PLAE: Jogo para Auxiliar o Ensino de Princípios e Valores Humanos para Crianças e Adolescentes em uma ONG

Luís Ricardo Mello Friess[1], Gustavo Stangherlin Cantarelli[1]

[1]Curso de Sistemas de Informação – Universidade Franciscana
CEP 97010-032 – Santa Maria – RS – Brasil

l.ricardo@ufn.edu.br, gus.cant@gmail.com

Abstract. *This article presents the development of a game aimed at Non-Governmental Organizations (NGOs), where mini-games are allowed for differents age groups and aimed at learning human principles and values. The game will be developed in C# language using Unity game engine, being built both 2D and 3D games. The methodologies chosen for game development were XGD and GAMA, two methodologied aimed at the development of digital games.*

Resumo. *Este artigo apresenta o desenvolvimento de um jogo intitulado PLAE voltado para as Organizações Não Governamentais (ONGs), que contará com um jogo para crianças de diferentes faixas-etárias voltados ao aprendizado de princípios e dos valores humanos. O jogo foi desenvolvido em linguagem C# com uso do motor de jogos Unity, tendo sido construído em 3D. As metodologias escolhidas para o desenvolvimento do jogo foram a XGD e a GAMA, duas metodologias voltadas ao desenvolvimento de jogos digitais.*

1. Introdução

As Organizações Não Governamentais (ONGs) são instituições sem fins lucrativos que buscam tirar pessoas - de qualquer faixa-etária - do mundo das drogas, do alcoolismo e do tráfico, ou até mesmo apenas conseguir um lar para que essas pessoas possam ter ao menos um local para dormir.

Dentre essas ONGs, existem algumas que possuem reuniões periódicas, são comandadas por instrutores e atendem ao público infanto-juvenil, com uma média de 6 aos 14 anos de idade. Nessas reuniões, são tratados assuntos que tendem a ensinar a esse público os valores humanos juntamente com os princípios presentes na organização.

A chegada dessas crianças e adolescentes nessas ONGs é um fato muito impactante na vida delas. Pode ser pelo fato de que terão um local mais seguro durante essa fase na vida, talvez pela mudança que vai acontecer na sua vida e/ou na de suas famílias. O certo é que haverá uma mudança e elas deverão ter de se adaptar à nova rotina e ao novo ambiente em que serão apresentadas.

Entretanto, a mudança muitas vezes é algo que as pessoas, e principalmente crianças e adolescentes, demoram a se acostumar e a conseguirem digerir as novas etapas que estão acontecendo e as que estão por vir. De acordo com Abdo (2018), mudanças mexem com as pessoas por causa do apego, da comodidade e da facilidades que o apego traz. Nessa rotina, a quebradura provoca angústia, por não saber como será o novo. Entender os sentimentos das crianças, ajuda muito na adaptação.

Considerando essa dificuldade de adaptação e o fato de se tratar de crianças e adolescentes, os meios presentes na grande parte das ONGs, por muitas vezes, é algo que não atrai muito a atenção deste público. Seja pelo fato de que alguns não tiveram a possibilidade de estarem em escolas e por isso não possuem a condição de ler livros expostos pelos instrutores ou talvez pela falta de recursos da ONG em sua estruturação, o que não a possibilita de ter outros meios para esse ensino.

Diante disso, o presente trabalho apresentou um novo método de ensino de um princípio presente em uma ONG, através do uso de boas práticas de duas metodologias, sendo elas a XGD (*Extreme Game Development*) e a GAMA (*Game Agile Methods Appplied*). Tem-se como principal elemento uma fórmula que traz figuras lúdicas, sons, cenários e todo o enredo voltado ao público infanto-juvenil.

1.1. Justificativa

Tendo em vista que as atividades realizadas em ONGs podem se tornar um tanto quanto monótonas ou repetitivas, surgiu a ideia da criação de um jogo para que o aprendizado destas crianças seja mais atrativo e lúdico, a fim de aprender-se os valores humanos e os princípios que essa Organização busca ensinar. O jogo pode ser trabalhado de várias formas diferentes, com a possibilidade de mostrar-se mais descontraído e interessante para o público infantil.

1.2. Objetivo geral

Este trabalho tem como objetivo o desenvolvimento e a implementação de um jogo (denominado PLAE – PLay Amor-Exigente) com várias figuras lúdicas (minijogo, músicas e cores), voltado a um princípio dentro da ONG, para ajudar o aprendizado de crianças presentes em uma Organização Não Governamental sobre os valores humanos, tendo em vista que muitas vezes elas não passam por essa etapa quando ainda pequenas.

1.3. Objetivos específicos

Os objetivos específicos deste trabalho são:

- Pesquisar e estudar jogos em Unity;
- Utilizar boas práticas dos métodos XGD e GAMA para auxiliar no desenvolvimento do trabalho;
- Criar um jogo estilo *tower defense* com o intuito de auxiliar na aprendizagem de valores humanos;
- Estudar e utilizar a linguagem de programação C#.

2. Referencial Teórico

Nesta seção, são apresentadas as pesquisas relacionadas com o objetivo de fundamentar o trabalho com um estudo sobre ONGs e, em seguida, as tecnologias que foram utilizadas no trabalho, sendo elas o motor de jogos Unity e a linguagem de programação C#.

2.1. Organizações não governamentais

As Organizações Não Governamentais (ONGs) são entidades que não têm fins lucrativos e realizam diversos tipos de ações solidárias para públicos específicos. Podem atuar em

várias áreas, como saúde, educação e economia, e podem ser desde âmbito local até internacional [Sebrae 2019].

De acordo com Faria (2017), as ONGs também podem ser chamadas de "terceiro setor". Essas organizações sem fins lucrativos podem ser particulares ou públicas, desde que o objetivo não seja a geração de lucros e, se houver geração de lucros, que sejam destinados para o fim que se dedica a organização.

2.1.1. Organização Amor-Exigente (AE)

A organização Amor-Exigente foi fundada em 1984 com o intuito de auxiliar familiares de dependentes químicos e pessoas com comportamentos inadequados. Através de um programa de auto e mútua ajuda, o AE busca a reorganização familiar, sensibilizando as pessoas e levando-as à percepção de mudar as suas vidas a partir de si mesmas. A AE já está presente por todo o Brasil, Argentina e Uruguai [Amor-Exigente 2018b].

Este programa é praticado por meio de 12 princípios éticos e 12 princípios básicos, sendo eles: identificador, humanizador, protetor, valorizador, libertador, influenciador, preparador, esperançador, apoiador, cooperador, organizador e compensador [Amor-Exigente 2016].

2.1.1.1. Amor-Exigentinho

O Amor-Exigentinho é um subgrupo do AE e propõe-se a promover a prevenção do uso de drogas com crianças e adolescentes com idades entre 6 e 14 anos por meio de ações educativas, baseando-se na arte, no lúdico e na literatura infantil. Essa abordagem, além de ser uma ação preventiva, também promove valores morais e éticos, motivando as crianças e adolescentes a se distanciarem de tudo que os torna vulneráveis ao crime, às drogas e à corrupção em qualquer de suas formas [Amor-Exigente 2018a].

Os encontros acontecem semanalmente com duração de duas horas com as crianças, adolescentes e jovens, separados por faixas etárias. Nesses encontros, são realizadas algumas atividades, como brincadeiras que estimulam o lazer construtivo, brincadeiras para socialização da criança, leitura de livros, atividades para propiciar a cooperação, entre outras [Amor-Exigente 2018a].

2.2. Tecnologias

2.2.1. *Games Engines* (Motores de jogos)

Um motor de jogos (*game engine)* é um software - ou um conjunto de bibliotecas - que simplifica o desenvolvimento de jogos ou de simulações em tempo real para videogames e computadores. A funcionalidade fornecida por um motor de jogos inclui um motor de renderização (*rendering engine*), sendo para gráficos 2D ou 3D, um motor de física (*physics engine*), um suporte à sons, uma linguagem de *script*, um porte à animação, suporte à inteligência artificial e um *scene graph* [Kobashikawa 2007].

