BLUCRAFT: UMA FERRAMENTA PARA A CRIAÇÃO DE JOGOS DE RPG DIGITAIS

Guilherme Paz Silva, Dalton Solano dos Reis – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

guilhermepaz@furb.br, dalton@furb.br

Resumo: *Este artigo apresenta o desenvolvimento de um editor de jogos de RPG digitais com o objetivo de facilitar a criação de histórias interativas, podendo serem utilizadas em um contexto pedagógico ou de entretenimento. O usuário do editor utiliza-se de um editor visual para a confecção de mapas e programação visual através de eventos para a criação de lógicas de jogo. O editor foi desenvolvido através da ferramenta Unity. Os objetivos foram atingidos, resultando em um editor de jogos de RPG funcional com customização de mapas e de lógicas através de programação visual, auxiliados pelas ferramentas escolhidas que cumpriram os papéis esperados no projeto. As propostas de extensão foram a adição de novos eventos, implementação de pathfinding e adição dos conceitos de itens e batalhas ao editor.*

Palavras-chave: *Editor. Jogos. RPG. Unity. Desenvolvimento.*

1 INTRODUÇÃO

Segundo Squire (2003, p. 1, tradução nossa), "o desenvolvimento contemporâneo de jogos, principalmente histórias interativas, ferramentas de autoria digital e mundos colaborativos, sugere novas e poderosas oportunidades para mídia educacional". Visto que jogos se mostram presentes na história de quase todas as culturas e sociedades (HUIZINGA, 1954 apud ZAGAL, 2010, p. 11), pode-se afirmar que o desenvolvimento cultural através do uso de jogos tem um embasamento histórico.

Os jogos de Role Playing Game (RPG) são "jogos de representação de papéis, onde a cooperação e a criatividade são seus principais elementos" (GRANDO; TAROUCO, 2008). A tradução da sigla é "jogo de interpretação de papéis" (em tradução livre) e, em sua forma original, é comumente classificado como uma brincadeira de contar histórias (ALVES, Lynn et al, 2004). Este trabalho visa explorar uma versão digital dos jogos deste tipo.

Visto de um espectro educacional, o engajamento de crianças no desenvolvimento de jogos "[...] pode permitir o desenvolvimento da imaginação e criatividade na infância e consequentemente das funções psicológicas superiores, como habilidades de concentração, atenção, raciocínio, memória [...]" (ALVES, 2017, p. 4). Além disso, segundo Grando e Tarouco (2008), "as características principais que auxiliam o jogo de RPG a se tornar uma excelente ferramenta educacional são: socialização, cooperação, criatividade, interatividade e interdisciplinaridade". Desta forma, é plausível sugerir que o uso de jogos, mais especificamente do tipo de RPG, seria benéfico no ambiente pedagógico, tanto em um contexto de criação quanto de aplicação.

Diante do exposto, o objetivo do projeto foi desenvolver um editor de jogos no estilo RPG que disponibilize ferramentas para a criação de contextos e objetivos customizáveis pelo usuário. Os objetivos específicos são: permitir a criação de entidades executáveis através de programação visual; permitir a customização de cenários interconectados; executar e compartilhar os jogos criados através do editor através de persistência.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção descreve os principais conceitos para o melhor entendimento do trabalho desenvolvido. A seção 2.1 descreve o que são e quais os objetivos de jogos de RPG, de sua criação até sua versão digital; a seção 2.2 apresenta o conceito de Entity Component System (ECS); a seção 2.3 explora as técnicas para uso da programação visual; e a seção 2.4 os trabalhos correlatos.

2.1 ROLE PLAYING GAMES (RPG)

A primeira versão do que viria a ser o RPG ("role playing game" ou "jogo de interpretação de papéis" em tradução livre) nasceu em 1974 através do jogo "Dungeons and Dragons" de Gary Gygax (GRANDO; TAROUCO, 2008). Conforme demonstra a Figura 1, a versão analógica do jogo é composta do narrador (também chamado de "mestre" no contexto do jogo) e os jogadores que, juntos, formam uma estória interativa (PEREIRA, 1992). Através da discussão e interpretação de ações faladas ou escritas, os jogadores narram acontecimentos enquanto o narrador descreve as consequências e desdobramentos destes acontecimentos.

