MIGUEL RAMSAUER NETO

CRIAÇÃO DE UM JOGO PARA MOBILE UTILIZANDO A FERRAMENTA UNITY

Assis

2015

CRIAÇÃO DE UM JOGO PARA MOBILE UTILIZANDO A FERRAMENTA UNIY

Trabalho de conclusão de curso apresentado ao curso superior de Analise e Desenvolvimento de Sistemas do Instituto Municipal de Ensino de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, com objetivo de estudar a ferramenta Unity e desenvolver um jogo para dispositivos moveis.

Orientando: Miguel Ramsauer Neto

Orientador: Doutor Luiz Ricardo Begosso

Área de Pesquisa: Ciências Exatas e da Terra

Assis

2015

Ficha Catalográfica

NETO, Miguel Ramsauer

Criação de um jogo para móbile utilizando a ferramenta Unity / Miguel Ramsauer Neto. Fundação Educacional do Município de Assis – FEMA - Assis, 2015.

42 p.

Orientador: Dr. Luiz Ricardo Begosso

Trabalho de Conclusão de Curso

Instituto de Educação Superior de Assis – IMESA.

1. Unity 2D, 2. C#.

CDD: 001.61

CRIAÇÃO DE UM JOGO PARA MOBILE UTILIZANDO A FERRAMENTA UNITY

Miguel Ramsauer Neto

Trabalho de conclusão de curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do curso de Analise e Desenvolvimento de Sistemas, analisado pela seguinte comissão examinadora:

Orientador: Doutor Luiz Ricardo Begosso

Analisadora: Diomara Martins Reigato Barros

Assis

2015

Resumo

Este trabalho descreve o estudo da game engine Unity, seu funcionamento e importância na indústria de videogames.

Unity é uma ferramenta muito usada atualmente principalmente por oferecer suporte a diversas plataformas de distribuição de software produzido com ela e fornecer suporte para a criação desde pequenos jogos por uma pessoa ou pequenos grupos até a criação de jogos de grande porte feitos por estúdios renomados e consolidados na indústria.

O objetivo final do trabalho é a criação de um jogo utilizando a ferramenta Unity, para dispositivos moveis e web browsers com intenção de adquirir experiência na área de desenvolvimento de jogos.

Palavras-chave: Unity, desenvolvimento de jogos.

Abstract

This paper describes the study of the Unity game engine, its function and relevance in the games industry.

Unity is a widely used nowadays due to its cross platform nature and possibility to be used in small projects by one or few people or by large teams by renowned studios in large projects.

The final objective of this project is the creation of one game for mobile and web browsers using Unity with the intent to acquire experience in game development.

Keywords: Unity, game development.

Lista de Figuras

Figura 1 – Interface da Unity 19

Figura 2 – Tela Scene 20

Figura 3 – Tela Game 21

Figura 4 – Tela Hierarchy 21

Figura 5 – Teste do jogo 22

Figura 6 – Tela de Projeto 22

Figura 7 – Inspetor de Objetos 23

Figura 8 – *Build Settings* 24

Figura 9 – *Player Settings* 25

Figura 10 – Loja Virtual da Unity 26

Figura 11 - Menu 27

Figura 12 – Script *DontDestroy* 28

Figura 13 – Tela de opções 28

Figura 14 – Script para alterar volume 28

Figura 15 – Comparação entre as câmeras 29

Figura 16 – Script *DestroyByBoundary* 30

Figura 17 – Função para criar inimigos 30

Figura 18 – Elementos do jogador 31

Figura 19 – Script do jogador mostrado no *Inspector* 32

Figura 20 – Variáveis do personagem principal 33

Figura 21 – Função Start to personagem principal 33

Figura 22 – Função responsável pelo movimento 33

Figura 23 – Função responsável pela colisão com inimigos 34

Figura 24 – Função Update 35

Figura 25 – Código responsável pelos disparos 36

Figura 26 – Código responsável pela bola de fogo 36

Figura 27 – Código que define o movimento dos inimigos 37

Figura 28 – Código usado nos disparos inimigos 37

Figura 29 – 'Cortando' o sprite 38

Figura 30 – Criando um clipe animado com a Unity 38

Figura 31 – Controle de animação 39

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Objetivos 11
1.2 Justificativas 11
1.3 Motivações 12
1.4 Publico Alvo 12
1.5 Estrutura do trabalho 12
2 INTRODUÇÃO AOS VIDEOGAMES 13
2.1 Breve História dos Videogames 13
2.2 Conceitos sobre videogames 13
2.2.1 Engines 14
2.2.2 Gêneros 14
2.2.3 Plataformas 16
2.2.4 Mods 17
3 TECNOLOGIAS UTILIZADAS 18
4 DESENVOLVIMENTO 27
4.1 Menu 27
4.2 Desenvolvimento do Jogo 29
4.2.1 Câmeras e elementos de suporte 29
4.2.2 Personagem Principal 31
4.2.3 Inimigos 37
4.2.4 Animações 38
5 Considerações finais 40
5.1 Conclusão 40
5.2 Trabalhos futuros 40
6 Referências 41

1 INTRODUÇÃO

Videogames são a maior indústria de entretenimento do mundo, superando tanto a indústria da música como a do cinema.

Videogames misturam artes visuais, música e literatura com os aspectos técnicos da computação, códigos em C++, C#, Java, JavaScript, LUA e inúmeras outras linguagens são utilizados em conjunto com os recursos artísticos para a criação de softwares que empurram os limites dos hardwares modernos.

Com a popularização dos dispositivos moveis videogames estão cada vez mais populares, de crianças a adultos quase todos já jogaram algum jogo, e o mercado de videogames vem crescendo muito nos últimos anos, chegando até um ponto onde ele esteja saturado com jogos de baixa qualidade devido ao apelo da chance de sucesso de jogos *indie* ou independentes, que geralmente são feitos por um indivíduo ou pequeno grupo de pessoas.

Contudo videogames continuam oferecendo entretenimento e escapismo para pessoas de todos as classes sociais, gêneros, idades e convicções.

1.1 Objetivos

O objetivo deste trabalho é o estudo da ferramenta Unity e a criação de um jogo para dispositivos moveis, visando adquirir experiência na área de game design para poder ingressar nessa área de trabalho.

Visando o estudo não só da ferramenta Unity, mas também do processo de desenvolvimento, distribuição, marketing de videogames em geral.

1.2 Justificativas

Com este projeto espero adquirir os conhecimentos necessários para ingressar na indústria de videogames, que não estão disponíveis no curso da instituição.

1.3 Motivações

Sempre fui fascinado por videogames e seu funcionamento e espero ingressar nessa indústria após o término do curso e o desenvolvimento de um trabalho sobre esse tema é um incentivo e um desafio para alcançar esse objetivo.

1.4 Publico Alvo

O publico alvo desse projeto é qualquer pessoa que tenha acesso a internet e a dispositivos moveis, pois videogames são jogados no mundo inteiro, sem distinção entre as pessoas.

1.5 Estrutura do trabalho

O primeiro capítulo apresenta introdução, o objetivo, as motivações, o público alvo e a justificativa para o desenvolvimento deste trabalho.

O segundo capítulo introduz ao leitor conceitos sobre videogames e a sua historia e importância.

O terceiro capitulo apresenta as tecnologias e ferramentas utilizadas para o desenvolvimento deste trabalho.

O quarto capitulo descreve o desenvolvimento do jogo e seus conceitos.

O quinto capitulo apresenta os trabalhos futuros e as conclusões.

O sexto capitulo contem as referências que foram utilizadas neste trabalho.

2 INTRODUÇÃO AOS VIDEOGAMES

2.1 Breve História dos Videogames

A história dos videogames começa por volta da década de 50 quando programadores e professores começaram a desenvolver pequenos jogos, simulações e conceitos de inteligência artificial em suas pesquisas sobre computação, mas somente nos anos 70 com Nolan Bushnell os videogames começaram a ser vistos como forma de entretenimento comercial, mas não fizeram muito sucesso, devido à necessidade de possuir um computador para jogá-los.

Ralph Baer foi um dos primeiros a desenvolver um console especifico para jogar videogames ligado a um aparelho de TV, sem a necessidade de utilizar computadores pessoais.

A partir da década de 70 a indústria dos videogames foi evoluindo, com o surgimento das máquinas de arcade e o Atari videogames se espalharam pelo mundo, mas sendo vistos pela maioria do publico como brinquedos.

No fim da década de 70 e começo de 80 foi a era dourada das máquinas de arcade e existiu um crescimento acelerado da indústria dos videogames, expandindo os gêneros dos jogos e os consoles para jogá-los e o surgimento de computadores pessoais capazes de servir como plataforma para jogos.

Em 1983 houve uma crise na indústria de videogames na América do Norte, onde muitas empresas declararam falência e houve a ''invasão'' das empresas Japonesas no cenário americano.

Até o final da década de 90 houve inúmeras inovações em relação aos videogames, incluindo o surgimento de gráficos em 3D e melhores tecnologias em relação ao software e hardware.

A partir dos anos 2000 a indústria dos videogames começou um crescimento estrondoso, com gráficos exuberantes, consoles e computadores ficando cada vez mais comuns, a popularização da internet e dos dispositivos moveis.

