

ESCOLA TÉCNICA ESTADUAL DE PALMARES

ENSINO MÉDIO INTEGRADO EM DESENVOLVIMENTO DE SISTEMAS

HIAGO LUIZ M. ARRUDA

JOÃO V. SILVA

MATEUS FLÁVIO H. FARIAS

DÁVILA P. REIS

DESENVOLVIMENTO DE JOGOS PARA INICIANTEs

Palmares

2022

HIAGO LUIZ M. ARRUDA
JOÃO V. SILVA
MATEUS FLÁVIO H. FARIAS
DÁVILA P. REIS

DESENVOLVIMENTO DE JOGOS PARA INICIANTEs

Projeto Integrador apresentado ao Curso
Desenvolvimento de Sistemas, da instituição
Escola Técnica Estadual de Palmares, a ser
tratado como o trabalho de conclusão de curso.

Palmares
2022

HIAGO LUIZ M. ARRUDA
JOÃO V. SILVA
MATEUS FLÁVIO H. FARIAS
DÁVILA P. REIS

DESENVOLVIMENTO DE JOGOS PARA INICIANTEs

Relatório final, apresentado a Escola Técnica Estadual de Palmares como parte das exigências para a obtenção do título de Técnico em desenvolvimento de sistemas.

Aprovado em 28 de novembro de 2022

Bancada examinadora:

KARINA JACQUELINE DE SOUZA ARAUJO
Escola Técnica Estadual de Palmares

ALBERTO FRAGA
Escola Técnica Estadual de Palmares

LAURO ANDRÉ MELO SABINO ALVES
Escola Técnica Estadual de Palmares

PALMARES / 2022

RESUMO

Com a chegada da era digital, cada dia que se passa mais pessoas ficam viciadas em conteúdos como vídeos, postagens em redes sociais, e principalmente jogos eletrônicos, para manter essas medias em constante atualização existem os desenvolvedores, uma pesquisa do NPD Group feita em 2016 constatou que os games estão entre as principais atividades dos adolescentes, jovens e adultos do país: cerca de 82% da população do país entre 13 e 19 anos joga algo nas mais diversas plataformas, sejam PCs, consoles, dispositivos móveis ou portáteis. O projeto em questão trata-se de mostrar para o espectador o desenvolvimento de jogos desktop e como fazê-lo, para isso será utilizada a game engine Unity, mostrando desde o desenvolvimento dos sprites até o funcionamento do código.

Palavras-Chave: Unity, Engine, Desenvolvimento.

ABSTRACT

With the arrival of the digital age, every day that searches for content, more people become addicted to social networks, and especially integrated games, to keep these media in constant update from the developers, one of the NPD Group made in 2016 Know what games are the main activities of teenagers, young people and adults in the country: about 2% of the country's population 13 and 19 years old plus cross-platform devices, whether PCs, consoles, mobile or portable. What in question is to show for the viewer project makes the development of desktop games and how to code it, to be used a Unity game engine this design, from the development of the games to the operation of the game.

Keywords: Unit, Engine, Development.

Sumário

1.	Introdução	07
1.1	Objetivo geral	07
1.2	Objetivos	07
1.2.1.	Geral	07
1.2.2.	Específicos	07
1.3	Justificativa	07
2.	Os Jogos eletrônicos e sua popularidade	08
2.1	O Mercado dos Jogos eletrônicos no Brasil	08
2.2	A história dos Jogos eletrônicos	08
2.3	A Chegada dos jogos no Brasil	09
3.	O Desenvolvimento de Jogos dos Primórdios à Modernidade	11
3.1	Desenvolvimento em Binário	11
3.2	O Avanço das tecnologias de 2D para o 3D	11
3.3	Os Motores Gráficos	12
4.	O Motor Gráfico Unity	14
4.1	O Que é a Unity Engine	14
4.1.1.	A Origem da Unity	14
4.1.2	Como funciona a Unity Engine	14
4.2	Game Objects	14
4.3	Criando os Controles do Jogador	16
4.3.1	Movimentação	17
4.3.2	Ataque	18
4.4	Criação de Inimigos	21
4.4.1	Inimigo Perseguidor	21
4.4.2	Dano do Jogador e do Inimigo	23
4.4.3	Geração de Inimigos Aleatória	24
4.5	Inserindo Gráficos	25
4.5.1	Terreno e Tilemap	25
4.5.2	Animações	25
5.	Conclusão	27

1. Introdução

Este Capítulo ira apresentar as principais motivações e objetivos que levaram a escolher o tema do presente trabalho.

1.1. Considerações Iniciais

O desenvolvimento de jogos está em crescimento constante, de acordo o valor investe do globo o mercado de jogos, teve entre 2019 e 2020 um crescimento de 105%, e no início da pandemia apresentou o crescimento de 140%, o público-alvo desses jogos são jovens que acabaram de entrar no mercado de trabalho ou estão entrando.

1.2. Objetivos

1.2.1. Geral

O desenvolvimento de um jogo do tipo Action Role Playing Game (ARPG) demonstrando todo o passo a passo desde a criação dos sprites, o desenvolvimento de uma história e o funcionamento dos códigos na linguagem utilizada.

1.2.2. Específicos

Demonstrar para as pessoas que nunca programaram ou tiveram muito contato com o desenvolvimento de jogos o funcionamento de seu desenvolvimento, utilizando de ferramentas de desenvolvimento de jogos como a Unity, que é muito utilizada para o desenvolvimento de jogos de celular, computadores etc. Afinal esse programa compatível com esses outros aparelhos não sendo necessário criar o jogo do zero novamente para funcionar em um dispositivo diferente do qual foi desenvolvido inicialmente.

1.3. Justificativa

A principal razão pela qual esse tema foi escolhido foi para demonstrar para iniciantes na área da informática o desenvolvimento de um jogo e sua facilidade, com intuito de os estimularem a estudar essa área.

2. Os Jogos eletrônicos e sua popularidade

Este capítulo falara sobre a popularidade dos jogos eletrônicos e sua história.

2.1. O Mercado dos Jogos eletrônicos no Brasil

Os jogos eletrônicos estão no mercado desde a década de 1980 com os primeiros lançamentos da Atari, desde então os jogos cresceram no mercado de forma que em 2021 de acordo com estimativas da Confederação Brasileira de Texas Hold'em (CBTH), mais de 7 milhões de jogadores estão atualmente ativos no Brasil, a Pesquisa Game Brasil (PGB) constatou em 2020 que 73,4% dos brasileiros dizem jogar jogos eletrônicos com uma margem de crescimento de 7,1% ao ano. Esse publico não se restringe apenas a crianças e adolescentes, 33,6% desses pessoas tem entre 25 e 34 anos. O mercado de trabalho para o publico Gamer é abrangente nas mais diversas áreas, desde streamers, youtubers, reporteres especializados e desenvolvedores.