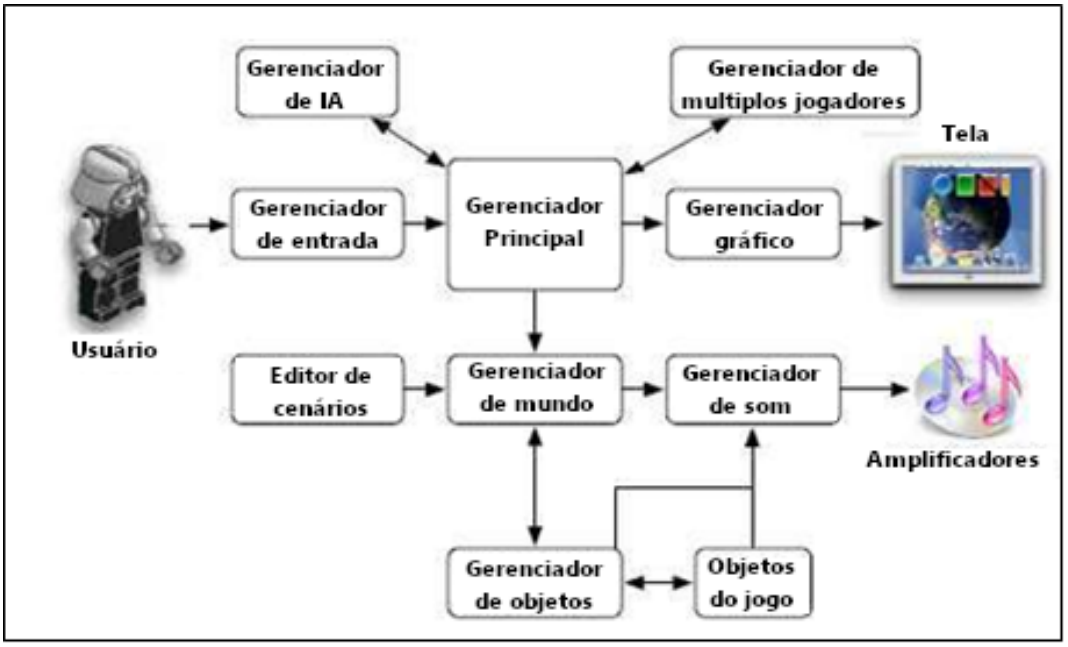

Figura 1. Arquitetura genérica de um motor de jogos [Kobashikawa 2007]

Conforme ilustra a Figura 1, um motor de jogos possui basicamente essa estrutura, que, conforme Battaiola *et al.* (2001), pode ser vista com mais detalhes abaixo:

- *Gerenciador de Entrada:* Encarregado de identificar eventos nos dispositivos de entrada e encaminhar para outro módulo que irá processá-lo;
- *Gerenciador Gráfico:* Encarregado de realizar o processamento necessário para transformar a cena criada em dados que sejam suportados pelas rotinas do sistema para desenho na tela;
- *Gerenciador de Som:* Encarregado pela execução de sons a partir de eventos, devendo ser capaz de converter e processar amostras de som;
- *Gerenciador de Inteligência Artificial:* Encarregado de gerenciar o comportamento dos objetos controlados pela máquina, realizando ações de acordo com o atual estado do jogo e seguindo algumas regras;
- *Gerenciador de Múltiplos Jogadores:* Encarregado de permitir que jogadores do mesmo jogo se comuniquem entre si. Logo, gerenciando a conexão e a troca de informações entre os jogadores;
- *Gerenciador de Objetos:* Encarregado de gerenciar um grupo de objetos do jogo, envolvendo armazená-los em alguma estrutura de dados e controlar o ciclo de vida deles. Em geral, um jogo possui vários gerenciadores de objetos;
- *Objeto do Jogo:* Representa uma entidade que faz parte do jogo e todas as informações necessárias para que seja gerenciado e manipulado. Tais informações podem ser posição, velocidade, dimensão, etc.;
- *Gerenciador de Mundo:* Encarregado por armazenar o estado atual do jogo, contando com diversos gerenciadores de objetos para realizar essa tarefa. Geralmente, associa-se a um editor de cenários o estado inicial de cada nível do jogo;

- *Editor de Cenários:* Ferramenta para a descrição do estado do cenário do jogo de forma visual e sem necessidade de programação. Geralmente, utilizada para gerar instâncias de jogos (níveis);
- *Gerenciador Principal:* Faz a ligação entre a troca de informações e algumas partes do jogo.

2.2.2. Motor de jogos Unity

De acordo com Gasparotto (2014), o motor de jogos Unity, além de auxiliar muito na construção de jogos 3D e 2D, é uma ferramenta que possui um estilo de programação e organização muito especial, além de simples. O ponto alto da ferramenta se dá ao fato de ter objetos já prontos, facilitando, assim, a criação de cenários e deixando mais livre para o desenvolvedor se deter aos *scritps* de programação.

Uma das vantagens da Unity, em relação a outros motores de jogos, é que ela traz a possibilidade de utilizar elementos criados por outros usuários em criações próprias através da *Asset Store*, uma loja presente dentro da plataforma que disponibiliza vários pacotes com modelos e, por vezes, até projetos completos, onde o desenvolvedor pode conhecer e aprender mais sobre a Unity. Muitos desses pacotes são gratuitos para todos os desenvolvedores, mas há também alguns pagos [Gasparotto 2014].

Conforme Dias *et al.* (2015), a Figura 2 mostra uma captura de tela do ambiente principal da Unity (*Unity Editor*), no qual possibilita-se a criação de cenários, personagens, objetos, entre outros, através do "arrastar e soltar" (*drag-and-drop*). Esse ambiente contém 6 painéis principais altamente configuráveis, onde o desenvolvedor consegue manipular objetos, adicionar texturas, testar o jogo em seu estado atual, adicionar novos objetos, etc.

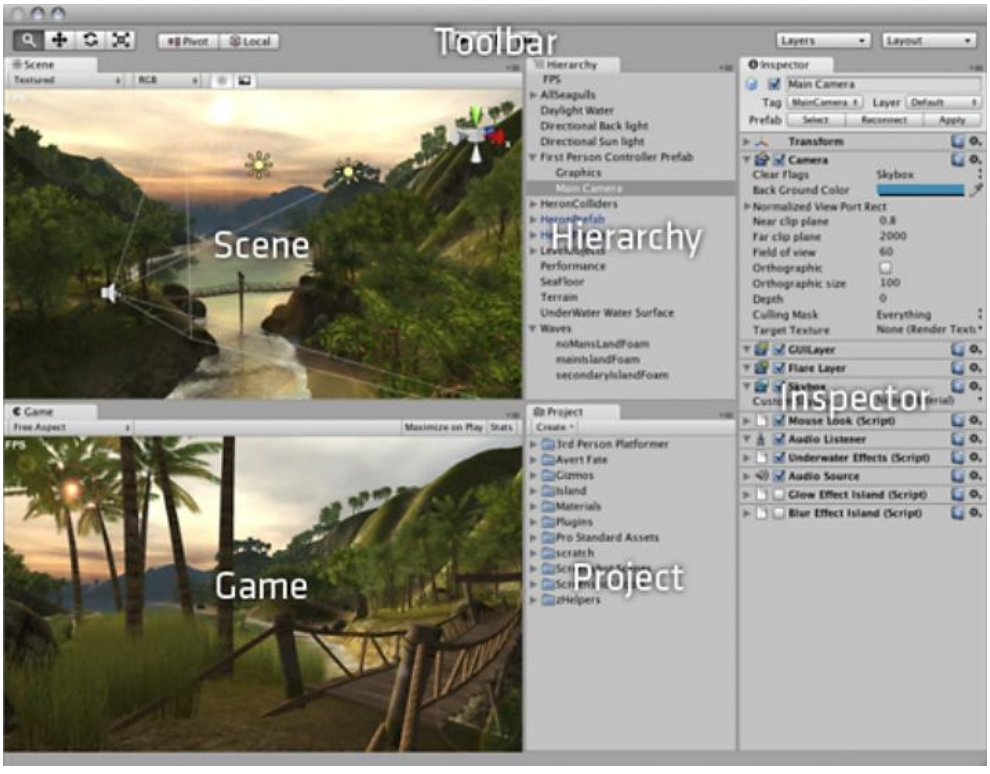

Figura 2. Ambiente de Desenvolvimento na ferramenta Unity [Dias 2015]

Ainda conforme Dias *et al.* (2015), seus principais painéis podem ser descritos de forma funcional presentes no Anexo A, contendo uma breve descrição sobre cada um dos painéis principais do *Unity Editor*.

