BLUCRAFT: UMA PLATAFORMA 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 visto que as funcionalidades propostas foram completadas com êxito [O que mais de conclusão?]. As propostas de extensão foram [PROPOSTAS DE EXTENSÃO].*

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 RPG são "jogos de representação de papéis, onde a cooperação e a criatividade são seus principais elementos" (GRANDO, 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 (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 (Role Playing Game), de sua criação até sua versão digital; a seção 2.2 apresenta o conceito de ECS (Entity Component System); a seção 2.3 explora as técnicas para uso da programação visual.

2.1 RPG (ROLE PLAYING GAMES)

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, 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 (Precisa de fonte?).

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

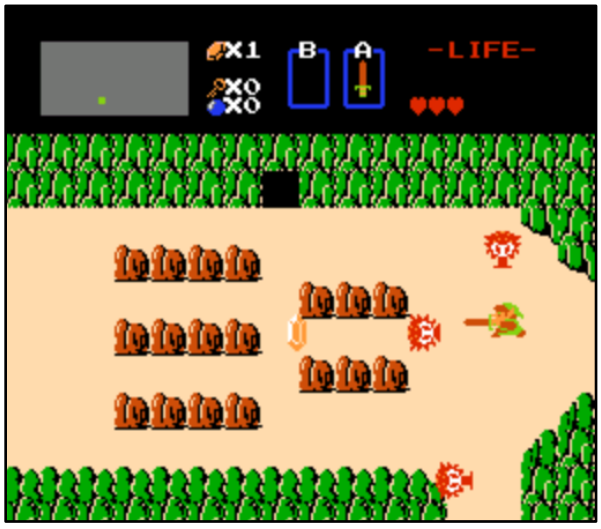
Fonte: Geek and Sundry (2016).

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 apud Frias, 2016). Uma ferramenta que permite a criação destes jogos é a ferramenta RPG4ALL (ver seção 2.4).

2.2 ECS (ENTITY COMPONENT SYSTEM)

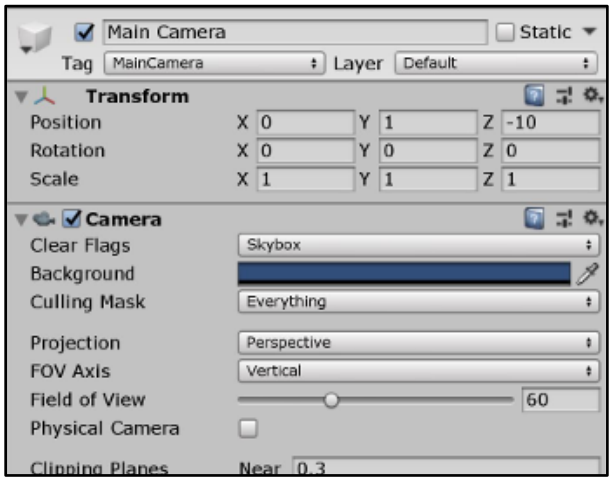
Entity Component System é uma arquitetura de código que se baseia na composição de entidades, componentes e sistemas para resolução de problemas, em contraponto a outra metodologia como a orientação de objetos, por exemplo. Através destas camadas, é possível que uma entidade representada por um identificador 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 6, 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 [FONTE?].

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). [Mais alguma coisa aqui]

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.

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.