Segundo Bittencourt e Giraffa (2003), "esta mecânica refere-se a forma tradicional de jogar RPG bastante conhecida pela comunidade como 'RPG de mesa', pelo fato de comumente ser jogado em torno de uma mesa com lápis, papel e dados". Desde sua criação, diversas outras modalidades baseadas em cima do conceito de "interpretação de personagens" surgiram. Dentre estas modalidades, o RPG digital surge em meados dos anos 70 (CRAWFORD, 1982). Desde então, o ramo de jogos digitais cresceu vertiginosamente, com estimativas de mais de 150 bilhões de dólares de consumo em 2019 (WIJMAN, 2019).

Figura 1 - Representação de uma mesa de RPG

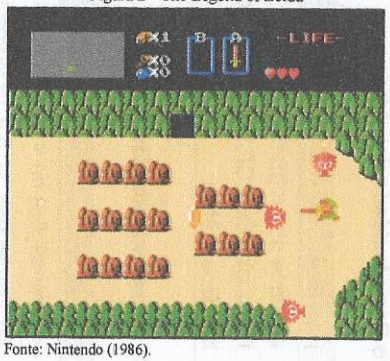
Fonte: Pinchefsky (2016).

Segundo Chagas (2010), "a proliferação dos RPGs em diversas modalidades, como jogos de tabuleiro, jogos de cards [...] ou nos jogos produzidos para computadores e videogames, populariza o hábito, exercícios e conduta relacionados à construção de novas narrativas pessoais". Na modalidade digital, "o RPG continua sendo uma representação de papéis, um jogo de faz-de-conta e permitindo vivenciar mundos imaginários, só que o grupo de pessoas não se reúnem presencialmente, mas no ciberespaço" (BITTENCOURT; GIRAFFA, 2003).

Em contramão dos RPGs *online* que permitem a conexão de diversos jogadores e foi uma modalidade popularizada principalmente pelo jogo Diablo, da Blizzard (BITTENCOURT; GIRAFFA, 2003), uma modalidade similar mas à parte é a de jogos de RPG de jogador único, onde não existe interação com um segundo jogador. Dada a tecnologia, esta foi a primeira forma digital de jogos de RPG.

Exemplos não faltam para o contexto de jogos, mas um dos mais ilustres e representativos foi The Legend of Zelda (Zeruda no Densetsu Za Hairaru Fantajī, no Japão), como mostrado na Figura 2. O jogo de 1986 da Nintendo apresenta Link, um personagem controlado pelo jogador através de uma visão aérea. É do objetivo do jogador decifrar enigmas, encontrar moradores e comerciantes e enfrentar criaturas em um mundo de fantasia (NINTENDO OF AMERICA, 1986). O jogo é considerado do gênero "Japanese RPG", uma vertende do RPG ocidental com características próprias como batalha em turnos, foco no enredo e visuais próprios derivados do estilo mangá.

Figura 2 - The Legend of Zelda



Fonte: Nintendo (1986).

Fora da indústria do entretenimento, existe uma vertente que explora os chamados Jogos Sérios, em ebulição no mundo acadêmico (GURZYNSKI; HOUNSELL; KEMCZINSKI, 2016). Jogos educativos estão abrangidos dentro desta categoria. Através do conceito de Jogos Digitais Sérios do estilo RPG (RPGDS) é possível contextualizar as ações de um jogador dentro de um ambiente montado através de uma ferramenta (GURZYNSKI; HOUNSELL; KEMCZINSKI, 2016, apud Frias, 2009). Uma ferramenta que permite a criação destes jogos é a ferramenta RPG4ALL (ver seção 2.4).

2.2 ENTITY COMPONENT SYSTEM (ECS)

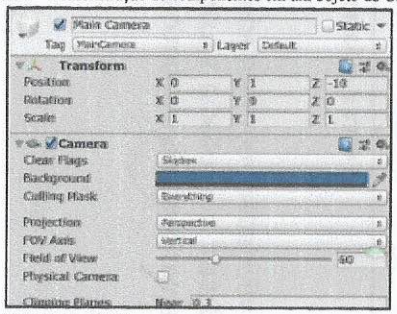
Entity Component System (ECS) é uma arquitetura de código que se baseia na composição de entidades, componentes e sistemas para resolução de problemas, em contraponto a outra metodologia como a orientação de objetos, por exemplo. Através destas camadas, é possível que uma entidade representada por um identificador possua componentes atrelados a ela contendo informações que, por sua vez, são lidas e executadas através de sistemas que executam uma lógica (HALL, 2014).