2.2.3. Linguagem de Programação C#

O C# é uma linguagem elegante e fortemente tipada orientada a objetos. O C# permite que os desenvolvedores criem muitos tipos de aplicativos seguros e robustos que são executados no ecossistema do .NET. [Microsoft 2020].

Com a linguagem de programação C#, é possível fazer o *scripting* (programação orientada a eventos) no Unity, que serve para dizer como seu *GameObject* (objetos presentes dentro do jogo) deve se comportar. Com isso, os *scripts* gerados com C# são associados ao *GameObject* e essa interação compõe a *gameplay*. A criação dos *scripts* é diferente de uma programação pura, pois se trata de um objeto e seus comportamentos [Unity 2020].

2.3. Modelagem

Existem diversas metodologias para o desenvolvimento de jogos, como: *Game Waterfall Process* (GWP), *Game Unified Process* (GUP), Scrum, entre outros. E, dentre elas, estão presentes a *Extreme Game Development* (XGD) e a *Game Agile Methods Applied* (GAMA), que terão suas boas práticas utilizadas.

2.3.1. *Extreme Game Development* (XGD)

O XGD foi criado em 2003 por Thomas Demachy através de um artigo enviado ao portal Gamasutra [Demachy 2003]. Sendo definido no artigo como "método ágil para produção de jogos", é fortemente ligado ao método ágil *Extreme Programming* (XP), tendo seus princípios e a grande maioria de suas práticas sendo aplicadas às duas metodologias [Brauwers 2011].

No XGD existe a preocupação em diminuir problemas encontrados em projetos de jogos eletrônicos, como desenvolvimento atrasado, mudanças de requisitos por parte das *publishers*, entre outros. Baseia-se fortemente nos princípios publicados no Manifesto Ágil e nos quatro valores defendidos pelo XP: simplicidade, comunicação, *feedback* e coragem.

Algumas características e práticas da metodologia XGD são:

- Utilização da *Unifield Modeling Language* (UML);
- Versionamento;
- Ciclos de desenvolvimento de 3 semanas;
- Integração da equipe suportada por ferramentas apropriadas.

2.3.2. *Game Agile Methods Appplied* (GAMA)

A metodologia GAMA foi criada em 2008 por Petrillo e utiliza de algumas boas práticas da UML, como o uso dos diagramas de casos e uso e diagramas de atividades [Brauwers 2011]. Tem-se como resultado deste estudo um conjunto de práticas ágeis agrupadas em forma de metodologia específica para o desenvolvimento de jogos [Petrillo 2008].

Segundo Petrillo (2008), o GAMA, como um conjunto de práticas, pode ser organizado para a execução básica de um projeto de desenvolvimento de jogos. Para isso, é dividido em duas fases principais: fase exploratória e fase de execução.

A fase exploratória compreende na etapa onde os conceitos básicos do jogo são definidos, geralmente com uma ideia/inspiração de como um jogo pode ser feito, e é formada pelas seguintes atividades: elaborar conceito do jogo, determinar autores e histórias de usuário, elaborar histórias de usuário e planejar interações.

A fase de execução compreende no conjunto de atividades de planejamento e execução das tarefas do jogo, sendo formada pelas seguintes atividades: definir e estimar tarefas, reunião de pé, sessão de modelagem útil, sessão de implementação, executar atividades de qualidade, integrar versão, reunião de retrospectiva e entregar versão final.

3. Trabalhos Relacionados

3.1. Jogo Educacional EDUCPLAY

EDUCPLAY é um jogo desenvolvido com o propósito de ajudar o desenvolvimento de alunos em atividades relacionadas ao letramento do ensino básico, sendo elas um jogo de caça-vogais e outro baseado em descobrir nomes das imagens que aparecem na tela. Tem como tecnologias utilizadas o Unity e a linguagem de programação JavaScript e o SGBD utilizado foi o MySQL [Ferreira *et al*. 2018].

3.2. Jogo Educacional Apae Games

No trabalho de Mota e Alencar (2015), o jogo Apae Games tem como objetivo auxiliar a aprendizagem e a educação diária de crianças com deficiência intelectual. É dividido em 3 grandes módulos para ajudar nessas principais áreas, tendo inicialmente os módulos de Higiene, Alimentação e Memória. As tecnologias utilizadas para o desenvolvimento desse jogo foram o motor de jogos Unity juntamente com a linguagem de programação C# para a programação dos *scripts* e outras ferramentas adversas do Unity, como Adobe Photoshop, CorelDraw e Audacity, para a produção dos personagens e os sons do jogo.

3.3. Jogo Educacional Geneticats

Geneticats é um jogo educacional voltado ao ensino de genética explorando a área da 1° Lei de Mendel. Bastante ilustrativo e lúdico, tem como objetivo a simulação de cruzamento de gatos, de modo que, de acordo com os gatos disponíveis, os resultados serão diferentes. O desenvolvimento do jogo foi dado pela utilização do motor de jogos Unity e a linguagem de programação C# para os *scripts*, tendo como produto final um jogo para a plataforma Android [Madureira *et al.* 2018].

3.4. Considerações finais sobre os trabalhos correlatos

A partir dos trabalhos correlatos, notou-se que cada um dos mesmos tem pontos nos quais podem servir de inspiração, tanto na parte das figuras lúdicas quanto na parte do som e até mesmo na parte dos minijogos. São focados em jogadores com a faixa-etária baixa por se tratar de jogos educativos com foco no aprendizado de crianças.

O trabalho de Ferreira *et al.* (2018) tem como foco crianças na fase de aprendizagem do letramento, o que traz várias figuras infantis, um cenário mais focado

em cores e sons, o que se assemelha com o trabalho que está sendo proposto, tendo um foco maior em crianças e trazendo um cenário mais fantasioso e lúdico.

No caso do jogo de Mota e Alencar (2015), os diferentes minijogos e módulos presentes são pontos em que se assemelha com o trabalho a ser desenvolvido. Também sendo construído com base de cenário e som focados em crianças, alguns minijogos tiveram como inspiração artes e conceitos utilizados neste trabalho estudado.

Observa-se no trabalho de Madureira *et al.* (2018) um jogo focado em uma faixa-etária um pouco mais alta, mas ainda assim não perdendo a essência de jogo lúdico e com bastante figuras com variadas cores e sendo elaborado com um grau de dificuldade um pouco mais elevado. Servindo de base e inspiração à estrutura pré-jogo, com a parte de menu e seleção de fases dos jogos.

O diferencial do trabalho proposto é o agrupamento de vários conceitos presentes em cada um dos trabalhos, tendo assim, em um único jogo, vários minijogos com figuras lúdicas, cenários com muitas cores e com sons focados no público infanto-juvenil.

4. Metodologia

Nesta seção é apresentado o projeto do jogo utilizando as metodologias XGD e GAMA. A escolha dessas metodologias se deu pelo fato de que as mesmas se utilizam de diagramas da UML, por possuírem versionamento durante as versões desenvolvidas e também por terem seu foco voltado ao desenvolvimento de jogos eletrônicos.