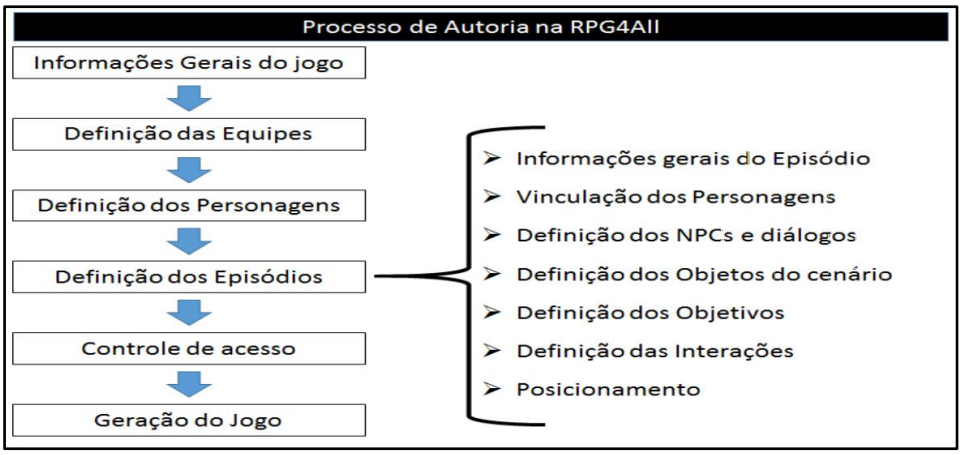
Quadro 3 - RPG4ALL

Referência	Pessini, Kemczinski, Hounsell (2015)
Objetivos	Ferramenta para especificação de Jogos Sérios por docentes sem conhecimento em desenvolvimento de jogos.
Principais funcionalidades	Importação de mapas da ferramenta Tiled. Definição de episódios, objetos e personagens. Posicionamento de objetos em cena. Reprodução dos jogos através do Módulo de Execução.
Ferramentas de desenvolvimento	Tiled Map Editor; a tecnologia dos Módulos não é discutida no trabalho.
Resultados e conclusões	A ferramenta encontrava-se em fase de homologação junto a docentes e alunos de cursos de licenciatura na época do artigo. Os autores citaram a validação das funcionalidades da ferramenta como uma etapa futura.

Fonte: elaborado pelo autor.

O RPG4ALL disponibiliza uma ferramenta para a definição de diversas etapas de um RPG em um contexto de JS (Jogos Sérios). O intuito é a utilização dentro de ambiente pedagógico e com o uso por docentes e alunos de pedagogia. Conforme a Figura 4, são utilizadas diversas etapas sequenciais dna especificação dos JS dentro do Módulo de Autoria para que, posteriormente, o produto do processo de geração seja executado por um segundo módulo, o Módulo de Execução.

Figura 4 - Processo de Autoria do RPG4ALL



Fonte: Uma Ferramenta de Autoria para o desenvolvimento de Jogos Sérios do Gênero RPG (2015).

Quadro 4 - RPG Maker

Referência	Enterbrain (2015)
Objetivos	Ferramenta comercial para o desenvolvimento de jogos de RPG através de programação visual.
Principais funcionalidades	Programação lógica através de eventos e programação visual. Edição de mapas e objetos. Definição de itens e personagens. Exportação para múltiplas plataformas como Windows, Mac, mobile e HTML5. Programação avançada através de JavaScript.
Ferramentas de desenvolvimento	A ferramenta é proprietária e não são divulgadas as tecnologias utilizadas.
Resultados e conclusões	A ferramenta é comercial; não foram encontradas conclusões acerca de conclusões do projeto.

Fonte: elaborado pelo autor.

Conhecido no desenvolvimento de jogos independentes, o RPG Maker é considerado uma das primeiras engines de jogos acessíveis a amadores, lançando sua primeira versão em 1999. O foco da ferramenta é a disponibilização de cadastros e programação visual através de eventos para permitir o design de mapas e definição de lógicas de jogo de forma acessível. Além disso, é possível utilizar JavaScript caso o usuário seja proficiente no desenvolvimento de software.

3 DESCRIÇÃO DA FERRAMENTA

Esta seção dedica-se a apresentar os detalhes de especificação, implementação e usabilidade da ferramenta. Com esse objetivo, a seção foi dividida em três partes. A primeira parte, na seção 3.1, apresenta uma visão geral da ferramenta e seu fluxo de funcionamento. A segunda parte, na seção 3.2, detalha os processos de implementação mais significativos dentro do projeto. Por fim, a seção 3.3 apresenta as características de uso da ferramenta com especificações de sua usabilidade.