Componentes não possuem quaisquer funções, somente estados com informações; são os sistemas que executam utilizando múltiplos componentes como entrada para execução de sua lógica (FORD, 2017). Desta forma, sistemas como "renderização de personagem" ou "câmera" possuem uma lógica de execução, enquanto componentes como "posição" ou "imagem" possuem as informações necessárias para que estes sistemas sejam executados. Segundo Ford (2017, tradução nossa), "ECS é a cola do Overwatch; a arquitetura te ajuda a integrar diversos sistemas disparates com acomplamento mínimo".

Existem vertentes que usam a metodologia ECS como base mas modificam algumas de suas propriedades. Um exemplo é a plataforma Unity, que utiliza uma arquitetura baseada em entidades (chamadas de GameObject no contexto da ferramenta) e componentes; porém, estes componentes possuem não só dados como funcionalidades, eliminando assim a existência de sistemas (UNITY, 2020).

Conforme visualizado na Figura 3, os componentes da ferramenta Unity possuem informações como os componentes descritos pela ECS, mas executam a lógica que seria executada pelo sistema na segunda arquitetura. No caso da Figura 3, o componente "Camera" possui o código para a movimentação da câmera, por exemplo, além de suas informações. Esta abordagem permite o desacoplamento de componentes interdependentes, mas perde a soberania de sistemas especializados em diversos tipos de componentes como no ECS original.

Figura 3 - Demonstração de componentes em um objeto do Unity



Fonte: Unity (2020).

2.3 REFLEXÃO EM C#

Sistemas computacionais refletivos permitem observar e modificador seu próprio comportamento, principalmente em situações que são tipicamente observadas de um ponto de vista externo (SOBEL; FRIEDMAN, 1996, p. 1).

Na linguagem C#, a reflexão é disponibilizada através de objetos do tipo `Type` que descrevem informações diversas como assemblies, módulos e outros tipos, com os quais é possível criar instâncias de um tipo ou executar métodos de um objeto existente através de seu objeto de tipo (MICROSOFT, 2015). Estas ferramentas podem ser extrapoladas para a utilização de sistemas que instanciam objetos de tipos dinamicamente informados, sem conhecimento explícito do código-fonte. O código do Quadro 1 mostra o uso de reflexão para instanciar um objeto de um tipo obtido somente pelo seu nome completo.

Quadro 1 - Reflexão utilizada para instanciar objetos

```
var typeName = superTypeName;
if (data.ObjectValue.TryGetValue("_TYPE", out var specificTypeName))
    typeName = specificTypeName.StringValue;

var type = Type.GetType(typeName);
if (type == null)
{
    Debug.LogError("Could not find class for component '" + typeName + "'");
    yield break;
}

if (!typeof(T).IsAssignableFrom(type))
{
    Debug.LogError("Type '" + typeName + "' is not T");
    yield break;
}

if (!superType.IsAssignableFrom(type))
{
    Debug.LogError("Type '" + typeName + "' is not from file supertype '" +
    superTypeName + "'");
    yield break;
}

var _instance = (T) Activator.CreateInstance(type);
_instance.SetData(new PersistenceData(data.ObjectValue));
yield return _instance;
```

Fonte: elaborado pelo autor.

Conforme mostra o Quadro 1, o objeto `Type` é instanciado através da classe `Activator` após diversas verificações de existência de tipo com o nome informado e compatibilidade com o método `IsAssignableFrom`. Após isso, o método `SetData` passa as informações necessárias para a configuração do objeto em questão e é responsabilidade do objeto preencher suas propriedades.

2.4 TRABALHOS CORRELATOS

Nesta seção, serão apresentados três trabalhos correlatos que abordam temas relacionados a este trabalho. No Quadro 2 é apresentado o EasyEdu, trabalho desenvolvido por Corso (2017) que se trata de uma ferramenta web equipada para o desenvolvimento de jogos educacionais por professores e crianças. No Quadro 3 é apresentado o trabalho RPG4ALL desenvolvido por Pessini, Kemczinski, Hounsell (2015) que se trata de uma Ferramenta de Autoria para o desenvolvimento de jogos. Por fim, o Quadro 4 apresenta o produto RPG Maker MV da empresa Enterbrain (2015), uma ferramenta para a criação de jogos de RPG.

