



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO



SKULL SURVIVAL

- Game Design Document -

Projeto Final de Programação

INF2102 - 2018.1



DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO

Autor

Abel González Mondéjar

Matrícula: 1713266

Orientador

Alberto Barbosa Raposo

Rio de Janeiro

Junho / 2018

Resumo

O desenvolvimento de um jogo eletrônico é uma tarefa complexa que envolve diversas etapas como, por exemplo, criação de enredo, cenário, personagens, edição de som, imagens e vídeos, programação, entre outras. O presente trabalho tem como objetivo desenvolver um jogo eletrônico 3D tipo *survival* levando em consideração a aplicação de Engenharia de Software nesse processo, através do levantamento de metodologias traduzidas para contexto de jogos e de ferramentas específicas. O jogo será desenvolvido para PC, e alternativamente os usuários interagirão com o jogo por meio do Microsoft Kinect para movimentar o jogador na cena. A ferramenta a usar é o conhecido motor de jogos Unity3D e a linguagem de programação C#. Será apresentada no *Game Design Document* a fundamentação teórica, o processo de desenvolvimento e ao fim, destacam-se os principais testes aplicados que garantem a qualidade do jogo desenvolvido.

Sumário

Sumário

Introdução	7
Motivação.....	7
Objetivo.....	8
Estrutura do trabalho.....	8
Convenções	9
1- Estado da arte	10
1.2- Principais Empresas	10
1.3- Comparação de Características.....	10
2- Análise do problema	11
2.1- Mercado-Alvo.....	11
2.2- Características Principais	11
2.2.1- Gênero.....	12
2.2.2- Plataforma	12
2.3- Analise dos requerimentos	12
2.3.1- Requerimentos funcionais	12
2.3.2- Requerimentos não funcionais	14
2.4- Casos de uso.....	15
2.4.1- Casso de uso Usuário	15
2.4.2- Casso de uso Jogador.....	18
2.5- Análise Técnica.....	19
2.5.1- Elementos Experimentais	19
2.5.2- Riscos.....	19
2.5.3- Recursos Estimados.....	20
2.5.4- Cronograma Estimado.....	20
3- Desenho da solução	21
3.1- Game Design	21

3.1.1- Arte e Vídeo.....	21
3.1.2- Elementos do Game Play	21
3.1.3- Animação e Arte 3D.....	22
3.1.4- Som e Música	22
3.2- Arquitetura da aplicação	23
3.2.1- Logica da aplicação	23
3.2.2- Diagrama de classes	25
3.3- Controle de versões.....	25
3.4- Organização do projeto.....	26
4- Implementação.....	28
4.1- Interface e interação	28
4.1.1- Interfaces do Jogo	28
4.1.2- Controles	30
4.2- Core Game Play	31
4.2.1- Jogador.....	31
4.2.2- Inimigos	31
4.2.3- Elementos do Game Play	33
4.3- Scripting e Component Based Design	34
4.4- Técnicas de Inteligência Artificial	35
4.4.1- Máquinas de Estado Finitas	35
4.4.2- Path finding.....	37
5- Testes	41
5.1- Testes alfas.....	41
5.2- Testes de sistema.....	42
5.3- Testes automatizados.....	44
Conclusões.....	48
Bibliografia	49

Sumário de tabelas

Tabela 1 RF01 Configurar dificuldade	12
Tabela 2 RF02 Configurar gráficos	13
Tabela 3 RF03 Calibrar Kinect	13
Tabela 4 RF04 Manipular player	13
Tabela 5 A1 Caso de Uso 1 – Jogar	16
Tabela 6 A1 Caso de uso 2 - Configurar jogo	16
Tabela 7 A1 Caso de uso 3 - Selecionar dificuldade	16
Tabela 8 A1 Caso de uso 4 - Configurar Gráficos	17
Tabela 9 A1 Caso de uso 5 - Sair do Jogo	17
Tabela 10 Riscos do desenvolvimento	19
Tabela 11 Cronograma de desenvolvimento do jogo	20
Tabela 12 Legenda das figuras usadas no diagrama de fluxo	23
Tabela 13 Controles com teclado e Kinect	30
Tabela 14 Teste jogador - Pausar jogo	42
Tabela 15 Teste jogador - Caminhar	42
Tabela 16 Teste Jogador - Agachar-se	43
Tabela 17 Teste jogador - Soco direito	43
Tabela 18 Teste jogador - Pontapé direito	43
Tabela 19 Teste jogador - Pegar objeto	44
Tabela 20 Teste jogador - Dano	44

Sumário de figuras

Imagem 1 Casos de uso do ator Usuário	15
Imagem 2 Casos de uso do ator Jogador	18
Imagem 3 Diagrama de fluxo da aplicação	24
Imagem 4 Diagrama de classes do projeto	25
Imagem 5 Hierarquia do projeto desde Unity	26
Imagem 6 Hierarquia do projeto desde VS Community	27
Imagem 7 Tela inicial do jogo.....	28
Imagem 8 Tela opções do jogo	29
Imagem 9 Jogo em execução	29
Imagem 10 Tela jogo em pausa	30
Imagem 11 Avatar do jogador	31
Imagem 12 Inimigo esqueleto branco	32
Imagem 13 Inimigo esqueleto vermelho.....	32
Imagem 14 Inimigo Esqueleto armado.....	33
Imagem 15 Inimigo porquinho voador	33
Imagem 16 Componentes do GameObject Inimigo1	34
Imagem 17 MEF do Inimigo	36
Imagem 18 MEF do inimigo – Walk.....	36
Imagem 19 MEF do inimigo – Death	37
Imagem 20 Nav Mesh do Terreno.....	38
Imagem 21 Navigation Mesh - Inimigo	39
Imagem 22 Classe EnemyMovement.....	39
Imagem 23 Método Update da classe EnemyMovement	40
Imagem 24 Classe NavMeshAgent	40
Imagem 25 Teste - Carga da cena.....	45
Imagem 26 Teste - GameObject com Rigidbody é afetado pela fisica	46
Imagem 27 Teste - Carga de um GameObject tipo Inimigo desde prefab	46
Imagem 28 Teste - Unity Edit Mode	47
Imagem 29 Teste - Unity Play Mode	47

Introdução

Neste projeto, descreve-se o processo de desenvolvimento do videogame chamado Skull Survival. Trata-se de um jogo estilo *survival* que foi criado utilizando Unity 2018.1.0f2f. O jogo consiste na ideia de sobreviver num terreno hostil dominando por inimigos. Desenvolvido sobre uma visão 3D, o jogo é ambientado num terreno onde o jogador precisará de se defender de adversários que o perseguem. O jogador precisará de pegar munições para sobreviver naquele cenário.

Este projeto foi desenvolvido com fins acadêmicos, embora aporte um ar fresco e interessante ao usar Microsoft Kinect para movimentar o jogador. As ferramentas escolhidas para sua realização decidiram-se ao suporte e ampla comunidade que possuem [1] [2] [3] e a experiência previa do desenvolvedor com elas.

Motivação

Há algum tempo, a indústria de videogames tem experimentado um boom. Todos os anos, há números recordes relacionados às vendas [4], que continuam a crescer ano após ano, apesar da crise global que tem sido vivenciada recentemente e permanece na ordem do dia. Ao mesmo tempo, o uso de Microsoft Kinect tornou-se popular como dispositivos orientados ao entretenimento relacionados a videogames [5] ou tratamento de doenças [6] [7], uma vez que muitos estão quase no mesmo nível de muitos computadores ou consoles convencionais em termos de desempenho.

A fase de desenvolvimento de um videogame sempre foi muito cara e precisou de muita gente para completá-lo; no entanto, existe agora a cultura da empresa independente ou como eles se chamam de empresas *indie*. Existem muitas ferramentas que hoje facilitam a criação de videogames para pequenos e médios equipamentos, entre eles, o motor que é utilizado neste projeto, Unity, que permite reduzir significativamente os tempos de desenvolvimento e, portanto, aumentar a oferta deste tipo de produtos no mercado.

Com este trabalho, tentamos fazer uma proposta diferente para os jogos que costumam ser vistos pelas lojas das principais plataformas de videogame e reviver um gênero que carece de ideias. O principal incentivo para a sua construção é perceber a possibilidade de movimentar com o Kinect a o *player*.

Objetivo

O objetivo deste trabalho é o desenvolvimento de um videogame desde zero com a ajuda do motor Unity, complementado com Microsoft Kinect pode se obter uma solução interessante. Além disso, neste projeto nos propor compreender as interações entre personagens e ambiente (máquinas físicas e de estado).

Não se pretende fazer vários níveis no *videogame*, mas sim uma demonstração para mostrar as possibilidades que estão disponíveis. Uma vez feito isso, a possível extensão do projeto e possíveis melhorias serão consideradas, assim como a viabilidade de todas as ferramentas externas que foram usadas.

Para a realização deste, um documento de design (GDD) foi elaborado, que será a pedra angular do projeto incorporando-se artefatos da engenharia de software. Descreveremos o design da interface, personagens e cenários, bem como sua funcionalidade incluído deste documento.

Estrutura do trabalho

Este documento apresenta a visão do Skull Survival, sendo o documento que formaliza o processo de desenvolvimento do jogo. Ele está dividido em cinco partes:

Estado da arte: serão apresentadas as principais empresas neste nicho do mercado e comparações de características com outras soluções similares.

Análise do problema: define-se o mercado alvo, as características principais do videogame, a análise dos requerimentos funcionais e não funcionais. Por outra parte, se apresenta a análise técnica sobre os riscos e cronograma estimado.

Desenho da solução: amostrasse itens referentes ao game design (arte e vídeo, elementos do Game Play, animação e som & música). Na arquitetura da aplicação inclui-se a lógica e o diagrama de classes. É amostrado o controle de versões e a organização do projeto que serve de base para a fase de implementação.

Implementação: neste apartado, é mostrado ementas sobre a interface e interação. É explicada a importância dos scripts no Unity e como foram usadas no projeto. As ações do jogador e inimigos são apresentadas assim como as técnicas de inteligência artificial aplicadas.

Testes: se apresentam os principais testes aplicados ao jogo desenvolvido e as validações feitas para demonstrar a qualidade do produto desenvolvido.

Convenções

O texto deste projeto incluirá as seguintes normas de marcação:

- As palavras idioma estrangeiro serão remarcadas em cursiva, excluindo os nomes próprios, companhias e o nome do projeto.
- Serão colocadas as referências bibliográficas nas citas textuais, externas da obra.
- Algumas palavras de notável importância serão remarcadas em negrita.

1- Estado da arte

Neste ponto, é apresentada as principais empresas que trabalham no mercado de desenvolvimento de videojogos e se mencionam alguns jogos que se tomaram como referência para o desenvolvimento da aplicação.

1.2- Principais Empresas

Existem diversas empresas trabalhando com jogos no mundo inteiro. Dentre as principais, destacam-se:

- Microsoft
- SEGA
- Blizzard
- Sony
- Ubisoft

Estas empresas são referência de excelência no mercado mundial de jogos, estando entre as maiores receitas atualmente [8]. Devido a isso, as práticas dessas empresas são observadas por grande parte dos demais fabricantes de jogos.

1.3- Comparação de Características

No desenvolvimento das características do Skull Survival foram observadas funcionalidades existentes em *The Forest*, *Plants Vs Zombies*, *I am alive*.

- *The Forest*: trata-se de um jogo onde você sofre um acidente e cai no meio de uma floresta selvagem, no início você conta com um manual de instruções e equipamentos básicos como uma machadinha, a partir daí você aprende a recolher recursos da natureza para sobreviver. [9]

-*Plants Vs Zombies*: é um jogo *tower defense* desenvolvido e publicado pela PopCap Games para Windows, Mac OS X e iPhone OS. O jogador assume o controle de várias plantas, cada uma com suas características e peculiaridades. As plantas devem resistir aos ataques dos inimigos, que, no caso, são zumbis. [10]

I am alive: trata-se da história do personagem que sai de viagem para fora da civilização, e quando volta encontra um cenário desolador, a cidade totalmente destruída e um silêncio mortal no ar, estava tudo um caos, pontes quebradas, carros capotados, corpos para todo lado, você precisa se movimentar em um cenário apocalíptico. O jogo pode ser jogado tanto em primeira pessoa ou em terceira pessoa. [11]

2-Análise do problema

2.1- Mercado-Alvo

O público-alvo do Skull Survival é formado por jogadores casuais, com idade acima de sete anos. Não há distinção de gênero ou limite máximo de idade para o público-alvo, dado sua característica de jogador casual.

2.2- Características Principais

Single-player: Skull Survival foi desenvolvido sobre uma concepção *single-player* e não fornecerá um suporte *multi-player*.