4.1. Roteiro

O jogo foi focado principalmente no princípio protetor, por se tratar do estilo em que o jogo foi desenvolvido e pelo fato de ser um dos mais importantes princípios presentes dentro da ONG. A seguir, é apresentada a descrição do jogo, juntamente com seu funcionamento e suas regras:

- *PLAE:* Um jogo de estilo *tower defense* que tem como princípio proteger sua família dos riscos e dos inimigos que possam vir a feri-los de alguma forma, sendo fisicamente ou verbalmente. O jogador tem uma certa quantidade de moedas e precisa posicionar estrategicamente os objetos que vão proteger sua família dos inimigos. Serão 10 ondas de inimigos diferentes e, a cada onda que passa, a quantidade e os tipos de inimigos serão diferentes da anterior, dificultando, assim, sua missão. Ao final do jogo, aparece a quantas ondas de inimigos conseguiu proteger a sua família. Caso não consiga completar, ao final aparece uma mensagem de reflexão e o jogador é redirecionado novamente ao menu, podendo tentar novamente quantas vezes quiser.

4.2. Requisitos funcionais e requisitos não-funcionais

Segundo Kerr (2015), requisitos funcionais dizem respeito ao que o sistema deve fazer ou fornecer, ao modo como o sistema deve responder as entradas específicas ou determinadas situações que venham a aparecer. Já os requisitos não-funcionais referem-se a características globais do *software*, a critérios que qualificam os requisitos funcionais e aplicam-se ao sistema como um todo.

O jogo PLAE, tem seus requisitos funcionais e requisitos não-funcionais detalhados no Apêndice A, juntamente com suas funcionalidades.

4.3. Diagrama de atividades

De acordo com Sommerville (2007), os diagramas de atividades são destinados a mostrar as atividades que compõem um processo de sistema e o fluxo de controle de uma atividade para a outra. O início de um processo é indicado por um círculo preenchido; o fim, por um círculo preenchido dentro de outro círculo. Os retângulos com cantos arredondados representam atividades, ou seja, os subprocessos específicos que devem ser realizados.

O diagrama de atividades é utilizado pela metodologia GAMA para representar as suas tarefas, na qual são divididas em duas fases, e também é utilizado para representar o fluxo de como o jogo deverá se comportar [Petrillo 2008].

Na Figura 3, está representado o Diagrama de Atividades do PLAE, que coloca o jogador com o objetivo de defender sua família dos perigos que estão por vir. O jogador tem uma quantia de moedas e deve posicionar suas defesas, que são de três diferentes tipos, para não deixar que os perigos consigam completar todo o seu percurso. Ele pode adquirir novas defesas, evoluir as defesas possuintes e vendê-las. Caso ele consiga manter sua família longe dos perigos por dez ondas de inimigos, ele vence o jogo.

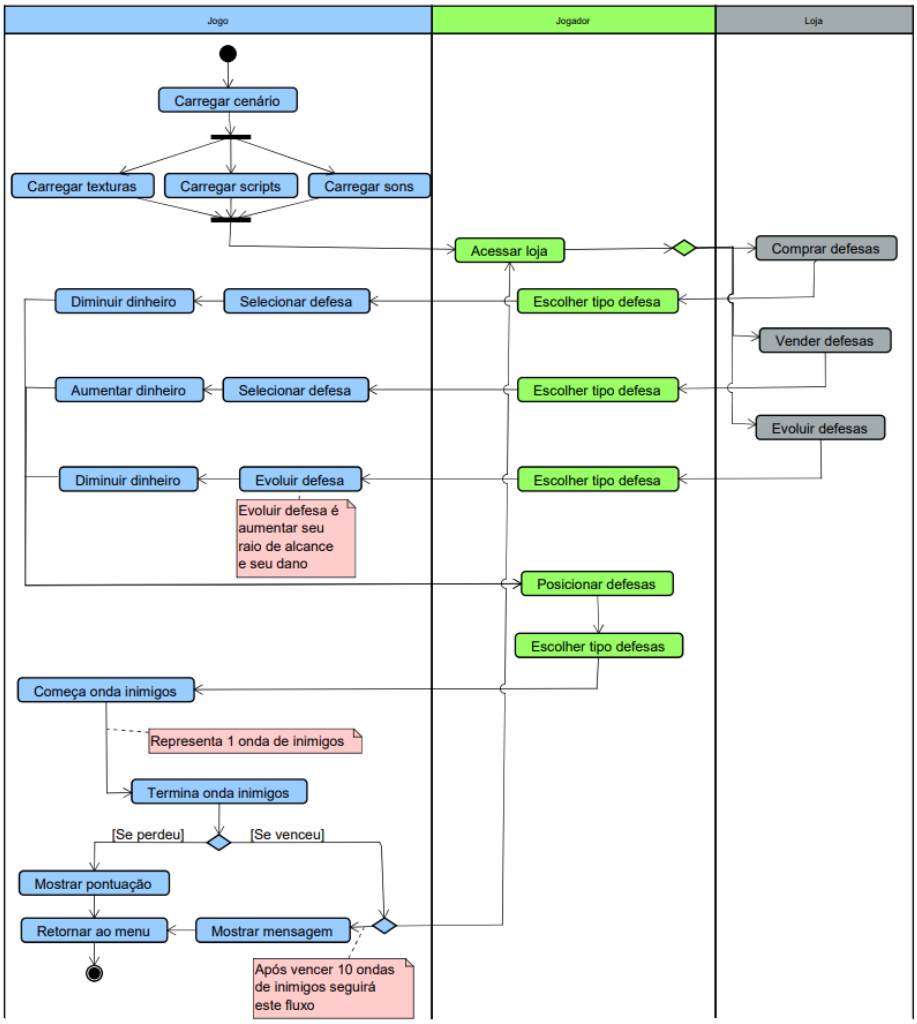

Figura 3. Diagrama de Atividades do PLAE

No Apêndice A, é apresentado o Diagrama de Casos de Uso e o Descritivo de Casos de Uso do Caso de Uso mais importante presente no jogo.

4.4. Diagrama de Classes

Segundo Sommerville (2007), os diagramas de classe são usados no desenvolvimento de um modelo de sistema orientado a objetos para mostrar as classes de um sistema e as associações entre essas classes. Em poucas palavras, uma classe de objeto pode ser pensada como uma definição geral de um tipo de objeto do sistema. Uma associação é um *link* entre classes que indica algum relacionamento entre essas classes.

A metodologia XGD, por ser baseada na metodologia XP, utiliza o diagrama de classes para representar como é a estrutura do jogo e também para a criação de "objetos genéricos", que podem ser reutilizados em outras partes do jogo [Demachy 2003].

O jogo PLAE tem seu Diagrama de Classes, podendo ser visualizado na Figura 4, trazendo as classes que estarão presentes no jogo, juntamente com seus métodos e como é feita a comunicação entre as classes.