Quadro 2 - EasyEdu

Referência	Corso (2017)
Objetivos	Desenvolvimento de jogos educacionais em ambiente web.
Principais funcionalidades	Criação de jogos educacionais através do uso de templates. Compartilhamento dos jogos através da internet e QR Code. Personalização com o envio de imagens e cadastro de palavras.
Ferramentas de desenvolvimento	AngularJS, Google Drive, QR Code.
Resultados e conclusões	De acordo com o autor, os templates de quebra-cabeças e memória foram inclusos no planejamento mas não puderam ser desenvolvidos em tempo hábil. Porém, a equipe de pedagogia responsável pela aplicação em sala de aula obteve os resultados esperados.

Fonte: elaborado pelo autor.

Com o EasyEdu é possível desenvolver jogos baseados em templates com regras pré-definidas e executar estes jogos a partir de uma galeria. Através da ferramenta, o professor é capaz de desenvolver e compartilhar os jogos desenvolvidos através de QR Code e armazenamento no Google Drive. Um ponto importante da ferramenta é a personalização por parte do professor encarregado da utilização.

3.2 IMPLEMENTAÇÃO

Para o desenvolvimento da ferramenta foi utilizado o motor de jogos Unity e a biblioteca `SimpleJSON` (BOLDAI AB, 2011) para auxílio na persistência dos dados no formato `JSON`. Os ícones e imagens utilizados foram do pacote `DawnLike` (DRAGONDEPLATINO, DAWNBRINGER, 2013). Para algumas edições gráficas das imagens usadas foi utilizado o software Aseprite.

Como a engine Unity utiliza-se de cenas para realizar suas operações, foram criadas três cenas para melhor organizar a ferramenta:

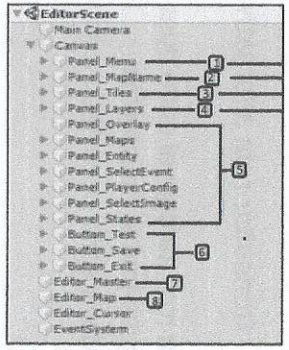
a) `MenuScene`: cena que possui o fluxo de menu inicial, levando a criação de um novo projeto ou ao carregamento de um projeto existente;
b) `EditorScene`: cena do editor da ferramenta que possui as janelas para a programação visual e montagem do jogo;
c) `PlayScene`: cena do executor que realiza o loop de jogo conforme a programação realizada no editor.

O principal ponto em comum dentro destas três cenas é a modelagem dos elementos e a persistência dos dados, ambas explicadas na seção 3.2.3.

3.2.1 EditorScene

Conforme mostrado na Figura 6, a maior parte da cena do editor da ferramenta é composta de elementos dentro de um `Canvas`. Em seguida, o Quadro 5 traz uma breve descrição do funcionamento de cada objeto em cena. Vale notar que os painéis não mostrados no lado direito da Figura 6 são ativados durante as interações do usuário, conforme o fluxograma da Figura 5.

Figura 6 – Hierarquia dos objetos utilizados na cena do editor



Fonte: elaborado pelo autor.

Quadro 5 - Objetos em cena no cena do editor

Identificador	Descrição
1	Painel contendo os botões para abertura das telas para cadastro de mensagens e configurações do jogador.
2	Painel mostrando o nome do mapa atual em edição.
3	Painel com a ferramenta de pintura do mapa atual, contendo os `sprites` disponíveis para pintura assim como opção de `Adicionar` ou `Remover` um *sprite*.
4	Painel para escolha da camada de edição, seja de pintura (camadas 1, 2 e 3) ou criação de entidades (camada 4).
5	Objetos contendo as telas de configuração disponibilizadas pelo editor.
6	Botões do fluxo de execução, salvamento e saída da ferramenta.
7	Objeto que contém o script `Editor_MasterController`, responsável por orquestrar todo o fluxo principal do editor.
8	Objeto que contém os objetos visuais e de interação com o mapa em edição atualmente, seja para a pintura de *sprites* ou para a criação de entidades.