Gráficos em três dimensões (3D): Skull Survival foi feito em Unity, para agilizar o processo de desenvolvimento, foi necessário utilizar scripts de terceiros. A modelagem do personagem principal foi feita com a ferramenta online Autodesk Character Generator e modificado com 3D Max 2017.

Cenário: Skull Survival é ambientado num terreno hostil, dominado por esqueletos que saem de diversos pontos do cenário.

Jogabilidade: O produto obtido pode ser jogado com o *keyboard* ou com Microsoft Kinect. As duas variantes proponham novos desafios à lá sobrevivência do *player*.

2.2.1- Gênero

Skull Survival é um jogo no consagrado estilo *survival*, onde o jogador deverá tentar sobreviver num ambiente hostil. O aproveitamento das munições é fundamental para garantir sua supervivência.

2.2.2- Plataforma

O jogo foi desenvolvido para PC. Para jogá-lo, alternativamente pode ser usado Microsoft Kinect para movimentar o jogador.

2.3- Analise dos requerimentos

A aplicação resultante deve satisfazer certos requisitos específicos no final do mesmo. Estes determinarão a solução finalmente escolhida para a implementação do projeto e também dependerão das conclusões e resultados que confirmam até que ponto o trabalho chegou.

2.3.1- Requerimentos funcionais

Nesta parte do trabalho, se apresenta as diferentes funções do jogo. Esses requisitos estabelecem como o sistema se comporta para cada um deles.

Identificação do requerimento	RF01
Nome do requerimento	Configurar dificuldade
Características	Os usuários poderão modificar a dificuldade do jogo
Descrição do requerimento	O sistema oferecerá aos usuários distintas dificuldades de nível
Prioridade do requerimento	Alta

TABELA 1 RF01 CONFIGURAR DIFICULDADE

Identificação do requerimento	RF02
Nome do requerimento	Configurar gráficos
Características	Os usuários poderão modificar a qualidade dos gráficos do jogo
Descrição do requerimento	O sistema oferecerá aos usuários distintas escalas de qualidade
Prioridade do requerimento	Alta

TABELA 2 RF02 CONFIGURAR GRÁFICOS

Identificação do requerimento	RF03
Nome do requerimento	Calibrar Kinect
Características	Os usuários poderão calibrar o Kinect
Descrição do requerimento	O sistema oferecerá aos usuários mensagens informando a posição que este deverá adotar para calibrar corretamente o Kinect
Prioridade do requerimento	Alta

TABELA 3 RF03 CALIBRAR KINECT

Identificação do requerimento	RF04
Nome do requerimento	Manipular player
Características	Os usuários poderão realizar ações com o player.
Descrição do requerimento	O sistema apresentará um player o qual será movimentado pelos controles do teclado ou Kinect.
Prioridade do requerimento	Alta

TABELA 4 RF04 MANIPULAR PLAYER

2.3.2- Requerimentos não funcionais

A seguir, serão detalhados os requisitos que podem ser usados para qualificar as operações do sistema, ou seja, eles não descrevem as informações a serem armazenadas, nem as funções a serem executadas, mas as características operacionais.

Requisitos de desempenho

O sistema deve ser capaz de responder com uma resposta de menos de 2 segundos.

Requisitos de segurança

A segurança do usuário não deve ser comprometida a qualquer momento. O aplicativo não violará a privacidade do usuário.

Requisitos de confiabilidade

As operações que compõem o sistema devem ser desenvolvidas sem incidentes.

Requisitos de manutenção

A tarefa de administrar o sistema deve ser a mais simples possível, de modo a não sobrecarregar a pessoa encarregada de realizar essa atividade. Por outro lado, o sistema tem que ser escalável se fosse necessário adicionar mais funcionalidades.

Requisitos de disponibilidade

A disponibilidade do sistema deve ser em tempo integral e em caso de falha, reativar o serviço deve ser realizado no menor tempo possível.

2.4- Casos de uso

Esta seção descreve os casos de uso do aplicativo, que, pelas características do *software*, consiste em dois atores: usuário e jogador.

2.4.1- Caso de uso Usuário

Tratasse do usuário que ainda tem iniciado o jogo (Imagem 1).

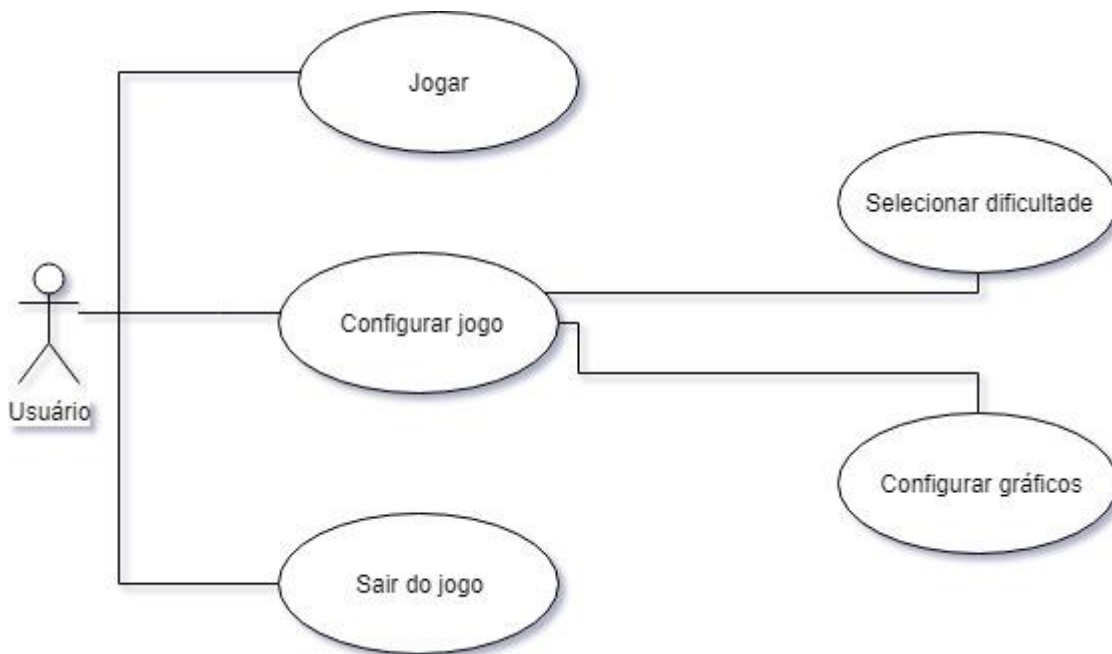


IMAGEM 1 CASOS DE USO DO ATOR USUÁRIO

A continuação se explicam cada um dos casos de uso da Imagem 1 tendo em conta as precondições, a sequência a seguir, os erros ou alternativas e as post-condições.

Nome	Jogar
Descrição	O usuário começa o jogo
Precondição	Nenhuma
Sequencia principal	<ol style="list-style-type: none"> 1. O usuário inicia o jogo 2. O usuário pulsa em Jogar
Erros/ Alternativas	No
Post-condição	O jogo começa.

TABELA 5 A1 CASO DE USO 1 – JOGAR

Nome	Configurar jogo
Descrição	O usuário pode configurar o jogo
Precondição	Nenhuma
Sequencia principal	<ol style="list-style-type: none"> 1. O usuário inicia o jogo 2. O usuário pulsa em Configurar jogo.
Erros/ Alternativas	No
Post-condição	-

TABELA 6 A1 CASO DE USO 2 - CONFIGURAR JOGO

Nome	Selecionar dificuldade
Descrição	O usuário pode selecionar a dificuldade do jogo
Precondição	Nenhuma
Sequencia principal	<ol style="list-style-type: none"> 1. O usuário inicia o jogo 2. O usuário pulsa em Configurar jogo. 3. O usuário seleciona a dificuldade.
Erros/ Alternativas	No
Post-condição	Dificuldade do jogo selecionada.

TABELA 7 A1 CASO DE USO 3 - SELECIONAR DIFICULDADE

Nome	Configurar gráficos
Descrição	O usuário pode configurar a qualidade dos gráficos do jogo
Precondição	Nenhuma
Sequencia principal	<ol style="list-style-type: none"> 1. O usuário inicia o jogo 2. O usuário pulsa em Configurar jogo. 3. O usuário seleciona a qualidade dos gráficos.
Erros/ Alternativas	No
Post-condição	Qualidade gráfica do jogo selecionada.

TABELA 8 A1 CASO DE USO 4 - CONFIGURAR GRÁFICOS

Nome	Sair do jogo
Descrição	O usuário pode parar a execução da aplicação.
Precondição	Nenhuma
Sequencia principal	<ol style="list-style-type: none"> 1. O usuário inicia o jogo 2. O usuário seleciona Sair do jogo.
Erros/ Alternativas	No
Post-condição	-

TABELA 9 A1 CASO DE USO 5 - SAIR DO JOGO

Embora consista o mesmo caso de uso decidiu-se colocar em casos de uso separados para uma melhor compreensão deste roteiro.

2.4.2- Caso de uso Jogador

Tratasse do usuário que já começou o jogo (Imagem 2).

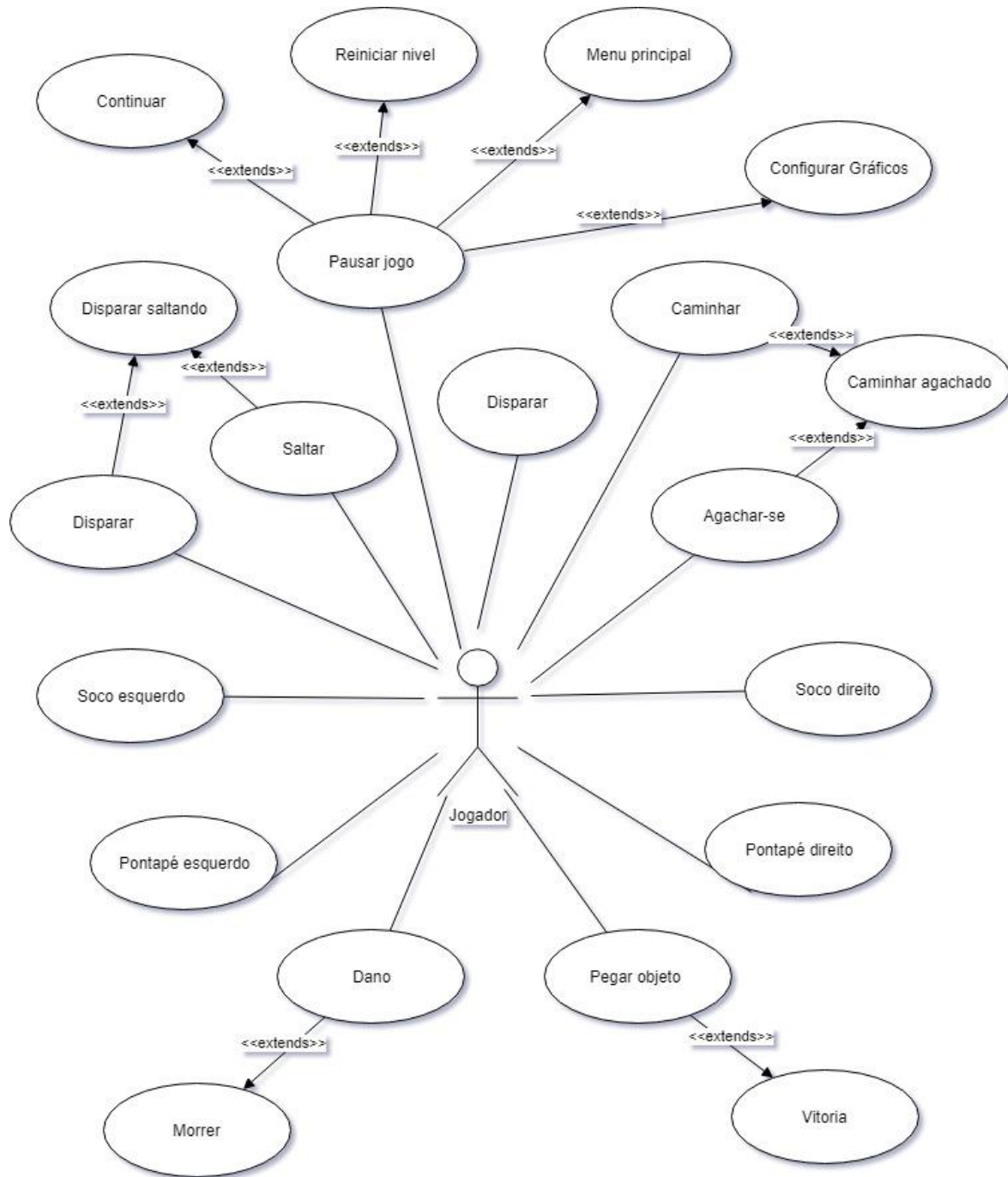


IMAGEM 2 CASOS DE USO DO ATOR JOGADOR

2.5- Análise Técnica

2.5.1- Elementos Experimentais

O desenvolvedor do jogo é inexperiente no manejo do Kinect, e o jogo foi desenvolvido focando o aprendizado. Dito isso, não serão utilizados elementos experimentais no jogo. Para o desenvolvedor, a criação do jogo em si já envolve uma série de tecnologias novas a serem aprendidas, e o uso de inovações será postergado para desenvolvimentos de jogos futuros, quando o conhecimento básico para a feitura de jogos já estiver bem sedimentado.

