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

[1]Isaac Beserra [1]Lucas Câmara [2]Charles Madeira [3]Rogério Tavares

[1]Universidade Federal do Rio Grande do Norte, Departamento de Informática e Matemática Aplicada, Brasil
[2]Universidade Federal do Rio Grande do Norte, Instituto Metrópole Digital, Brasil
[3]Universidade Federal do Rio Grande do Norte, Departamento de Artes, Brasil

Resumo

Motores de desenvolvimento de jogos têm ganhado muito espaço nestes últimos anos, ajudando a facilitar de forma considerável o desenvolvimento destas aplicações. Unity é um exemplo de motor que está tendo grande sucesso atualmente no mercado. 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 auxilia na etapa de posicionamento dos diversos objetos no mundo a ser simulado. O presente trabalho propõe um editor para auxiliar a construção de cenários 2D com Unity através de um processo de distribuição facilitada dos objetos, permitindo assim alavancar bastante a produtividade. Este editor foi experimentado e validado com diversos desenvolvedores e não-desenvolvedores de software. Os resultados obtidos são bastante promissores pois permitem a criação de cenários mais rapidamente.

Contato dos autores:
[1]isaacciencomp@gmail.com
[2]lucas_tac18@hotmail.com
[3]charles@imd.ufrn.br
[4]rogertavares@gmail.com

Palavras-chave:: Motores de Jogos, Unity, Jogos 2D, Editor de Cenários

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].

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]. Possui vários recursos que permitem agilizar o processo de desenvolvimento de jogos, tanto em 2D quanto em 3D. Estes recursos podem ser internos (disponibilizados pelo próprio motor) ou externos (adicionados por desenvolvedores independentes), podendo ser encontrados na loja oficial do Unity (Asset Store).

Mesmo com todas essas 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, 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.

Este artigo introduz o TEd2D, um editor de cenários para jogos em 2D que permite suprir as limitações do Unity pelo fato de facilitar o posicionamento e a organização dos objetos em ambientes de jogos em 2D. Para isto, são adicionados novos elementos gráficos na interface de edição do Unity para permitir 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 é alinhada com os demais objetos, fazendo com que não exista a necessidade de controlar manualmente o posicionamento. Para avaliar a eficácia desta solução, experimentos foram realizados no contexto da construção de cenários de jogos em 2D, obtendo resultados bastante promissores [Beserra 2015].

2 Trabalhos Relacionados

Existem diversos editores de cenários para Unity. A maioria deles voltado para a criação de cenários em 3D. Alguns exemplos mais conhecidos são apresentados a seguir..

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[8]. 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.

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

[6]http://www.rpgmakerweb.com/
[7]https://www.yoyogames.com/studio
[8]Objetos pré-fabricados, utilizados para agilizar o processo de criação do jogo, podendo ser instanciados em qualquer parte do jogo, sem a necessidade de recriá-los

proTile Map Editor[9] é bem mais completo. Diferentemente 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[10], também se chama Tile Editor, igualmente ao primeiro editor citado nessa seção. Diferentemente 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.

Em seguida, temos o editor para jogos em 2D desenvolvido por [Branicki 2013]. Ao contrário do anterior, ele faz uso dos prefabs, eliminando a limitação de somente construir cenários estáticos. Infelizmente, ele tem a mesma falha que o TileEditor (Zerofield), 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.

Além desses, existem outros editores, mas todos eles possuem características semelhantes aos que já foram citados. A maioria dos editores são de certa forma similares, mas cada um possui características únicas, que podem ser úteis ou não para os mais variados projetos. Um comparativo entre os diversos editores de cenário apresentados pode ser encontrado na Tabela 1.

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
Ajustar View	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
TileSets	Sim	Sim	Sim	Sim
Cenários 3D	Sim	Sim	Não	Não

3 Editor de Cenários 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 et al. 2013].

3.1 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 tem um sistema de auto-ajuste, uma vez que cada prefab possui dimensões diferentes e, consequentemente, precisa de espaços de tamanhos diferenciados no cenário.

[9]http://protilemapeditor.com

[10]https://www.assetstore.unity3d.com/en/!/publisher/9316

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.

A Figura 1 mostra o resultado obtido depois da Grid criada. Ela ainda não possui funcionalidade, pois apenas desenha as linhas na tela do editor do Unity.

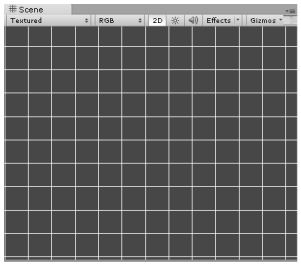

Figura 1: *Grid*

3.2 Grid Editor

Uma vez a *Grid* criada, os objetos precisam 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. Um evento chamado *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 ideais 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.

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.

3.3 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. 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 2.

3.4 Inspector GUI

A interface que será modificada é a Inspector GUI (ver Figura 3). Essa interface é responsável por mostrar as informações de cada

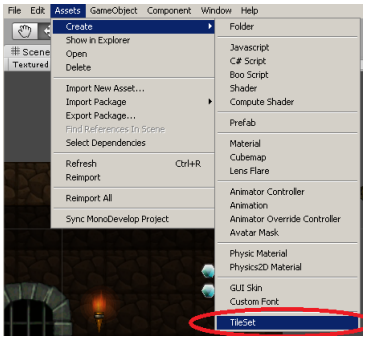

Figura 2: *Menus de itens do Unity com a adição da criação dos TileSets*

objeto. Alguns exemplos de informações contida nela são: Transform, RigidBody2D e Scripts. Nessa interface, são inseridos os elementos visuais para facilitar as escolhas dos prefabs.

