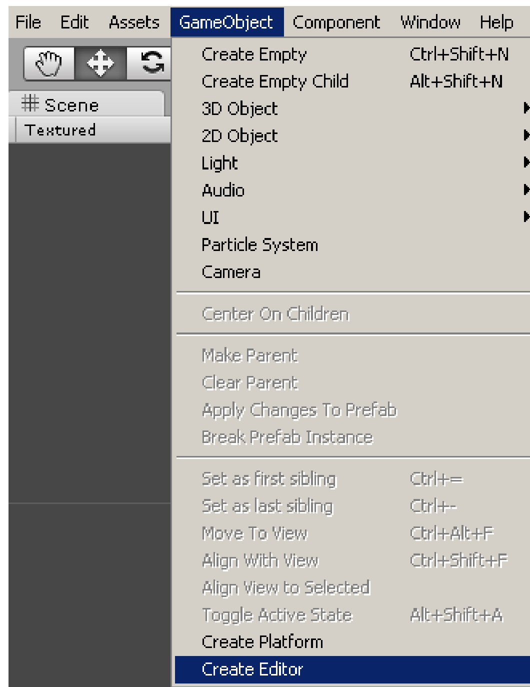

Figura 37: Caminho para criar o TEd2D

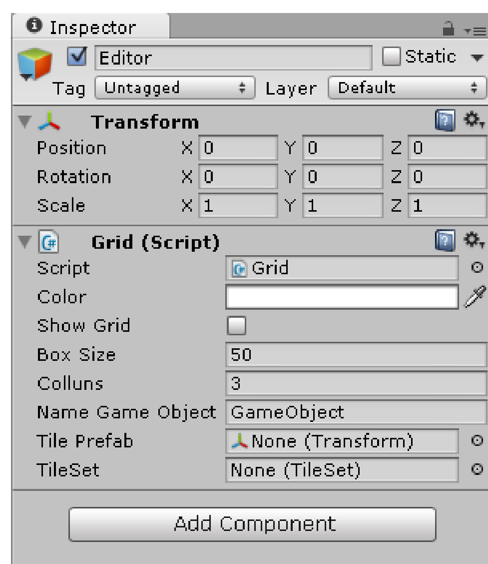

Figura 38: TEd2D sem nenhum TileSet Selecionado

Vejamos que o elemento TileSet está vazio. Entretanto, como já foram criados alguns

TileSet, podemos clicar e selecionar algum deles no projeto.

Para visualizar a Grid no cenário, basta marcar a opção Show Grid. Dependo da cena, a Grid pode atrapalhar um pouco a visualização.

3.2.4 Passo 4 - Criando uma Cena com o TEd2D

Depois de selecionar o TileSet, já será possível criar um cenário com os prefabs existentes. Para isso, precisamos clicar com o botão esquerdo em qualquer objeto e em seguida clicar na cena. O objeto será criado no local onde o clique foi efetuado. Se quiser criar vários objetos do mesmo tipo, basta segurar o clique e arrastar em qualquer direção. Para deletar um objeto de do cenário, basta clicar sobre ele com o botão direito do mouse.

Depois disso o trabalho se torna mais rápido e simples, pois a medida que o TileSet necessário para cada parte do cenário estiver selecionado, basta clicar na View Scene do Unity e selecionar o posicionamento desejado. A figura 39 mostra um pequeno exemplo de uma plataforma criada usando o TEd2D.