2.5.2- Riscos

Como em qualquer desenvolvimento de software, diversos riscos estão associados à produção do Skull Survival. A tabela abaixo mostra os principais riscos que aparecerem durante o desenvolvimento do Skull Survival:

Nome	Descrição	Plano de Mitigação	Plano de Contingência
Uso de novas tecnologias	Dado que o desenvolvedor possui pouca experiência no desenvolvimento de jogos com Kinect, e que diversas técnicas e tecnologias deverão ser aprendidas, as tecnologias utilizadas para o desenvolvimento de jogos podem tornar a produção inviável dentro do prazo disponível	Distribuição de tutoriais específicos para utilização das tecnologias necessárias; Membros do Tecgraf e outras matérias mais experientes podem ministrar exemplos. Assistir tutoriais oficiais de Unity em Youtube.	Migrar para tecnologias mais simples.
Prazo curto	Há um prazo relativamente curto (3 a 4 meses) para realizar as diversas tarefas necessárias ao desenvolvimento do jogo	Estabelecer um cronograma e gerenciar o tempo, de forma a terminar tarefas no prazo especificado	Consultar desenvolvedores com mais experiência.

TABELA 10 RISCOS DO DESENVOLVIMENTO

2.5.3- Recursos Estimados

Os recursos utilizados para o desenvolvimento do Skull Survival foram:

- Desenvolvedor: 1 pessoa.
- Software: Sistema Operacional Windows 10 x64; Unity 2018.1.0f2 Personal 64 bits; IDE Microsoft Visual Studio Community Edition 2017, 3D Max2017, Kinect Studio v2.0.
- Hardware: Dell Inspiron 5559, com 6 Gb de RAM, placa de vídeo de 2 Gb e placa de som (ambas as placas compatíveis com DirectX 10 ou superior).

2.5.4- Cronograma Estimado

Fase	Marco	Data de início	Data de término
Concepção	Apresentação do Conceito do Jogo	05/03/2018	09/03/2018
Design	Release do Documento de Game Design	10/10/2018	13/10/2018
Documentação	Em definição	12/03/2018	23/03/2018
Treinamento nas ferramentas	Conclusão do curso de treinamento interno	26/03/2018	13/04/2018
Implementação	Release da versão alfa	16/04/2018	12/06/2018
Testes	Release da versão beta e apresentação prévia do jogo	13/06/2018	28/06/2018
Entrega	Release da versão oficial e apresentação do jogo	29/06/2018	03/07/2018

TABELA 11 CRONOGRAMA DE DESENVOLVIMENTO DO JOGO

3- Desenho da solução

Nesta seção serão descritas características do projeto modular da aplicação tais como a lógica, a arquitetura, diagrama de arquitetura e os critérios de projeto utilizados.

3.1- Game Design

3.1.1- Arte e Vídeo

Overall Goals

O Skull Survival é um jogo 3D, e, portanto, o uso de gráficos em 3 dimensões deve se sobressair nos elementos principais do jogo (com os quais o jogador interage) para criar um ambiente de imersão.

Animação e Arte 2D

-GUI

Serão utilizados os elementos básicos da GUI (telas, janelas, botões, etc.).

-Terreno

O terreno foi construído por o próprio desenvolvedor desde Unity. As montanhas feitas e o ambiente limpo de outros modelos com uma alta quantidade de polígonos, poderiam afetar a fluidez do jogo num ordenador com modestas prestações. Para não deixar o terreno sem item se incluíram 1 modelo duma casa antiga e 1 modelo de arvore. Além disso, foi colocada algumas plantas no início das montanhas e um vento leve para dar sensação de realismo.

3.1.2- Elementos do Game Play

Da mesma forma que os terrenos, também devem haver texturas adequadas aos elementos do *game play*. O editor utilizado para elementos 3D deve ser utilizado para colocar as texturas 2D sobre os elementos. Neste caso colocaram-se nas pastas indicadas seguindo os tutoriais oficiais de Unity.

3.1.3- Animação e Arte 3D

Os jogadores e itens que interagem com os jogadores foram desenhados com a menor quantidade possível de polígonos para garantir a fluidez da aplicação. O trabalho de criação das figuras 3D só foi necessário para alguns itens somente.

3.1.4- Som e Música

Os sons do Skull Survival seguem uma temática misteriosa, ajudando a criar a identidade do jogo. O uso do som relacionado aos elementos do jogo também deve ser utilizado para provocar uma maior imersão do jogador, a partir de uma integração da parte sonora com a visual. A existência de itens será explorada pela sonoridade, ajudando o jogador a perceber os elementos distintos que compõem o jogo.

Efeitos de Som: Os efeitos de som utilizados deverão ser compatíveis com a tecnologia adequada. A continuação segue a descrição dos diversos efeitos de som por elementos de jogo.

-GUI

Elementos de GUI (cliques de botão nas telas, etc.) irão utilizar os sons padrões do jogo.

-Personagens

Jogador: O mesmo faz um som quando bate a disparos aos esqueletos dando sentido de profundidade o som escolhido. Quando este é abatido por um inimigo, um som notifica ao player do dano.

Inimigos: Os inimigos quando são abatidos pelo jogador fazem um som para indicar que o disparo do jogador deu certo. Quando morrem faz um grito e desaparecem do jogo.

-Elementos do Game Play

Cada elemento do game play descrito anteriormente (subseção Elementos do Game Play em Mecânica do Jogo) possui um som próprio ao ser ativado, para alertar os jogadores.

3.2- Arquitetura da aplicação

3.2.1- Logica da aplicação

A continuação, o comportamento do jogo é explicado por meio de um diagrama de fluxo (Imagem 3), mostrando as ações que podem ser executadas nele. A legenda das figuras usadas no diagrama são apresentadas na Tabela 12 Legenda das figuras usadas no diagrama de fluxo:

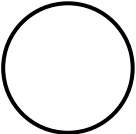


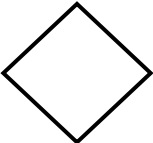
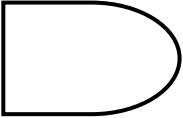

	Este ícone representa o início da aplicação
	Este ícone representa as diferentes vistas do aplicativo.
	Determina as ações a serem executadas na exibição.
	Representa situações em que o usuário, ao executar uma ação, é avaliado pelo sistema e determina o caminho a seguir.
	Representa um delay para executar a seguinte ação
	Este ícone representa o final da execução do aplicativo.

TABELA 12 LEGENDA DAS FIGURAS USADAS NO DIAGRAMA DE FLUXO

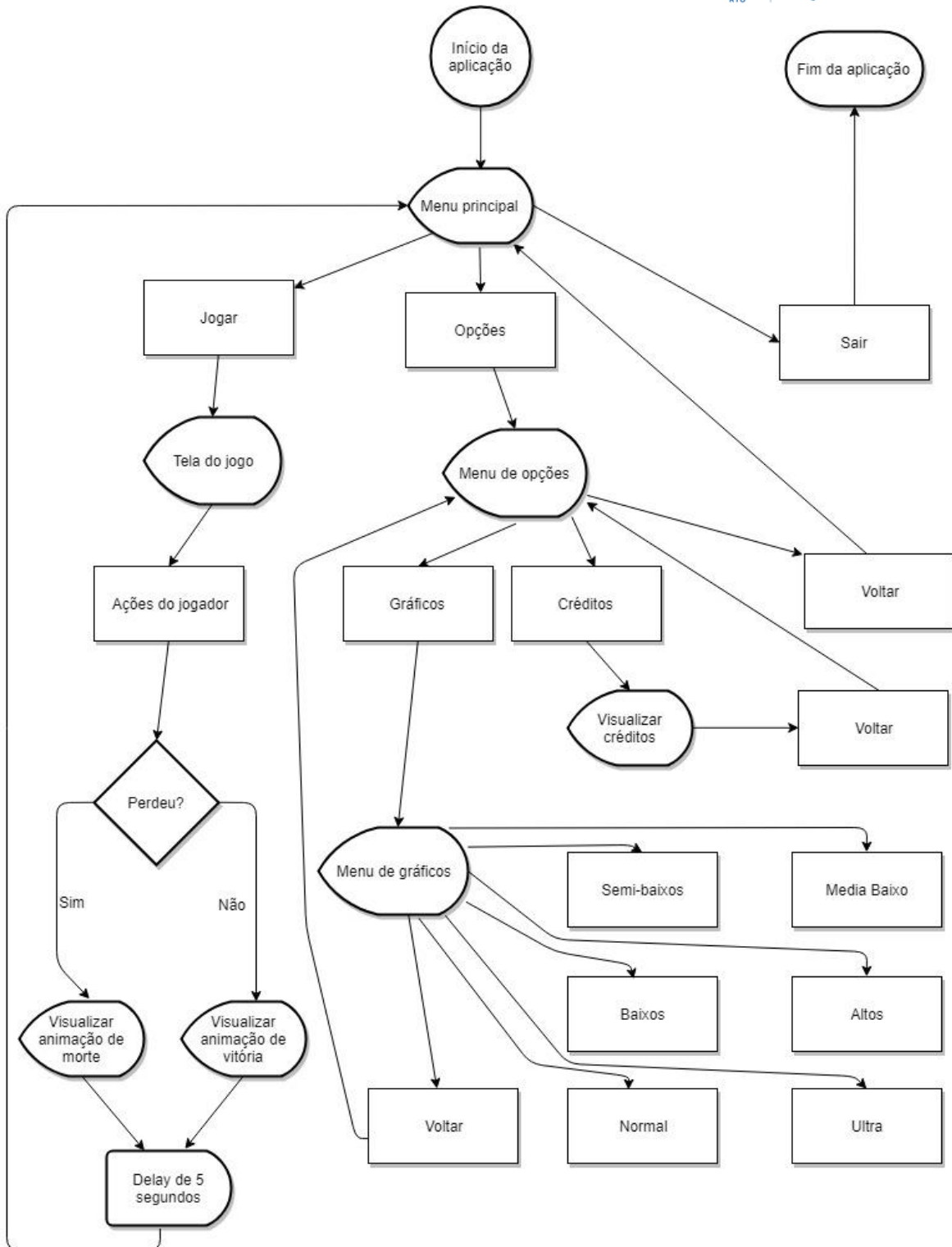


IMAGEM 3 DIAGRAMA DE FLUXO DA APLICAÇÃO

Pode se observar que a aplicação é bem básica e a maioria dos elementos giram em torno ao menu principal.

3.2.2- Diagrama de classes

A continuação se apresenta o Diagrama de Classes do projeto na Imagem 4. Foi colocada na esquerda os scripts referentes ao inimigo (EnemyMovement, EnemyManager e, EnemyAttack e EnemyHealth), no centro da imagem os referentes ao jogador (PlayerHealth e PlayerShooting) além dos plug-ins de terceiros necessários para a movimentação do jogador com o Kinect (ThirdPersonUserControl, ThirdPersonCharacter e BodySourceManager) neste caso preferiu-se deixar com os nomes originais. No canto inferior pode ser observado um conjunto de scripts auxiliares que foram necessários para funciones com a interface de usuário.