Na semana em que o Ministro da Saúde declarou o fim do estado de emergência sanitária no Brasil foi feita uma pesquisa mostrou que durante o período pandêmico de Covid-19, 72,2% dos brasileiros se divertiram com jogos eletrônicos. Essa é uma das principais conclusões da nona edição da Pesquisa Game Brasil (PGB), desenvolvida pelo Sioux Group e Go Gamers em parceria com a Blend New Research e a Escola Superior de Propaganda e Marketing. Em 2022, o estudo ouviu 13.051 pessoas em 26 estados e no Distrito Federal.

Um levantamento do Instituto Data Favela, em parceria com a Locomotiva — Pesquisa e Estratégia e a CUFA (Central Única das Favelas), aponta que 96% dos jovens moradores de comunidades do Brasil gostariam de ser gamers profissionais. De acordo com o estudo, homens jovens com renda de até um salário mínimo foram os que mais demonstraram essa vontade, isso porque a carreira de gamer profissional aparece como uma oportunidade de transformação da realidade. Os jogos podem trazer uma ascensão rápida que pode chegar a permitir com que o gamer possa ajudar toda a sua família financeiramente.

2.2. A História Dos Jogos Eletrônicos

O primeiro jogo eletrônico foi desenvolvido na década de 1950 sendo um dos primeiros um jogo similar ao pong que foi lançado na década de 1970, nessa época devido ao alto custo, grande consumo de energia e a necessidade de se empregar uma equipe altamente treinada para manter e operar as máquinas, a tecnologia da computação ficou limitada para

organizações maiores. Por conta disso, a criação dos primeiros jogos eletrônicos limitou-se a testes e demonstrações de teorias relacionadas a áreas como a interação humano-computador, a aprendizagem adaptativa e estratégia militar. De certa forma se assemelhando a Internet que era muito limitada na época, Por causa da documentação é difícil afirmar qual foi de fato o primeiro jogo porem o que mais se destaca é o tennis for two que foi o jogo falado anteriormente. Ainda assim os jogos só começaram a serem desenvolvidos para lazer como Lunar Lander, que foi o primeiro jogo voltado ao lazer e comercializado com gráficos vetoriais, na forma de wireframes, ou seja, os objetos eram formados por linhas como se fossem o esqueleto de um modelo 3D nesse nasceu o antecessor dos gráficos poligonais, usados na maioria dos jogos da atualidade, Atualmente jogos 3D usam polígonos mesmo que não pareça como o jogo god of war de 2018 onde apenas o rosto do protagonista Kratos tem cerca de 80.000 polígonos.

Voltando um pouco na década de 80 nasceu um jogo muito icônico conhecido com Space Invader que foi lançado para Atari 2600, nessa década também nasceu outros jogos icônicos como o pac-man para arcade sendo o mais famoso, nesta década foi onde ocorreu a popularização dos jogos eletrônicos com Mário Bros, Metal gear, Vampire Killers(antecessor da franquia castlevania), The Legend of Zelda, entre vários outros que estão vivos até os dias de hoje podendo ser considerado o “boom” dessa indústria, mas antes dessa popularização, houve a crise norte-americana dos jogos eletrônicos que foi uma grande recessão nessa indústria que ocorreu de 1983 até 1985 nos Estados Unidos. A saturação do mercado de jogos eletrônicos na segunda geração de consoles junto más decisões da líder Atari, e a ascensão do computador pessoal fizeram várias companhias de consoles quebrarem ou abandonarem o meio.

2.3. A Chegada dos jogos no Brasil

Em 1991 a indústria brasileira de games, era composta basicamente de consoles e jogos, a Tec Toy (Master System, Mega Drive e Game Gear) e famiclones (Nome popular dado aos diversos tipos clones do videogame Japonês Farmicon da Nintendo) de diversos fabricantes, além de consoles anteriores (Atari etc.), movimentou 100 milhões de dólares, segundo uma revista de videogames da época(Nintendo Power), o Brasil era considerado nos anos 1990, como ainda é, o 4º. maior mercado de games do mundo, atrás de Japão, Estados Unidos e Europa. Mas nas gerações seguintes o mercado foi minguando e a enxurrada de pirataria na era PlayStation e PlayStation 2, que beirava os 90%, fez destruir o mercado gamer, que era restrito a produtos oficiais Nintendo (Game Boy Color, Nintendo 64, Game

Boy Advance e GameCube) e Sega (Dreamcast, Saturn, Mega Drive e Master System). No entanto a Sega, através de seu representante, a Tec Toy Ind. de Brinquedos também era muito prejudicada pela pirataria, com o Dreamcast e o Saturn, ao contrário da Nintendo que usava cartuchos como mídia principal. Já com o contra-ataque recente através de recursos digitais das empresas à pirataria o mercado nacional voltou a crescer e fez diminuir sensivelmente a pirataria.

3. O Desenvolvimento de Jogos dos Primórdios à Modernidade

Este capítulo explicará o funcionamento do desenvolvimento de jogos desde as tecnologias básicas que os compõem até as mais notáveis como os gráficos.

3.1. Desenvolvimento em Binário

As primeiras aplicações a serem criadas para computadores utilizavam um sistema de cartões perfurados no qual os furos em posições específicas podiam significar funções predefinidas que poderiam ser interpretadas pelas máquinas projetadas para isso, onde perfurado poderia ser interpretado como sim ou verdadeiro (True) e onde não estavam perfurados poderiam ser interpretados como não ou falso (False). O sistema utilizado pelos computadores na atualidade, o código binário, pode ser visto como uma versão aprimorada dos cartões perfurados, ele é originário do trabalho do matemático inglês Charles Bull, ele publicou o sistema lógico que precedeu a lógica utilizada em todos os hardwares e softwares na atualidade.

O código binário assim como também é conhecido a língua do computador, foi desenvolvida durante a Segunda Guerra Mundial e a corrida espacial, e ainda que o código binário seja um pouco velho ele é utilizado até os dias de hoje, porque ele possui algumas vantagens muito boas por exemplo a sua simplicidade já que ele é composto por 0 e 1, Por exemplo a frase “trabalho de PI Desenvolvimento de jogo” em código binário é “01110100 01110010 01100001 01100010 01100001 01101100 01101000 01101111 00100000 01100100 01100101 00100000 01010000 01001001 00100000 01000100 01100101 01110011 01100101 01101110 01110110 01101111 01101100 01110110 01101001 01101101 01100101 01101110 01110100 01101111 00100000 01100100 01100101 00100000 01101010 01101111 01100111 01101111”, então ele é facilmente entendido pelas máquinas além da segurança dada já que o código binário possibilita que uma grande massa de informações seja gravada de forma íntegra e segura já que o código binário possibilita que uma grande massa de informações seja gravada de forma íntegra e segura, e o motivo de não ser visto no dia a dia em todas as máquinas é simplesmente porque os códigos binários já são convertidos automaticamente na chamada *Linguagem de auto nível* na qual o desenvolvedor pode escrever o código com palavras e símbolos que tem significados predefinidos.