3.1 VISÃO GERAL DA FERRAMENTA

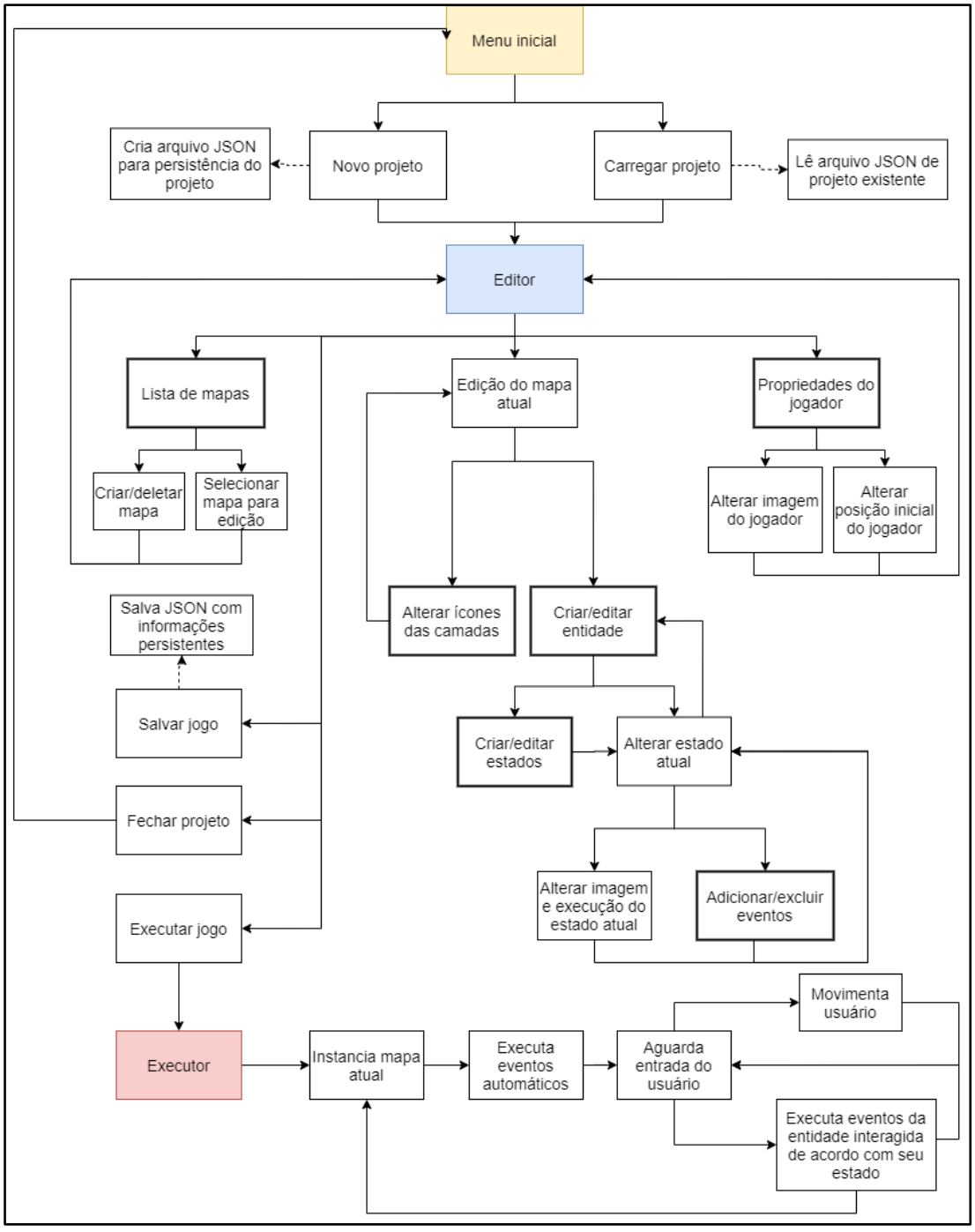
A ferramenta desenvolvida se trata de um editor de jogos com foco nas mecânicas normalmente utilizadas no gênero RPG. Para isso, foram utilizados os conceitos de mapas estáticos e entidades dinâmicas formadas de eventos (vide seção 2.2), assim como programação visual através de janelas de cadastro e configuração. Assim, o fluxo da ferramenta é composto de duas partes:

a) **Editor**: composto de uma interface com diversas janelas de configuração, recebe as entradas da programação visual realizada pelo usuário pelas janelas, assim como a customização dos mapas e dos atributos do jogador;
b) **Executor**: após a leitura dos dados persistidos salvos através do editor, realiza a execução dos mapas e eventos das entidades, recebendo as entradas dos botões de direcionais (movimentação) e interação do

jogador e alterando os estados do jogo de acordo com a programação dos eventos.