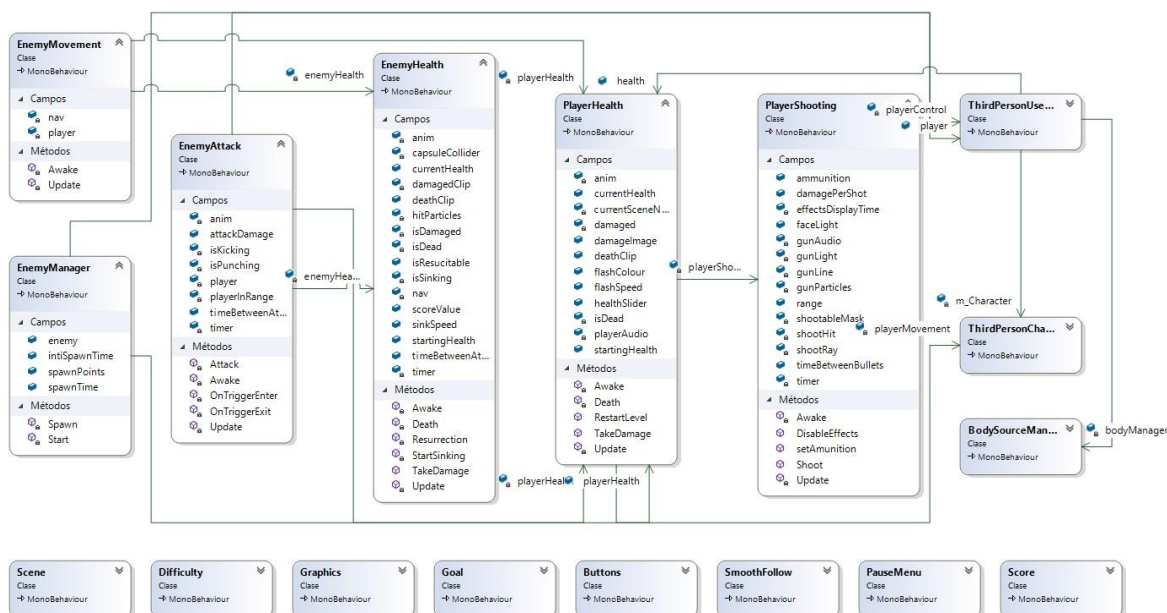


IMAGEM 4 DIAGRAMA DE CLASSES DO PROJETO

3.3- Controle de versões

O desenvolvimento foi feito de forma incremental, utilizando-se o repositório de arquivos na nuvem do Unity e o GIT para controle de versão. O uso do Git ajudou consideravelmente a realização de um desenvolvimento em partes pequenas e com a menor probabilidade de erros.

3.4- Organização do projeto

O projeto foi dividido em duas bibliotecas, a primeira com os componentes que formam o jogo e a segunda por componentes necessários para a vinculação com o Kinect contendo livreiras de terceiros.

No primeiro componente, tendo em conta recomendações e experiências em projetos anteriores decidiu-se estruturar na seguinte forma:

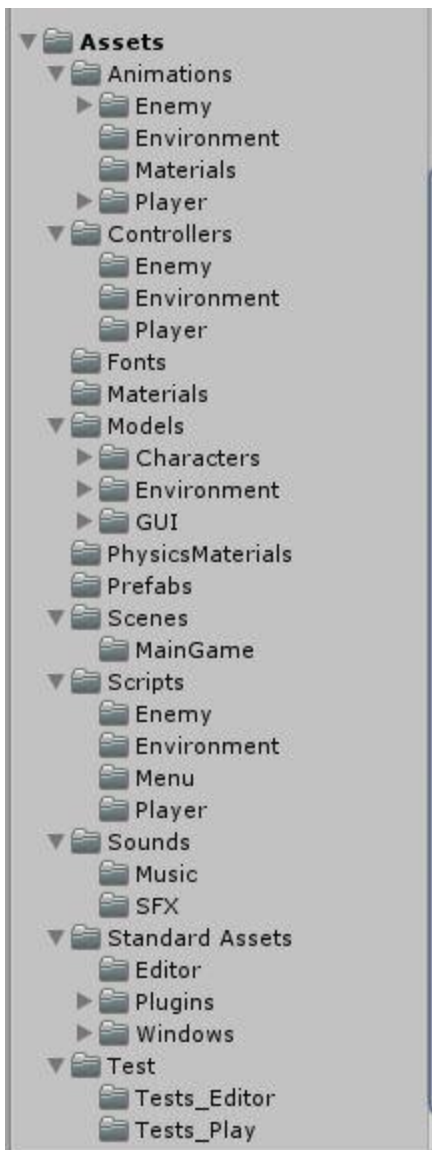


IMAGEM 5 HIERARQUIA DO PROJETO DESDE UNITY

- Animations: Contém todas as animações do jogo, segmentados em inimigos, ambiente e jogador.
- Controller: Contém as máquinas de estado dos inimigos, o jogador e os clips a reproduzir quando o jogador ganhe ou perda.
- Fonts: Inclui as fontes usadas no jogo.
- Materials: é guardado os materiais usados no ambiente do jogo.
- Models: Contém os personagens que interatuam no jogo.
- PhysicsMaterials: Inclui efeitos físicos a serem aplicados aos personagens e scena.
- Prefabs: é um tipo de asset que pode ser entendido como um GameObject armazenado no Project. Todas as instâncias de cada prefab estão conectadas com o prefab original.
- Scenes: Contém as cenas disponíveis no jogo.
- Script: Contém os scripts do jogo (Camera, Enemy, Goal, Menu, Player). Em seções

posteriores serão explicadas a importância das mesmas.

- Sound: Inclui os arquivos de música e ambiente do jogo.
- Test: Contém as classes de teste usadas no projeto, estas serão explicadas com mais detalhes na secção Testes deste documento

Na segunda biblioteca, chamada Standard Assets, é alocado os scripts e bibliotecas de terceiros usados no jogo, neste caso, para a vinculação do Kinect com o avatar.

Por outra parte, desde o *Solution Explorer* do Visual Studio Community 2017 é possível observar a hierarquia do projeto o qual se apresenta na Imagem 6.

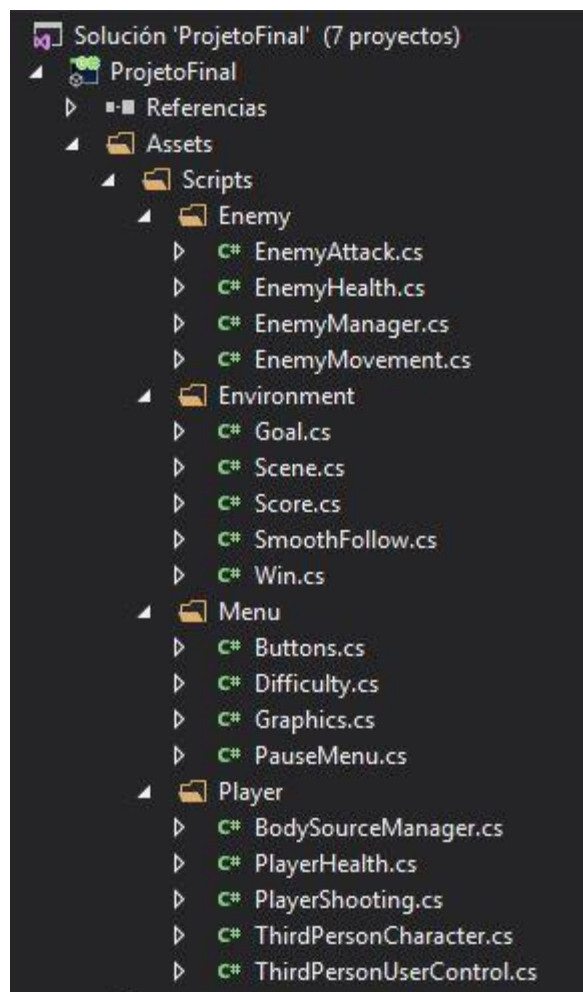


IMAGEM 6 HIERARQUIA DO PROJETO DESDE VS COMMUNITY

4- Implementação

4.1- Interface e interação

Neste apartado, serão apresentadas as interfaces do jogo Skull Survival e os controles do mesmo. Pelas características do software, optou-se por interfaces simples e controles familiares para os jogadores.

4.1.1- Interfaces do Jogo

O jogo começa com o menu básico, onde o jogador pode escolher uma das seguintes opções: Jogar (inicia uma nova campanha), Configurações (gráficos do jogo, créditos) e sair (sai do jogo).

Quando começa uma nova campanha, irá aparecer a tela onde o jogador deverá pegar o mais rápido possível os objetivos antes que os esqueletos comecem a sair de suas respectivas casas!

Abaixo, seguem as telas de navegação do usuário no jogo. São basicamente 3 telas na navegação padrão: a tela de abertura do jogo; a tela da partida em si e a tela de opções.

Na Imagem 7 observa-se a tela inicial do jogo

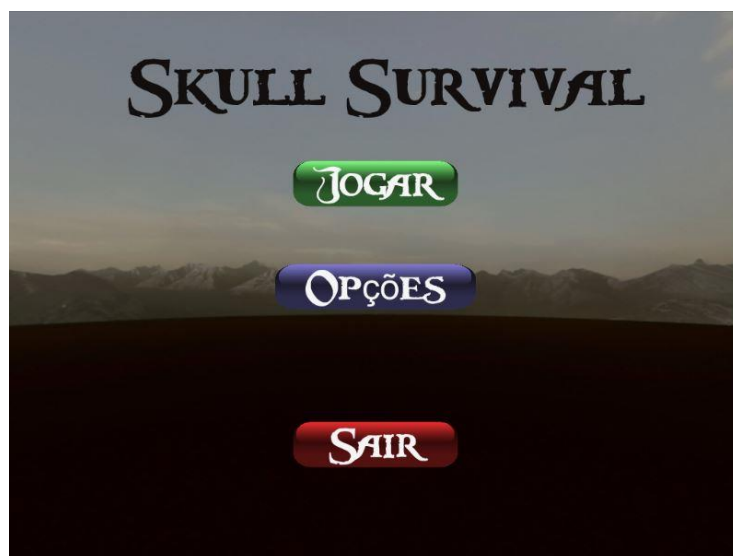


IMAGEM 7 TELA INICIAL DO JOGO

A Imagem 8 apresenta a tela opções do jogo onde pode-se configurar a qualidade dos gráficos e os autores do trabalho.

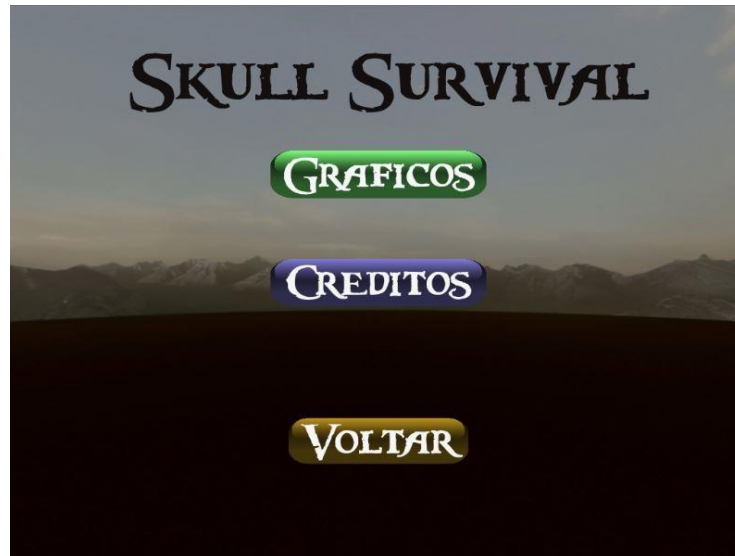


IMAGEM 8 TELA OPÇÕES DO JOGO

A Imagem 9 apresenta a tela do jogo pronto para começar

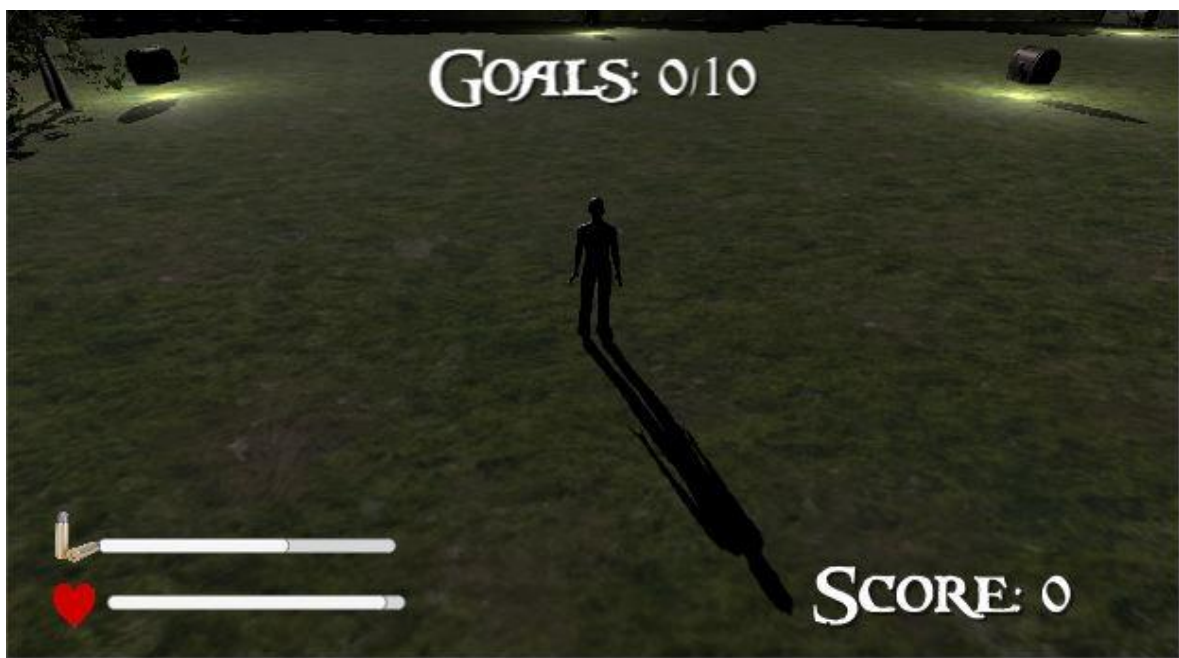


IMAGEM 9 JOGO EM EXECUÇÃO

Ao pressionar a tecla **Esc** o levantar a mão direita usando Kinect, o jogo ficara em pausa e mostrara a seguinte tela de opções da Imagem 10:

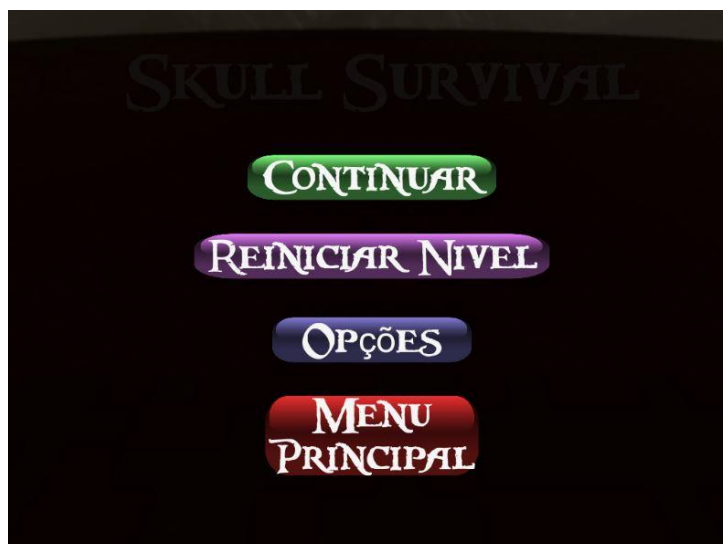


IMAGEM 10 TELA JOGO EM PAUSA

Ao finalizar o jogo (ganhado ou perdido) o mesmo começara de novo proporcionando um novo cenário para disfrutar do mesmo.

4.1.2- Controles

Na tabela 1 observa-se as ações permitidas com o teclado e o gesto com Kinect que equivalente previa calibração ao começar o jogo se estivesse um Kinect conectado ao PC.

Ação	Teclado	Kinect
Caminhar adiante	Up / W	Corpo ao frente
Caminhar atrás	Down / S	Corpo ao atrás
Caminhar direita	Right / D	Corpo ao direita
Caminhar esquerda	Left / A	Corpo ao esquerda
Disparar	Ctrl	Fechar mão direita/esquerda
Saltar	Espaço	Baixar o corpo e subir
Bater com o pé direito	V	Bater com o pé direito
Bater com o pé esquerdo	B	Bater com o pé esquerdo
Agacha-se	C	Agacha-se
Bater com a mão direita	Z	Bater com a mão direita
Bater com a mão esquerda	X	Bater com a mão esquerda

TABELA 13 CONTROLES COM TECLADO E KINECT

4.2- Core Game Play

4.2.1- Jogador

O modelo do jogador, na consiste em um *avatar*, o qual se apresenta na Imagem 11, o qual possui um conjunto de características que faram o jogo mais dinâmico.



IMAGEM 11 AVATAR DO JOGADOR

4.2.2- Inimigos

Os inimigos são três tipos de esqueletos e um porquinho voador que atacam em ondas cuja dificuldade vai progredindo o jogo mesmo. O jogador deverá defende-se dos inimigos fazendo uso da sua arma de fogo o, quando seja iminente o ataque, batendo golpes o com seus pés, no caso contrário os inimigos atingirão contra o jogador quantas vezes for necessário até causar a morte dele. Os esqueletos saem de 4 casas localizadas nas 4 esquinas o cenário, por outra parte, os porquinhos vão sair dos arvores. Os mesmos, vão a procurar a posição do jogador sempre que o mesmo esteja vivo e não se encontre entre os arvores do cenário.

Ações dos inimigos

Esqueleto branco: Este tipo de inimigo da Imagem 12 se caracteriza por caminhar mais devagar. A força de seu ataque é a mais baixa dos inimigos apresentados.

Esqueleto vermelho: Este inimigo, apresentado na Imagem 13 possui características e comportamento similar ao anterior com a característica que sua velocidade é o dobro do anterior e pode ressuscitar 1 vez.



IMAGEM 12 INIMIGO ESQUELETO
BRANCO



IMAGEM 13 INIMIGO ESQUELETO
VERMELHO

Esqueleto armado:

Este inimigo, apresentado na Imagem 14, apresenta o mesmo comportamento do esqueleto branco com o diferencia que a força de ataque é a maior dos inimigos apresentados.

Porquinho voador:

Este inimigo da Imagem 15 não é inofensivo, o mesmo tem a característica que pode perseguir ao jogador ao longo do todo o terreno, embora este esteja nas árvores o porquinho pode entrar em algumas áreas da floresta.



IMAGEM 14 INIMIGO ESQUELETO ARMADO



IMAGEM 15 INIMIGO PORQUINHO VOADOR

4.2.3- Elementos do Game Play

Há diversos elementos com os quais o jogador pode interagir no ambiente do Skull Survival. Tem arcos do poder, com luz amarela ressaltante que serão indispensáveis para o jogador e condição necessária para ganhar.

Aos esqueletos não gostam das árvores, esse será um lugar de paz para o jogador, mas, não deve confiar-se muito tempo em sua zona de conforto, o tempo passa e os inimigos seguem saindo a defender seu terreno.

4.3- Scripting e Component Based Design

O sistema de scripting da Unity3D é abrangente e flexível o que permite o desenvolvimento de jogos completos sem a necessidade do uso de C/C++. [12] Internamente, os scripts são executados através de uma versão modificada da biblioteca Mono, uma implementação de código aberto para o sistema .Net. Essa biblioteca, permite que os scripts sejam implementados em qualquer uma de três linguagens, à escolha do programador: javascript, C# ou Boo (um dialeto de Python).

De forma consistente à arquitetura desenvolvida, scripts no Unity3D são acoplados como **componentes** de GameObjects [13]. Desta forma, é importante projetar os scripts de maneira modular, ganhando com isso a flexibilidade do reuso.

Logo, os componentes tornam nosso código mais puro (sem variáveis e funções desnecessárias), mas também tornam o processo de criação de inimigos muito mais flexível e agradável. Com cada componente de funcionalidade inimiga configurada como um componente, podemos arrastar e soltar aspectos de inimigos (Imagem 16) e ver como eles se comportam - e até mesmo em tempo real, ao usar o Unity.

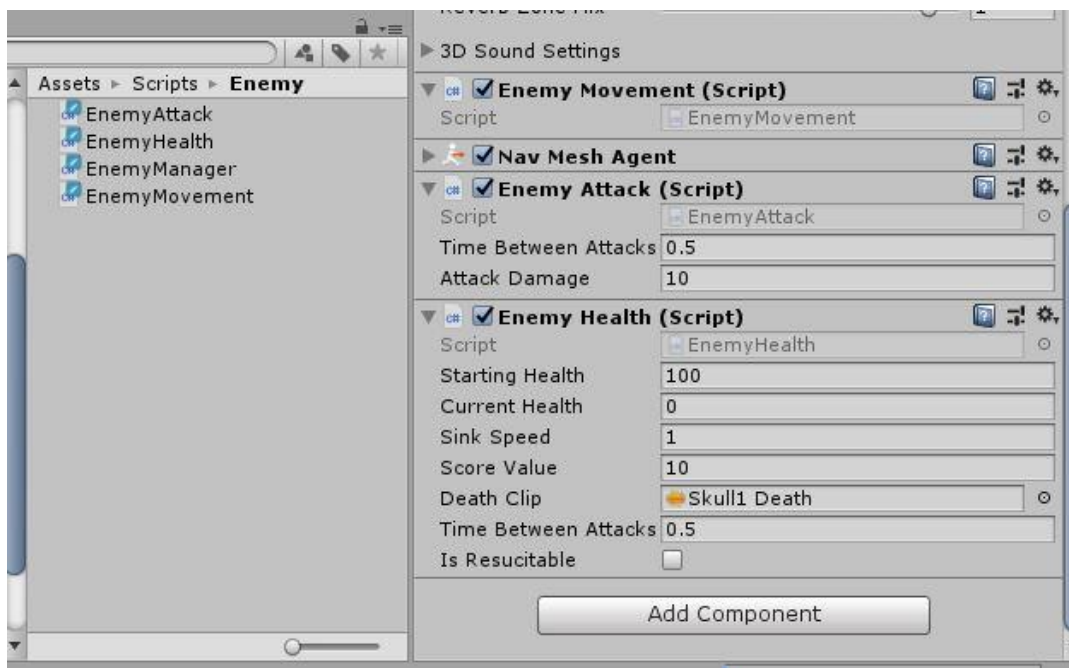


IMAGEM 16 COMPONENTES DO GAMEOBJECT INIMIGO1

4.4- Técnicas de Inteligência Artificial

O termo “IA em jogos” (ou “Game AI”) é geralmente usado de forma muito ampla, variando desde a representação e controle de comportamento de personagens não controlados pelo jogador no jogo (*nonplayer characters* - NPCs).

Neste projeto foram utilizadas as técnicas máquinas de estado finitas (*Finite State Machine* - FSMs), o *Path Finding* através do popular algoritmo A^* para o cálculo de caminhos e sistemas baseados em regras (*Rule Based Systems* - RBSs).

A continuação se explica cada uma delas e como foi utilizada no projeto desenvolvido para os inimigos do jogador.

4.4.1- Máquinas de Estado Finitas

Uma máquina de estados finita (de aqui por diante MEF) é basicamente composta por um conjunto de estados e um conjunto de regras de transição entre estes estados [14].

Na figura Imagem 17 se apresenta a MEF dos inimigos. Na execução do comportamento (no início do jogo está em *Entry*), a MEF encontra-se em seu estado (*Walk*); a cada iteração (neste caso, cada frame), as regras das transições que deixam o estado corrente são avaliadas; se alguma delas for disparada (cercania do jogador, por exemplo), a transição é então realizada, e o estado de chegada desta regra se torna o novo estado corrente (*Walk* ou *Idle*) e a animação associada ao mesmo vai ser executada.

Neste caso observa-se que o estado *Exit* não possui uma conexão com nenhum estado, isto faz sentido porque a morte do inimigo vai ligado a um evento que só é disparado baixo um conjunto de condições que será explicado em uma técnica posterior.

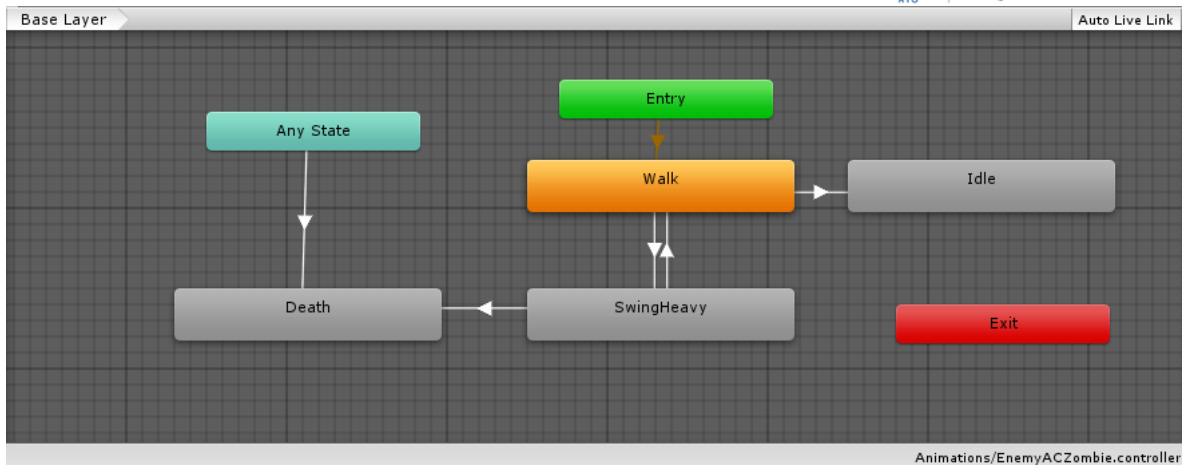


IMAGEM 17 MEF DO INIMIGO

Na Imagem 18 a continuação, observa-se a animação quando o inimigo vai para onde está o jogador (o como vai ser explicado mais diante). Observe que a linha azul em *Walk* é a duração total da animação, neste caso a mesma vai se executar até o final e verificar as condições necessárias para volte-se a executar ou bem, mudar o estado a outra transição (*Idle*, *SwingHeavy*).