3.2. O Avanço das tecnologias de 2D para o 3D

Os gráficos dos jogos é aplicações é feito com base na renderização de imagens sejam elas criadas a partir do código ou através da importação de um arquivo, porem esse sistema consistia apenas em duas dimensões (2D) que dentro do código eram representadas como em

um plano cartesiano onde a letra X representava largura e a letra Y representava altura porem não havia uma forma de calcular a profundidade tendo apenas essas duas dimensões (x, y).

A Criação de gráficos em três dimensões (3D) começou na década de 80 porem com as tecnologias da época era muito custoso criar esse tipo de gráfico no qual alem da largura e da altura também é adicionada a profundidade com um terceiro eixo o Z (X, Y, Z) para o efeito de profundidade ser possível tem que serem adicionados cálculos em tempo real de luz e sombras o que não eram totalmente possível naquela época, o primeiro jogo em 3D só veio ser lançado de verdade em 1994 com o jogo Star Fox para o Super Nintendo Entertainment System(SNES) no qual o cartucho do jogo tinha um chip extra responsável por fazer esses cálculos.

Na atualidade as tecnologias evoluíram o suficiente para que o 3D seja utilizado na maioria quando não em todos os jogos, sendo esse 3D moderno quase tão realista que dependendo da situação é difícil diferenciar se é um jogo ou um filme, tecnologias como o traçado de raios (Ray Tracing) permitem que os cálculos de dispersão da luz e sombras sejam tão realistas como a dispersão da luz no mundo real, para isso as placas gráficas que são as responsáveis pelos cálculos executam estes cálculos executam bilhões de operações por segundo.

3.3. Os Motores Gráficos

O Motor Gráfico, Engine Gráfica ou Motor de Jogo é uma biblioteca ou conjunto de ferramentas direcionadas a facilitar o desenvolvimento de jogos, essas ferramentas poupam tempo para escrever linhas de códigos ou funções que são utilizadas com muita frequência como por exemplo um código de colisão ou da inteligência artificial de um inimigo.

A primeira Engine a ser utilizada comercialmente foi a Freescape em 1987, ela foi usada na criação de diversos jogos de FPS (First Person Shooter) como Driller e Dark Side porem o termo “*Engine*” só veio a ser criado na década seguinte onde os jogos do gênero se popularizaram e ela passou a ser usada constantemente na criação desses, o trio dos famosos jogos: Doom, Quake e Wolfenstein foram os principais responsáveis pela popularização dos FPS.

Na Atualidade as Engines são as principais responsáveis na criação de jogos, algumas como: Unreal Engine, Unity Engine, Source Engine, Construct 3 e Game Maker Studio permitem que alem do seu uso profissional, também possam ser utilizadas de forma pessoal ou estudantil no desenvolvimento de jogos apenas por estudo, a Game Maker Studio por exemplo não utiliza diretamente de uma linguagem de programação, apenas através de sua interface é possível criar jogos completos, o título de sucesso Undertale foi criado totalmente

nela, Enquanto outras como a Unity e a Unreal disponibilizam dentro de sua própria interface ou site oficial, documentações e tutorias de vídeo que ensinam o seu uso.

4. Motor Gráfico Unity

Neste capítulo será abordado o Motor Gráfico Unity e o desenvolvimento de jogos através dele.

4.1. O Que é a Unity Engine

4.1.1. A Origem da Unity

O Motor Gráfico Unity Engine foi criado pela empresa Unity Technologies que foi fundada na Dinamarca em 2004 com o nome de “*Over the Edge Entertainment*” que fora rebatizada em 2007 para o nome atual, o seu principal produto é a Unity Engine que foi lançada em 2005 em uma conferencia da Apple, ela foi criada originalmente para ser utilizada apenas no MAC mas posteriormente teve uma versão liberada para Windows apenas em navegadores. Em apenas 2 anos a Unity teve uma aceitação do publico bastante clorosa e um desenvolvimento absurdo, onde em suas atualizações mais de 50 novos recursos para os desenvolvedores.

Segundo o próprio site oficial do software a Unity Engine, antes chamado de Unity 3D, é um software que centraliza tudo o que é necessário para poder desenvolver videojogos. Ou seja, a Unity é uma ferramenta que permite criar jogos para diversas plataformas (PC, consoles, mobile, VR e AR) utilizando um editor visual e programação através de scripting, oferecendo aos utilizadores ferramentas profissionais, capazes de preencher os requisitos de qualquer jogo, prova disso, são os jogos muito famosos criados com a Unity como: “Monument Valley”, “GRIS”, “Cuphead”, “Hollow Knight”, “HearthStone”, “Rust”, “Genshin Impact”, e outros tantos que, provavelmente, preencheriam uma página completa.

4.1.2. Como funciona a Unity Engine

O motor de jogos Unity 3D possui uma interface bastante simples e amigável para o desenvolvimento de jogos de diversos gêneros. Sua área de trabalho é composta de várias janelas chamadas “views”, cada uma com um propósito específico. É possível criar games para iOS, Android, BlackBerry, Windows Phone ou Windows. No programa Unity são utilizados diversos tipos de linguagem de programação, principalmente as linguagens C++ e C#, nos scripts dos objetos do jogo. Estas mesmas também são aplicadas em outros jogos, mesmo fora da Unity, por serem versáteis e terem um bom desempenho. Os games do Unity são baseados em cenas, os Game Objects são praticamente todos os objetos dentro da cena.

4.2. Game Objects

Game objects ou em português objetos de jogo, como citado anteriormente são tudo que

esta dentro de uma cena na unity, ele tem como base um sistema de coordenadas nas quais eles estão situados dentro do dos eixos X, Y e Z, independentemente de se o projeto tem como base o 3D ou o 2D, esses sistemas de coordenadas são utilizados em conjunto de métodos lógicos para definir se um objeto se move, esta estático ou se movera apenas em situações específicas e no resto do tempo ficara estático.

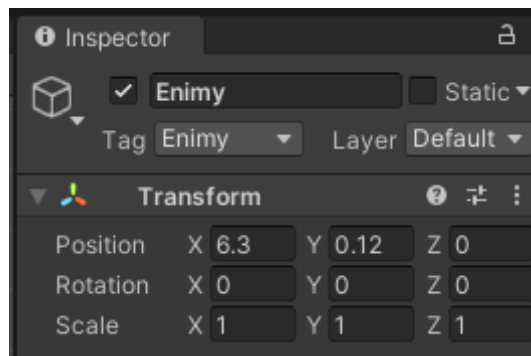


Figura 01 - GameObject chamado Enemy

A função mais comum dos GameObjects é receber componentes, esses componentes são códigos ou bibliotecas completas nos quais tem funções comumente utilizadas entre desenvolvedores como: Iluminação, Camera de Jogo, Colisão, entre outras inúmeras funções, porem códigos criados pelo programador também são adicionados como componentes e podem interagir com os demais componentes, tornado eles ativados ou desativados. Para criar um GameObject é necessário apenas clicar com o botão direito do mouse na View Hierarchy, apos isso irão aparecer diversas opções de GameObjects: 2D, 3D, Empty e alguns mais avançados, clicando em Empty será criado um objeto como nome escolhido por você no qual terá apenas o componente básico transform, se quiser adicionar outros componentes basta clicar na opção que esta na View “Inspector” Add Component.