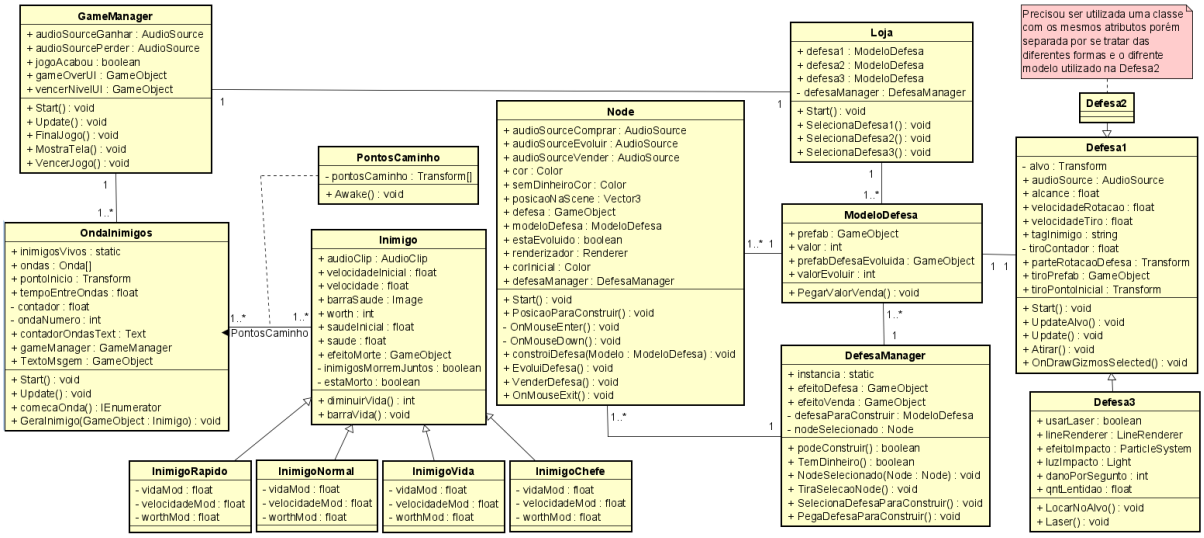

Figura 4. Diagrama de Classes do PLAE

4.5 Trechos de código

Nesta seção poderá ser visualizado alguns trechos de códigos que foram implementados para o funcionamento do jogo. Será apresentado trechos de dois *scripts* diferentes, um deles sendo o Node.cs e o outro sendo o DefesaManager.cs.

A Figura 5 represente o trecho do código referente ao DefesaManager.cs, no qual, é realizado dois testes (*pulic bool* PodeConstruir() e *public bool* TemDinheiro()) para verificar se o jogador possui moedas suficientes e se está tentando posicionar em uma posição correta. Após isso, é feito um teste com a função *public void* NodeSelecionado() para saber se o jogador está clicando em cima de uma defesa já posicionada, se sim, abrirá o menu da loja pra o jogador evoluir ou vender a sua defesa. Caso não esteja clicando em cima de uma defesa já posicionada, ele apenas posicionará a defesa no cenário.

A função *public void* TiraSelecaoNode() serve para quando o jogador clicar em algum local da tela, após a loja estar aberta, a mesma seja retirada da tela. E as funções *public void* SelecionaDefesaParaConstruir() e *public* ModeloDefesa PegaDefesaParaConstruir() são utilizadas para fazer o posicionamento da defesa no jogo, após o jogador já ter as selecionado anteriormente na loja.

```
//Testa caso o usuário pode ou não construir naquela posição (retorna true se pode construir e false se não pode construir)
2 referências
public bool PodeConstruir {
    get {
        return defesaParaConstruir != null;
    }
}

//Teste caso o usuário não tenha dinheiro suficiente para construir uma defesa
1 referência
public bool TemDinheiro {
    get {
        return EstatisticasJogador.Dinheiro >= defesaParaConstruir.valor;
    }
}

//Método para saber em qual node a defesa está selecionada para poder abrir a loja
1 referência
public void NodeSelecionado(Node node) {
    if(nodeSelecionado == node) {
        TiraSelecaoNode();
        return;
    }

    nodeSelecionado = node;
    defesaParaConstruir = null;

    nodeUI.SetAlvo(node);
}

//Para quando clicar em outro node fechar a loja
4 referências
public void TiraSelecaoNode() {
    nodeSelecionado = null;
    nodeUI.EscondeLoja();
}

//Método para que o jogador só consiga posicionar a sua defesa após escolher ela na loja
3 referências
public void SelecionaDefesaParaConstruir(ModeloDefesa defesa) {
    defesaParaConstruir = defesa;

    TiraSelecaoNode();
}

1 referência
public ModeloDefesa PegaDefesaParaConstruir() {
    return defesaParaConstruir;
}
```

Figura 5. Trecho do código DefesaManager.cs

Na Figura 6 pode ser visualizado o método *private void* OnMouseEnter() e *private void* OnMouseDown(). O método *private void* OnMouseEnter() é responsável por fazer o teste de onde o jogador está com o mouse posicionado, antes dele ser clicado. Esse teste é responsável por indicar ao jogador em forma de uma cor mais azulada/esverdeada caso o jogador possa posicionar a sua defesa, e uma cor mais avermelhada caso ele não possua moedas suficientes para fazer a compra da defesa.

Já no método *private void* OnMouseDown() é testado o clique do jogador, no primeiro teste, é feito para caso o jogador esteja clicando em uma posição válida no cenário. Já o segundo teste, é feito para caso ele esteja clicando em cima de uma defesa já posicionada, o que não o permitirá posicionar a defesa e sim abrir a loja. Também é feito um teste caso ele tente posicionar alguma defesa “por cima” do ícone da loja, o que não será permitido. Se nos testes anteriores foi permitido que o jogador posicionasse a defesa, será executada a função *private void* ConstroiDefesa() que será a responsável por fazer a construção da defesa na posição em que o jogador deseja.

```
//Função para detectar qual node o jogador está com o mouse (para mudar a cor do node)
Mensagem do Unity | 0 referências
private void OnMouseEnter() {

    //Função para caso o jogador tente posicionar uma defesa clicando em um icone "por cima" da loja, ele não deixar que a defesa seja poiscionada
    if (EventSystem.current.IsPointerOverGameObject()) {
        return;
    }

    //Caso o jogador não escolha nenhuma das defesas, a clicar em um lugar apenas retornar
    if (!defesaManager.PodeConstruir) {
        return;
    }

    //Caso o jogador possua dinheiro para construir pinta node de uma cor, caso não tenha pinta de outro
    if (defesaManager.TemDinheiro) {
        renderizador.material.color = cor;
    } else {
        renderizador.material.color = semDinheiroCor;
    }

}

//Função para posicionar a defesa conforme o clique do jogador
Mensagem do Unity | 0 referências
private void OnMouseDown() {

    //Função para caso o jogador tente posicionar uma defesa clicando em um icone "por cima" da loja, ele não deixar quea  defesa seja poiscionada
    if (EventSystem.current.IsPointerOverGameObject()) {
        return;
    }

    //Caso tente posicionar a defesa em um node inválido, retornar qual é o node que foi tentado posicionar
    if(defesa != null) {
        defesaManager.NodeSelecionado(this);
        return;
    }

    //Caso o jogador não escolha nenhuma das defesas, a clicar em um lugar apenas retornar
    if (!defesaManager.PodeConstruir) {
        return;
    }

    ConstroiDefesa(defesaManager.PegaDefesaParaConstruir());
    audioSourceComprar.Play();
}
```

Figura 6. Trecho do código Node.cs

A Figura 7 tem duas funções: *void* ConstroiDefesa() e *public void* EvoluiDefesa(). A função *void* ConstroiDefesa() é a responsável por fazer a construção da defesa na posição em que o jogador a escolheu. Nela é feita um teste para saber se o jogador possui moedas suficientes para fazer a compra da defesa e caso possua, será diminuído o valor da defesa em que o jogador escolheu posicionar. Após isso, é criado um clone da defesa escolhida pelo jogador e a mesma é posicionada no cenário, juntamente com uma animação de partículas para demonstrar que a defesa foi construída.