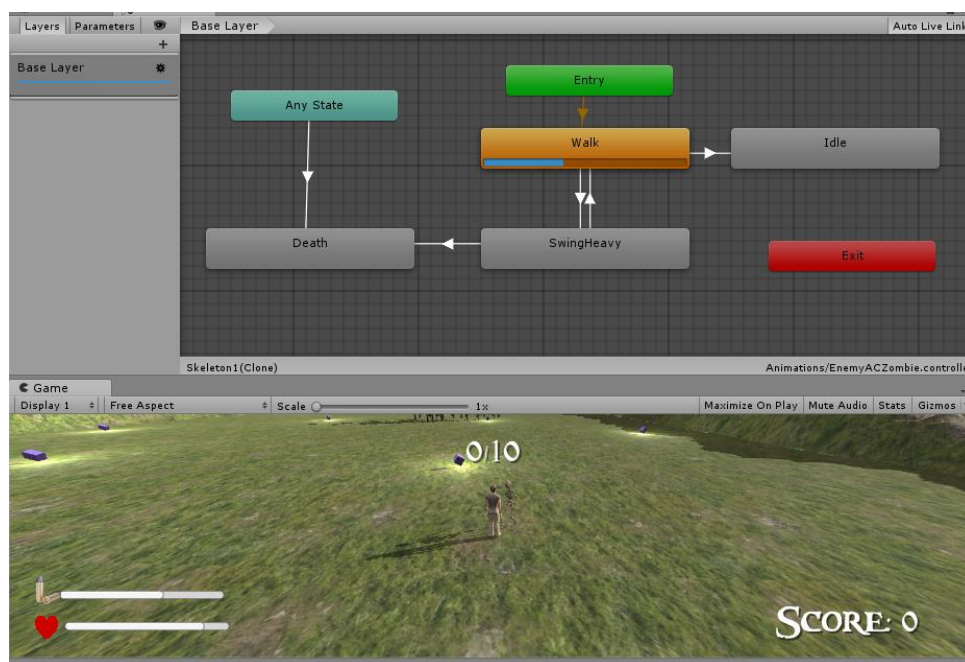


IMAGEM 18 MEF DO INIMIGO – WALK

Neste outro (Imagem 19) caso o jogador atingiu a quantidade necessária de disparos sobre o inimigo, porém o estado dele, vai mudar a animação *Death*. É válido esclarecer que o estado *Death* vai ser executado em qualquer momento que o mesmo seja morto. Em uma técnica posterior vai se explicar este estado.

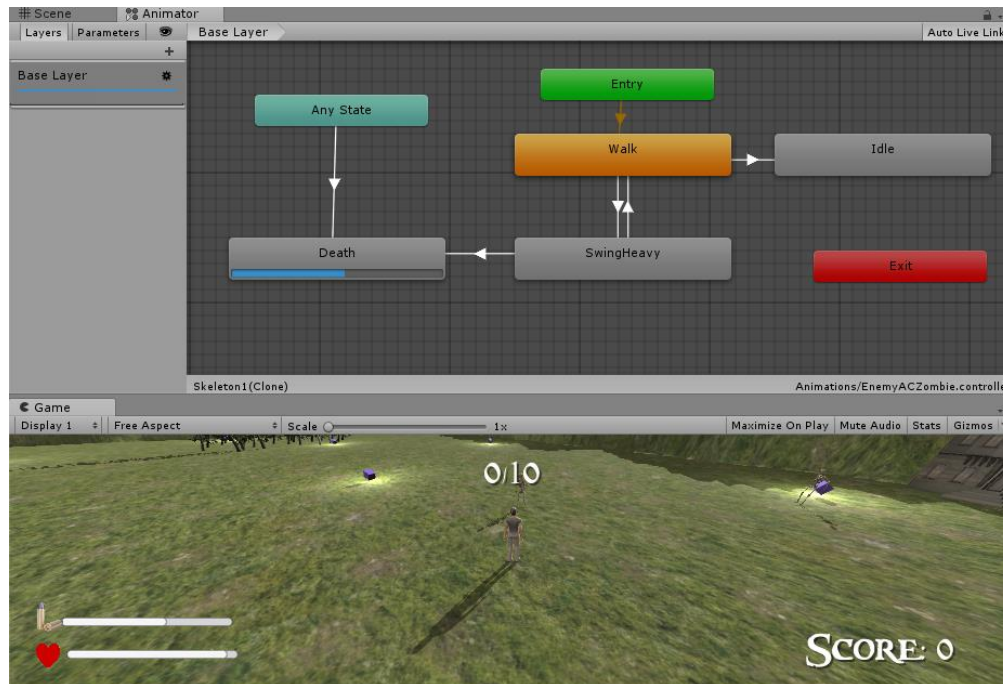


IMAGEM 19 MEF DO INIMIGO – DEATH

4.4.2- Path finding

Mover-se de um lugar a outro utilizando um caminho razoável, ao mesmo tempo em que se desvia de obstáculos, é um requisito fundamental para qualquer entidade que queira demonstrar algum sinal de inteligência em um jogo. Um dos aspectos mais importantes relacionados à implementação de funcionalidades de IA em jogos, e de impacto visual mais obvio, é então a determinação de caminhos (*path-finding*) [15]

De modo a tratar esse problema, a abordagem geralmente utilizada é executar um algoritmo de busca sobre os dados da cena de modo a encontrar um caminho entre

a posição de origem e a posição de destino. Esse é um ponto em que a IA para jogos aproveita bem as soluções da IA clássica, especificamente na forma do algoritmo de busca A*. Implementar o A* num contexto de jogos digitais requer diversas melhorias/adaptações.

O algoritmo em si é apenas parte do problema, de modo a melhorar seu desempenho é necessário (principalmente) tentar simplificar o espaço de busca. Uma das maneiras de representar esse espaço e seguimento dum personagem em Unity é através de malhas de navegação (*Navigation Meshes*), as quais foram usadas para o terreno.

A Imagem 20 apresenta, sinalizado em azul, as áreas onde os inimigos vão a se movimentar. Sendo delimitando o terreno em forma de quadrado e no centro, nas árvores, colocando uma zona onde o jogador pode se proteger de seus adversários.

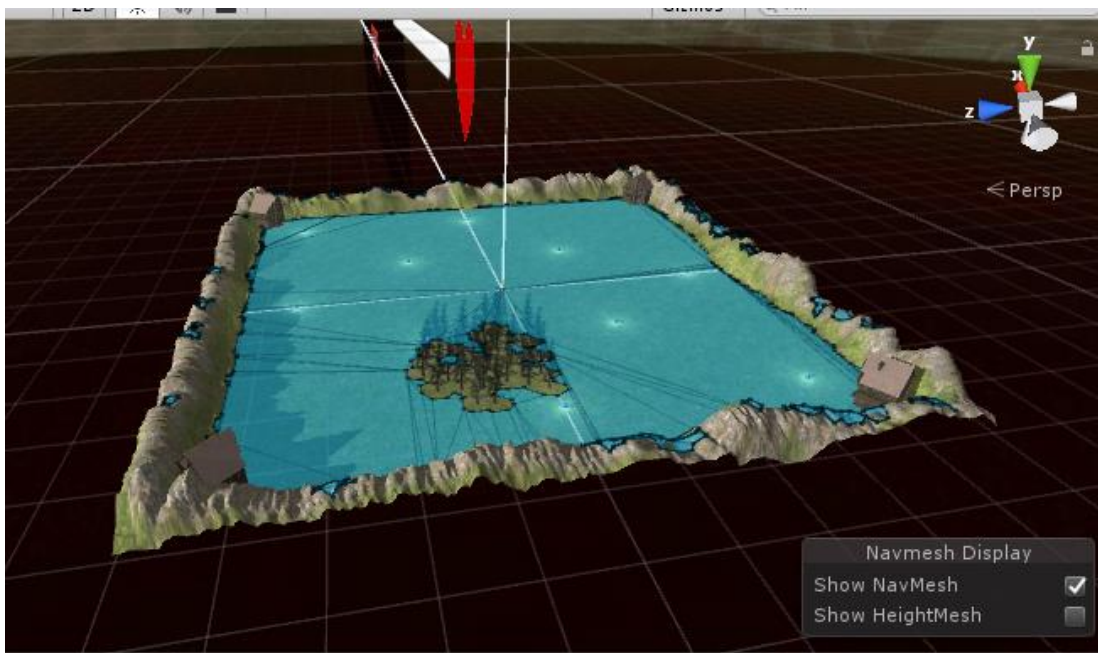


IMAGEM 20 NAV MESH DO TERRENO

Uma vez delimitada a área onde os inimigos vão a se movimentar, vai se definir os parâmetros para que os NPC's interagem com o jogador como se apresenta na Imagem 21.

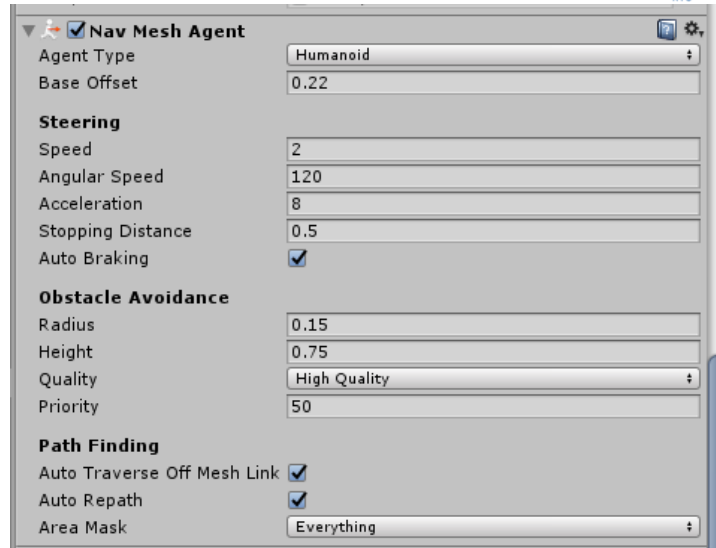


IMAGEM 21 NAVIGATION MESH - INIMIGO

Unity permite implementar scripts com certa flexibilidade para manipular os atores do jogo e o mesmo adquire um comportamento otimizado da A*, a classe criada [EnemyMovement](#) se amostra na Imagem 22.