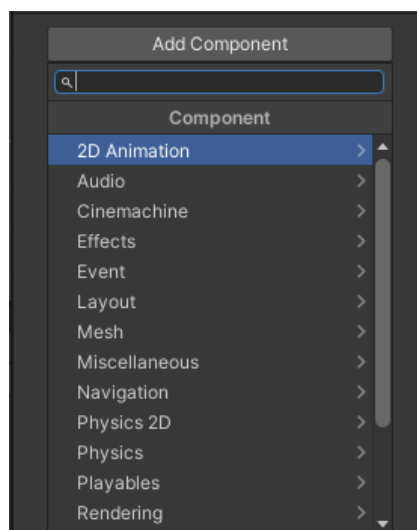


Figura 02 - Tela de Adição de Componentes

Lá você poderá escolher um componente ou um Script para definir funções predefinidas ou personalizadas para o objeto criado, criando um dos objetos citados anteriormente como 2D ou 3D eles já virão com alguns componentes que são responsáveis por coisas como a renderização dos polígonos no caso de objetos 3D e de Sprites no caso de Objetos 2D.

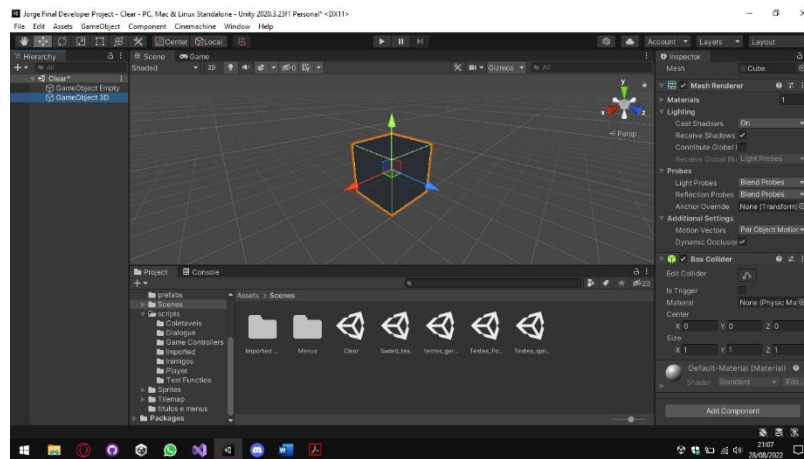


Figura 03 - Objeto 3D

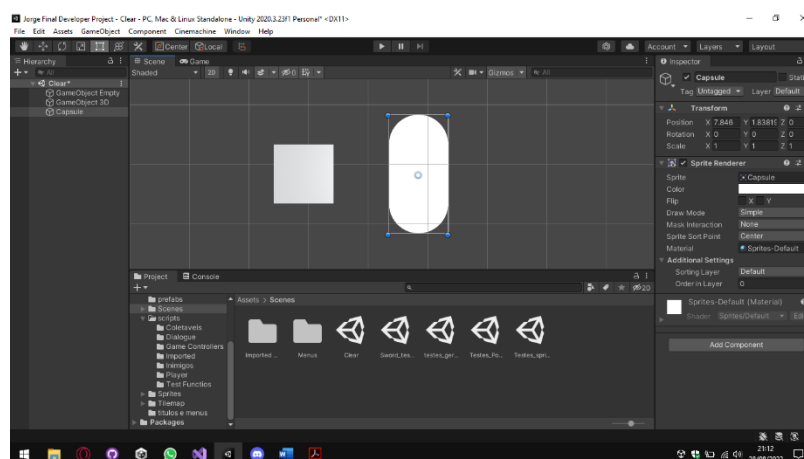


Figura 04 – Objeto 2D

Como observado nas figuras 03 e 04 o mesmo objeto que na primeira é visto como um polígono em 3d na segunda é visto como um sprite em 2d isso ocorre porque a unity utiliza um sistema de cameras híbrido, no qual a camera é um componente que pode ser configurado através da interface, sem a necessidade de codificar.

4.3. Criando os Controles do Jogador

Algumas funções para um GameObject tem de ser definidas através de códigos que apos importarem as bibliotecas dos componentes citados anteriormente podem alterar ou definir parâmetros no componente transform. De acordo com a documentação da Unity Engine para Importar as bibliotecas básicas de componentes indispensável na Unity você pode criar um script na linguagem de programação C# através da própria engine com o botão esquerdo do

mouse ou você pode realizar a importação manualmente digitando na primeira linha do script o comando “*Using UnityEngine;*” e criando métodos vazios nomeados Start e Update.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class DemonstrationScript : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
```

Figura 05 – Script criado através da Unity

Os métodos funcionam da seguinte forma o Start executa todos os comandos inseridos nele quando o jogo é aberto, enquanto o Update executa todos os comandos constantemente cerca de 30 a 60 vezes por segundo.

4.3.1. Movimentação

Para realizar a movimentação de um objeto na Unity será utilizado apenas o componente básico Transform que controla: a posição, a rotação e a escala do objeto, desta forma mudando gradualmente a posição do objeto se obtém uma impressão de movimento. O comando para mudar a posição de objeto na Unity consiste em *Transform.Position += (mudança em X, Mudança em Y) * Tempo decorrido * Velocidade;* Isso significa que: o atributo position da classe Transform é incrementado em determinado valor nas posições X, Y e Z de acordo com o tempo e a velocidade.

```
5 public class DemonstrationScript : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     public float speed;
9     void Start()
10    {
11    }
12
13    // Update is called once per frame
14    void Update()
15    {
16        Move();
17    }
18    void Move()
19    {
20        Vector3 movimento = new Vector3(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"), 0f);
21        transform.position += movimento * Time.deltaTime * speed;
22    }
23 }
```

Figura 06 – Código Básico de movimentação com a UnityEngine