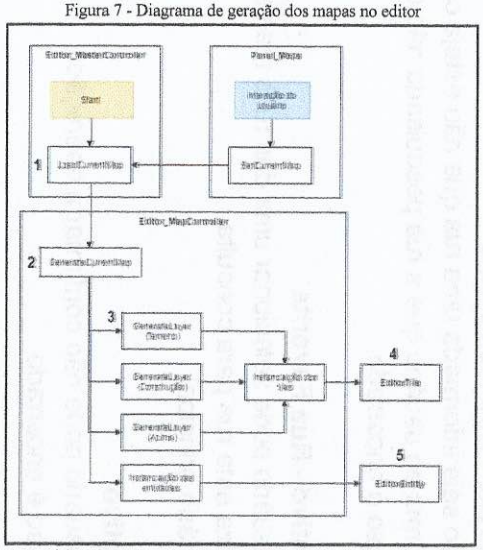
Fonte: elaborado pelo autor.

Através do fluxo trazido pelos objetos mostrados na Figura 6, são realizadas as ativações dos painéis citados no identificador 5 do Quadro 5. Para a padronização dos painéis do editor, foi realizada a criação da interface `IEditorPanel`. Esta interface possui somente o método `DialogOpened` para o carregamento das informações de responsabilidade do painel. Os painéis são criados utilizando o `Canvas` do Unity, uma biblioteca destinada a criação de interfaces visuais para o usuário.

Através do método `OpenPanel` e `ClosePanel` da classe `Editor_MasterController` é possível realizar a abertura e fechamento destes painéis, recebendo uma instância do mesmo para a configuração apropriada por parte do responsável pela abertura. Os painéis são responsáveis por receber entradas do usuário. Informações relacionadas ao uso destes painéis podem ser verificadas na seção 3.3.

O ponto principal do editor são a criação dos mapas, comandada pela classe `Editor_MapController`. Conforme mostrado na Figura 7, a classe é chamada através do método `LoadCurrentMap` (Figura 7-1), que pode ser executado na abertura do editor ou através de interação do usuário. O principal método da rotina de criação do mapa é o `GenerateCurrentMap` (Figura 7-2) que usa várias chamadas ao método `GenerateLayer` (Figura 7-3) para instanciar os `prefabs` de `tiles`. Estes `tiles` possuem o componente `EditorTile` (Figura 7-4), responsável por tratar os cliques do usuário e alterar seu `sprite` de acordo com o selecionado no editor. Posteriormente, o método `GenerateCurrentMap` faz o instanciamento das entidades cadastradas que possuem o componente `EditorEntity` (Figura 7-5) que, similar ao componente `EditorTile`, trata os cliques para a edição das entidades.

Figura 7 - Diagrama de geração dos mapas no editor



Fonte: elaborado pelo autor.

3.2.2 PlayScene

A cena `PlayScene` possui como intuito a execução do jogo criado dentro do editor. Para isso, possui o `GameMasterBehaviour` como classe principal; a classe `GameState` possui o estado do jogo atual; as classes `EntityBehaviour` e `PlayerBehaviour` para controle dos `GameObject` de entidades e jogador, respectivamente; e a classe `MessagePanel` para exibição de mensagens em tela.

A principal implementação no lado da execução são os eventos que herdam da classe `GameEvent`, presente nos modelos (ver seção 3.2.3). Cada evento deve implementar os seguintes métodos:

a) `GetNameText`: retorna o nome do evento para ser usado como informativo do usuário em tela;
b) `GetDescriptionText`: retorna uma descrição dos parâmetros para ser usado como informativo do usuário em tela;

Quadro 8 - Código de obtenção de valores utilizando reflexão e biblioteca `SimpleJSON`

```
public T Get<T>(string key, T defaultValue = default)
{
    var obj = Get(key);
    if (obj == null)
        return defaultValue;

    if (typeof(T) == typeof(string))
        return (T) (object) obj.StringValue;
    if (typeof(T) == typeof(int))
        return (T)(object)int.Parse(obj.StringValue);
    if (typeof(T) == typeof(bool))
        return (T)(object)obj.BooleanValue;
    if (typeof(T) == typeof(PersistenceData))
        return (T)(object)new PersistenceData(obj.ObjectValue);
    if (typeof(T) == typeof(List<PersistenceData>))
    {
        var l = new List<PersistenceData>();
        foreach (var v in obj.ArrayValue)
        {
            l.Add(new PersistenceData(v.ObjectValue));
        }

        return (T)(object)l;
    }
    if (typeof(T) == typeof(Sprite))
        return (T) (object) obj.StringValue.ToSprite();
    if (typeof(T) == typeof(Vector2))
        return (T)(object)obj.StringValue.ToVector2();


    return (T)(object)obj;
}
```

Fonte: elaborado pelo autor.