Hoje o hardware voltado para videogames, tanto consoles como PCs, são geralmente de alta tecnologia e liderando as inovações em muitos campos, além de infraestruturas como servidores para jogos online suportarem milhões de jogadores.

2.2 Conceitos sobre videogames

Neste capítulo serão tratados conceitos básicos sobre videogames, que servirão de fundação para os capítulos posteriores.

Um videogame é um jogo eletrônico, onde um ou mais usuário (ou jogador) interage com o software para obter resultados, esses resultados variam de acordo com o jogo em questão.

2.2.1 Engines

Do mesmo jeito em que softwares como o Microsoft Visual Studio e Eclipse são utilizados para a produção de sistemas, existem diversas frameworks voltadas para a produção de videogames.

Essas *engines*, como são chamadas, possuem vários níveis de complexidade e podem ser para a criação de estilos de jogos específicos ou abrangem diversos gêneros e estilos.

Uma engine deve ser capaz de habilitar o usuário programar o funcionamento do jogo e possibilitar o manuseio e aplicação de recursos gráficos e sonoros.

Assim como simulações de física, e outros recursos que possam ser adicionados ao produto final.

Exemplos: Unity, Unreal Engine.

2.2.2 Gêneros

Existem inúmeros gêneros de jogos e a cada dia mais gêneros e tipos de jogos são pensados e desenvolvidos, além de quem existem jogos que englobam diversos gêneros para uma experiência mais completa e envolvente, os gêneros dos jogos não limitam o que um jogo pode oferecer, mas sim indicam ao consumidor o que esperar do mesmo.

A seguir serão descritos alguns dos gêneros de videogames mais comuns, não serão apresentados gêneros de nicho e específicos, mas sim uma visão mais ampla dos jogos que estão sendo produzidos na atualidade.

Os gêneros mais comuns são os seguintes:

Puzzles: Jogos que envolvem a resolução de quebra-cabeças e desafios, geralmente envolvendo coordenação motora e destreza com os controles ou pensamento lógico e solução de problemas.

Exemplos: Candy Crush, Tetris.

Shooters: Jogos que envolvem o jogador atirando contra oponentes, depende do tempo de reação e precisão do jogador para alcançar seus objetivos, existem dezenas de subgêneros.

Exemplos: Doom, Counter Strike, Battefield, River Raid.

Plataformers: Jogos em que o jogador atravessa um level que é constituído por ''plataformas'', geralmente o objetivo é atravessar um level de seu começo ao fim passando por diversos obstáculos.

Exemplos: Sonic, Super Mario World, Meatboy.

RPGs: Um dos gêneros mais abundantes e de maior sucesso, jogos onde o jogador assume o controle de um ou mais personagens e contam uma historia através do jogo. Combate, progressão de personagem e enredo envolvente são comuns a esse gênero que pode ser para um jogador ou para milhões online.

Exemplos: Jogos da franquia The Elder Scrolls, Fallout, Legend of Zelda, World of Warcraft.

Simuladores: Jogos que buscam simular algo do mundo real, gênero extremamente abrangente, que varia desde jogos de futebol até simuladores de empilhadeiras.

Exemplos: FIFA, PES, The Sims, Cities Skylines.

Racing: Jogos de corrida, normalmente de carros e motos, porém o limite é a imaginação dos desenvolvedores.

Exemplos: Need For Speed, Mario Kart.

Action/Adventure: Jogos de ação e/ou aventura, que combinam combate, resolução de problemas e quebra-cabeças e destreza com os controles.

Exemplos: Tomb Raider, Uncharted, Assassins Creed.

Fighters: jogos onde a ênfase é o combate podem variar entre Arena Fighters, Spectacle Fighters, Beat'em Up e muitos outros.

Exemplos: Devil May Cry, Mortal Kombat, God of War.

Strategy: Jogos de estratégia variam entre tempo real e por turnos,

São jogos onde conhecimentos, estratégias e planejamento são essenciais para a vitória, podendo ser jogados em tempo real ou por turnos.

Exemplos: Franquia Sid Meyer, Age of Empires, Starcraft II.

Esses são apenas alguns gêneros de videogames que existem e cada um desses se divide em diversos outros, levando em consideração o estilo de arte, tipo de controles, objetivos, temática, etc.

2.2.3 Plataformas

Videogames estão disponíveis em diversas plataformas, que serão demonstradas a seguir.

Dispositivos moveis – Smartphones, tablets.

Consoles – Aparelhos que se conectam a uma TV ou monitor, como Playstation, Xbox, Wii.

E aparelhos que já vem com uma tela, *handhelds* são chamados os consoles portáteis como o Nintendo DS e o PSVITA.

Arcades – Maquinas eletrônicas que contem apenas um jogo e possuem um SO e sistema de controles próprio, essas máquinas são comumente encontradas em shoppings pelo Brasil.

Computadores pessoais – PCs comuns com sistemas operacionais Windows, Mac ou baseados em Linux.

2.2.4 Mods

Mods é o nome dado a modificações feitas por pessoas que não são os desenvolvedores de um jogo, essas modificações podem afetar todos os aspectos do jogo, de modificações visuais até modificações de como o jogo funciona.

Muitos desenvolvedores oferecem suporte a modificações em seus jogos e muitas vezes até as ferramentas para facilitar a modificação dos mesmos.

Existem casos em que jogos completos são criados baseados nessas modificações ou são inspirados por elas, por exemplo: Dota, League of Legends e Conter Strike.

3 TECNOLOGIAS UTILIZADAS

3.1 Softwares auxiliares

Esses softwares foram utilizados para criar e editar os recursos utilizados no projeto final, mas não envolveram código.

Paint.NET e GIMP foram utilizados para a criação e edição dos gráficos.

Sony Vegas foi utilizado para a edição de áudio.

Gyazoo foi utilizado para editar e criar as telas mostradas neste trabalho.

3.2 Unity3D

3.2.1 Introdução a Unity

Unity é um framework completo para a criação de jogos, pode ser adquirida em sua versão grátis ou proprietária e permite que o usuário criar seus jogos para diversas plataformas, como Android, Windows Phone, IOS, Web e PCs.

Outra grande vantagem é o suporte aos mais diversos tipos de arquivos de imagem, vídeo e som, assim como integração com sistemas de anúncios para a monetização dos jogos criados, permitindo aumentar os ganhos dos desenvolvedores assim como da própria empresa desenvolvedora da Unity.

Esta ferramenta possui 45% do mercado de *game engines* e tem mais de quatro milhões de desenvolvedores registrados para utilizar a ferramenta que oferece suporte para a criação de jogos 2D e 3D de todos os gêneros imagináveis.

Inicialmente a Unity foi usada por desenvolvedores independentes e pequenas empresas, porém nos últimos anos essa ferramenta foi utilizada para a criação de projetos de grande porte e com aclamação de críticos e fãs, jogos feitos com a Unity incluem:

Hearthstone: Heroes of Warcraft, Ori and the Blind Forest e Cities: Skylines, resaltando que o primeiro jogo desse exemplo foi desenvolvido pela gigante dos jogos Blizzard Entertainment e o segundo por um estúdio da Microsoft.

Os jogos criado na Unity podem ser comercializados sem a necessidade de adquirir a versão paga do software, porém ao alcançar renda superior a $100.000,00 em ano fiscal é necessário adquirir a versão paga para continuar a disponibilizar o jogo.

A versão gratuita e paga não tem diferenças drásticas em desempenho e funcionalidade, o diferencial da versão paga é o suporte a projetos grandes e ferramentas avançadas de teste e analise, o preço é de $75,00/mês segundo o site oficial.

Apesar de que o desenvolvimento de jogos é o foco da ferramenta ela pode ser usada para a criação de diversos outros softwares como livros interativos, simulações, artes gráficas, treinamentos virtuais e outros.

3.2.2 Funcionamento da Unity

Todo o processo de criação de um jogo é baseado no princípio de orientação a objetos, cada objeto representa um elemento do jogo e vários são instanciados no decorrer da execução.

Uma das características mais marcantes da Unity é sua interface fácil de usar e configurável, permitindo que tanto usuários iniciantes como experientes possam usufruir de suas funcionalidades.

Será utilizado um projeto pronto de uma tutorial para exemplificar as diferences funcionalidades da interface e o funcionamento da Unity.

A figura a seguir mostra a interface da Unity em sua forma padrão, no decorrer deste trabalho os principais aspectos serão analisados e estudados.