```
Vector3 movimento = new Vector3(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"), 0f);  
Vector3 movimento = new Vector3(+1f, -1f, 0f);
```

Figura 07 – Exemplificação de como seriam lidos os Inputs

Para definir de uma única vez os valores de X, Y e Z a Unity contém os métodos Vector2 e Vector3 que podem conter respectivamente 2 e 3 valores que já serão identificados como representantes de X, Y e Z. Para não ter a necessidade de serem criados longos scripts de controles a Unity já tem integrado um sistema de Input que pode ser configurado na interface e chamado com simples comandos iniciados com a palavra Input seguida da palavra Get e função de controle requerida, neste caso foi utilizado o Axis que permite que mais de uma tecla seja lida por vez de forma ao menos uma tecla quando pressionada responda com o número 1 de forma positiva (+1) e outra quando pressionada responda como o número 1 de forma negativa (-1).

4.3.2. Ataque

Existem diversas formas de realizar um ataque em um jogo de visão de cima também conhecido pelo termo *Top Down*, é possível realizar ataques corpo a corpo em que o jogador precisa estar próximo ao inimigo para o atingir como com uma espada, faca ou mesmo os próprios punhos dos personagens, ataques a longa distância no qual o jogador arremessa algum projétil como arcos e flechas, armas de fogo ou armas de arremesso, ou ataques de tempo determinado nos quais o jogador arremessa ou solta algum objeto que causará dano no inimigo depois de um tempo como bombas ou granadas ou durante um tempo como venenos ou artefactos que diminuem a vida dentro de uma área, vale ressaltar que os ataques de tempo determinado também podem valer como ataque em área já que na maioria dos seus casos eles provocam dano em todas as entidades que estão na sua área sendo inimigos ou jogadores.

Para este exemplo será utilizada uma espada que é uma arma corpo a corpo, inicialmente é criado um segundo objeto dentro do objeto do jogador, esse objeto receberá o componente de colisão, na Unity existem diversos tipos de colisores para diversas situações, neste caso foi utilizado o colisor em forma de quadrado chamado Box Collider 2D

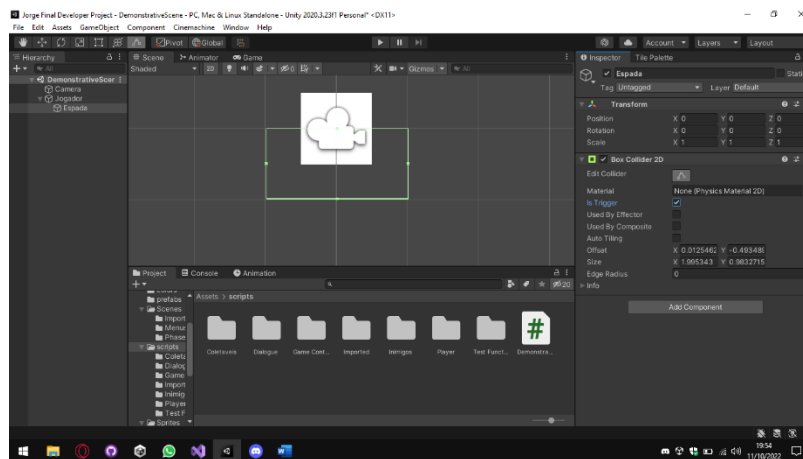


Figura 08 – Objeto Da Espada Configurado

Este colisor foi configurado com um formato em que ele entre em contato com os inimigos há uma distância considerável do jogador. Também foi selecionada a opção Is Trigger na qual ao invés do colisor simular uma colisão real e parar o movimento de quem entrar em contato com ele o colisor ficara impercetível e apenas enviara para o código a informação que ele entrou em contato com alguma coisa e se solicitado o que entrou em contato. Vale ressaltar que apos terminar de configurar o objeto ele deve ser desativado, dessa forma apenas quando a arma for ativada ele ira realizar o teste se ela entra ou não em contato com alguma coisa. Dentro do código deve ser criada uma variável da Unity do tipo GameObject, dentro dela você colocara o objeto a qual você esta se referindo quando a chamar, essa variável foi nomeada *Espada* e dentro dela foi passado o objeto como o colisor, depois disso um novo método chamado *Wepon* foi criado e nele utilizaremos novamente o Input desta vez usaremos o *GetButtonDown* que significa detectar quando a tecla for pressionada, então desta forma a ação só será feita uma vez, e para repeti-la será necessário apertar novamente o botão, vale ressaltar que a ação contraria também pode ser detectada caso necessário, para testar se a tecla foi solta o comando é *GetButtonUp*

```

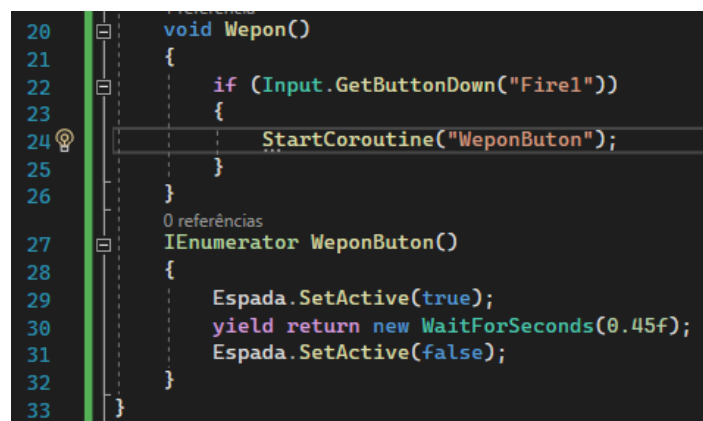
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  Script do Unity (1 referência de ativo) | 0 referências
6  public class DemonstrationScript : MonoBehaviour
7  {
8      public float speed;
9      public GameObject Espada;
10     // Update is called once per frame
11     void Update()
12     {
13         Move();
14     }
15     1 referência
16     void Move()
17     {
18         Vector3 movimento = new Vector3(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"), 0f);
19         transform.position += movimento * Time.deltaTime * speed;
20     }
21     0 referências
22     void Wepon()
23     {
24         if (Input.GetButtonDown("Fire1"))
25         {
26         }
27     }
28 }

```

Figura 09 – Atualização do Script

Desta vez o Input foi colocado dentro de uma condicional e o que for colocado dentro dela

será executado quando o botão configurado no Fire1 for pressionado. Como o objeto dos colisores ficara desativado até que seja pressionado o botão é preciso que ele seja desativado novamente apos um tempo, a forma mais simples de fazer essa contagem de tempo é através de uma Corrotina que é um componente já pronto que esta disponível na maioria das linguagens de programação e tem como função exatamente a inicialização e finalização de serviços e tarefas, para criar uma corrotina fora de um método deve ser posto o comando *IEnumerator NomeDaCorrotina () {}*



```

20 void Wepon()
21 {
22     if (Input.GetButtonDown("Fire1"))
23     {
24         StartCoroutine("WeponButon");
25     }
26 }
27 0 referências
28 IEnumerator WeponButon()
29 {
30     Espada.SetActive(true);
31     yield return new WaitForSeconds(0.45f);
32     Espada.SetActive(false);
33 }

```

Figura 10 – Corrotina criada e configurada

dentro das chaves será posto o que ela deve fazer e desfazer, neste caso foi mandado ativar e desativar o objeto da espada para ativar ou desativar um objeto é preciso apenas colocar o nome da variável que contem o objeto ponto SetActive e dentro de parenteses se você quer ativado ou não (True ou False). A questão do controle de tempo foi feita com um comando da Unity chamado WaitForSeconds com ele é possível fazer uma pausa na leitura do código pelo tempo em segundos colocado dentro dos parenteses, no final a sequência que será lida é, ative o objeto espada, espere 0.45 segundos, desative o objeto espada. Para usar o WaitForSeconds deve ser posto: `yield return new WaitForSeconds(tempo)` que na tradução literal colheita retorna novo espere por segundos, isso significa que o código voltara a ser lido após um número de segundos.

Com isso feito sempre que o jogador pressionar um botão definido a área definida da espada testara se ela entrou e contato com algum inimigo, caso entre o inimigo terá uma parte de sua vida reduzida, porem da forma que está apenas a parte de baixo realizaria o teste, para resolver isso o objeto com o colisor deve ser rotacionado de acordo com a região que o personagem está olhando. Para rotacionar o objeto será utilizado o componente Transform, como o jogador poderá virar para qualquer lado a qualquer momento o objeto não pode ser rotacionado com incrementos como foi no caso da movimentação e sim devem ser utilizadas

rotações fixas para as 4 posições possíveis, os ângulos que correspondem são os de 0, 90, 180 e 270 graus. Então no fim o código ficaria, se o personagem está virado para baixo a espada esta rotacionada em 0 graus, se o personagem está virado para a direita a espada esta rotacionada em 90 graus, se o personagem estiver virado para cima a esta estará rotacionada em 180 graus e se o personagem estiver virado para a esquerda a espada estará rotacionada em 270 graus.