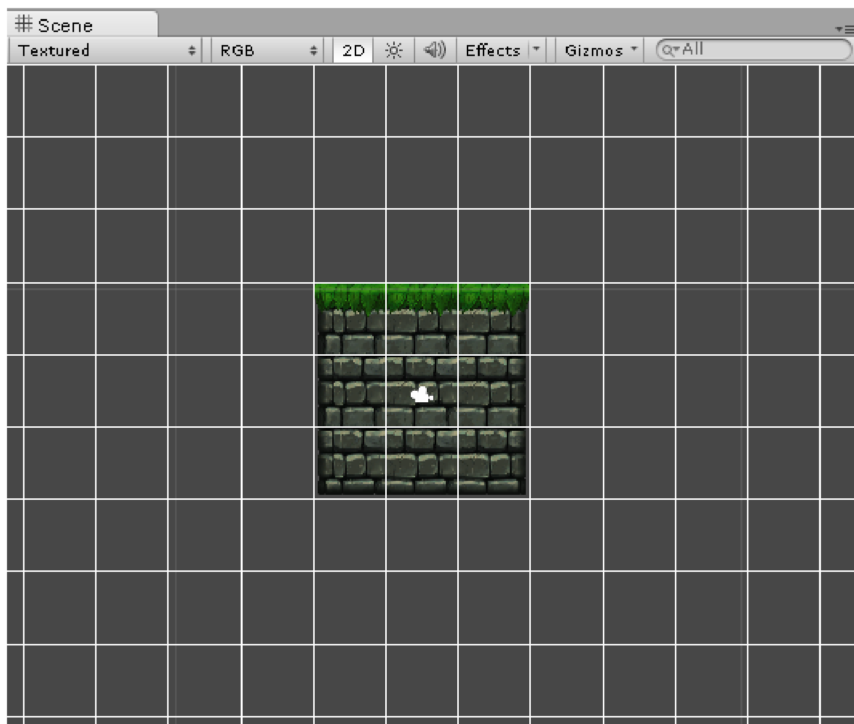

Figura 39: Plataforma simples criada com o TEd2D

A tabela 2 mostra que o TEd2D eliminou as desvantagens existentes nos editores estudados, com exceção do suporte ao desenvolvimento de cenários em 3D.

Tabela 2: Tabela de comparação

Características	TEd2D	TileEditor	proTile	Zerofield	Branicki
Uso de Prefabs	Sim	Sim	Sim	Não	Sim
Pré-Visualização	Sim	Não	Sim	Sim	Não
Ajuste de Visualização	Sim	Não	Sim	Não	Não
Gratuito	Sim	Não	Não	Não	Sim
Auto-Ajustar	Sim	Sim	Sim	Não	Sim
Grid (On/Off)	Sim	Sim	Não	Sim	Sim
Color Grid	Sim	Não	Não	Não	Sim
Divisão de TileSets	Sim	Sim	Sim	Sim	Sim
Cenários 3D	Não	Sim	Sim	Não	Não

4 Metodologia Experimental

Para verificar se o TEd2D estava atendendo aos objetivos propostos de facilitar a construção de cenários para jogos em 2D, era necessário experimentá-lo com alguns desenvolvedores de software familiarizados com Unity e alguns não-desenvolvedores.

Os experimentos com TEd2D ocorreram em dois momentos. Primeiramente, tendo como objetivo a tarefa de construção de um cenário simples, no qual 10 participantes estiveram envolvidos. Em seguida, tendo como objetivo a tarefa de construção de fases completas de 3 jogos, no qual o autor deste trabalho esteve envolvido.

4.1 Construção de cenários simples

Nestes experimentos, um grupo de 10 pessoas foi submetido ao procedimento de criação de um cenário simples de duas formas: utilizando o TEd2D e sem utilizá-lo. Para tentar minimizar o bias que poderia existir pelo fato do participante já construir o cenário demandado antes de usar o editor, e assim diminuir o tempo médio de construção do mesmo não somente devido ao uso propriamente dito do TEd2D, mas pelo fato de já conhecer o cenário, decidimos variar a ordem da tarefa. Portanto, alguns participantes iniciaram o experimento sem utilizá-lo e em seguida utilizando-o. Já outros participantes realizaram as tarefas na ordem inversa.

Os experimentos ocorreram da seguinte forma: uma imagem impressa contendo um modelo de cenário em 2D (ver figura 40) foi entregue a cada participante para eles construírem o mesmo cenário com Unity das duas formas: com o TEd2D e sem ele. Um cronômetro foi usado para contabilizar o tempo que eles gastaram para construir cada uma das versões do cenário.

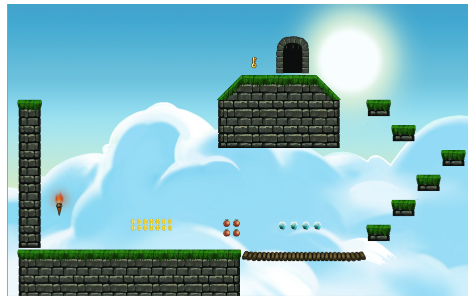

Figura 40: Imagem do cenário modelo utilizado para os experimentos. O cenário contém prefabs de plataformas, moedas, itens, etc.