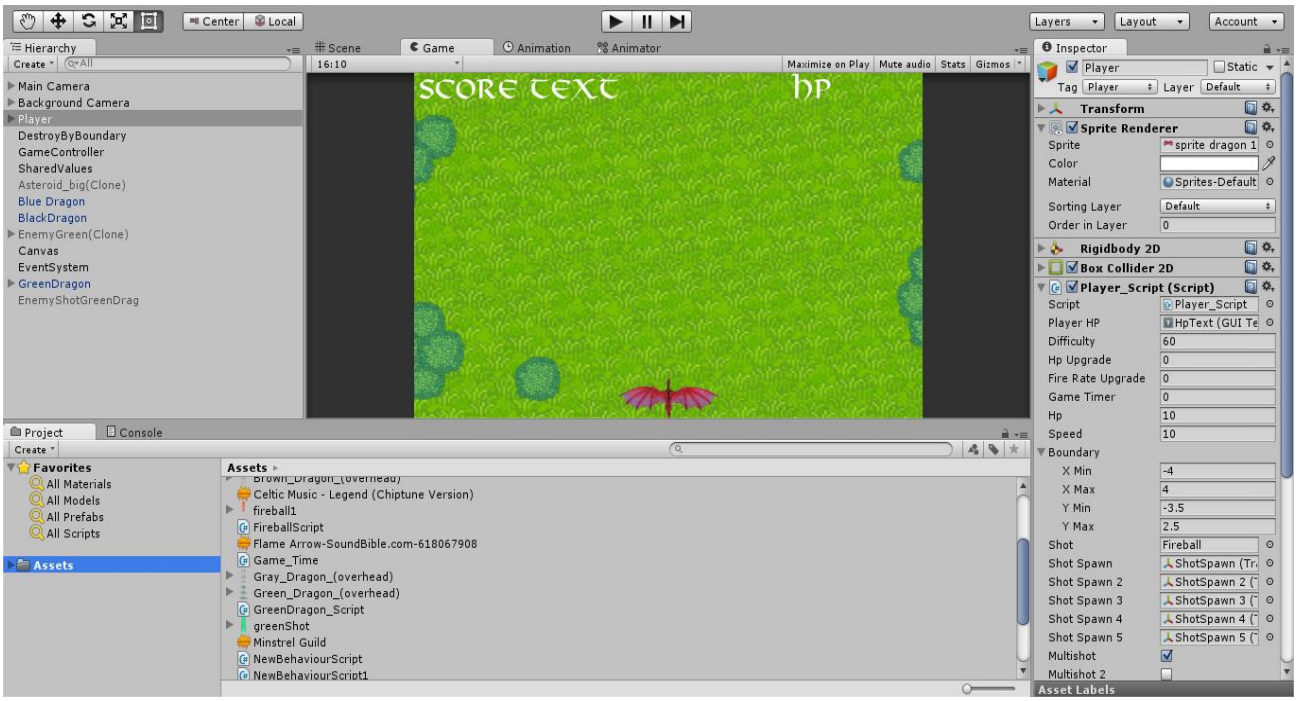

Figura 1 – Interface da Unity

O conceito principal da organização dos jogos produzidos com a Unity é o de cenas, cada cena pode conter inúmeros objetos e também é possível criar um jogo utilizando apenas uma cena.

Essas cenas contem os objetos criados pelo desenvolvedor e são o ambiente de execução do programa, uma das vantagens de se usar mais de uma cena é que os objetos só são carregados na cena que está ativa, então quando ocorre a mudança de cenas os recursos utilizados pela cena anterior são liberados tornando a execução do jogo mais eficiente.

A figura abaixo mostra a tela *scene* onde o desenvolvedor pode modificar os objetos diretamente na cena.

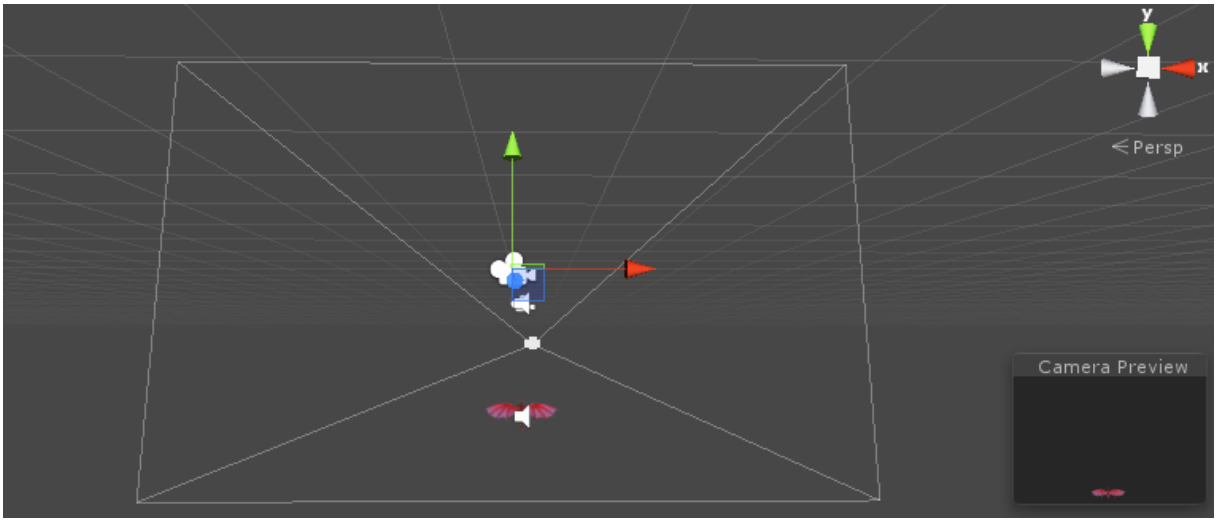

Figura 2 – Tela Scene

Essa parte da interface da Unity mostra todos os objetos que compõem a cena, até mesmo objetos que são invisíveis e desconhecidos do jogador.

Para mostrar como o projeto final está sendo feito a Unity também tem a tela *Game* que mostra como a cena será mostrada para o jogador.

Essa tela também é utilizada para testar a cena ativa no editor e é de suma importância para o desenvolvimento ter acesso rápido ao teste de execução do jogo.

Durante o teste do jogo é possível adicionar ou remover objetos e alterar os valores de seus atributos e observar os resultados em tempo real.

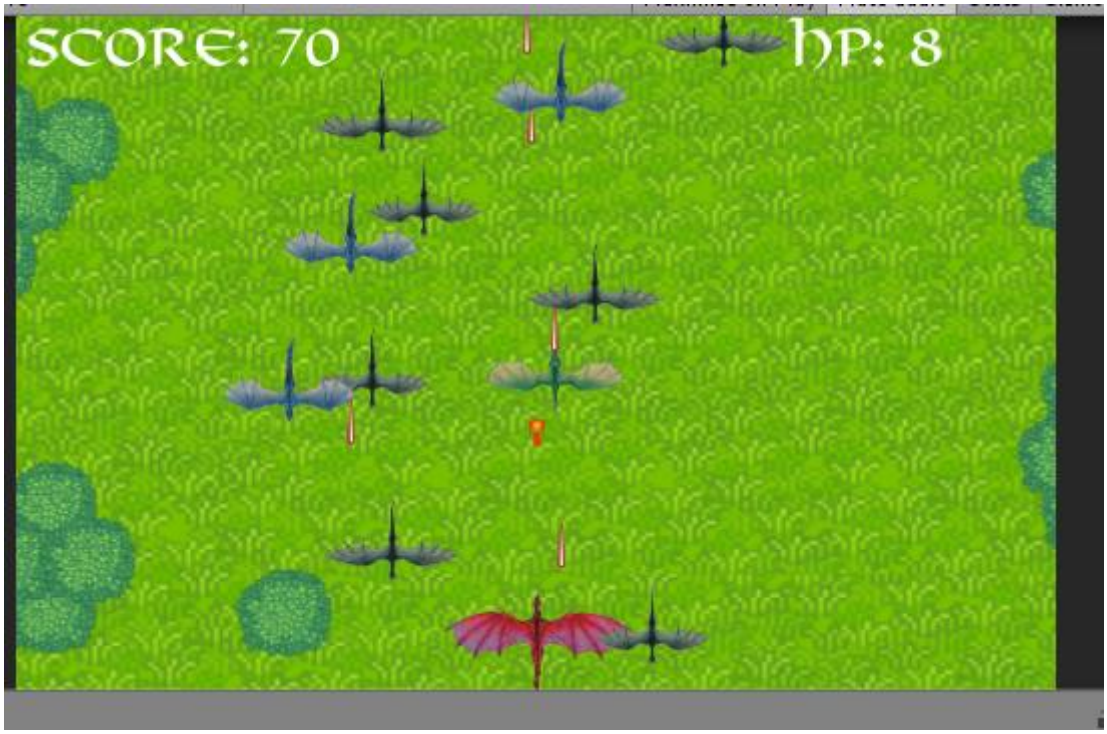

Figura 3 – Tela Game

Cada cena tem uma hierarquia de objetos, essa hierarquia contem os objetos que estão presentes no inicio da execução, como mostra a imagem a seguir:

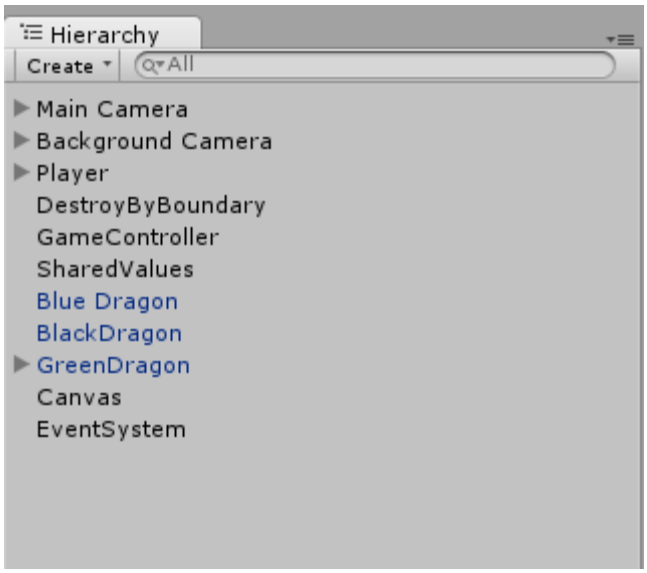

Figura 4 – Hierarchy

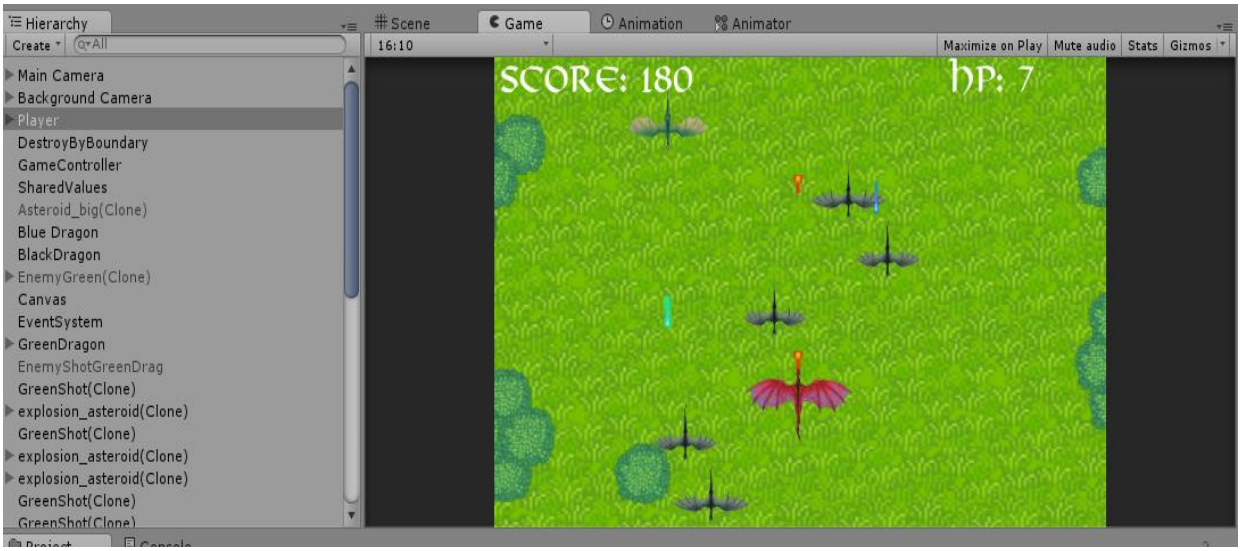

Figura 5 – Teste do jogo

Durante o teste do jogo, mostrado na figura acima, os objetos que são instanciados durante a execução do código são mostrados na tela de *hierarchy* e são vitais para o desenvolvedor entender como cada objeto interage com os outros na cena.

Os objetos que serão instanciados durante a execução do jogo são criados com o tipo *prefab*, que é permite que a Unity crie copias desse objeto facilmente para serem utilizados na cena, e qualquer mudança no *prefab* vai mudar todos os objetos criados a partir do mesmo.

Pode-se observar na tela de hierarquia na figura 5 que existem diversos objetos com a palavra (Clone) seguindo seus nomes, esses objetos foram criados utilizando o recurso de *prefab*.

Para permitir o acesso rápido e fácil a todos os arquivos que compõe o seu projeto na Unity existe uma tela de projeto que mostra todos os diretórios e arquivos contidos neles, podendo simplesmente arrastar os arquivos com o mouse para a cena atual e eles serão criados automaticamente.

Figura 6 – Tela de projeto

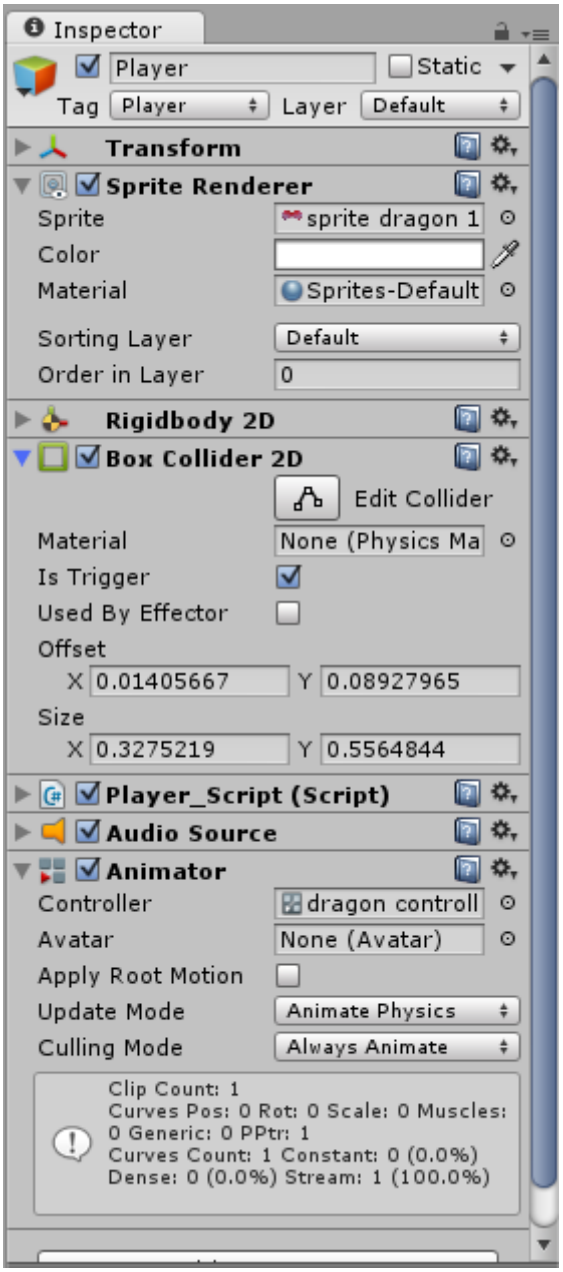

Figura 7 – Inspetor de objetos

Para cada objeto criado é possível configurar seus atributos e elementos através da tela *Inspector* que permite adicionar e modificar com simplicidade elementos e atributos a qualquer objeto selecionado.

Variáveis publicas nos códigos também podem ser alteradas através dessa tela.

Esses códigos escritos em C#, JavaScript ou Boo são a principal parte dos jogos, pois eles ditam o comportamento dos objetos e suas interações com o jogador e outros objetos durante a execução.

A Unity oferece integrado a sua ferramenta o editor de textos MonoDevelop, que é utilizado para escrever os scripts e oferece funcionalidades como completar código e bibliotecas próprias da Unity como padrão.

Essas várias bibliotecas contem funções e métodos que simulam física, leitura de entradas e interação entre objetos do jogo, além de inúmeras outras funcionalidades próprias para o desenvolvimento de jogos.

Uma das vantagens da Unity é sua portabilidade, podendo desenvolver jogos para PC, Mac, Linux, Android, IOS, Web browsers e consoles, todas essas opções são definidas na tela chamada de *Build Settings*.