As entidades persistidas pela classe `GameEntity` e que geram a lógica dos jogos construídos são modeladas visando implementar uma vertente de ECS (ver seção 2.2). `GameEntity` são as entidades e os eventos persistidos por `GameEvent` que fazem o papel de componente (contendo dados) e sistemas (executando operações), similar a arquitetura utilizada na própria engine Unity.

3.3 USO DA FERRAMENTA

Esta seção se destina a descrever a utilização de todas as funcionalidades da ferramenta. Na primeira seção são apresentados conceitos relacionados ao fluxo e ao jogador, na segunda seção são apresentados os conceitos de entidades e seus eventos e na terceira seção são apresentadas informações de como utilizar estados, interruptores e variáveis para a criação de lógica.

3.3.1 Execução, Mapas e Jogador

O fluxo dos jogos criados pela ferramenta é baseado nos mapas criados. Para criar um mapa, deve-se clicar no botão Mapas (Figura 10-1). Através deste menu, é possível preencher nome e tamanho para o mapa criado. Após a criação, é possível selecionar o mapa para edição em tela através do botão `Selecionar mapa`.

Após selecionado, o mapa pode ser editado visualmente utilizando os painéis de *sprites* e camadas (Figura 10-3 e Figura 10-4, respectivamente). O painel de *sprites* permite selecionar um ícone para ser pintado em tela com o botão esquerdo do mouse. Os botões azul e vermelho permitem adicionar e remover um ícone do mapa, respectivamente. O painel de camadas permite pintar ícones em níveis distintos.

a) Camada de Terreno: esta camada será renderizada abaixo do jogador e das outras camadas;
b) Camada de Construção: esta camada será renderizada acima da Camada de Terreno e abaixo da Camada Acima do Jogador, além do jogador não poder se movimentar através dos ícones desta camada;
c) Camada Acima do Jogador: esta camada será renderizada acima do jogador e das outras camadas;
d) Camada de Entidades: esta camada não é editada visualmente e serve para a criação de entidades, conforme seção 3.3.2.

Figura 10 - Interface da ferramenta

Fonte: elaborado pelo autor.

Além da edição de mapas, é possível editar as configurações do jogador através do botão Jogador (Figura 10-2). É possível mudar a imagem do jogador e definir qual será o mapa e coordenadas X e Y que o jogador começará o jogo.

Os controles utilizados pelo jogador são as teclas direcionais para realizar movimento e o caractere "Z" para executar a interação com entidades. Para executar um jogo criado, é possível utilizar duas opções:

a) Menu inicial: o botão "Executar jogo" leva direto ao jogo criado pelo arquivo atual;
b) Testar: o botão "Testar", após o carregamento de um jogo, leva a execução do mesmo.

3.3.2 Entidades e Eventos

Após selecionar a Camada de Entidades (ver seção 3.3.1) é possível clicar com o botão direito em uma posição do mapa para criar uma entidade. A tela de edição da entidade aparecerá, onde será possível definir seu nome, imagem, passável/impassável, tipo de execução e eventos.

O nome da entidade é utilizado como identificador e é usado por alguns eventos. O ícone é a imagem que será mostrada para representar a entidade no mapa, e pode ser deixada em branco. É possível selecionar se a entidade será "passável" (jogador pode se movimentar através da entidade) ou não-passável (jogador não poderá se movimentar através da entidade). O tipo de execução define quando os eventos serão executados, conforme a lista:

a) Interação: os eventos serão executados quando o botão de interação for pressionado junto a entidade (ver seção 3.3.4);
b) Contato: os eventos serão executados quando o jogador se movimentar em cima da entidade (a entidade precisa ser passável);
c) Automática: os eventos serão executados quando o jogador entrar no mapa atual.