Conforme apresentado no fluxograma da Figura 5, o editor é composto de janelas para criação e edição dos mapas, criação e edição das entidades e criação e edição dos eventos, de acordo com o fluxo mostrado. Após o editor, o executor realiza o carregamento do modelo de jogo criado e segue seu loop próprio. Este fluxo também sugere a hierarquia dos objetos na modelagem utilizada (vide seção 3.2).

Figura 5 - Fluxograma do editor



Fonte: elaborado pelo autor.

3.2 IMPLEMENTAÇÃO

Para o desenvolvimento da ferramenta foi utilizado o motor de jogos Unity e a biblioteca `SimpleJSON` 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:

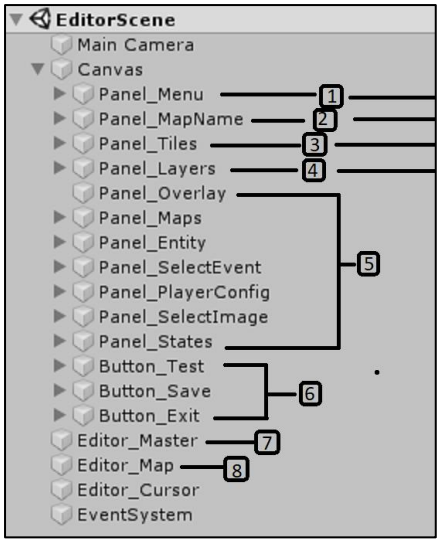
a) `MenuScene`: cena que possui o fluxo de menu inicial, levando a criação de um novo projeto ou ao carregamento de um projeto existente;
b) `EditorScene`: cena do editor da ferramenta que possui as janelas para a programação visual e montagem do jogo;
c) `PlayScene`: cena do executor que realiza o loop 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.

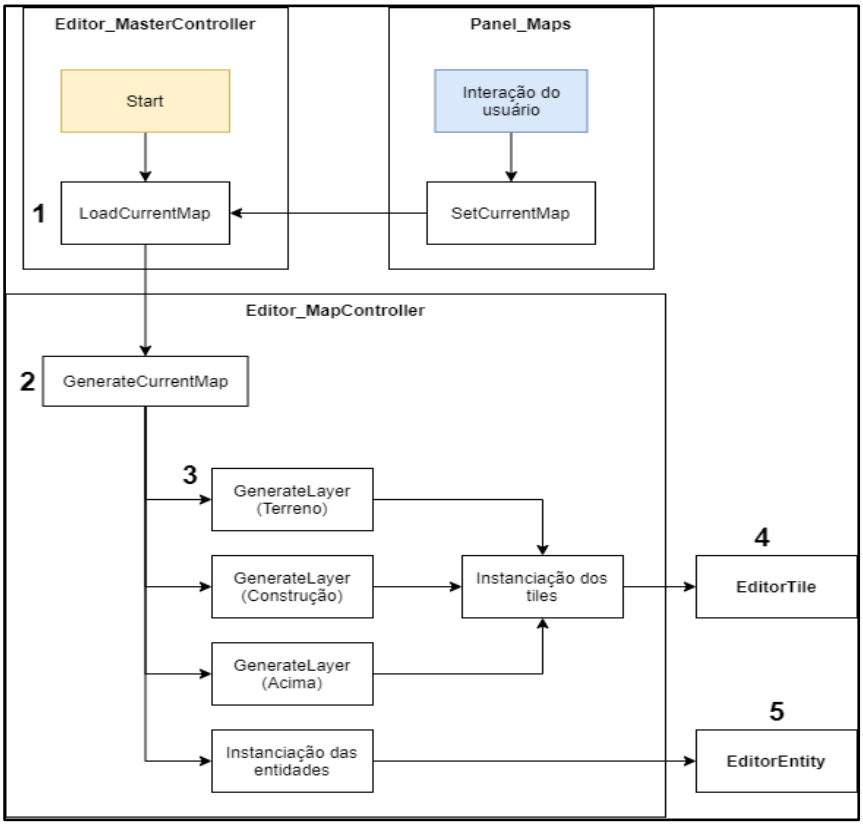
Através do fluxo trazido pelos objetos mostrados na Figura 6, são realizadas as ativações dos painéis citados no identificador 5 do Quadro 5. Para a padronização dos painéis do editor, foi realizada a criação da interface `IEditorPanel`. Esta interface possui somente o método `DialogOpened` para o carregamento 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 8, a classe é chamada através do método `LoadCurrentMap` (identificador 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` (identificador 2) que usa várias chamadas ao método `GenerateLayer` (identificador 3) para instanciar os `prefabs` de `tiles`. Estes `tiles` possuem o componente `EditorTile` (identificador 4), responsável por ouvir cliques do usuário e alterar seu `sprite` de acordo com o selecionado no editor. Posteriormente, o método `GenerateCurrentMap` instancia as entidades cadastradas que possuem o componente `EditorEntity` (identificador 5) que, similar ao componente `EditorTile`, ouve 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;
c) `Execute`: método chamado na execução do evento;
d) `Update`: método chamado em cada loop de Update do Unity, para caso o evento seja de execução contínua;