O método *public void* EvoluiDefesa() é o responsável por fazer a evolução da defesa, com isso, aumentando o seu raio de alcance e o dano que a defesa dá. Para isso, primeiramente é feito um teste para saber se o jogador possui moedas suficientes para fazer a evolução da sua defesa. Caso ele possua, o jogo vai diminuir a quantidade de moedas responsável pela evolução da defesa escolhida e após isso será destruído o modelo antigo da defesa que já estava posicionada no jogo. Após a destruição do modelo antigo, é gerado um clone do modelo da defesa evoluída e também é gerada uma animação de partículas, para indicar que a defesa foi evoluída. Por último, a variável estaEvoluido tem seu estado trocado de *false* para *true*, indicando assim que a defesa já está evoluída.

```
void ConstroiDefesa (ModeloDefesa modelo) {
    if (EstatisticasJogador.Dinheiro < modelo.valor) {
        Debug.Log("Dinheiro insuficiente para comprar!");
        return;
    }

    EstatisticasJogador.Dinheiro -= modelo.valor;

    GameObject _defesa = (GameObject)Instantiate(modelo.prefab, PosicaoParaConstruir(), Quaternion.identity);
    defesa = _defesa;

    modeloDefesa = modelo;

    GameObject efeito = (GameObject)Instantiate(defesaManager.efeitoDefesa, PosicaoParaConstruir(), Quaternion.identity);
    Destroy(efeito, 5f);

    Debug.Log("Defesa constuída!");
}

//Função para evoluir as defesas
1 referência
public void EvoluiDefesa() {
    if (EstatisticasJogador.Dinheiro < modeloDefesa.valorEvoluir) {
        Debug.Log("Dinheiro insuficiente para evoluir!");
        return;
    }

    EstatisticasJogador.Dinheiro -= modeloDefesa.valorEvoluir;

    //Destruir a defesa antiga
    Destroy(defesa);

    //Construir a nova defesa evoluída
    GameObject _defesa = (GameObject)Instantiate(modeloDefesa.prefabDefesaEvoluida, PosicaoParaConstruir(), Quaternion.identity);
    defesa = _defesa;

    GameObject efeito = (GameObject)Instantiate(defesaManager.efeitoDefesa, PosicaoParaConstruir(), Quaternion.identity);
    Destroy(efeito, 5f);

    estaEvoluido = true;

    audioSourceEvoluir.Play();

    Debug.Log("Defesa evoluída!");
}
```

Figura 7. Trecho do código Node.cs

5. Resultados

Para a avaliação do jogo, foram feitos testes individuais de todos os níveis e todas as funcionalidades que estão presentes no jogo. Salientando também, como resultado obtido, o fato de o trabalho ter sido publicado como artigo no XXIV Simpósio de Ensino, Pesquisa e Extensão – SEPE do ano de 2020.

O jogo foi apresentado à Organização Amor-Exigente em uma reunião realizada pelo Google Meet. Nessa reunião, estavam presentes alguns integrantes da Amor-Exigente juntamente com integrantes da Amor-Exigentinho, esta última sendo o foco do trabalho. O jogo foi recebido com muitos elogios, críticas construtivas e expectativa por parte da Organização, por se tratar de uma nova forma para a realização de atividades dentro da Instituição.

Devido ao grande número de interfaces e níveis que possui o jogo, serão apresentadas as de maior relevância para o trabalho. As imagens foram retiradas do produto final que constam dentro do jogo.

Na Figura 8a, é apresentado o menu principal do jogo. Nele é apresentada a logo do jogo, juntamente com os botões para entrar no menu de opções, no menu de instruções de como jogar, o botão para sair do jogo e o botão para jogar. Ao clicar no botão jogar, o jogador é levado à Figura 8b, que é a tela de escolha dos níveis, onde se pode escolher

qualquer um dos 10 níveis que estão presentes no jogo. Os níveis são de diferentes formatos, cores e dificuldades, trazendo assim, uma diversidade maior de níveis ao jogador.

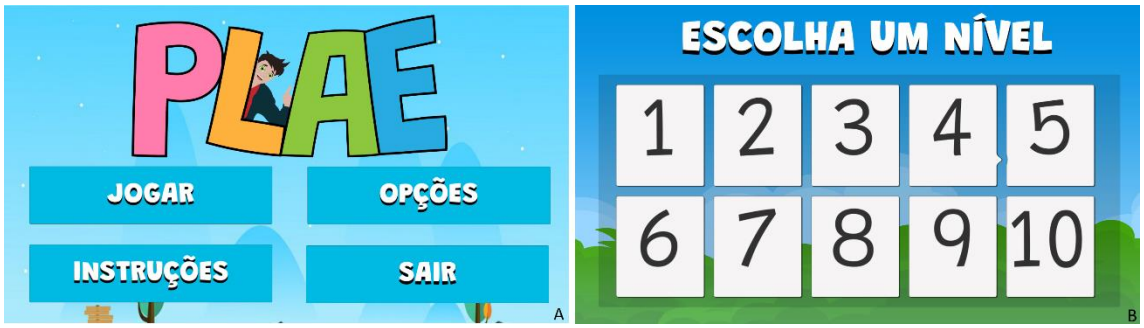

Figura 8. Menu Principal e Menu de escolha de níveis

Após o jogador escolher o nível, a Figura 9a mostra como é uma das telas apresentadas ao jogador, com uma breve descrição do nível, que é diferente entre os 10 níveis presentes do jogo. E, após o jogador clicar em jogar, na Figura 9b, pode-se visualizar a tela do nível 1.

Figura 9. Descrição do Nível 1 e Tela Inicial do Nível 1

6. Conclusão

Com o estudo realizado sobre Organizações Não Governamentais e como são realizadas as atividades de ensino de valores humanos e princípios, este trabalho trouxe uma nova forma de realizar este processo, um jogo digital com o intuito de auxiliar nas atividades presentes nessas organizações, trazendo de forma intuitiva e lúdica os ensinamentos buscados por essas instituições.

Na metodologia, por se ter optado pelo uso de boas práticas das metodologias XGD e GAMA, foram realizados versionamentos a cada etapa em que jogo foi construído, o que foi de suma importância, pois por diversas vezes, o jogo teve de ter sua versão regredida em função a *bugs* referentes a Unity. Também foram utilizados diagramas presentes na UML, o que facilitou a visualização de fluxos, como as classes do sistema tendem a realizar a comunicação entre si e como o usuário poderá fazer escolhas durante o jogo.

A escolha da Unity como motor de jogos possibilitou o desenvolvimento do jogo em 3D e trouxe grande ajuda nos modelos 3D utilizados no trabalho por possuir a *Asset Store*. Também foi de grande ajuda na parte das animações, visto que este motor de jogos já possui internamente uma parte exclusiva para poder tratar das animações que foram feitas durante o desenvolvimento do jogo.

A Linguagem de programação C#, por ter sido utilizada, trouxe uma grande ajuda no desenvolvimento do jogo, pois houve a possibilidade de ser realizada a programação orientada a objetos, o que facilitou na construção de classes e também no reaproveitamento de código.

Embora o jogo não tenha sido testado em uma Instituição, como pôde-se ver nos Resultados, ele foi muito bem aceito na Organização Amor-Exigente, a qual já demonstrou interesse em ter o jogo o mais breve possível para conseguir trazer essa nova dinâmica para dentro da Instituição. Também demonstrou, em um futuro, interesse em levar o jogo para outros países em que a Amor-Exigente já se faz presente.

Para trabalhos futuros, sugere-se que sejam desenvolvidos mais jogos e minijogos de diferentes tipos, plataformas e que abranjam outros temas relacionados a valores humanos, para que se possa ter uma abrangência maior de Instituições e Organizações. Com isso, será possível usufruir, de uma maneira mais lúdica, o ensinar para crianças e adolescentes.

Referências

Abdo, Ângela. (2018) “Como ajudar a criança a lidar com o processo de mudança?”, https://formacao.cancaonova.com/familia/pais-e-filhos/como-ajudar-a-crianca-a-lidar-com-as-mudancas-de-casa-escola-cidade/, Dezembro.

Amor-Exigente. (2018a) “Amor-Exigentinho”, https://amorexigente.org.br/amor-exigentinho/, Março.

Amor-Exigente. (2016) “Princípios”, https://amorexigente.org.br/principios/, Março.

Amor-Exigente. (2018b) “Quem Somos Amor-Exigente”, https://amorexigente.org.br/quem-somos/, Março.

Battaiola, A. L. *et al.* (2001) “Desenvolvimento de Jogos em Computadores e Celulares”, http://www.inf.ufrgs.br/~revista/docs/rita08/rita_v8_n2_p7a46.pdf, Março.

Brauwers, R. (2011) “Estendendo e Instanciando o *Game Agile Methods Applied* (GAMA)”, https://lume.ufrgs.br/bitstream/handle/10183/31035/000782125.pdf?se, Maio.