Os eventos são comportamentos que executam atividades sequencialmente. Cada estado (ver seção 3.3.3) de uma entidade pode possuir diversos eventos que serão executados sequencialmente de acordo com o seu tipo de execução. Os eventos são:

a) Mensagem: mostra uma mensagem em tela que pode ser fechada pressionando o botão de interação (ver seção 3.3.4);
b) Definir variável: define, soma ou subtrai um valor na variável informada (ver seção 3.3.3);
c) Ativar/desativar interruptor: define se um interruptor está ativo ou inativo (ver seção 3.3.3);
d) Mover entidade: move a entidade do nome informado para um local no mapa ou relativo a sua posição (utilizar o nome "Jogador" para mover o jogador);
e) Mudar mapa: realiza a transição para outro mapa;
f) Mudar imagem de entidade: altera a imagem da entidade informada.

3.3.3 Estados, Interruptores e Variáveis

Cada entidade pode possuir diversos estados, representando comportamentos de acordo com o contexto atual do jogo. Um estado pode ser ativado através de variáveis ou interruptores. Através do botão `Estados` (Figura11-1), é possível criar novos estados definindo um nome e ativações. Uma ativação representa uma verificação de uma variável, um interruptor ou ambos. No cenário da verificação informada ser verdadeira, o estado será ativado.

Para editar a imagem, tipo de execução e eventos de uma entidade, é necessário clicar no botão `Selecionar`. Estes eventos e tipo de execução serão utilizados caso este estado esteja ativo.

Figura 11 - Exemplo do painel de edição de entidades

Fonte: elaborado pelo autor.

4 RESULTADOS

Esta seção se destina a descrever os testes realizados com a ferramenta. Para validar as funcionalidades da ferramenta, foi realizada a criação de pequenos jogos de RPG utilizando todos os eventos e funcionalidades de estado disponibilizados. O jogo desenvolvido através da ferramenta está disponível no repositório `GIT` do projeto e detalhes de sua edição podem ser encontradas no Apêndice A.

Os resultados foram satisfatórios do ponto de vista de implementação, visto que a ferramenta Unity se mostrou útil e capaz de realizar todas as funcionalidades necessárias. A principal utilização da ferramenta Unity foi o `Canvas` (ver seção 3.2.2) que se mostrou de grande valia para a criação rápida dos diversos painéis de cadastro necessários para a criação do editor.

A primeira implementação foi realizada utilizando os conceitos puros de ECS (ver seção 2.2); porém, tais conceitos se mostraram demasiadamente abstratos para a implementação do editor. A solução foi utilizar conceitos mais brandos e substituir a execução simultânea dos componentes e sistemas por eventos que seriam executados linearmente, similar ao trabalho correlato RPG Maker (ver seção 2.4).

Os eventos criados para a utilização do editor foram satisfatórios e a lógica disponível com a utilização das variáveis e interruptores é suficiente para um jogo pequeno de RPG. Mais conceitos que interessantes de serem abordados eram itens e batalhas, comuns dentro do gênero de RPG, mas a complexidade ultrapassaria o escopo do projeto. Além disso, a movimentação pode ser melhorada utilizando um algoritmo de `pathfinding` como o `A*`.

A utilização do editor se tornou mais complexa do que o necessário, levando em conta o leiaute das telas e a relação dos modelos apresentados. Como os conceitos da seção 3.3 são abstratos, existe uma necessidade de exemplos práticos para uma boa compreensão do usuário.

5 CONCLUSÕES

Diante dos resultados apresentados, conclui-se que a ferramenta se provou capaz de criar jogos de RPG utilizando lógicas de programação visual e customização de mapas. Parte das funcionalidades propostas do editor foram alcançadas, ainda que sua usabilidade tenha deixado a desejar do ponto de vista de facilidade de uso.

A ferramenta Unity se mostrou poderosa, embora burocrática em alguns pontos (ver seção 4.1). A biblioteca SimpleJSON foi de grande valia para as rotinas de persistência e abstraíram muito da manipulação dos arquivos JSON.

As possíveis extensões propostas para seguir a implementação são: a) adicionar novos eventos para lógicas mais complexas ou funcionalidades visuais; b) implementar uma lógica de `pathfinding` como A* para a movimentação de entidades; c) adicionar o conceito de itens conforme usado normalmente no gênero RPG; d) implementar batalhas conforme usado normalmente no gênero RPG.