```
20 void Wepon()
21 {
22     if (Input.GetAxis("Vertical") < 0 && Input.GetAxis("Horizontal") == 0)
23     {
24         Espada.transform.eulerAngles = new Vector3(0f, 0f, 0f);
25     }
26     if (Input.GetAxis("Horizontal") > 0 && Input.GetAxis("Vertical") == 0)
27     {
28         Espada.transform.eulerAngles = new Vector3(0f, 0f, 90f);
29     }
30     if (Input.GetAxis("Vertical") > 0 && Input.GetAxis("Horizontal") == 0)
31     {
32         Espada.transform.eulerAngles = new Vector3(0f, 0f, 180f);
33     }
34     if (Input.GetAxis("Horizontal") < 0 && Input.GetAxis("Vertical") == 0)
35     {
36         Espada.transform.eulerAngles = new Vector3(0f, 0f, 270f);
37     }
38
39     if (Input.GetButtonDown("Fire1"))
40     {
41         StartCoroutine("WeponButon");
42     }
43 }
```

Figura 11 – Código de Rotação da Espada

Para realizar esses testes de para onde o personagem está olhando foi utilizada como condição a função input, de forma que para a direção em que a direção que o personagem andou por último é a direção que ele está olhando.

4.4. Criação de Inimigos

Dentro dos jogos pode haver diversos tipos de inimigos, ou até mesmo pode não haver necessariamente um inimigo claro, os inimigos podem ser desde simples monstros como slimes, saqueadores que tem os mesmos padrões de ataques que o jogador a bestas colossais que nas quais o jogador precisa criar uma grande estratégia para poder destruir, mas também o inimigo pode ser o próprio intelecto do jogador, ou seja o objetivo é que seja completada uma serie de desafios e puzzles para vencer o jogo. Dentro deste projeto o objetivo do jogo é que o jogador consiga matar o máximo de inimigo dentro do menor tempo possível, as três maiores pontuações ficaram salvas na tela de titulo.

4.4.1. Inimigo Perseguidor

Para o sistema pelo qual o jogador acumula pontos ao matar inimigos quanto mais simples o inimigo melhor, isso pois o que mais o jogador deve querer nessa situação é velocidade e praticidade, e isso não seria possível se ele tivesse de pensar em estratégias para cada inimigo que fosse matar. O inimigo mais simples que pode ser projetado dentro de um jogo do tipo Top Down que é o estilo utilizado neste exemplo seria um inimigo persegue o jogador e ao entrar em contato com ele o jogador recebe dano. Para criar um inimigo que persiga o jogador na Unity é preciso apenas uma única linha

de código, isso se dar, pois, essa função é algo tão utilizado na criação de jogos que dentro da biblioteca da Unity existem métodos já prontos que executam essas funções sendo preciso apenas passar os parâmetros. Criando um script para ser adicionado ao inimigo ele precisara de algumas variáveis primeiramente uma variável do tipo Transform que servirá para definir a posição atual do jogador e outra do tipo Float para definir a velocidade do inimigo, com isso no método start a variável com a posição do jogador recebera como valor a própria posição do jogador, desta forma não importa aonde o jogador esteja o inimigo saberá aonde ele está.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 // Script de Unity | 0 referências
6 public class Desmonstrative_Ennmy : MonoBehaviour
7 {
8     private Transform posPlayer;
9     public float EniSpeed;
10
11     // Mensagem do Unity | 0 referências
12     void Start()
13     {
14         posPlayer = GameObject.FindGameObjectWithTag("Player").transform;
15     }
16     // Mensagem do Unity | 0 referências
17     void Update()
18     {
19         SeguirPlayer();
20     }
21
22     // 1 referência
23     void SeguirPlayer()
24     {
25         if (posPlayer.gameObject != null)
26         {
27             transform.position = Vector2.MoveTowards(transform.position, posPlayer.position, EniSpeed * Time.deltaTime);
28         }
29     }
30 }
```

Figura 12 – Código que permitem o inimigo perseguir o jogador

Com isso o inimigo já pode seguir o jogador aplicando dentro do transform.position o método MoveTowards com ele o objeto que esta com o script se moverá entre 2 pontos em uma determinada velocidade, sendo assim a forma que foi escrita pode ser lida como: “se mova da sua posição até a posição do Jogador em determinada velocidade em relação ao tempo”.

Aplicando o script dentro da Unity também devem ser configuradas as Tags do inglês etiquetas que são o que permite que o código reconheça o que é cada objeto, no inimigo é aplicada uma tag como o nome inimigo e no jogador uma com o nome jogador. Se o jogo for iniciado agora o inimigo irá automaticamente perseguir o jogador, para que o inimigo persiga o jogador apenas se o jogador chegar a uma determinada distancia do inimigo é necessário criar uma forma que reconheça se o jogador entrou ou não há uma certa distancia do inimigo, para isso poderia ser utilizado o componente Collider2D porem como o inimigo já teria um colisor próprio para não causar conflitos ou métodos excessivos, uma forma diferente de reconhecer se o jogador está há uma determinada distancia do inimigos é criar uma variável do tipo Collider2D pela qual dentro do próprio código é configurado seu tamanho e posição, de certa forma esta variável não estará no inimigo porem ela sempre receberá como posição central a posição do inimigo

```

33 void SeguirPlayer()...
40 1 referência
41 void AvistouTeste()
42 {
43     Collider2D MuitoPerto = Physics2D.OverlapCircle(transform.position, radius, playerLayer);
44     if (MuitoPerto != null)
45     {
46         Avistou = true;
47     }
48 }

```

Figura 13 – Sistema de reconhecimento de distância do jogador

Dentro do componente criado foi passado o método Physics2D que é responsável pela física da Unity, ou seja, por tudo que envolva: colisão, gravidade, forças ou impulsos, e dentro dele foi mandado criar um círculo pelo qual se algo entrar em contato com a “layer” que fora passada como parâmetro a variável receberá como valor o nome do objeto que entrou em contato e continha a tag que fora passada como parâmetro. Desta forma agora o Inimigo apenas se moverá quando o Jogador entrar em uma distância definida através do código do inimigo.