Alguns dos participantes dos experimentos foram convidados a construir o mesmo cenário uma terceira vez, caso tivessem efetuado as tarefas na seguinte ordem: com TEd2D e sem o TEd2D. Isso só foi feito quando o tempo gasto para a construção do cenário ficou muito próximo nos dois casos. De forma sistemática, na terceira vez os participantes sempre conseguiram reduzir bastante o tempo gasto na construção. Este participantes, em particular, já estavam acostumados com a criação de cenários com Unity, por isso conseguiram construir o cenário rapidamente sem a ajuda do TEd2D. Porém, depois de se acostumar com o TEd2D, superaram as nossas expectativas.

Tabela 3: Tempo gasto na seguinte ordem de execução da tarefa: sem TEd2D e com TEd2D

Nome	Sem TEd2D	Com TEd2D	Usuário do Unity
Tester01	8:17	4:23	Sim
Tester02	12:50	3:48	Sim
Tester03	15:43	6:06	Não

Tabela 4: Tempo gasto na seguinte ordem de execução da tarefa: com TEd2D e Sem TEd2D

Nome	Com TEd2D	Sem TEd2D	Com TEd2D	Usuário do Unity
Tester04	6:54	7:48	3:25	Sim
Tester05	4:26	7:49	-	Sim
Tester06	6:43	12:33	4:22	Não
Tester07	7:32	16:23	-	Não
Tester08	8:05	11:21	-	Não
Tester09	2:32	11:00	-	Sim

O gráfico da Figura 41 mostra o tempo gasto em segundos com cada um dos participantes do experimento. Em nenhum dos casos, os participantes demoraram mais para construir o cenário com o TEd2D. Independente da ordem em que eles foram submetidos ao teste, sempre o cenário foi construído mais rapidamente com o uso do TEd2D.

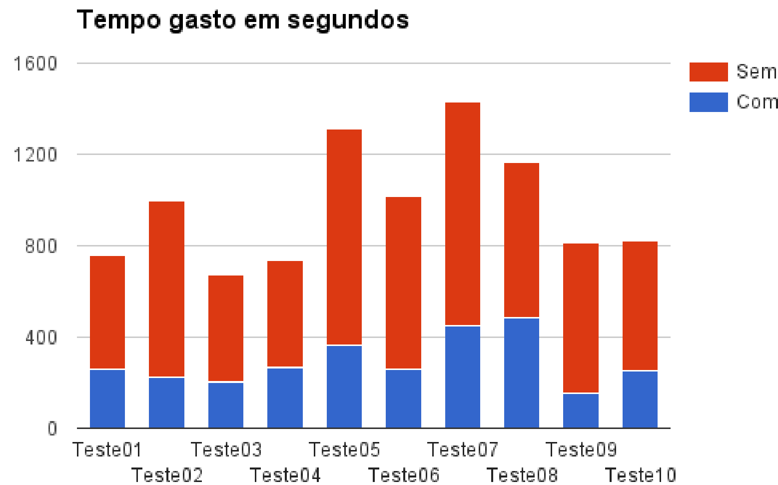

Figura 41: Gráfico comparativo do tempo gasto, em segundos, para construção do cenário modelo com o TEd2D e sem o TEd2D.

De acordo com o gráfico mostrado na figura 41, temos um ganho significativo, porém são apenas segundo. Quando o editor foi utilizados para criação de jogos completos, o que será mostrado mais a frente, o editor se mostrar mais eficiente, reduzindo três dias de trabalho, para apenas uma tarde.

Uma vez terminados os experimentos, um questionário foi aplicado com todos os participantes afim de verificar o impacto do uso do editor de cenários em aspectos tais como usabilidade, eficácia e aprendizagem. Este questionário foi desenvolvido contendo perguntas objetivas com as seguintes possibilidades de escolha: 1) Discordo Totalmente, Discordo, Indiferente/Não sei, Concordo e Concordo Totalmente; e 2) Sim e Não. Os resultados obtidos através deste questionário são bastante positivos, pois todos os participantes deram feedbacks bastante positivos quanto ao uso do TEd2D.