REFERÊNCIAS

ALVES, Adriana G. "Eu fiz meu game": um framework para desenvolvimento de jogos por crianças. In: Congresso Brasileiro de Informática na Educação, 6., 2017, Recife. **Anais...** Itajaí: Univali, 2017, p. 1-9. Disponível em: . Acesso em: 05 jul. 2020.

ALVES, Lynn et al. Ensino On-Line, jogos eletrônicos e RPG: Construindo novas lógicas. In: **Conferência eLES**. 2004. p. 49-58.

BOLDAI AB. SimpleJSON.NET. Disponível em: <https://github.com/mhallin/SimpleJSON.NET>. Acesso em: 05 jul. 2020.

BITTENCOURT, João Ricardo. GIRAFFA, Lucia Maria. Modelando Ambientes de Aprendizagem Virtuais utilizando Role-Playing Games. In: Simpósio Brasileiro de Informática na Educação, 14, 2003, Rio de Janeiro. **Anais...** Porto Alegre: PUCRS, 2003. p. . Disponível em: < http://www.nce.ufrj.br/sbie2003/publicacoes/paper71.pdf>. Acesso em: 02 jun. 2020.

CHAGAS, Artur Alves de Oliveira. **O transbordamento do lúdico e da biopolítica em jogos Massive Multiplayer Online**: um estudo sobre World of Warcraft. 2010. 157 p. Tese (Mestrado em Educação) - Universidade de São Paulo, São Paulo, 2010.

CORSO, Felipe L. **EasyEdu**: Editor web para jogos multitoque. 2017. 93 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

CRAWFORD, C. **The Art of Computer Game Design**, Washington State University, 1982.

DRAGONDEPLATINO. DAWNBRINGER. **DawnLike**: 16x16 Universal Rogue-Like Tileset v1.81 | OpenGameArt.org. 2013. Disponível em: < https://opengameart.org/content/dawnlike-16x16-universal-rogue-like-tileset-v181>. Acesso em: 02 jun. 2020.

ENTERBRAIN. Make Your Own Game With RPG Maker. Disponível em: <https://www.rpgmakerweb.com/>. Acesso em: 02 jun. 2020.

FORD, Timothy. Overwatch Gameplay Architecture and Netcode. In: **Game Developer's Conference**. 2017. Disponível em: <https://www.youtube.com/watch?v=W3aieHjyNvw&t=839s>. Acesso em: 02 jun. 2020.

GRANDO, Anita; TAROUCO, Liane Margarida Rockenbach. **O uso de jogos educacionais do tipo RPG na educação**. RENOTE-Revista Novas Tecnologias na Educação, v. 6, n. 1, 2008.

GURZYNSKI, Cleber; HOUNSELL, Marcelo; KEMCZINSKI, Avanilde. Análise de um jogo RPG educacional produzido pelo próprio docente, auxiliado por Ferramenta de Autoria. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. 2016. p. 617.

HALL, Daniel Masamune. **ECS Game Engine Design**. [S.l.]. 2014. Disponível em: < https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1138&context=cpesp>. Acesso em: 02 jun. 2020.

HUIZINGA, Johan. **Homo Ludens**. São Paulo: Editora Perspectiva, 2000.

MICROSOFT. **Reflexão (C#)**. Disponível em : <https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/concepts/reflection>. Acesso em: 02 jun. 2020.

NINTENDO OF AMERICA, Inc. **The Legend of Zelda Instruction Booklet**. Japão. 1986. p. 28.

PEREIRA, C.K., Andrade, F.M., Freitas, L.E.R. **Desafio dos Bandeirantes – Aventuras na Terra de Santa Cruz**, GSA, 1992.

PESSINI, Adriano; KEMCZINSKI, Avanilde; DA SILVA HOUNSELL, Marcelo. **Uma Ferramenta de Autoria para o desenvolvimento de Jogos Sérios do Gênero RPG**. Anais do Computer on the Beach, p. 071-080, 2015.

PINCHEFSKY, Carol. **Role-Playing Gamers Have More Empathy Than Non-Gamers**. [S.l]: Geek and Sundry, 2016.