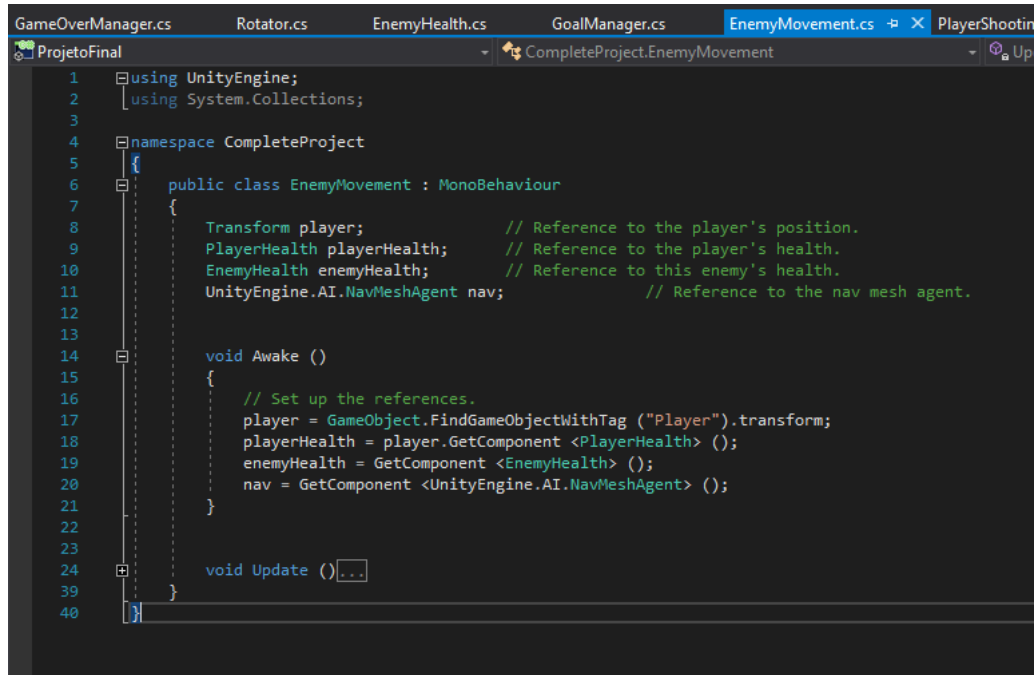


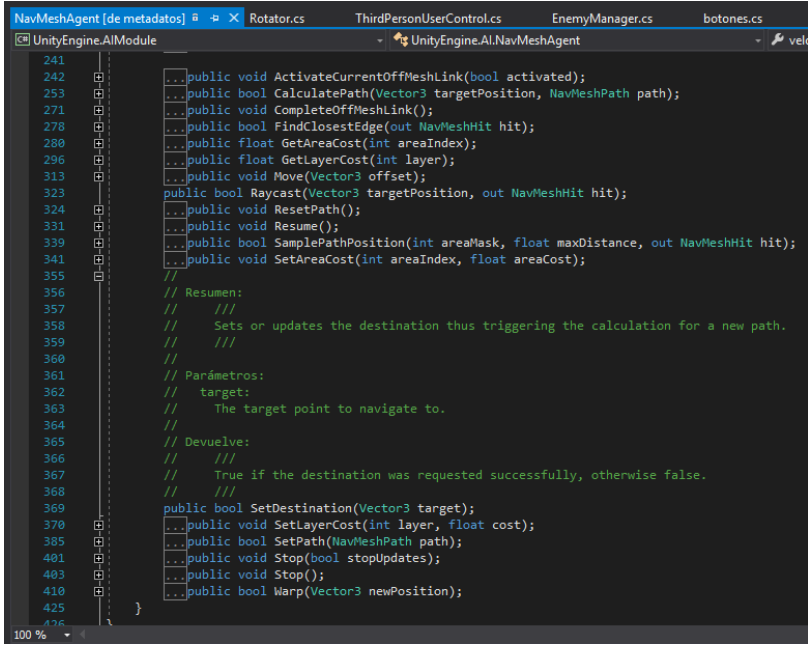
IMAGEM 22 CLASSE ENEMYMOVEMENT

Por outra parte o método `Update` implementado, reposiciona o inimigo em direção ao jogador em cada frame do jogo, o mesmo se apresenta na Imagem 23.

```
void Update ()
{
    // If the enemy and the player have health left...
    if(enemyHealth.currentHealth > 0 && playerHealth.CurrentHealth > 0)
    {
        // ... set the destination of the nav mesh agent to the player.
        nav.SetDestination (player.position);
    }
    // Otherwise...
    else
    {
        // ... disable the nav mesh agent.
        nav.enabled = false;
    }
}
```

IMAGEM 23 MÉTODO UPDATE DA CLASSE ENEMYMOVEMENT

Uma vez implementada esta funcionalidade, a mesma vai chamar o método `SetDestination` da `public sealed class NavMeshAgent`. Esta classe permita a execução do algoritmo A* mais otimizada, na Imagem 24 se apresentam os métodos que avaliam a anterior afirmação, porém não é necessário implementar o método A* desde zero sino utilizar uma funcionalidade otimizada do próprio Unity.



```
241 ...public void ActivateCurrentOffMeshLink(bool activated);
242 ...public bool CalculatePath(Vector3 targetPosition, NavMeshPath path);
243 ...public void CompleteOffMeshLink();
244 ...public bool FindClosestEdge(out NavMeshHit hit);
245 ...public float GetAreaCost(int areaIndex);
246 ...public float GetLayerCost(int layer);
247 ...public void Move(Vector3 offset);
248 ...public bool Raycast(Vector3 targetPosition, out NavMeshHit hit);
249 ...public void ResetPath();
250 ...public void Resume();
251 ...public bool SamplePathPosition(int areaMask, float maxDistance, out NavMeshHit hit);
252 ...public void SetAreaCost(int areaIndex, float areaCost);
253 //
254 // Resumen:
255 //
256 // Sets or updates the destination thus triggering the calculation for a new path.
257 //
258 // Parámetros:
259 // target:
260 // The target point to navigate to.
261 //
262 // Devuelve:
263 //
264 // True if the destination was requested successfully, otherwise false.
265 //
266 public bool SetDestination(Vector3 target);
267 ...public void SetLayerCost(int layer, float cost);
268 ...public bool SetPath(NavMeshPath path);
269 ...public void Stop(bool stopUpdates);
270 ...public void Stop();
271 ...public bool Warp(Vector3 newPosition);
272 }
```

IMAGEM 24 CLASSE NAVMESHAGENT

5- Testes

Neste apartado se realizam as provas ao jogo os quais permitem comprovar a qualidade deste produto para o nível desejado, o que constitui um dos passos mais importantes no desenho e implementação de uma solução.

5.1- Testes alfas

Os testes alfa ocorreram nos últimos dias de desenvolvimento, com pessoas não ligadas diretamente ao jogo, principalmente colaboradores. O intuito dos testes alfa eram encontrar bugs e encontrar possíveis pontos de melhoria através de um questionário e entrevistas.

No entanto, os testes foram realizados com um grupo muito reduzido de colaboradores, apenas 5. Embora não seja uma escolha muito abrangente, os resultados tiveram pouquíssima divergência tendo apenas duas perguntas com respostas diferentes. Apenas nas perguntas “Ficou claro o que era preciso fazer?” e, “Os itens estavam disponibilizados corretamente” houve 50% de respostas negativas.

Ficou claro que o questionário realizado foi muito curto para o teste alfa. Era preciso avaliar outros itens como a satisfação com as músicas, fluxo de jogo, feedback e outros. No final, foi executada apenas o questionário para teste alfa cobrindo apenas os itens de: o jogo; concentração; desafio e controle.

Depois de percorrer o jogo avaliando os bugs, os colaboradores ajudaram indicando os principais pontos de melhoria. Isso levou aos principais pedidos de atualização para próximas versões:

- O jogo precisa de mais cenários e mais personagens;
- Implementação de eventos.

Outra *feature* muito requisita foi a implementação do conceito evolução do personagem, através de aquisição de novas habilidades, características e itens úteis para evolução do jogo, a exemplo de jogos como Megaman [16].

5.2- Testes de sistema

Trata o comportamento de todo do sistema ou produto definido pelo escopo de um projeto ou programa de desenvolvimento. Os testes de sistema podem ser baseados em especificação de riscos e/ou de requisitos, processos de negócios, casos de uso, dentre outras descrições de alto nível do comportamento, interações e recursos do sistema. [17] Em nosso caso decidimos centra-nos nos casos de uso principais do ator Jogador, as quais se apresentam a continuação.

Caso de Uso	Pausar jogo.
Descrição	Verificar se é possível pausar o jogo.
Condições de execução	O jogo deve estar em execução
Sequencia principal	1. Se pressiona a tecla Esc.
Resultado esperado	O jogo fique em pausa e apareça a interface “Jogo em Pausa”.
Avaliação	Satisfatória

TABELA 14 TESTE JOGADOR - PAUSAR JOGO

Caso de Uso	Caminhar.
Descrição	Verificar se o jogador pode caminhar pelo terreno.
Condições de execução	O jogo deve estar em execução
Sequencia principal	1. Se pressiona a tecla Up/W, Down/S, Left/A e Right/D.
Resultado esperado	O jogador vai se movimentar pelo terreno
Avaliação	Satisfatória

TABELA 15 TESTE JOGADOR - CAMINHAR

Caso de Uso	Agachar-se
Descrição	Verificar se o jogador pode agachar-se na cena.
Condições de execução	O jogo deve estar em execução
Sequencia principal	1. Se pressiona a tecla C.
Resultado esperado	O jogador vai se agachar
Avaliação	Satisfatória

TABELA 16 TESTE JOGADOR - AGACHAR-SE

Caso de Uso	Soco direito
Descrição	Verificar se o jogador pode fazer um soco direito.
Condições de execução	O jogo deve estar em execução
Sequencia principal	1. Se pressiona a tecla Z.
Resultado esperado	O jogador vai fazer um soco
Avaliação	Satisfatória

TABELA 17 TESTE JOGADOR - SOCO DIREITO

Caso de Uso	Pontapé direito
Descrição	Verificar se o jogador pode fazer um pontapé direito.
Condições de execução	O jogo deve estar em execução
Sequencia principal	1. Se pressiona a tecla B.
Resultado esperado	O jogador vai fazer um pontapé
Avaliação	Satisfatória

TABELA 18 TESTE JOGADOR - PONTAPÉ DIREITO

Caso de Uso	Pegar objeto
Descrição	Verificar se o jogador pode pegar um objeto (goal).
Condições de execução	O jogo deve estar em execução
Sequencia principal	<ol style="list-style-type: none"> 1. O jogador passa por um goal. 2. O goal some.
Resultado esperado	Ao passar o jogador pelo goal, este último some e é acrescentado o valor do mesmo no contador do goals.
Avaliação	Satisfatória

TABELA 19 TESTE JOGADOR - PEGAR OBJETO

Caso de Uso	Dano
Descrição	Verificar se o jogador recebe dano ao estar perto dos inimigos.
Condições de execução	O jogo deve estar em execução e inimigos espalhados na cena do jogo.
Sequencia principal	<ol style="list-style-type: none"> 1. O jogador passa perto dos inimigos. 2. A vida do jogador diminui.
Resultado esperado	Ao passar o jogador perto dos inimigos, a vida do mesmo é reduzida.
Avaliação	Satisfatória

TABELA 20 TESTE JOGADOR - DANO

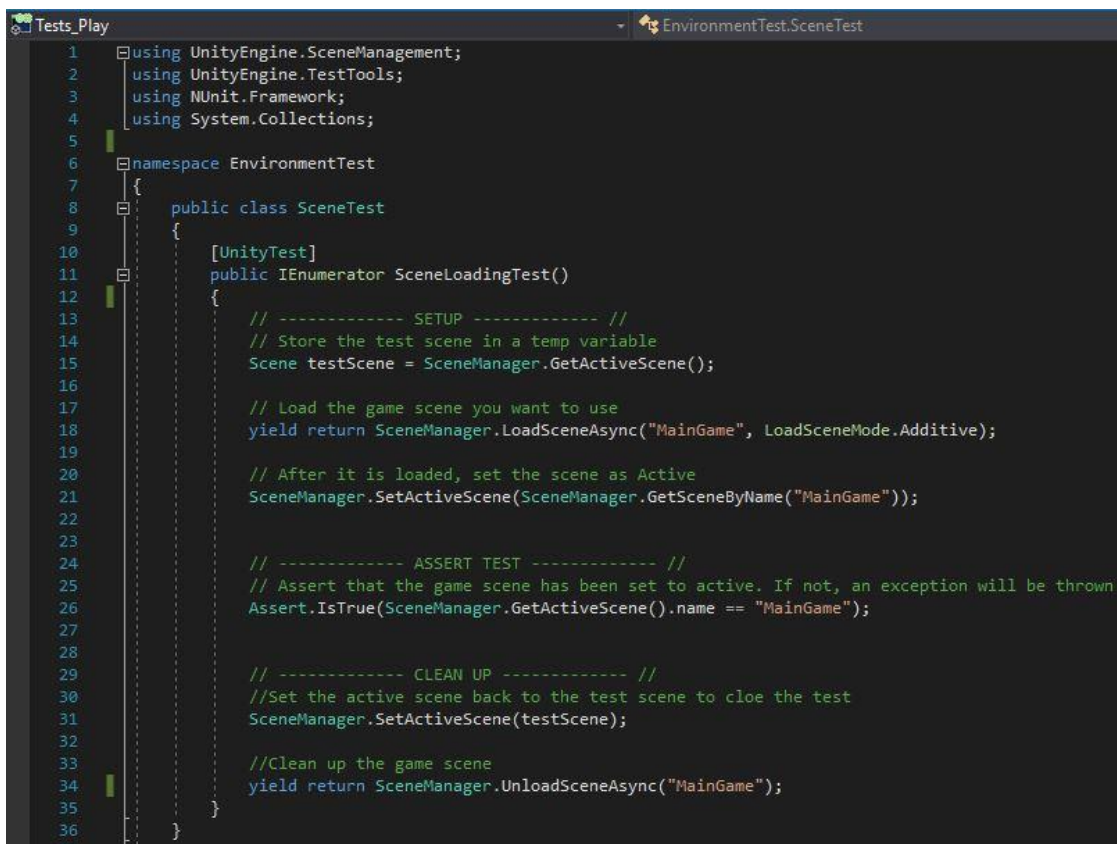
5.3- Testes automatizados

Após de pesquisar foi encontrado uma solução integrada no próprio Unity chamada Unity Test Runner que permite o desenvolvimento de testes usando a biblioteca NUnit para criar e executar estes testes. A vantagem do Editor Tests Runner

integrado ao editor em um test runner de um IDE (como o MonoDevelop ou o Visual Studio) é a possibilidade de invocar uma determinada API do Unity. Algumas APIs do Unity não estarão acessíveis se são acessadas fora do editor (por exemplo, instanciando GameObjects). Os testes são executados no editor, no modo não-reproduzível, tudo em um quadro. Não é possível pular um quadro e / ou executar uma API que requer pular quadros.