Conforme o código do Quadro 6, é possível verificar o exemplo do evento `MessageEvent`, utilizado para exibir mensagens em tela. No método Execute é realizada a definição da mensagem e exibição em tela e, no método `Update`, aguarda-se uma confirmação do usuário para que esta mensagem seja fechada posteriormente. A variável `finishedExecution` é utilizada para que o evento seja considerado finalizado.

Quadro 6 - Código do evento de exibição de mensagem

```
public class MessageEvent : GameEvent
    {
        public string message;

        public override string GetNameText()
        {
            return "Mensagem";
        }

        public override string GetDescriptionText()
        {
            return message;
        }

        public override void Execute()
        {
            MessagePanel.main.SetMessage(message);
            MessagePanel.main.Toggle(true);
        }

        public override void Update()
        {
            if (Input.GetKeyDown(KeyCode.Z))
            {
                MessagePanel.main.Toggle(false);
                finishedExecution = true;
            }
        }
```

Fonte: elaborado pelo autor.

Assim como o editor, a cena de execução possui uma rotina similar de instanciamento de mapas dentro da classe `GameMasterBehaviour` seguindo os métodos `InstantiateMap`, `InstantiateLayer` e `InstantiateEntities`; porém, eles utilizam menos componentes de interação visto que a lógica é ditada principalmente pela própria classe `GameMasterBehaviour` e pela classe `PlayerBehaviour`, que faz o controle de movimento e interação do jogador para a execução dos eventos supracitados. A Figura 9 mostra a comparação do instanciamento do mapa do editor com o mapa da execução do jogo.

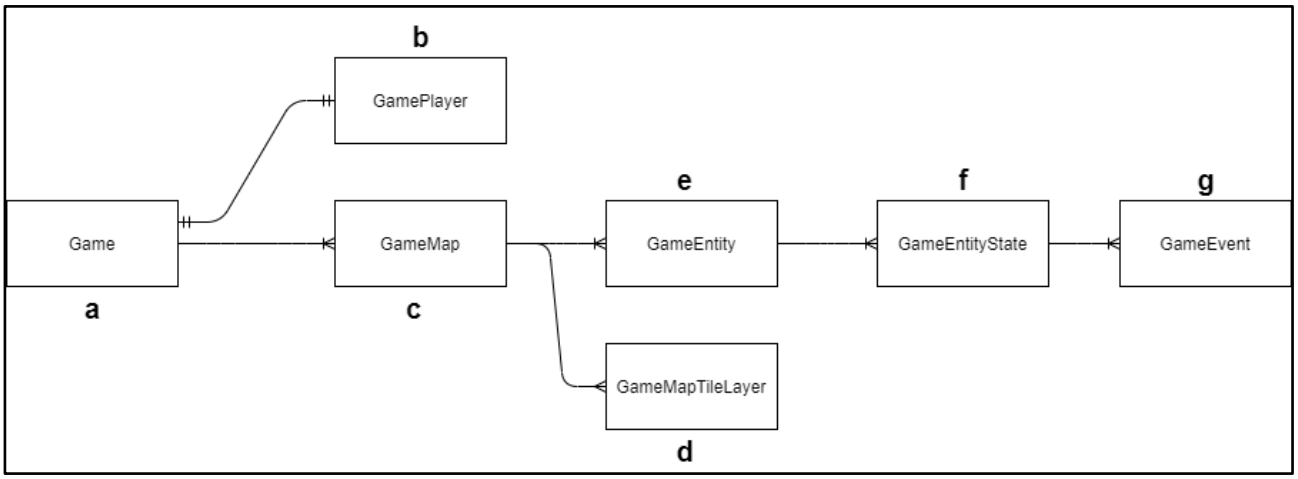
Figura 8 - Comparação do mapa de edição e mapa de execução do jogo