Demachy, T. (2003) “Extreme Game Development: Right on Time, Every Time”. https://www.gamasutra.com/view/feature/131236/extreme_game_development_right_on_.php, Maio.

Dias, D. R. C. *et al.* (2015) “A Realidade Virtual e o Motor de Jogos Unity”, https://www.researchgate.net/profile/Diego_Roberto_Dias/publication/277598033_A_Realidade_Virtual_e_o_Motor_de_Jogo_Unity/links/557a1cce08ae75363756fd91/A-Realidade-Virtual-e-o-Motor-de-Jogo-Unity.pdf, Março.

Faria, C. (2017) “ONGs (Organizações não-governamentais)”, https://www.infoescola.com/geografia/ongs-organizacoes-nao-governamentais/, Março.

Ferreira, S. A. G. *et al*. (2018) “APAE GAMES: Um jogo digital como ferramenta de aprendizagem para crianças com deficiência intelectual”, https://www.researchgate.net/profile/Leandro_Furtado2/publication/328346630_APAE_GAMES_Um_jogo_digital_como_ferramenta_de_aprendizagem_para_criancas

_com_deficiencia_intelectual/links/5be3499b92851c6b27aee08b/APAE-GAMES-Um-jogo-digital-como-ferramenta-de-aprendizagem-para-criancas-com-deficiencia-intelectual.pdf, Março.

Gasparotto, H. M. (2014) “Unity 3D: Introdução ao desenvolvimento de games”, https://www.devmedia.com.br/unity-3d-introducao-ao-desenvolvimento-de-games/30653, Março.

Kerr, Eduardo Santos. Gerenciamento de Requisitos. São Paulo: Pearson Education do Brasil, 2015. Disponível em PEARSON Biblioteca online.

Kobashikawa, D. (2007) “Estudo comparativo de ferramentas para motores de jogos RPG”, http://revista.uniplac.net/ojs/index.php/tc_si/article/download/771/483, Março.

Madureira, O. R. A. *et al.* (2018) “Geneticats”, http://www.sbgames.org/sbgames2018/files/papers/EducacaoShort/188193.pdf, Março.

Microsoft (2020) “Introdução a Linguagem C# e .NET”, https://docs.microsoft.com/pt-br/dotnet/csharp/getting-started/. Dezembro.

Mota, C. R. e Alencar, S. A. (2015) “ECUCPLAY – Jogo educativo destinado ao Letramento no Ensino Infantil”, https://www.researchgate.net/profile/Marcio_Alencar/publication/283719151_EDUCPLAY_-_Jogo_Educativo_destinado_ao_Letramento_no_Ensino_Infantil/links/56449a0308ae9f9c13e55328/EDUCPLAY-Jogo-Educativo-destinado-ao-Letramento-no-Ensino-Infantil.pdf, Março.

Petrillo, F. S. (2008) “Práticas Ágeis no Processo de Desenvolvimento de Jogos Digitais”, https://www.lume.ufrgs.br/handle/10183/22809, Maio.

Sebrae. (2019) “O que são ONGs?”, https://www.sebrae.com.br/sites/PortalSebrae/artigos/o-que-e-uma-organizacao-nao-governamental-ong,ba5f4e64c093d510VgnVCM1000004c00210aRCRD, Março.

Sommerville, Ian. Engenharia de Software. 10. ed. São Paulo: Prentice Hall, 2019. Disponível em PEARSON Biblioteca online.

Sommerville, Ian. Engenharia de Sofware. 9. ed. São Paulo: Prentice Hall, 2007. Disponível em PEARSON Biblioteca online.

Unity. (2020) “Codificação em C# em Unity para Iniciantes”, https://unity3d.com/pt/learning-c-sharp-in-unity-for-beginners, Março.

Anexo A. Principais Painéis da Unity

Tabela 2. Detalhamento dos painéis principais presentes na Unity

Painel	Descrição
Project	Por meio deste painel, pode-se acessar e gerenciar os recursos pertencentes ao projeto Unity, localizados no diretório escolhido ao se criar o projeto, apresentando tais recursos e as árvores de diretórios nos quais estes são armazenados.
Scene	Onde são criados os AVs (Ambientes Virtuais), dispondo os recursos disponíveis no painel Project por meio do "arrastar e soltar". Pode-se navegar neste painel de forma livre fazendo uso de teclado e mouse, visualizando e editando o posicionamento dos objetivos dentro do AV.
Hierarchy	Apresenta todos os contidos no AV, ou seja, os dispostos no painel Scene. Por meio deste, é possível o agrupamento de objetos, criando um sistema de coordenadas local para os objetos filhos com origem na posição global do objeto pai.
Game	Permite a visualização e interação com a execução do que vem sendo criado no painel Scene.
Toolbar	Consiste em controles básicos, como opções de navegação e posicionamento dos objetos para o painel Scene, e botões para execução, pausa e quadro por quadro para o painel Game.
Inspector	Ao selecionar qualquer objeto no painel Project, Scene ou Hierarchy, é apresentado no painel Inspector os componentes (malhas, textura, sons, animações, *scripts*, entre outros) do objeto selecionado. Pode-se alterar as propriedades dos componentes apresentados, visualizando tais alterações na edição do AV no painel Scene, ou em tempo execução no painel Game.

Apêndice A. Requisitos funcionais e não-funcionais do jogo PLAE

Tabela 1. Requisitos funcionais e não funcionais do jogo PLAE

RF.1 Loja
Descrição: O jogo terá um sistema de loja inteiro para que possam ser feitas as compras, vendas e evoluções das defesas.
Complexidade: Alto
RF.1.1 Comprar defesas
Descrição: O jogo deve permitir que o jogador compre e posicione as suas defesas conforme a quantidade de dinheiro que possui.
Complexidade: Médio
RF.1.2 Vender defesas
Descrição: O jogo deve permitir ao jogador que vendas as suas defesas e receba o dinheiro de volta.
Complexidade: Médio
RF.1.3 Evoluir defesas
Descrição: O jogo deve permitir ao usuário que faça evolução de alguma de suas defesas para que aumente o seu raio de alcance ou seu dano.
Complexidade: Alto
RF.1.4 Tempo para posicionar defesas
Descrição: O jogo terá um contador antes de começar as *waves* de inimigos para dar tempo de o jogador posicionar suas defesas.
Complexidade: Médio
RF.1.5 Posicionar defesa
Descrição: O jogo deve definir lugares pré-determinados para que o jogador consiga posicionar suas defesas.
Complexidade: Médio
RF.1.6 Tipos de defesas
Descrição: O jogo terá 3 tipos diferentes de defesa, cada um deles custando uma quantidade de dinheiro diferente.
Complexidade: Médio
RF.1.7 Raio de alcance
Descrição: Cada um dos tipos de defesa terá um raio de alcance diferente, dependendo do seu preço.
Complexidade: Alto
RF.1.8 Defesa vence
Descrição: Caso a defesa não deixe nenhum inimigo atravessar do ponto inicial até o ponto final, o jogo retorna ao menu com a pontuação do jogador e quantas *waves* ele venceu.
Complexidade: Médio
RF.2 Inimigos
Descrição: O jogo terá inimigos que percorrerão um caminho pré-definido de um ponto inicial até um ponto final.
Complexidade: Médio
RF.2.1 Destruir inimigos