Questão	Sim	Não
Você já desenvolveu algum jogo no Unity	50%	50%
Você conhece algum editor de cenas similar ao usado no experimento?	60%	40%
Você conhece algum Editor de cenas para o Unity semelhante ao usado no experimento?	0%	100%

Figura 42: Lista de questões com possibilidade de respostas Sim ou Não.

Dos 10 participantes nos experimentos, 50% deles responderam já ter desenvolvido anteriormente algum jogo com Unity, os outros 50% não. 60% dos participantes responderam que já conheciam algum editor similar ao TEd2D, mas nenhum respondeu conhecer editores do mesmo tipo específicos para o Unity (ver figura 42).

Nenhum dos participantes conhecia algum editor para o Unity

	Concordo Totalmente	Concordo	Não sei / Indiferent	Discordo	Discordo Totalmente
Você achou a interface do Framework intuitiva?	30%	70%	0%	0%	0%
Você achou fácil encontrar novos elementos para colocar em cena?	30%	60%	10%	0%	0%
É mais fácil localizar um novo elemento com o Framework do que sem ele?	70%	20%	0%	10%	0%

Figura 43: Lista de questões com possibilidade de respostas: concordo Totalmente, Concordo, Não sei/Indiferente, Discordo e Discordo Totalmente

De acordo com os resultados apresentados na figura 43, todos os participantes acharam a interface do TEd2D intuitiva. 90% dos participantes concordaram que é fácil buscar elementos para inserí-los no cenário, nenhum discordando disso. Além disso, 90% dos participantes concordaram que é mais fácil buscar um novo elemento com o TEd2D do que sem ele. Apenas 10% deles discordaram. (ver figura 43)

	Sim	Não
Você aceitaria criar um cenários 10 vezes maior do que o usado no experimento SEM a ajuda do Framework?	0%	100%
Você aceitaria criar um cenários 10 vezes maior do que o usado no experimento COM a ajuda do Framework?	100%	0%

Figura 44: Lista de questões com possibilidade de respostas Sim ou Não.

De acordo com os resultados apresentados na figura 44, nenhum dos participantes topariam construir um cenário 10 vezes maior sem o TEd2D. No entanto, todos eles aceitariam o desafio de construir um cenário 10 vezes maior usando o TEd2D.

	Sim	Não	Depende do Jogo
Depois de conhecer o Framework, você ainda criaria seus cenários sem a sua ajuda?	0%	40%	60%

Figura 45: Lista de questões com possibilidade de respostas Sim, Não ou Depende do jogo.

De acordo com os resultados apresentados na figura 45, 60% dos participantes afirmaram que usariam o TEd2D dependendo do jogo a ser criado, como por exemplo jogos com cenários pequenos, exemplo: o jogo 2048[1]. 40% disseram que não criariam cenários sem o TEd2D após tê-lo conhecido.

[1]http://www.clickjogos.com.br/jogos/2048-flash/

Algumas questões subjetivas também foram feitas, mas sem obrigatoriedade de resposta. Portanto, só alguns dos participantes responderam às mesmas. Estas questões são as seguintes:

- O que você NÃO gostou no TEd2D?
- O que você gostou no TEd2D?
- Você mudaria algo na Interface do TEd2D?
- Deixe alguma sugestão

No geral, a maioria dos participantes nos experimentos não gostou da limitação existente para a movimentação da câmera quando usando o TEd2D. Para mover a câmera, é necessário selecioná-la, diferentemente de quando se usa o editor padrão do Unity.

Todos comentaram que gostaram do procedimento de alinhamento automático do TEd2D, pois não havia necessidade de se preocupar em ficar alinhando cada objeto um a um.

Enfim, um dos participantes fez um comentário sobre o tamanho dos botões. Provavelmente ele não percebeu que os botões são customizáveis, podendo ser redimensionados a qualquer momento.

Os participantes envolvidos nos experimentos foram colegas de curso e familiares, o que pode ter afetado um pouco o resultado dos mesmo. Em trabalhos futuros, os mesmo experimentos serão feito com a ajuda da comunidade do Unity, fazendo com que diversos usuários podem fazer uso da ferramenta.

4.2 Construção de jogos

Para avaliar o TEd2D no processo de construção de cenários mais complexos, foram desenvolvidas fases completas de três jogos: Mathmare, TryZ e Enigma.