4.4.2. Dano do jogador e do inimigo

Dentro de um jogo para que haja um nível de desafio é necessário que haja uma punição para o caso em que um jogador faça algo que não está dentro das regras do jogo ou que signifique um erro da parte do jogador, geralmente esta punição se dar por perder instantaneamente a partida, entretanto em jogos do estilo ARPG como é o caso deste, é muito mais comum utilizar um sistema de vidas pelo qual o jogador pode recuperar ou perder vidas até um certo ponto, caso ele perca completamente a vida ele perde. Criar um sistema de vidas para o jogador pode ser considerado uma das partes mais simples do jogo, pois é preciso criar apenas uma única variável e uma condicional que diga que se a vida chegar a zero o jogador morre.

Por mais que um sistema de condições seja uma das funções mais simples que podem ser programadas a condição para que a vida do jogador diminua ele deve entrar em contato com o inimigo, detectar a colisão entre dois objetos normalmente seria a parte mais complicado do código por ser necessário diversas formas matemáticas porém, dentro de engines como a Unity já vem implementados métodos de realizar testes de colisão entre objetos, no caso da Unity estes se chamam “OnCollisionEnter” e “OnTriggerEnter” eles servem respectivamente para: se o objeto com o método entrar em contato com um objeto sólido e se o objeto com o método entrar em contato com um objeto translúcido.

```

91 void OnCollisionEnter2D(Collision2D collision)
92 {
93     switch (collision.gameObject.tag)
94     {
95         case "Extra_Life":
96             Player_life += 1;
97             break;
98         case "Enemy":
99             Player_life -= 1;
100             break;
101     }
102 }
103
104
105
106
107

```

Figura 14 – Teste de Colisão do Jogador com o inimigo

Sendo assim quando o jogador entrar em contato com um objeto com a tag “inimigo” ele perderá um ponto de vida e se ele entrar em contato com um objeto com a tag “Vida_Extra” ele ganhará um ponto de vida. Já para o inimigo tomar dano é necessário apenas repetir o mesmo código entretanto neste caso: se o inimigo entrar em contato com um objeto com a tag “Espada” ele morrerá instantaneamente.

4.4.3. Geração de Inimigos Aleatória

A aleatoriedade é considerada como parte da mecânica, a partir dela permite-se que jogos digitais ou analógicos sejam mais desafiadores e permitam o jogador tomar decisões que eram inesperadas. De forma resumida os objetos serão diferentes ou estarão posicionados em outros locais a cada nova partida, os inimigos irão agir e tomar rotas distintas a cada partida também, trazendo assim um fator inesperado para o jogador, deixando o jogo mais divertido e menos monótono. Um sistema de geração de aleatoriedade pode ser encontrado em todas as linguagens de programação sendo neste caso a da linguagem C# quando for passar um valor para uma variável numérica sendo preciso apenas no lugar de colocar um numero específico seja colocado o comando “Random.Range(0, 10)” dentro destes dois atributos que foram colocados nos parenteses devem ser colocados o valor mínimo e o máximo respectivamente, sendo assim sorteado um numero entre esses valores, por exemplo nesta demonstração seria sorteado um numero entre zero de dez.