Descrição: O jogo deve diminuir a vida dos inimigos até que chegue a 0, conforme o dano de cada defesa que está no raio de alcance do mesmo.
Complexidade: Médio
RF.2.2 Caminho para inimigos percorrerem
Descrição: O jogo deve pré-determinar um caminho para os inimigos percorrerem do ponto inicial até o ponto final.
Complexidade: Médio
RF.2.3 Tipos de inimigos
Descrição: O jogo terá 3 tipos de inimigos que surgirão com o avanço das *waves*.
Complexidade: Médio
RF.2.4 Inimigos vencem
Descrição: Caso algum inimigo consiga percorrer o caminho do ponto inicial até o ponto final, o jogo deve retornar ao menu com uma mensagem de reflexão.
Complexidade: Médio
RF.2.5 *Waves* de inimigos
Descrição: O jogo terá um sistema de *waves* de inimigos que terá sua dificuldade gradualmente aumentada conforme for completada.
Complexidade: Alto
RF.2.6 Barras de vida
Descrição: O jogo terá de mostrar a barra de vida dos inimigos em cima dos mesmos.
Complexidade: Médio
RF.3 Dinheiro
Descrição: O jogador começa com uma quantia definida de dinheiro e o gasta conforme cada uma de suas defesas são posicionadas.
Complexidade: Baixo
RF.3.1 Aumentar dinheiro
Descrição: A cada *wave* que o jogador vencer, ele receberá uma quantia em dinheiro para gastar em sua defesa na *wave* seguinte.
Descrição: Médio
RF.4 Carregar jogo
Descrição: O jogo deve fazer o carregamento de cenário, *scripts*, texturas e personagens antes de mostrar a tela para o usuário.
Complexidade: Baixo
RF.4.1 Carregar cenário
Descrição: O jogo deve fazer o carregamento do cenário antes de mostrar a tela para o usuário.
Complexidade: Baixo
RF.4.2 Carregar texturas
Descrição: O jogo deve fazer o carregamento de texturas antes de mostrar a tela para o usuário.
Complexidade: Baixo
RF.4.3 Carregar *scripts*
Descrição: O jogo deve fazer o carregamento dos *scripts* antes de mostrar a tela para o usuário.
Complexidade: Baixo
RF.4.4 Carregar sons

Descrição: O jogo deve fazer o carregamento dos sons antes de mostrar a tela para o usuário.
Complexidade: Baixo
Requisitos não funcionais
RNF.1 O jogo será desenvolvido para a plataforma *Windows* para ser executado em computadores no formado *desktop.*
RNF.2 O jogo será desenvolvido na linguagem de programação C#.
RNF.3 O jogo utilizará o motor de jogos Unity para o desenvolvimento e emulação.

Apêndice B. Modelo de Caso de Uso

De acordo com Sommerville (2019), o Diagrama de casos de uso identifica as interações individuais com o sistema. Eles podem ser documentados por meio de texto ou de *links* com os modelos UML que desenvolvem o cenário mais detalhadamente.

A metodologia GAMA utiliza os diagramas de casos de uso como parte de uma de suas etapas que se faz presente na fase exploratória. É utilizada para a criação de atores e estórias de usuários (descrição da necessidade do usuário), para, no final, indicar os seus relacionamentos [Petrillo 2008].

Na Figura 10, pode-se ver o Diagrama de Casos de Uso do jogo PLAE, contendo os elementos presentes no jogo para a interação do jogador.

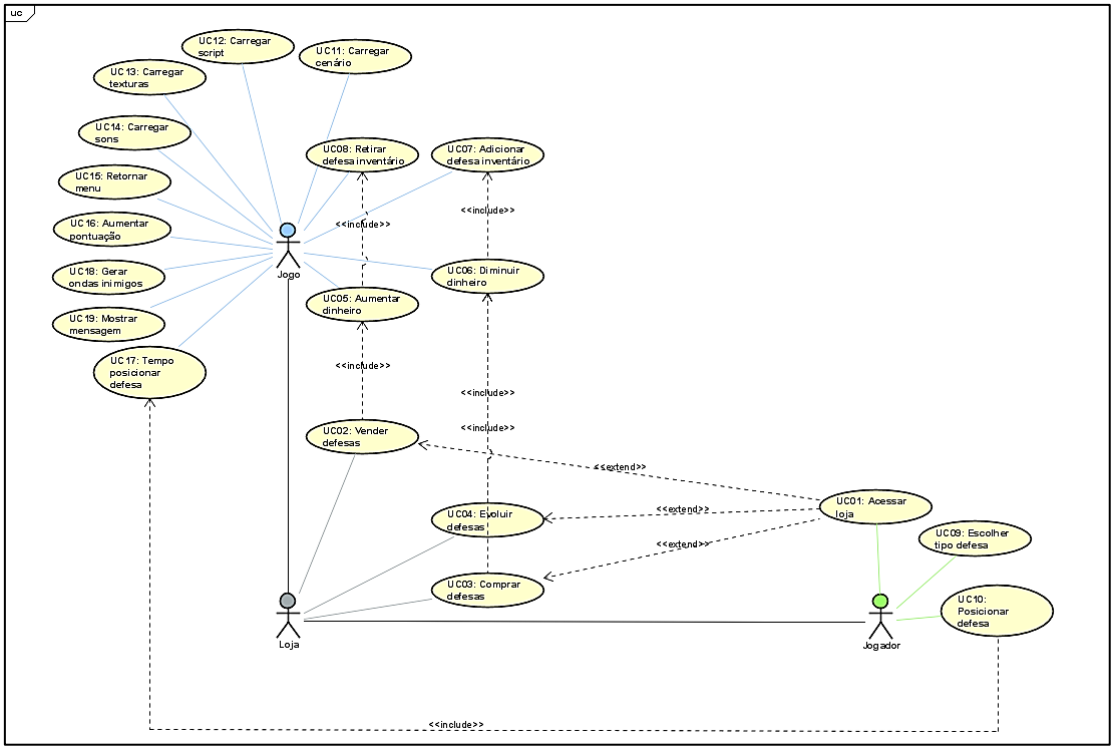

Figura 10. Diagrama de Casos de Uso do PLAE

Já na Tabela 3, pode-se observar o Descritivo do Caso de Uso Posicionar Defesa, que é um dos Casos de Uso mais importantes presentes no jogo.

Tabela 3. Descritivo do Caso de Uso Posicionar Defesa do PLAE

Identificação	UC10
Caso de Uso	Posicionar defesa.
Descrição	O jogador deve escolher e posicionar suas defesas antes que o tempo acabe.
Ator Principal	Jogador.
Ator(es) Secundário(s)	Jogo, Loja.

(Continua)

(Conclusão)

Pré Condições	
PC1	O jogador deve possuir as defesas no inventário para que possa posicioná-las.
Pós Condições	
PO1	O jogo deve retirar as defesas do inventário do jogador e as posicionar.
Fluxo Principal	
FP1	O jogador escolhe o tipo de defesa presente no seu inventário ou a compra na loja. [FA1] [FE1] [FE2] [FE4].
FP2	O jogador escolhe uma posição para que posicione sua defesa. [FE3] [FE4].
FP3	O jogo retira a defesa do inventário do jogador e a posiciona no cenário. [FE1] [FE2].
FP4	Fim do caso de uso.
Fluxo Alternativo	
FA1	1.1 - O jogador pode escolher as defesas: • Tipo1: Mais vida que as outras defesas. • Tipo2: Mais dano que as outras defesas. • Tipo3: Mais alcance que as outras defesas. 1.2 – O jogo retorna ao FP2.
Fluxo de Exceções	
FE1	Caso o jogador tenha a defesa escolhida no inventário, o jogo retorna ao [FP2].
FE2	Caso o jogador não tenha a defesa escolhida no inventário, o jogo deve informar ao jogador e retornar ao [FP1].
FE3	Caso o jogador tente posicionar a defesa em um lugar possível, o jogo posiciona a defesa e retorna ao [FP3].
FE4	Caso o jogador tente posicionar a defesa em um lugar errado, o jogo deve informar que não é possível, recoloca a defesa no inventário e retorna ao [FP1].
FE5	Caso o tempo acabe, o jogo deve começar as ondas de inimigos e retornar ao [FP4].
Regras de Negócio	
Casos de Uso Estendidos	
Casos de Uso Incluídos	UC17.
Observações	O jogador permanece no *looping* entre o [FP1], [FP2] e [FP3] entre as ondas de inimigos.