Não foi, no entanto, possível testar algumas classes, pois as mesmas dependiam diretamente do input do usuário, como, por exemplo, a classe que trata a entrada de teclado e o Kinect converte em movimento, critério compartilhado com outros autores [18].

Ainda assim, de modo geral há testes satisfatórios para carga de cena (Imagem 25), o afetação do GameObject pela física (Imagem 26) e carga de prefabs (Imagem 27).



```
1 using UnityEngine.SceneManagement;
2 using UnityEngine.TestTools;
3 using NUnit.Framework;
4 using System.Collections;
5
6 namespace EnvironmentTest
7 {
8     public class SceneTest
9     {
10         [UnityTest]
11         public IEnumerator SceneLoadingTest()
12         {
13             // ----- SETUP ----- //
14             // Store the test scene in a temp variable
15             Scene testScene = SceneManager.GetActiveScene();
16
17             // Load the game scene you want to use
18             yield return SceneManager.LoadSceneAsync("MainGame", LoadSceneMode.Additive);
19
20             // After it is loaded, set the scene as Active
21             SceneManager.SetActiveScene(SceneManager.GetSceneByName("MainGame"));
22
23             // ----- ASSERT TEST ----- //
24             // Assert that the game scene has been set to active. If not, an exception will be thrown
25             Assert.IsTrue(SceneManager.GetActiveScene().name == "MainGame");
26
27             // ----- CLEAN UP ----- //
28             //Set the active scene back to the test scene to close the test
29             SceneManager.SetActiveScene(testScene);
30
31             //Clean up the game scene
32             yield return SceneManager.UnloadSceneAsync("MainGame");
33         }
34     }
35 }
```

IMAGEM 25 TESTE - CARGA DA CENA

```
[UnityTest]
public IEnumerator GameObject_WithRigidBody_WillBeAffectedByPhysics()
{
    // ----- SETUP ----- //
    // Creating GameObject to be tested
    var go = new GameObject();

    // Adding Rigidbody component
    go.AddComponent<Rigidbody>();

    // Changing position
    var originalPosition = go.transform.position.y;

    // ----- ASSERT TEST ----- //
    // Waiting for execution FixedUpdate
    yield return new WaitForFixedUpdate();

    // Checking if are not equal a original position with other one
    Assert.AreNotEqual(originalPosition, go.transform.position.y);

    // ----- CLEAN UP ----- //
    // Finding all GameObjects from Scene test
    GameObject[] allObjects = Object.FindObjectsOfType<GameObject>();

    //Clean up the game scene
    foreach (var obj in allObjects)
        Object.Destroy(obj);
    yield return null;
}
```

IMAGEM 26 TESTE - GAMEOBJECT COM RIGIDBODY É AFETADO PELA FISICA

```
[UnityTest]
public IEnumerator Instantiates_GameObjectEnemy_From_Prefab_Test()
{
    // ----- SETUP ----- //
    // Creating o enemy
    var enemyPrefab = Resources.Load("Test/enemy");

    // Searching o enemy com o tag
    var spawnedEnemy = GameObject.FindWithTag("Enemy");

    // Getting the prefab where this enemy is allocated in scene
    var prefabOfTheSpawnedEnemy = PrefabUtility.GetPrefabParent(spawnedEnemy);

    // ----- ASSERT TEST ----- //
    // Checking if the loaded enemy is equal to will be spawned
    Assert.AreEqual(enemyPrefab, prefabOfTheSpawnedEnemy);

    // ----- CLEAN UP ----- //
    //Clean up the game scene
    foreach (var gameObject in GameObject.FindGameObjectsWithTag("Enemy"))
        Object.Destroy(gameObject);
    yield return null;
}
```

IMAGEM 27 TESTE - CARGA DE UM GAMEOBJECT TIPO INIMIGO DESDE PREFAB

Depois de ter testado os anteriores itens, foram obtidas as seguintes avaliações desde o Edit Mode (Imagem 28) e o PlayMode (Imagem 29) do Unity, validando os testes realizados.

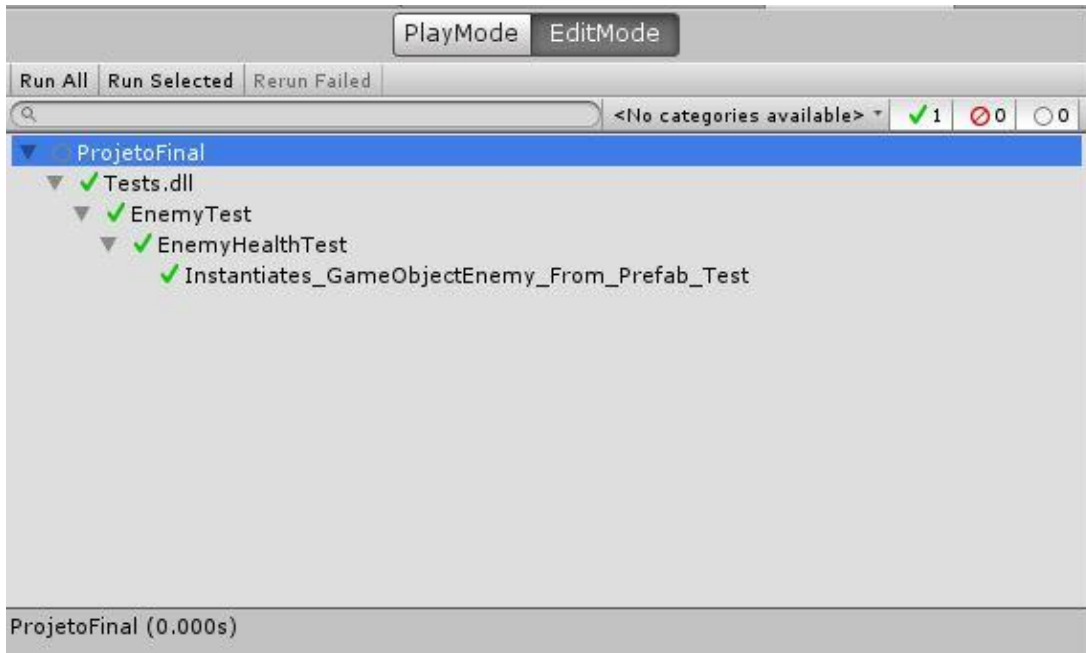


IMAGEM 28 TESTE - UNITY EDIT MODE

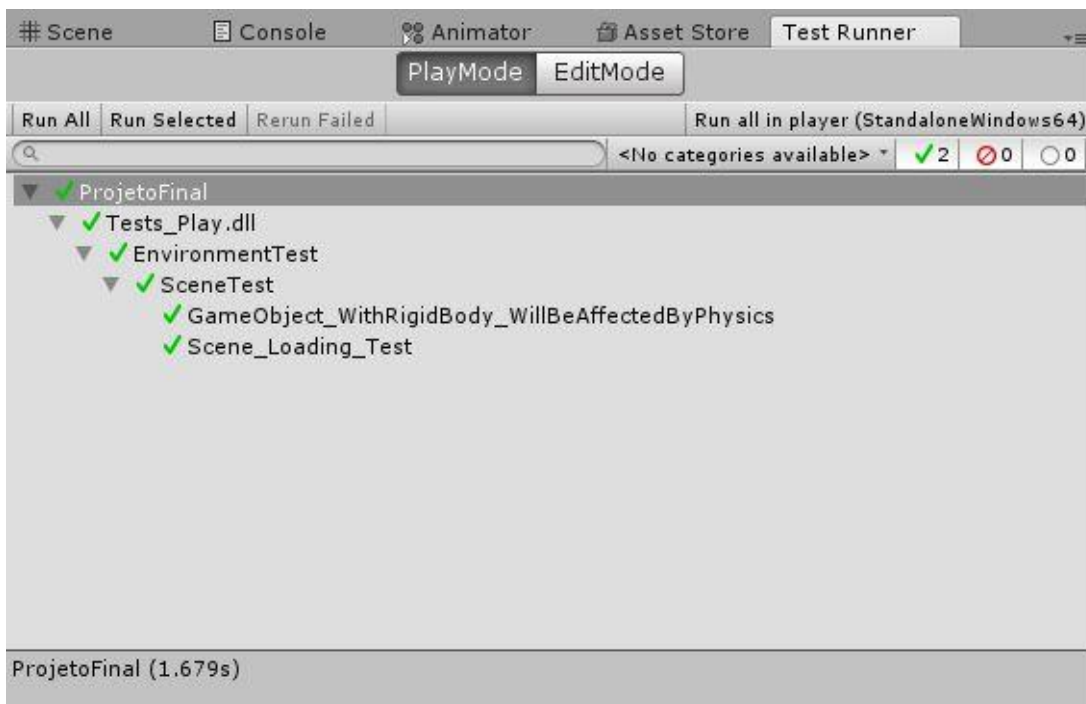


IMAGEM 29 TESTE - UNITY PLAY MODE

Conclusões

O presente trabalho consistiu em desenvolver um videojogo com o Unity3D e Microsoft Kinect para demonstrar as possibilidades destas. Foi realizada uma adaptação a um documento de design (GDD) com artefatos da engenharia de software.

O objetivo do projeto foi atingido, uma vez que o sistema proposto foi implementado dentro do prazo previsto, utilizando toda a documentação necessário para a criação de um software, inclusive testes, os quais garantem o correto funcionamento do sistema implementado.

Bibliografia

- [1] Unity, «Comunidade Unity,» 2018. [En línea]. Available: <https://unity3d.com/pt/community>.
- [2] «Forum Unity,» 2018. [En línea]. Available: <https://forum.unity.com/>.
- [3] Unity, «Unity Answers,» [En línea]. Available: <https://answers.unity.com/index.html>.
- [4] J. Bryce y J. Rutter, «In the Game – In the Flow: Presence in Public Computer,» *Computer Games & Digital Textualities*, 2001.
- [5] M. Kandroudi y T. Bratitsis, Exploring the Educational Perspectives of XBOX Kinect Based Video Games, Greece, 2013.
- [6] L. Taylor y N. Kerse, «Active Video Games for Improving Physical Performance Measures in Older People: A Meta-analysis,» *Journal of Geriatric Physical Therapy*, p. 108–123, 2018.
- [7] E. Lozano y J. Potterton, «The use of Xbox Kinect™ in a Paediatric Burns Unit,» *South African Journal of Physiotherapy*, 2018.
- [8] Woodside Capital Partners, «Game Industry Overview,» 2016. [En línea]. Available: <http://www.woodsidecap.com/wp-content/uploads/2016/12/WCP-Gaming-Industry-Overview-2016.pdf>.
- [9] S. Prescott, «PC Gamer - THE GLOBAL AUTHORITY ON PC GAMES,» 2018. [En línea]. Available: <https://www.pcgamer.com/the-forest-has-just-launched-out-of-early-access-so-heres-a-trailer-to-celebrate/>.
- [10] EA , «Plants vs. Zombies,» 2017. [En línea]. Available: <https://www.ea.com/pt-br/games/plants-vs-zombies/plants-vs-zombies-2>.
- [11] Techtudo, «Você consegue sobreviver aos perigos de I Am Alive?,» 2012. [En línea]. Available: <http://www.techtudo.com.br/tudo-sobre/i-am-alive.html>.
- [12] E. Baptista, J. R. Da Silva, E. Cardoso y P. T. Mourao, «Tutorial: Desenvolvimento de jogos com Unity3D,» de *VIII Brazilian Symposium on Games and Digital Entertainment*, Rio de Janeiro, 2009.

- [13] M. Porter, «Unity: Now You're Thinking With Components,» 2013. [En línea]. Available: <https://gamedevelopment.tutsplus.com/articles/unity-now-youre-thinking-with-components--gamedev-12492>.
- [14] B. Feijo, «Máquinas de Estado Finitas e Árvores de Comportamento,» Rio de Janeiro, 2017.
- [15] B. Feijo, «Um Middleware de Inteligencia Artificial para Jogos Digitais,» 2017. [En línea]. Available: https://www.maxwell.vrac.puc-rio.br/7861/7861_3.PDF.
- [16] E. Alves de Carvalho, *Os 5 Desprezíveis: Desenvolvimento de um Jogo Eletrônico Utilizando os Princípios de Engenharia de Software*, Brasília: Universidade de Brasília - UnB, 2003.
- [17] V. Borges Bernal, *Tipos de teste de software*, São Paulo: Escola Politécnica da USP, 2014.
- [18] P. Dam, *Ajuste Dinâmico dos Parâmetros de Estereocópia para Realidade Virtual*, Rio de Janeiro, 2011.