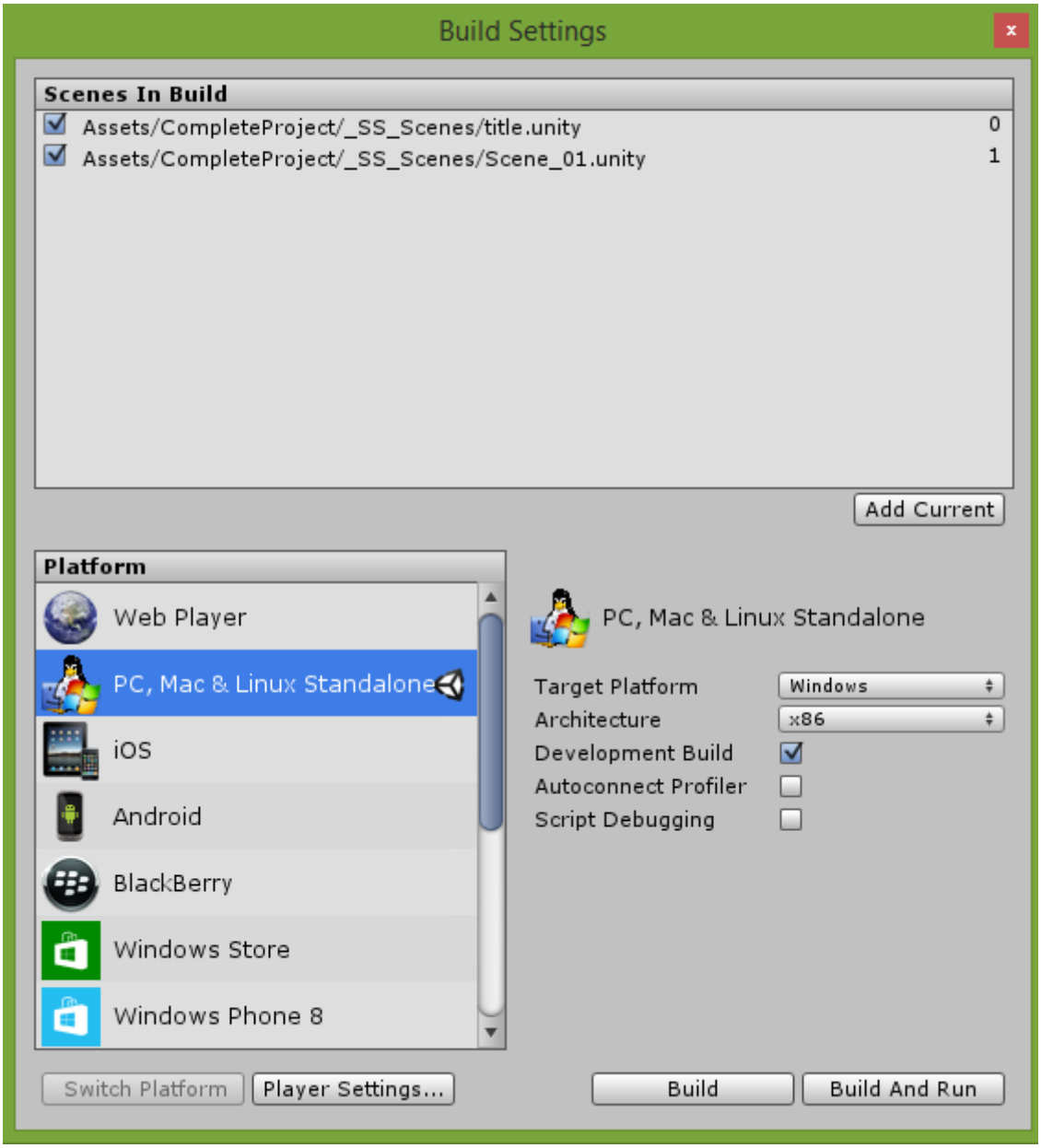

Figura 8 – *Build Settings*

Outra tela de configurações também é de extrema importância, a *Player Settings* que contem o tamanho e orientação da tela, os aspectos de tela suportados e outras opções como o ícone e o nome do arquivo e da empresa que é a autora.

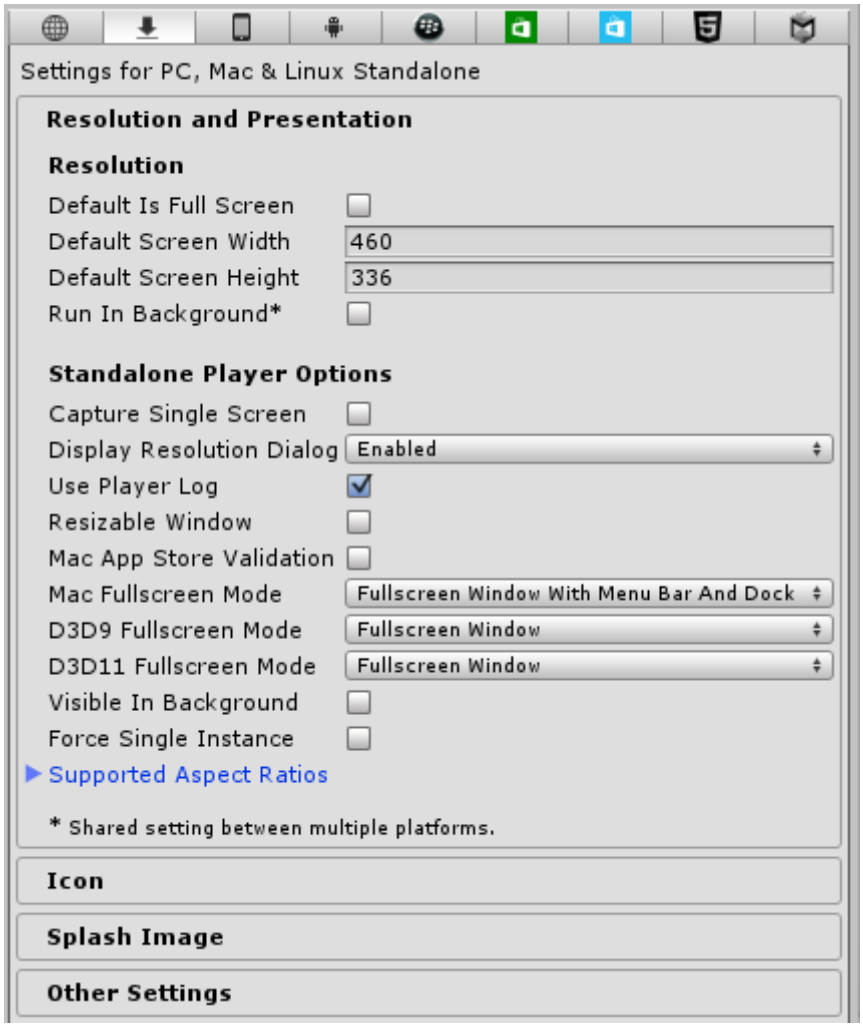

Figura 9 – *Player Settings*

3.2.3 Suporte a Unity

Após essas configurações estiverem configuradas pelo desenvolvedor a Unity cria automaticamente um arquivo compatível com a plataforma escolhida, pronto para a distribuição.

A Unity oferece a todos os usuários uma loja virtual onde qualquer usuário pode disponibilizar ou vender suas criações como objetos de um jogo ou até mesmo códigos fonte.

Nessa loja virtual também são encontradas as tutoriais da própria Unity e pacotes com recursos para novos desenvolvedores gratuitamente.

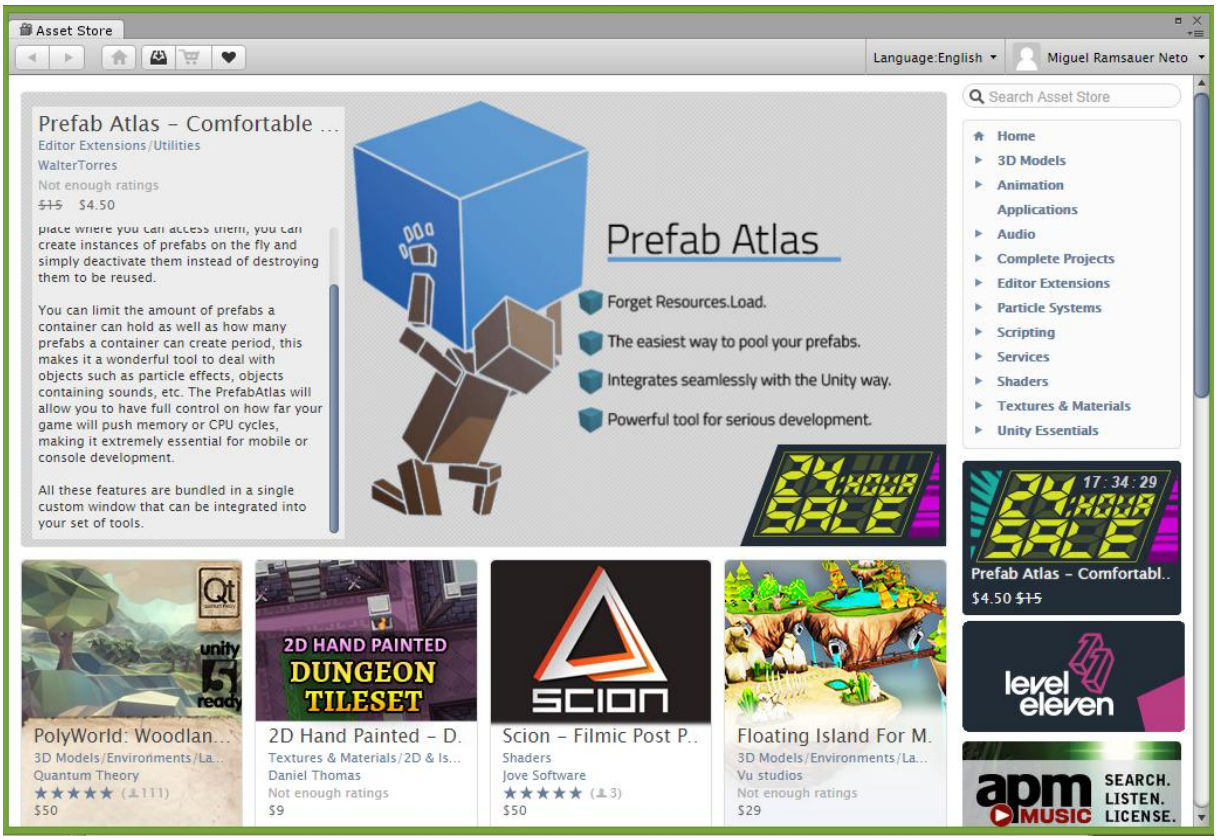

Figura 10 – Loja Virtual da Unity

E também são disponibilizados aulas e treinamentos gratuitos em como utilizar a ferramenta, um fórum para discussão e pedidos de ajuda além de uma documentação detalhada que mostra todas as funções e métodos que compõem as bibliotecas da Unity.

4 DESENVOLVIMENTO

O jogo a ser desenvolvido é um arcade shooter com graficos 2D e o jogo continuará gerando inimigos até que o jogador perca o jogo.

Cada inimigo irá recompensar o jogador com um certo número de pontos que serão adicionados a sua pontuação final, essa pontuação poderá ser comparada todos os outros jogadores.

A estrutura do jogo utilizará duas cenas, sendo uma com um menu inicial e outra com o jogo em si.