Fonte: elaborado pelo autor.

3.2.3 Persistência

Conforme mostrado no diagrama da Figura 10, a modelagem da ferramenta é realizada visando persistir os dados necessários para a execução do jogo de acordo com a configuração realizada através dos painéis do editor. Em seguida, o Quadro 7 descreve brevemente cada modelo utilizado e seu papel dentro do contexto da ferramenta.

Figura 9 - Modelagem dos objetos para persistência



Fonte: elaborado pelo autor.

Quadro 7 - Descrição dos objetos da modelagem para persistência

Identificador	Descrição
a	`Game` contém uma lista de mapas do jogo e a referência ao objeto com configurações do jogador;
b	`GamePlayer` contém informações do jogador, como imagem e mapa inicial do jogo;
c	`GameMap` possui informações do mapa em questão e referência às camadas pintadas no editor, além da lista de entidades existentes no mapa;
d	`GameMapTileLayer` contém um array bidimensional de inteiros que representam os sprites pintados no editor nesta camada, utilizados para renderização dos gráficos em tela;
e	`GameEntity` possui as informações de identificador e estados de uma entidade;
f	`GameEntityState` possui as condições de execução de um estado e uma lista dos eventos a serem executados;
g	`GameEvent` é uma classe abstrata que possui as informações referentes a um evento, variando de acordo com a implementação.

Fonte: elaborado pelo autor.

Para a persistência dos dados seguindo a modelagem, foi criada a interface `IPersistent` para obter e definir dados de uma classe que a implementa; a classe `PersistenceData`, que age como um dicionário de informações genéricas mas serializáveis; e a classe `Persistor`, que realiza a serialização e deserialização utilizando variáveis de tipo primitivas ou `PersistenceData`. Também foram criadas alguns métodos de extensão que convertem objetos complexos para tipos primitivos, como é o caso da `struct Vector2`.

O código do Quadro 8 mostra o processo de uso de reflexão para a obtenção de um valor dentro de um objeto do tipo `PersistenceData`, retornado pela interface `IPersistence` implementada por todas as classes mostradas no Quadro 6 referente a modelagem da ferramenta.

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 gerem a lógica dos jogos construídos são modeladas visando implementar uma vertente de ECS (ver seção 2.2), onde `GameEntity` são as entidades e os eventos persistidos por `GameEvent` 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.