4.2.1 Mathmare

O primeiro dentre os três jogos construídos nos experimentos foi um jogo educacional voltado ao ensino de matemática, chamado Mathmare, que tinha como objetivo inserir desafios matemáticos contendo conceitos do ensino médio para ser aplicado para os alunos

da disciplina Resolução de Problemas Matemáticos para Tecnologia da Informação do Bacharelado em Tecnologia da Informação (BTI) da UFRN.

A história do jogo gira em torno do personagem Dave Laze, que sempre foi viciado em jogos. Porém, Dave nunca considerou matemática uma coisa útil. Até que um dia, ele ganhou de um estranho um novo jogo chamado MathMare, do qual nunca tinha ouvido falar anteriormente. Dave, com seu instinto curioso, decidiu jogar para ver como era o jogo. Porém algo inesperado aconteceu. Dave desmaiou e acordou em um lugar estranho, quando uma voz começou a falar: "você está preso dentro de MathMare... a única maneira de sair é avançando até chegar ao final do jogo....". Nesse momento aparece uma mensagem em seu celular, de uma pessoa que afirma ser um ex-jogador do MathMare, e que se faz presente para ajudá-lo. Segundo a mensagem, o jogo foi feito para ser impossível de se completar, pois o objetivo era fazer com que ninguém conseguisse se salvar. A única forma de conseguir sair do jogo seria trapaceando. Para isso, Dave precisaria hackear o sistema do jogo com o seu o próprio celular. É a partir daí que ele começa a perceber que nos jogos que tanto gostava, existia muito mais matemática do que ele imaginava. Um vídeo contendo o trailer de Mathmare, pode ser acessado em https://www.youtube.com/watch?v=kKfqTnBSo-U

O primeiro cenário de Mathmare foi desenvolvido inicialmente sem o auxílio do editor TEd2D. Em seguida, para ser testado nas 4 turmas da disciplina do BTI citada acima, no semestre 2015.1, foi desenvolvida uma nova fase do jogo, usando o TEd2D, para ser jogada em 45 minutos.

Um fato importante notado durante a execução do jogo em sala de aula é que uma pequena parte inicial do cenário, na qual não foi feito uso de TEd2D, apresentou problema de alinhamento na plataforma, bloqueando a passagem do personagem. Isso ocorreu devido ao editor padrão do Unity deixar o usuário livre para posicionar os elementos do cenário, o que pode inserir problemas de imprecisão no alinhamento. Já com o TEd2D não há esse problema, pois todos os elementos são disponibilizados de forma uniforme.

Durante o desenvolvimento da fase do jogo a ser aplicada para os alunos do BTI, foi percebido que construir cenários não era mais uma tarefa enfadonha. Isso foi notado, porque antes já havíamos criado outras fases para Mathmare, e sempre demorava muito para um simples cenário ser construído. Com o TEd2D, durante uma tarde, foi possível criar todo o cenário da fase experimentada com os alunos e deixar o jogo totalmente funcional, tarefa que não era possível antes da utilização do mesmo. Sem fazer uso do TEd2D, seria preciso em média 3 dias para completar o cenário feito para os alunos do

BTI.

Mathmare foi desenvolvido em conjunto com o aluno Lucas Tomé Avelino Câmara, que está fazendo seu TCC em Ciência da Computação no tema do desenvolvimento de desafios matemáticos para este jogo, também tendo utilizado o TEd2D para construir seus cenários. A figura 46 mostra uma parte do cenário do jogo Mathmare.