4.1 Menu

A tela de menu foi feita utilizando os elementos *UI* no editor da Unity que são próprios para a criação de interfaces para usuários.

São 3 opções apenas, para iniciar o jogo, para modificar o volume do jogo e para sair do jogo.

Figura 11 - Menu

A opção de *start* tem um script em C# que faz a troca das cenas, *opitions* invoca uma tela de menu que está invisivel ao jogador e *quit* fecha aplicação.

A tela de opções não é destruida quando muda as cenas, para poder ser utilizada quando o jogo for parado pelo jogador, o seguinte código foi utilizado para não permitier que a Unity destrua um objeto durante a execução.

```
4 public class DontDestroy : MonoBehaviour {
5     void Start()
6     {
7         DontDestroyOnLoad(this.gameObject);
8     }
9 }
10
```

Figura 12 – Script *DontDestroy*

Existe também uma tela de pausa, que é igual a tela de opções e quando a tecla ‘ESC’ é pressionada durante a execução do jogo ela é executada.

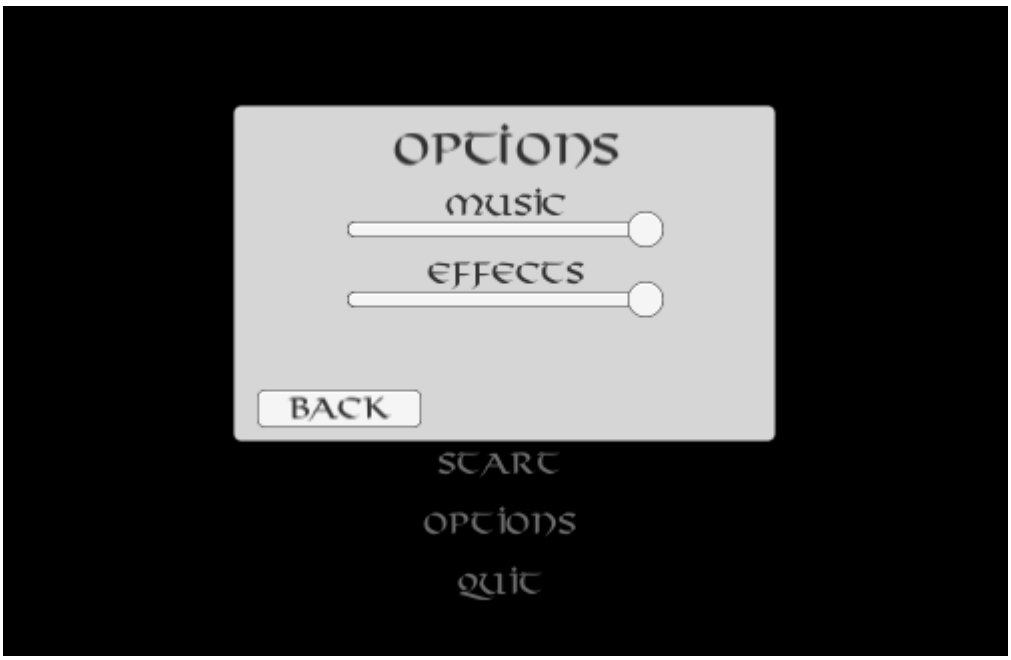

Figura 13 – Tela de opções

Na tela de opções é possível modificar a altura dos efeitos sonoros e da música do jogo através de um simples código.

```
public void SetMusicLevel(float musicLvl)
{
    mainMixer.SetFloat("musicVol", musicLvl);
}
public void SetSfxLevel(float sfxLevel)
{
    mainMixer.SetFloat("sfxVol", sfxLevel);
}
```

Figura 14 – Script para alterar volume

Cada ponto no controle deslizante tem um valor que corresponde ao volume nas configurações da Unity e esse código faz com que cada vez que o jogador mude o controle essas configurações são atualizadas.

4.2 Desenvolvimento do Jogo

4.2.1 Câmeras e elementos de suporte

Primeiramente serão descritos os objetos e scripts que não são visíveis ao jogador mas são cruciais para o funcionamento deste software.

A cena do jogo é mostrada para o jogador através de câmeras, no caso deste projeto foram criadas duas câmeras, uma principal e outra de fundo.

A câmera principal é responsável por mostrar ao jogador o seu personagem, os inimigos, os projéteis e os textos como a pontuação e mensagem de game over.

No caso da câmera de fundo ela só mostra ao jogador o plano de fundo do jogo, na imagem a seguir as duas câmeras estão mostrando a mesma área da cena, pode se observar a diferença entre as duas.

Figura 15 – Comparação entre as câmeras

Assim como as câmeras muitos outros objetos não são visíveis ao jogador e servem o propósito de configuração e manutenção da cena, como controlar a área onde o jogador pode se movimentar e a quantidade de inimigos na cena.

A seguir serão exemplificados e explicados tais objetos.

DestroyByBoundary é elemento que funciona como uma "caixa" que engloba toda a visão da câmera principal e destrói qualquer objeto do jogo que saia de dentro da mesma, para não sobrecarregar o jogo com objetos que não são mais necessários e seu funcionamento é devido a um pequeno código mostrado a seguir.

```
4  public class DestroyByBoundary_Script : MonoBehaviour
5  {
6      void OnTriggerExit2D(Collider2D other)
7      {
8          Destroy(other.gameObject);
9      }
10 }
```

Figura 16 – Script *DestroyByBoundary*

A lógica deste código é que qualquer objeto que deixar de fazer contato com sua área é destruído, o jogador é impedido de ultrapassar essa área através de outro script.

SharedValues é um objeto cujo propósito é iniciar e atualizar a pontuação do usuário na tela e mostrar a mensagem de fim de jogo quando o mesmo acontece.

GameController é um dos elementos mais importantes do projeto pois sua função é gerar os inimigos, no script desse elemento é determinado onde, com que frequência e quantos inimigos aparecem durante o jogo, a baixo será mostrado uma parte do código responsável pela criação de um dos inimigos.

```
IEnumerator enemyBlueSpawnWaves()
{
    yield return new WaitForSeconds (enemyBlue.StartWait);
    while (true)
    {
        for (int i = 0; i < enemyBlue.Count; i++)
        {
            Vector2 spawnPosition = new Vector2 (Random.Range (-spawnValues.x, spawnValues.x), spawnValues.y);
            Quaternion spawnRotation = Quaternion.identity;
            Instantiate (enemyBlue.enemyBlueObj, spawnPosition, spawnRotation);
            yield return new WaitForSeconds (enemyBlue.SpawnWait);
        }
        yield return new WaitForSeconds (enemyBlue.WaveWait);
    }
}
```

Figura 17 – Função para criar inimigos

Esse código utiliza um loop infinito para que o jogo continue a instanciar inimigos por tempo indeterminado, cada inimigo é criado em uma posição aleatória e respeita um

certo número de inimigos criados ao mesmo tempo e um tempo de espera entre cada nova onda de inimigos criados para não deixar o jogo com uma dificuldade exagerada.

4.2.2 Personagem Principal

O personagem principal do jogo é um dragão vermelho, o jogador controla a movimentação do personagem e o disparo dos projéteis é feita automaticamente.

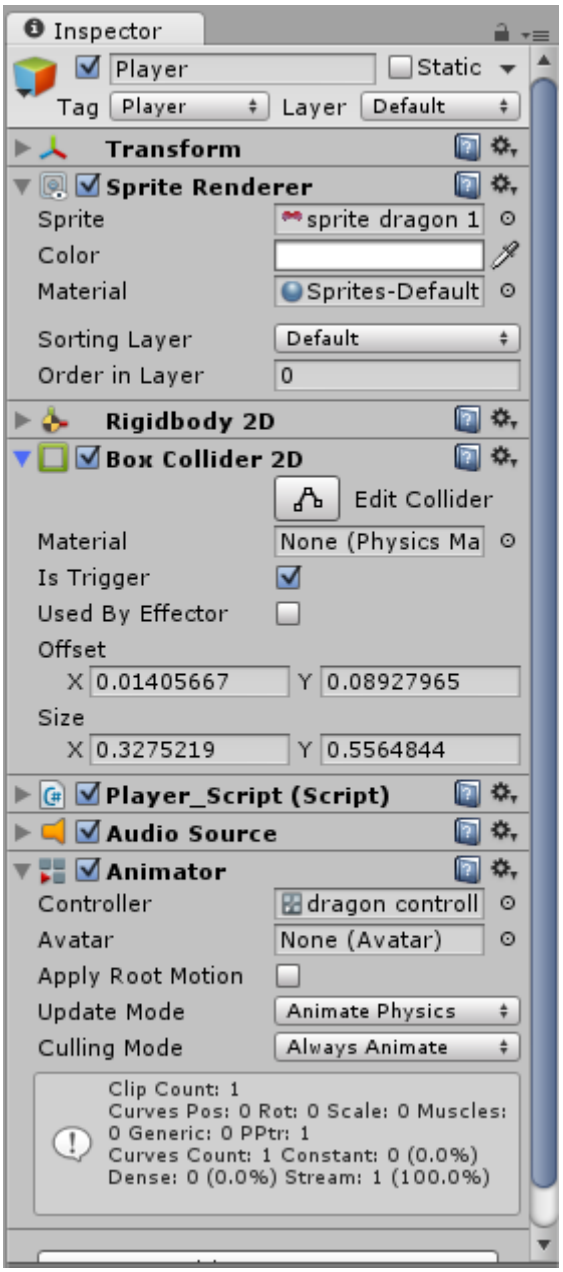

Figura 18 – Elementos do jogador

Os elementos que compõe o personagem principal e o script que determina seu comportamento serão explicados nesse capitulo.