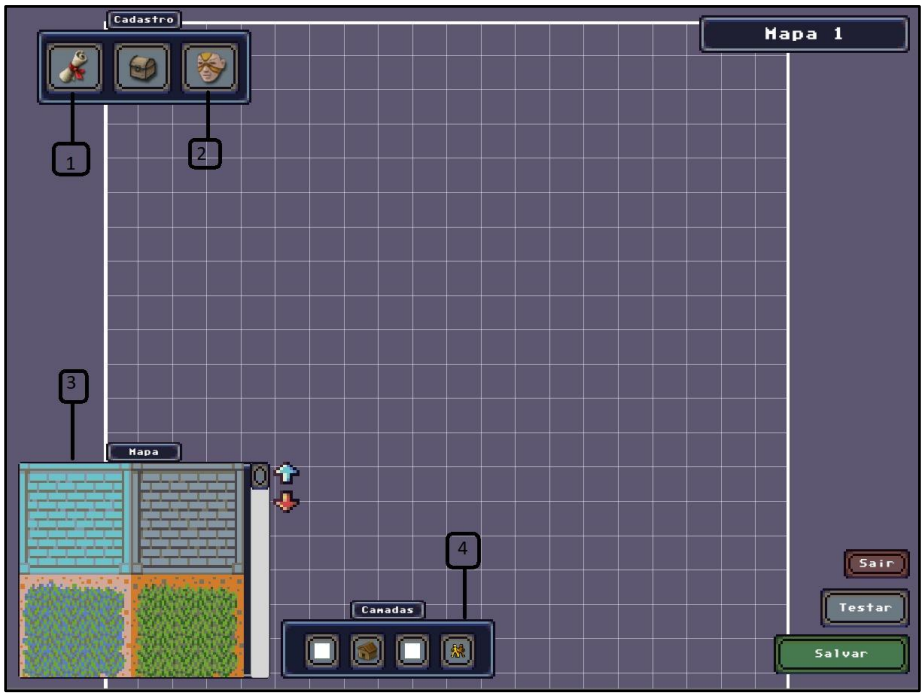
3.3.1 Mapas e Jogador

O fluxo dos jogos criados pela ferramenta é baseado nos mapas criados. Para criar um mapa, deve-se clicar no botão Mapas (identificador 1), conforme Figura 10. 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 (identificadores 3 e 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 (identificador 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.

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.

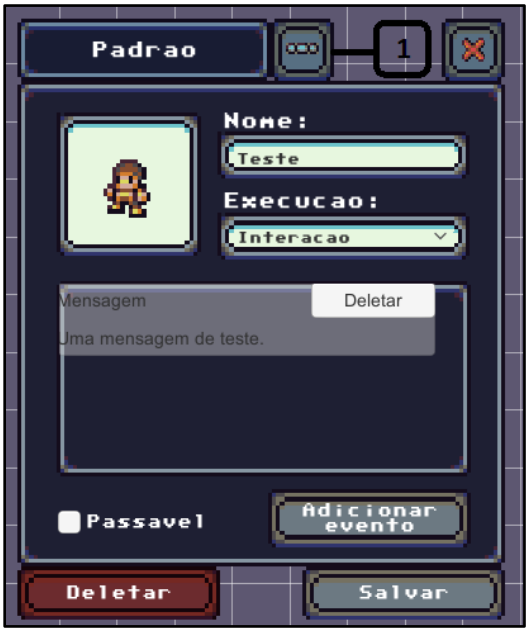
3.3.3 Estados, Interruptores e Variáveis

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 (identificador 1), demonstrado na Figura 11, 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.

3.3.4 Execução do Jogo

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.

Os controles utilizados são as teclas direcionais para movimento e o caractere "Z" para interação.

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.

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

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

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.

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.

SALDANHA, Ana Alayde; BATISTA, José Roniere Morais. **A concepção do role-playing game (RPG) em jogadores sistemáticos**. Psicologia: ciência e profissão, v. 29, n. 4, p. 700-717, 2009.

(A SER FORMATADO)

https://www.teses.usp.br/teses/disponiveis/48/48134/tde-30082010-104541/publico/ARTUR_ALVES_DE_OLIVEIRA_CHAGAS.pdf

http://www.nce.ufrj.br/sbie2003/publicacoes/paper71.pdf

https://geekandsundry.com/role-playing-gamers-have-more-empathy-than-non-gamers/

The Legend of Zelda Instruction Booklet. Nintendo of America, Inc. p. 28.

ECS Game Engine Design https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1138&context=cpesp

Overwatch ECS https://www.youtube.com/watch?v=W3aieHjyNvw&t=839s

IRDC US 2015 - Brian Bucklew, Data-Driven Engines of Qud and Sproggiwood https://www.youtube.com/watch?v=U03XXzcThGU&t=1389s

Postmortem: Thief: The Dark Project https://www.gamasutra.com/view/feature/3355/postmortem_thief_the_dark_project.php

A Data-Driven Game Object System https://www.gamedevs.org/uploads/data-driven-game-object-system.pdf

https://docs.unity3d.com/Manual/Components.html

https://web.archive.org/web/20100204091328/http://www.cs.indiana.edu/~jsobel/rop.html

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/concepts/reflection