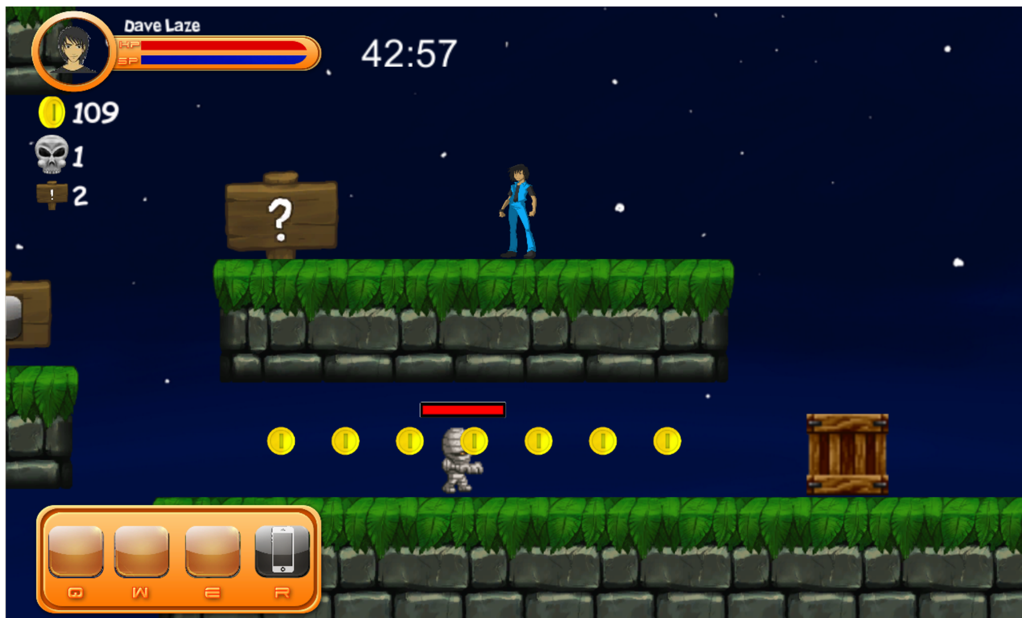

Figura 46: Cena do jogo Mathmare utilizado na disciplina

4.2.2 TryZ

O segundo jogo construído com o auxílio do TEd2D foi o TryZ, publicado na Play Store, desenvolvido pelo autor deste trabalho.[2]O objetivo deste jogo consiste em juntar as letras iguais para formar a próxima letra na sequência do alfabeto, até conseguir chegar na letra Z. A figura 47 ilustra uma cena do jogo TryZ.

Como o jogo TryZ possui apenas um cenário, o TEd2D foi utilizado para verificar se era viável utilizá-lo em um jogo pequeno, e como o cenário do TryZ possui uma Grid onde as letras ficarão dispostas, o TEd2D foi utilizado para fazer essa Grid, tarefa que demorou menos de um minuto. Sem o TEd2D, provavelmente eu teria utilizado algum script para criar a Grid para o cenário, e para fazer um script desses, gastariam em média uns dez minutos.

[2]https://play.google.com/store/apps/details?id=com.WellPlayStudios.Z

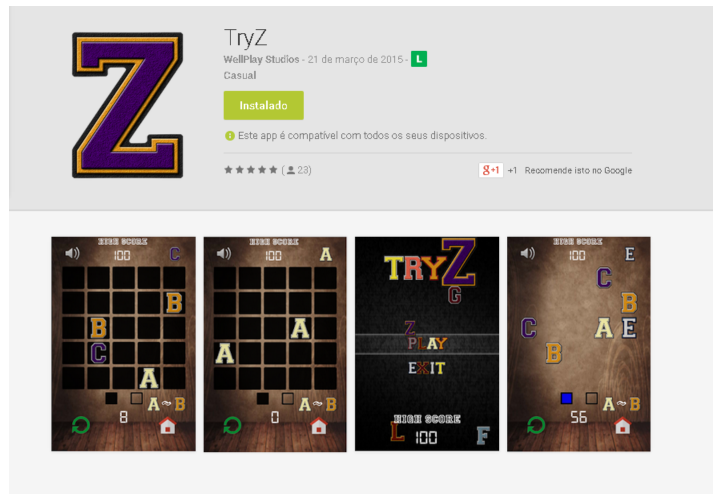

Figura 47: Exemplo de tela do jogo TryZ

4.2.3 Enigma

O terceiro jogo construído com o auxílio do TEd2D foi o Enigma (ver Figura 48), também desenvolvido pelo autor deste trabalho. O Enigma ainda se encontra em desenvolvimento. Ele possui diversos cenários bem mais complexos que o TryZ. Porém, com a ajuda do TEd2D, os cenários estão sendo feitos de forma rápida, permitindo assim nos concentrarmos nas mecânicas existentes no jogo. O jogo tem como objetivo encontrar em cada fase um chave, que pode ou não estar escondida dependendo do nível de dificuldade da fase. Cada fase tem um cenário diferente das outras, fazendo com que exista muito trabalho a ser feito. O TEd2D facilitou muito isso, pois 12 fases distintas foram feitas em apenas 1 dia.