Transform indica onde na cena o objeto está localizado.

Sprite Renderer é o componente que dá ao objeto *Player* sua representação gráfica.

Rigidibody 2D é usado para mostrar ao Unity que esse objeto tem massa e é afetado por gravidade e velocidade.

Box Collider 2D por sua vez cria uma área onde outros objetos podem colidir com o jogador, através dele que é possível as interações com os inimigos.

Audio Source é utilizado para definir o efeito sonoro que é utilizado a cada disparo.

Animator é o componente responsável pela animação do sprite, ele é composto por um *Animation Controller* que indica quando a animação deve ser reproduzida e de um clipe de animação feito com a própria ferramenta Unity.

O script que define o personagem principal será discutido em detalhes a seguir.

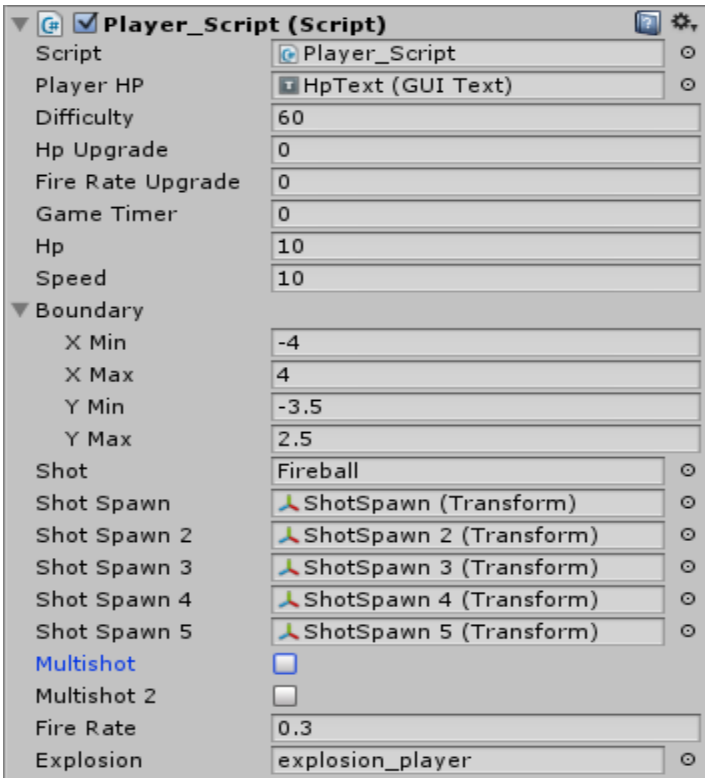

Figura 19 – Script do jogador mostrado no *Inspector*

```
public GUIText PlayerHP;
public float Difficulty = 60;
public float hpUpgrade = 0;
public float fireRateUpgrade = 0;
public float gameTimer = 0;
public float hp;
public float speed;
public Boundary boundary;
public GameObject shot;
public Transform shotSpawn;
public Transform shotSpawn2;
public Transform shotSpawn3;
public Transform shotSpawn4;
public Transform shotSpawn5;
public bool multishot;
public bool multishot2;
public float fireRate = 0.5F;
public GameObject Explosion;

private float nextFire = 0.0F;
```

Figura 20 – Variáveis do personagem principal

O código do personagem principal é responsável pela movimentação do mesmo, de limitar essa movimentação dentro da câmera principal, por disparar os projetéis que destroem os inimigos e por detectar quando um inimigo atinge o jogador.

```
void start(){
    hp = 10;
}
```

Figura 21 – Função Start do personagem principal

Quando o jogo começa a ser executado a variável ‘hp’ que determina quantas vezes o jogador pode ser atingido por inimigos recebe o valor 10.

```
void FixedUpdate ()
{
    float moveHorizontal = Input.GetAxis ("Horizontal");
    float moveVertical = Input.GetAxis ("Vertical");
    Vector2 movement = new Vector2 (moveHorizontal, moveVertical);
    GetComponent<Rigidbody2D>().velocity = movement * speed;
    GetComponent<Rigidbody2D>().position = new Vector2
        (
            Mathf.Clamp (GetComponent<Rigidbody2D>().position.x, boundary.xMin, boundary.xMax),
            Mathf.Clamp (GetComponent<Rigidbody2D>().position.y, boundary.yMin, boundary.yMax)
        );
}
```

Figura 22 – Função responsável pelo movimento

A função acima é executada toda vez que um comando direcional é utilizado pelo jogador, movimentando o personagem principal de acordo o jogador mas não permitindo que ele se movimente para fora da câmera principal.

```
void OnTriggerEnter2D(Collider2D other)
{
    if(other.tag == "Enemy" || other.tag == "EnemyShot")
    {
        Destroy(other.gameObject);
        if (hp > 0 ){
            hp--;
        }
        if (hp <= 0){
        Instantiate (Explosion, transform.position , transform.rotation);
        SharedValues_Script.gameover = true;
        Destroy(gameObject);
        }
    }
    if (other.tag == "GreenDragonShot") {
        Destroy(other.gameObject);
        if (hp > 0 ){
            hp = hp - 2;
        }
        if (hp <= 0){
            Instantiate (Explosion, transform.position , transform.rotation);
            SharedValues_Script.gameover = true;
            Destroy(gameObject);
        }
    }
}
```

Figura 23 – Função responsável pela colisão com inimigos

A função acima é responsável por controlar as colisões do personagem com os inimigos.

Cada vez que um objeto toca o personagem principal e ele é identificado pela ‘etiqueta’ como um inimigo ou um disparo inimigo ele é destruído depois essa função verifica se a variável ‘hp’ é menor ou igual a zero, se não for ela é diminuída por uma ou duas unidades dependendo de qual tipo de inimigo atingiu o personagem.

Se o valor da variável ‘hp’ for menor ou igual a zero a mensagem de fim de jogo é mostrada, um efeito de explosão é instanciado na posição do personagem principal que é destruído.

```
void Update ()
{
    PlayerHP.text = "HP: " + hp;
    multishot = false;
    multishot2 = false;
    gameTimer += Time.deltaTime;

    if (gameTimer > 90f + hpUpgrade) {
        hp = hp + 1;
        hpUpgrade = hpUpgrade + Difficulty;
        Difficulty = Difficulty + 20;
    }
    if (gameTimer >  Difficulty) {
        multishot = true;
    }
    if (gameTimer > Difficulty + Difficulty) {
        multishot2 = true;
    }
    if (gameTimer > 10f + fireRateUpgrade) {
        fireRate = fireRate -  0.02f;
        fireRateUpgrade = fireRateUpgrade + 30;
    }
```

Figura 24 – Função Update

Essa função é constantemente executada a cada quadro executado pelo jogo e é responsável pelo disparo dos projeteis pelo jogador e por modificar os atributos do personagem principal com o decorre do tempo.

Ao se passarem 60 segundos de jogo o jogador começa a disparar 3 bolas de jogo ao mesmo tempo e após mais 80 segundos ele pode disparar 5 bolas de fogo, tornando muito mais fácil destruir os inimigos, que em contra partida começam a aumentar em numero com o passar do tempo, para aumentar a dificuldade e diversão conforme o tempo avança.

A velocidade dos disparos e o numero de vezes que o jogador pode ser atingido também aumentam com o tempo para recompensar jogadores que conseguem se manter vivos com o decorrer do jogo.

Outra parte da função ‘Update’ é a de disparar, que utiliza objetos invisíveis posicionados a frente do personagem principal para direcionar os disparos, esses objetos são chamados de *shotSpawn* e cada um deles tem uma rotação diferente para cobrir a maior parte da tela possível.

O tempo entre disparos é determinado por uma variável que possui o valor de meio segundo e vai diminuindo seu valor como foi mencionado anteriormente.

```
if (Time.time > nextFire)
{
    nextFire = Time.time + fireRate;
    Instantiate (shot , shotSpawn.position ,shotSpawn.rotation);
    GetComponent<AudioSource>().Play ();
    if (multishot == true)
    {
        Instantiate (shot , shotSpawn2.position ,shotSpawn2.rotation);
        Instantiate (shot , shotSpawn3.position ,shotSpawn3.rotation);
    }
    if (multishot2 == true)
    {
        Instantiate (shot , shotSpawn4.position ,shotSpawn4.rotation);
        Instantiate (shot , shotSpawn5.position ,shotSpawn5.rotation);
    }
}
```

Figura 25 – Código responsável pelos disparos

O código acima verifica se o tempo entre verifica se o tempo entre os disparos já se passou e instancia o projétil na cena de acordo com as variáveis ‘multishoot’ e multishot2’ que determinam quantos projeteis serão disparados.