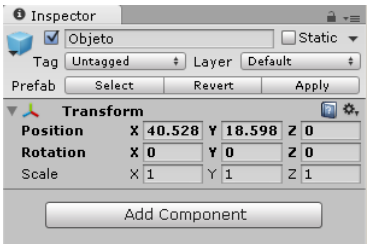

Figura 3: *Inspector GUI*

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 serão adicionados 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 4 mostra o resultado obtido depois das modificações.

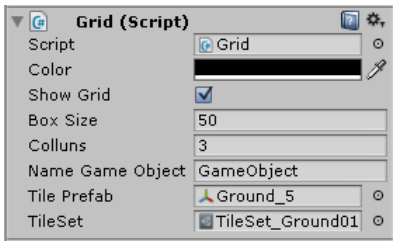

Figura 4: *Inspector*

Os seis primeiros elementos da Figura 4 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 4 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.

3.5 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.

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.

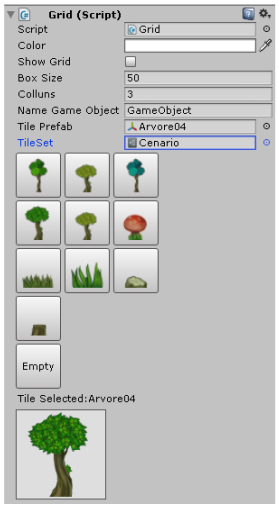

Figura 5: *Editor Final*

O Package do editor pode ser encontrado no seguinte domínio: http://migre.me/q9yo0.

4 Experimentos e Resultados Obtidos

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, a tarefa de construção de fases completas de um jogo no qual os autores deste trabalho estiveram envolvidos.

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 6) 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 6: *Imagem do cenário modelo utilizado para os experimentos. O cenário contém prefabs de plataformas, moedas, itens, etc.*

O gráfico da Figura 7 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.

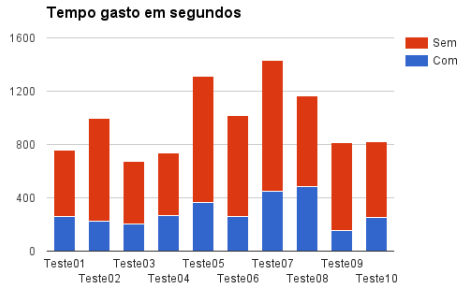

Figura 7: *Gráfico comparativo do tempo gasto, em segundos, para construção do cenário modelo com o TEd2D e sem o TEd2D.*

4.2 Construção de Fases Completas

Para avaliar o TEd2D no processo de construção de cenários mais complexos, foram desenvolvidas fases completas para o jogo Mathmare [Madeira et al. 2015].

Mathmare é um jogo educacional envolvendo desafios matemáticos do ensino médio com o objetivo de aumentar o interesse dos alunos pelas aulas de matemática neste nível específico da educação. Ele é composto por um cenário de plataforma em 2D com movimentação lateral (ver Figura 8).

O primeiro cenário de Mathmare foi desenvolvido inicialmente sem o auxílio do editor TEd2D. Em seguida, para ser testado com alunos ingressantes de graduação da UFRN, no semestre 2015.1, foi desenvolvida uma nova fase do jogo, usando o TEd2D, para ser jogada em 45 minutos.

Durante o desenvolvimento da fase do jogo a ser aplicada para os alunos, 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 experimentos com os alunos.

5 Conclusão

O presente trabalho consistiu no desenvolvimento de um editor de cenários para o Unity, tendo como objetivo facilitar a criação de

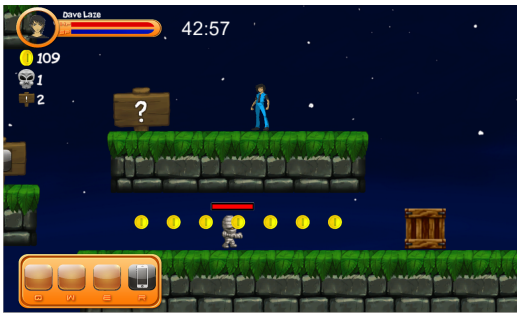

Figura 8: *Cena do jogo Mathmare utilizado com alunos ingressantes de graduação da UFRN*

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. Fases completas de um jogo de plataforma em 2D foram desenvolvidas com o uso do editor proposto, mostrando assim que é possível utilizá-lo eficazmente em um projeto. Durante a criação destas fases, ficou claro que sem o TEd2D não seria possível desenvolver todas elas 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

BESERRA, I. N. D. S. 2015. Ted2d: Um editor para criação de cenários de jogos 2d com unity. 2015. 62p. Trabalho de Conclusão de curso. Ciência da computação. UFRN. Natal/RN.

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

BRANICKI, D., 2013. How to add your own tools to unity?s editor. 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.

GREGORY, J. 2009. *Game engine architecture*. CRC Press.

JOHNSTON, C. R., 2013. New tool: Tileeditor. Sep., 2013. Disponível em: http://unitypatterns.com/new-tool-tileeditor. Acesso em Abril 21, 2015.

JORDAO, F., 2013. A nova guerra entre motores gráficos de games, July. 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.

MADEIRA, C., CÂMARA, L., BESERRA, I., AND TAVARES, R. 2015. Mathmare: um jogo de plataforma envolvendo desafios matemáticos do ensino médio. XIV Simposio Brasileiro de Jogos e Entretenimento Digital.

ROGERS, Y., SHARP, H., AND PREECE, J. 2013. *Design de Interação*, 3th ed. Bookman.