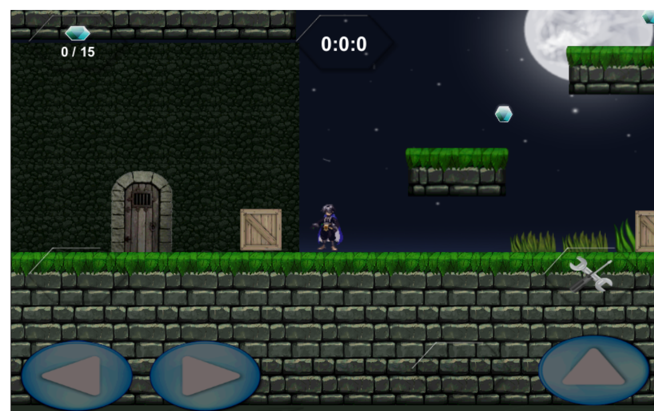

Figura 48: Exemplo de tela contendo parte do cenário de uma fase do Enigma

5 Considerações finais

O presente trabalho consistiu no desenvolvimento de um editor para o Unity, tendo como objetivo facilitar a criação de cenários para jogos em 2D. Com esse propósito, foi feito um estudo comparativo com outros editores de cenários existentes. Alguns dos editores de cenários estudados servem para criação de cenários de jogos em 3D, outros para jogos em 2D. Como resultado deste estudo, foram identificadas diversas lacunas nos editores existentes, gerando a necessidade de desenvolvermos um editor que pudesse suprir todos os requisitos desejados.

Os resultados obtidos através dos experimentos realizados são bastante positivos pois conseguimos demonstrar que usuários do Unity, assim como não usuários, conseguiram construir cenários com o editor TEd2D de forma bem mais rápida que sem o uso dele. Além disso, os cenários construídos se mostraram mais organizados quando fazendo uso do TEd2D, na maioria dos casos o tempo gasto sendo reduzido pela metade. Três jogos também foram desenvolvidos com o uso do editor proposto, mostrando assim que é possível utilizá-lo eficazmente em um projeto. Durante a criação destes jogos, ficou claro que sem o TEd2D não seria possível desenvolver todos eles em tão pouco tempo.

Em trabalhos futuros, o TEd2D poderá ser expandido, adicionando algumas novas funcionalidades para auxiliar na criação de eventos, além da construção de cenários. Uma outra evolução seria a criação de um editor híbrido, que daria suporte tanto para jogos em 2D como em 3D.

Referências

BITTENCOURT, F. S. O. J. R. Motores para criação de jogos digitais: Gráficos, Áudio, interação, rede, inteligência artificial e física. 2006.

BRANICKI, D. *How to Add Your Own Tools to Unity?s Editor*. 2013. Oct., 2011. Disponível em: <http://code.tutsplus.com/tutorials/how-to-add-your-own-tools-to-unitys-editor--active-10047>. Acesso em Abril 21, 2015.

CLUA E., B. J. Desenvolvimento de jogos 3d: Concepção, design e programação. In: *Anais da XXIV Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação*. [S.l.: s.n.], 2005. p. 1313–1356.

GREGORY, J. *Game engine architecture*. [S.l.]: CRC Press, 2009.

JOHNSTON, C. R. *New Tool: TileEditor*. 2013. Sep., 2013. Disponível em: <http://unitypatterns.com/new-tool-tileeditor/>. Acesso em Abril 21, 2015.

JORDAO, F. *A nova guerra entre motores gráficos de games*. jul. 2013. Jul., 2013. Disponível em: <http://www.tecmundo.com.br/video-game-e-jogos/42657-a-nova-guerra-entre-motores-graficos-de-games-video-.htm>. Acesso em Maio 05, 2015.

KOBASHIKAWA, D. Estudo comparativo de ferramentas para motores de jogos rpg. 2007.

ROGERS, H. S. Y.; PREECE, J. *Design de Interação*. 3th. ed. [S.l.]: Bookman, 2013.

WIKIPEDIA, a. e. l. *Unity*. 2015. Mai., 2015. Disponível em: <http://pt.wikipedia.org/wiki/Unity/>. Acesso em 28 de Maio de 2015.