Ele também toca um efeito sonoro a cada disparo.

```
public class FireballScript : MonoBehaviour {
        public float speed;

        void Start ()
        {
            GetComponent<Rigidbody2D>().velocity = transform.up * speed;
        }
        void OnTriggerEnter2D(Collider2D other) {
        if (other.tag == "enemy") {
            Destroy(gameObject);
        }

    }
```

Figura 26 – Código responsável pela bola de fogo

O código acima faz com que cada bola de fogo disparada pelo jogador se movimente e destrua os inimigos quanto os tocar, utilizando o comando ‘Destroy ()’.

Possui somente uma variável que dita a velocidade que o projétil se movimenta na cena.

4.2.3 Inimigos

Os inimigos possuem scripts e códigos baseados no do personagem principal, porem não possuem funções mais avançadas como controles de movimento, ao contrario o movimento dos inimigos é definido assim que eles são instanciados.

```
void Start ()
{
    health = 3;
    GetComponent<Rigidbody2D>().velocity = -1 * transform.up * speed;
}
```

Figura 27 – Código que define o movimento dos inimigos.

Do mesmo jeito que o personagem principal os inimigos tem uma variável que determina quantas vezes ele pode ser atingido, e a velocidade que o inimigo recebe é negativa, pois no plano 2D que é o jogo ocorre isso causa com que o inimigo se movimente na direção contraria dos disparos do jogador.

Alguns inimigos podem disparar contra o jogador, o código que define tais disparos é bem similar ao script utilizado para as bolas de fogo.

```
public class EnemyShot_Script : MonoBehaviour
{
    public float speed;
    void Start ()
    {
        GetComponent<Rigidbody2D>().velocity = -1 * transform.up * speed;
    }
}
```

Figura 28 – Código usado nos disparos inimigos.

A velocidade do disparo inimigo tem velocidade negativa como os próprios inimigos e nota-se que diferentemente do tiro disparado pelo jogador (figura 26) ele não destrói o objeto que atinge, pois o que determina se o jogador é destruído ou não é o código do próprio personagem principal.

4.2.4 Animações

O personagem principal e os inimigos possuem animações simples feitas com a própria Unity, a seguir o processo da criação de uma animação será descrito.

Primeiramente é necessário que todas as etapas da animação estejam em uma só imagem, que é importada para o projeto da Unity e então como a imagem a seguir mostra é preciso separar a imagem em cada etapa da animação, cada imagem dessa vira um *sprite*.

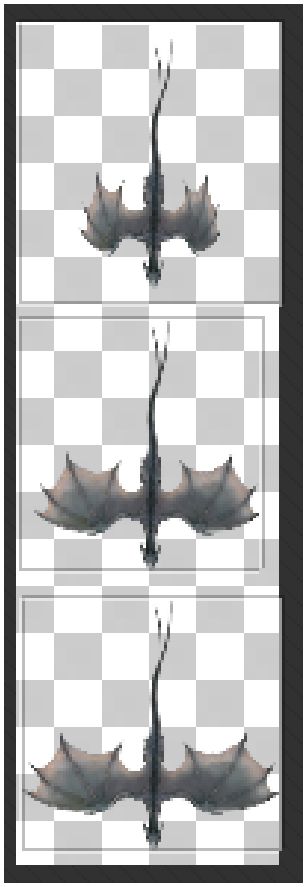

Figura 28 – 'Cortando' o sprite.

Depois que cada sprite é definido é preciso criar um clipe animado, colocando cada um dos sprites na ordem que o desenvolvedor desejar, após estarem todos ordenados é possível gravar o clipe animado para ser usado posteriormente.

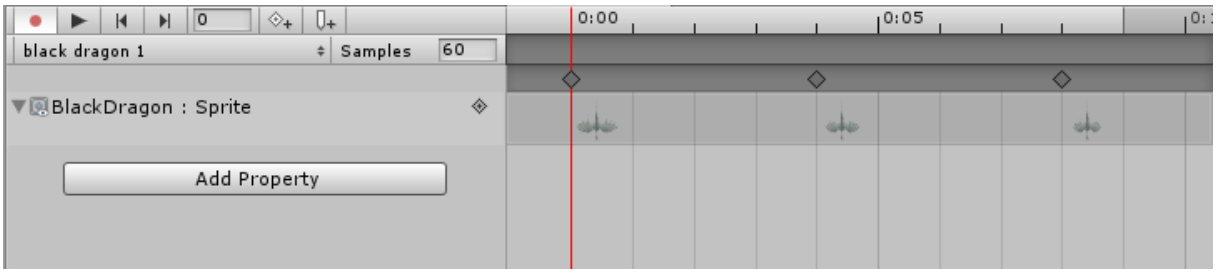

Figura 29 – Criando um clipe animado com a Unity.

Finalmente é preciso criar um controle para a animação, como as animações no jogo desenvolvido são simples o controle simplesmente repete as animações durante toda a execução do jogo.

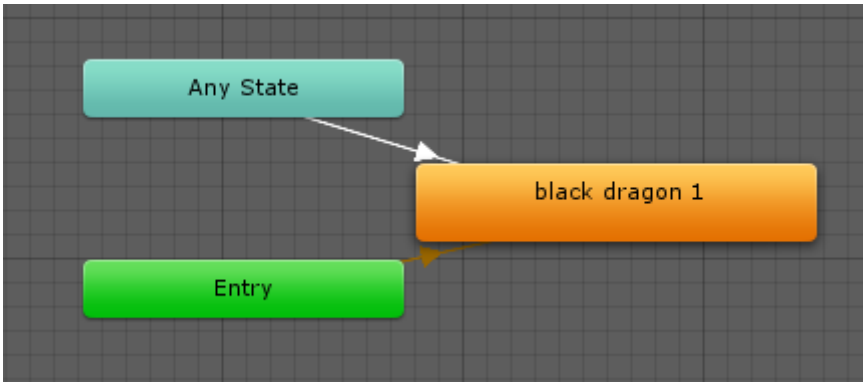

Figura 30 – Controle de animação

5 Considerações finais

5.1 Conclusão

No decorrer do projeto foi possível entender uma parte do processo da criação de um videogame e os aspectos artísticos e técnicos que são necessários para um produto final satisfatório.

O estudo da ferramenta Unity comprovou o que os números sugerem, uma poderosa ferramenta que permite a criação de jogos dos mais diversos gêneros e estilos com inúmeros recursos a disposição dos desenvolvedores, de aulas e treinamentos até recursos gráficos e códigos, a empresa auxilia todos que tiverem interesse em usar seu produto, sendo a opção mais popular no mundo inteiro para desenvolvedores novatos e veteranos.

Durante o desenvolvimento desse projeto o tema e gênero do jogo feito foram modificados inúmeras vezes, pois diferentemente de um sistema comercial não é claro que características o produto final deve ter, esse aspecto subjetivo do projeto causou certa confusão durante os trabalhos realizados.

Esse trabalho de conclusão de curso trouxe muitos conhecimentos sobre assuntos que não estão na grade curricular do curso e sobre uma indústria que vem crescendo a cada ano. Trazendo conhecimentos que serão usados tentando ingressar na indústria de videogames.

O produto final, o jogo titulado *Dragon Defence,* apesar de sua relativa simplicidade tem importante papel como o produto do aprendizado de todas essas tecnologias e conceitos aprendidos durante a realização deste projeto.

5.2 Trabalhos futuros

Aprimoramentos serão feitos ao jogo produzido para esse projeto e ele será distribuído para diversas plataformas para avaliar a recepção do público.

Com os conhecimentos adquiridos no trabalho planejo criar mais e mais jogos para construir meu conhecimento nessa área e ter mais chances de ingressar na indústria de videogames com a experiência adquirida.

6 Referências

Calabrese, David. Unity 2D Game Development. 1ª Edição. Birmingham: Packt Publishing Ltd, 2014.

Scolastici, Claudio. Unity 2D Game Development Cookbook. 1ª Edição. Birmnham: Packt Publishing Ltd, 2015.

Dreher, Thomas. History of Computer Art. Disponível em <http://iasl.uni-muenchen.de/links/GCA-VII.1e.html>. Acesso em 21 de março 2015.

Unity3D. Learn With Unity. Disponível em <http://unity3d.com/learn>. Acesso em 21 de março 2015.

Unity3D. About Us. Disponível em <http://unity3d.com/public-relations>. Acesso em 21 de março 2015.

Wikipédia. Video Game. Disponível em <http://en.wikipedia.org/wiki/Video_game>. Acesso em 21 de março 2015.

Wikipédia. History of Video Games. Disponível em <http://en.wikipedia.org/wiki/History_of_video_games>. Acesso em 21 de março 2015.

J. Louis, Sushil. History of Video games. Disponível em <http://www.cse.unr.edu/~sushil/class/games/notes/sushilHistory.ppt>. Acesso em 21 de março 2015.

Entertainment Software Rating Board, ESRB. How much do we know about Video Games?. Disponível em <http://www.esrb.org/about/video-game-industry-statistics.jsp>. Acesso em 21 de março 2015.

Unity3D. Unity Manual. Disponível em <http://docs.unity3d.com/Manual/index.html>. Acesso em 21 de março 2015.

Unity3D. Unity Answers. Disponível em <http://answers.unity3d.com/index.html>. Acesso em 21 de março 2015.

Unity3D. Gallery of software made with Unity. Disponível em <http://unity3d.com/showcase/gallery>. Acesso em 21 de março 2015.