Para este projeto a aleatoriedade pode ser aplicada de forma que ao jogo ser aberto serão gerados um número de inimigos aleatórios em posições aleatórias, assim sempre que o jogo for jogado os inimigos estarão em locais e quantidades diferentes aumentando a chance de ré-jogabilidade do jogo. Para criar os inimigos sem precisar programar na Unity existem os chamados “Prefabs” estes fazem com que você possa salvar um objeto com configurações prontas para que o desenvolvedor possa recriá-lo através do código ou exportá-lo para projetos diferentes. Dentro do código fora feito um laço de repetição que gera um novo inimigo em uma posição aleatória um número de vezes que é sorteado no início do jogo.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 // Script da Unity (1 referência de assets) 0 referências
6 public class GeraçãoDeInimigos : MonoBehaviour
7 {
8     Vector3 LugarDeSpawn = new Vector3(0,0,0);
9     public GameObject inimigo;
10    public int minimoDeInimigos = 1, maximoDeInimigos = 10; //ajuste o minimo e o maximo para o sorteio
11
12    private float X, Y;
13
14    public float RangeMinX, RangeMaxX, RangeMinY, RangeMaxY;
15
16    // Messages da Unity 0 referências
17    void Start()
18    {
19        int quantidade = Random.Range(minimoDeInimigos, maximoDeInimigos); // aqui acontece o sorteio da quantidade de inimigos
20        for (int x = 0; x < quantidade; x++) //Loop de Repetição
21        {
22            X = Random.Range(RangeMinX, RangeMaxX); //Sorteia a posição em X do Inimigo
23            Y = Random.Range(RangeMinY, RangeMaxY); //Sorteia a posição em Y do Inimigo
24            LugarDeSpawn = new Vector3(X, Y, 0f); //Junta as posições em X e em Y em um unico vetor
25            Instantiate(inimigo, LugarDeSpawn, transform.rotation); // instancia um inimigo aleatorio
26        }
27    }
28 }
```

Figura 15 – Script de geração de inimigos

4.5. Inserindo Gráficos

Os gráficos podem ser considerados como um cartão de visitas para um jogo, já que a parte visual é vai ser a primeira coisa que vai chamar a atenção do jogador, os gráficos são um dos fatores mais importantes para a venda dos jogos, já que dependendo podem os tornam mais imersivos como no jogo “Gran Turismo” que os fans reclamam pelo jogo parecer mais lento por ter uma pegada realista isso porque jogos como “Need for Speed” colocam efeitos para mostrar velocidade oque da ao jogador uma certa imersão. Os motores gráficos ou Engines tem como principal função exatamente isso, facilitar a inserção dos gráficos nos jogos, normalmente para colocar gráficos através de próprio código renderiam centenas de linhas, porem dentro das engines é possível adicionar imagens, desenhar cenários e utilizar efeitos de partículas e iluminação com poucos cliques do mouse.

4.5.1. Terreno e Tilemap

O cenário muitas vezes pode ser considerado muitas vezes um dos pontos chave mais importantes em um jogo, isso se dar afinal um bom cenário pode render cenas ou batalhas memoráveis dentro do jogo, um cenário pode ser previamente planejado e desenhado através de outro programa, porem uma forma mais volátil de criar cenários é através de peças que possam ser montadas de acordo com a criatividade do desenvolvedor e possam ser alteradas com facilidade caso necessário, dentro da Unity2D existe uma espécie de programa capaz de realizar essa “pintura de cenário” que se chama “TilleMap” este permite que seja criado um grid onde possam ser encaixadas essas peças foram desenhadas e configuradas para funcionar em paralelo com o tilemap

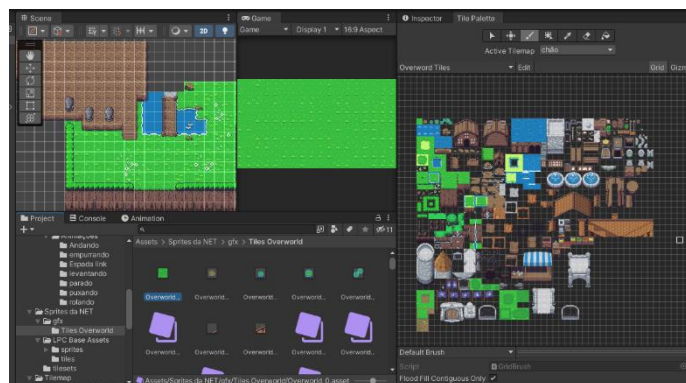


Figura 16 – Tile Palette sendo usada

Para este projeto foram utilizados sprites disponibilizados gratuitamente na internet para uso acadêmico porem em casos de jogos com objetivos comerciais que seriam publicados o uso de sprites sem permissão é crime.

4.5.2. Animações

Objetos que se movem como o próprio jogador, na maioria dos casos precisam que seus sprites sejam animados, de forma que torne a movimentação mais realista, pode se dizer que dentro da Unity criar uma animação é uma das coisas mais simples que podem ser feitas, isso pois é preciso apenas

levar os frames da animação a tela de nome “Animator” e uma caixa de de salvamento automaticamente será aberta onde será dado o nome daquela animação



Figura 17 – Animação do Jogador

Para esse projeto o jogador recebeu 9 sequencias de animação que foram tiradas do jogo “The Legend Of Zelda: The Minish Cap” essas foram as animações de: parado para baixo, parado para cima, parado para os lados, andando para baixo, andando para cima, andando para os lados, atacando para baixo, atacando para cima e atacando para os lados. Existem diversas formas de reproduzir as animações da Unity, porem a mais simples que é a que fora utilizada neste projeto é usar o comando “Animation.Play(“nome da animação”);” quando este comando for chamado a animação que foi selecionada é ativada instantaneamente.

5. Conclusão

De alguma forma o desenvolvimento de jogos é um mercado que está em ascensão durante as últimas décadas e que desperta interesse em boa parte dos jovens em seguir trabalhando na área, contudo desenvolver jogos pode se mostrar uma tarefa difícil então este projeto teve como intuito mostrar uma das formas mais simples de desenvolver os jogos que é através de uma Game Engine.

A Unity provou-se ser uma ferramenta de fácil utilidade por ter uma grande comunidade ativa com diversos fóruns e sites que auxiliam ao uso de suas ferramentas, todavia existem funções simples que na Unity acabam por se complicar tornando mais complicado de um leigo dominar a plataforma.

Conclui-se que a Unity é uma boa plataforma para iniciantes, contudo existem defeitos que não podem ser ignorados, sendo eles corrigidos com linhas desnecessárias, ou sendo preciso consultar fóruns na internet para resolver os seus problemas, ainda assim a Unity pode ser considerada uma das formas mais simples de desenvolver jogos não tendo conhecimento na área.

Referencial Bibliográfico

Moreira, Fábio Eduardo. **DESENVOLVIMENTO DE JOGOS ELETRÔNICOS MULTIUSUÁRIO PARA DISPOSITIVOS MÓVEIS**. 2011. Monografia para obtenção de título em Ciências da computação – Faculdade Farias Brito, Fortaleza, 2011. Disponível em: <https://fbuni.edu.br/sites/default/files/tcc-20102-fabio-eduardo-moreira.pdf>. Acesso em: 11/01/2022

Ferreira, Luís Henrique. **DESENVOLVIMENTO DE JOGOS ELETRÔNICOS UTILIZANDO A TECNOLOGIA UNITY**. 2015. Trabalho de conclusão de curso em Sistemas de Informação – Centro universitário de Araraquara, Araraquara, 2015.

A HISTÓRIA DE TUDO.; **História do Videogame**. Brasil, 2020. Disponível em: <https://www.historiadetudo.com/videogame>. Acesso em 24/11/2021.

DIGITAL, COLABORATIVO & INDEPENDENCE.; **História do videogame: relembre os consoles que marcaram época**. Agosto de 2020. Disponível em: [https://www.dci.com.br/tecnologia-e-games/historia-do-Videogame/4270/#:~:text=História%20do%20videogame%3A%20Magnavox%20Odyssey&text=Em%201972%20nasceu%20o%20primeiro,Brown%20Box%20\(Caixa%20Marrom\)](https://www.dci.com.br/tecnologia-e-games/historia-do-Videogame/4270/#:~:text=História%20do%20videogame%3A%20Magnavox%20Odyssey&text=Em%201972%20nasceu%20o%20primeiro,Brown%20Box%20(Caixa%20Marrom)). Acesso em 24/11/2021.

N.A.V.E.; **Quantos Gêneros de games você joga?** Julho de 2020. Disponível em: <https://blog.navegamer.com.br/quantos-generos-de-games-voce-joga>. Acesso em 11/01/2022.

Cartão perfurado. In: WIKIPEDIA: a enciclopédia livre. Outubro de 2020. Disponível em: https://pt.wikipedia.org/wiki/Cartão_perfurado. Acesso em 24/11/2022.

SHOWMETECH.; **Tecnologias que revolucionaram os games**. Dezembro de 2021. Disponível em: <https://www.showmetech.com.br/tecnologias-que-revolucionaram-os-games/>. Acesso em 11/01/2022.

TECMUNDO.; **O que é engine ou motor gráfico?** Março de 2011. Disponível em: <https://www.tecmundo.com.br/video-game-e-jogos/9263-o-que-e-engine-ou-motor-grafico-.htm>. Acesso em 18/04/2022.

CRIE SEUS JOGOS.; **como surgiu a Unity Engine?** Janeiro de 2022. Disponível em: <https://www.crieseusjogos.com.br/como-surgiu-a-unity-engine/>. Acesso em 20/04/2022.

MASTER.D.; **O que é o Unity e para que serve?** Julho de 2022. Disponível em: <https://www.masterd.pt/blog/o-que-e-o-unity-e-para-que-serve>. Acesso em 24/07/2022.

TREINAMENTO24.; **Como funciona a Unity?** [Entre 2020 e 2022]. Disponível em: <https://treinamento24.com/library/lecture/read/763480-como-funciona-a-unity>. Acesso em 02/08/2022.

BLOG IMPULSO.; **Unity: uma poderosa ferramenta para desenvolvimento de jogos.** Janeiro de 2022. Disponível em: <https://blog.impulso.network/unity-uma-poderosa-ferramenta-para-desenvolvimento-de-jogos-mfbp/>. Acesso em 02/08/2022

UNITY DOCUMENTATION.; **WaitForSeconds.** Setembro de 2022. Disponível em: <https://docs.unity3d.com/ScriptReference/WaitForSeconds.html>. Acesso em: 18/09/2022.