

FACULDADE INDEPENDENTE DO NORDESTE

ENGENHARIA DA COMPUTAÇÃO

DANIEL MOREIRA MACARIO SOUZA

O DESENVOLVIMENTO DE UM JOGO 2D

VITORIA DA CONQUISTA - BA

2020

DANIEL MOREIRA MACARIO SOUZA

O DESENVOLVIMENTO DE UM JOGO 2D

Trabalho Científico de Formatura apresentado junto ao curso de Engenharia de Computação da Faculdade Independente do Nordeste como requisito parcial e obrigatório a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. M.Sc. Marcelo Barbosa de Almeida.

VITÓRIA DA CONQUISTA – BA

2020

DANIEL MOREIRA MACÁRIO SOUZA

O DESENVOLVIMENTO DE UM JOGO 2D

Esta monografia foi julgada adequada à obtenção do título de bacharel em Engenharia da Computação e aprovada em sua forma final pelo curso de Engenharia da Computação, da Faculdade Independente do Nordeste.

Aprovado em: _____ de _____ de 2020

BANCA EXAMINADORA / COMISSÃO AVALIADORA

Faculdade Independente do Nordeste (FAINOR)

Faculdade Independente do Nordeste (FAINOR)

Prof. M.Sc. Marcelo Barbosa de Almeida – Orientador
Faculdade Independente do Nordeste (FAINOR)

VITÓRIA DA CONQUISTA – BA

20120

Dedico este trabalho aos meus pais: Cecilia e Alencar, aos meus irmãos Gabriel e Juliana, aos meus Tios e Tias e aos meus amigos.

AGRADECIMENTOS

Primeiramente agradeço a Deus, por ter me concedido o dom da vida e por sempre me guardar aonde quer vá.

Aos meus amados pais, Cecilia e Alencar, por toda dedicação, amor, compreensão e apoio ao longo da minha vida, sempre me motivando e incentivando a buscar pelos meus sonhos e sendo sempre meus principais alicerces.

Aos meus irmãos, Gabriel e Juliana, por ser verdadeiros irmãos e amigos e que apesar de todas as brigas de irmãos, sempre me prestigiam com momentos únicos e felizes.

Aos meus avós Valda e Aldemisso, por todo o amor e sempre servirem de inspiração de pessoa a ser.

Aos meus tios e tias, Acassio, Aldeir, Almerindo, Anuzia, Jerry, Marcia, Marcos e Viviane, por sempre buscarem o melhor pra mim e me tratarem como um filho.

Aos meus amigos e irmãos, Henrique, Carol e Carla, que dividiram apartamento comigo e sem dividas são minha família de verdade, sempre ao meu lado me “zoando” a todo momento.

Aos meus colegas e amigos de faculdade, Nainde, Marcelo, Paulo Henrique, Bianca Peçanha, por serem o melhores amigos que poderia ter feito na faculdade, nunca me deixando desanimar e muitas vezes me forçando(Nainde e Bia) a ir a faculdade.

E a todos os professores que me auxiliaram nessa jornada desde meus anos iniciais e em especial ao meu orientador, Prof. M.Sc. Marcelo Barbosa de Almeida, pelo auxílio e conhecimento para o desenvolvimento deste trabalho.

LISTA DE QUADROS

Quadro 1 – Requisitos não Funcionais do software.	32
Quadro 2 – Caso de uso detalhado.....	34
Quadro 3 – Programas utilizados no desenvolvimento o jogo.....	36

LISTA DE FIGURAS

Figura 1 – Perspectiva para de mercado.....	18
Figura 2 - Funcionamento do IF/else.....	20
Figura 3 – Assets na unity	21
Figura 4 - Script.	22
Figura 5 - Diagramas Estruturais.	26
Figura 6 - Diagramas Comportamentais.	26
Figura 7- Diagramas de Interação.	27
Figura 8 – Diagrama de caso de uso.	33
Figura 9 – Diagrama de Classes.....	35
Figura 10 – Diagrama de Transição de Estados.	36
Figura 11 – Importando assets.....	38
Figura 12 – criando o tile map.	39
Figura 13 – Colocando blocos na cena.	40
Figura 14 – Inserindo o tilemap Collider 2D.	41
Figura 15 - Criação de personagem.....	41
Figura 16 – Criação de personagem.	42
Figura 17 – Orange.	42
Figura 18 – Rock.....	43
Figura 19 – Fantasma.	43
Figura 20 – textos em tela.	44
Figura 21 – Botão de Menu.....	44
Figura 22 – Animação do player.....	45
Figura 23 – Prefabs.....	46
Figura 24 – Criando script.	47
Figura 25 - Script criado.	48
Figura 26 – Script player 1.	49
Figura 27 – Movimentação do player.	50
Figura 28 – Colisão do player com objetos.	51
Figura 29 – Script do GameController.....	52
Figura 30 – Script 1 Rock.....	53
Figura 31 – Colisão do “Rock”.....	53
Figura 32 – Tutorial 1.	54
Figura 33 – Tutorial 2	55
Figura 34 – Fase 2.	55
Figura 35 – Fase 2 Incompleta.....	56
Figura 36 – Build do jogo.....	57
Figura 37 – Executável do jogo.	58
Figura 38– Menu.	58
Figura 39 – Apresentação da Historia.	59
Figura 40 – Tutorial 1: Como funciona o C++.....	60
Figura 41 – Exemplo de código a ser seguido.	60

Figura 42 – Dialogo com a IA.	61
Figura 43- Fase 1.	61
Figura 44 – Fase 1 Completa.	62
Figura 45 - Exemplos de inimigos e objetos.	62
Figura 46– Tela de Game Over.	63
Figura 47- Botões de menu e resetar.	63
Figura 48 - Método Jump do Personagem.	68
Figura 49 – Script da Orange.	68
Figura 50 – Script CheckPoint.	69
Figura 51 – Script Fantasma.	69
Figura 52 – Script da pedra espinhosa.	70
Figura 53 - Tutorial 5.	70
Figura 54 - Tutorial 6.	71
Figura 55 - Tutorial 7.	71
Figura 56 - Tutorial 8.	72
Figura 57 - Tutorial 9.	72
Figura 58 - Tutorial 10.	73
Figura 59 - Fase 3.	73
Figura 60 - Fase 4.	74
Figura 61 - Fase 5.	74
Figura 62 - Fase 6.	75
Figura 63 – Fase 7.	75

RESUMO

O mundo está em constante evolução a todo momento, e as tecnologias avançaram de tamanha forma que hoje é comum as pessoas terem o “mundo” na palma de suas mãos, um exemplo disso são os celulares, e não tem como olhar para o futuro sem imaginar nos avanços e formas de usar a internet e os dispositivos computacionais, e isso interfere também nas profissões do “futuro” que já estão surgindo e crescendo e a tendência é crescer ainda mais e estão diretamente ligadas à programação. Dentro desta prerrogativa o trabalho de conclusão de curso a criação de um jogo 2D para computador com o intuito de mostrar as linguagens de programação de uma forma diferente associada a jogabilidade dos jogos. Além disso o mercado de trabalho tem buscado cada vez mais profissionais na área, por isso é muito importante atrair novos jovens interessados. A partir disso foi desenvolvido dentro da plataforma unity que é um game engine, juntamente com uma programação em C# através do editor de código visual code studio, um jogo em 2D que mescla diversão, com conceitos de lógica e linguagens de programação, a fim de estimular o interesse nesta área. Existem diversos aplicativos hoje com esse mesmo intuito nas diferentes áreas de ensino e a tendência é que com todos os avanços aumente cada vez mais.

Palavras-Chave: Jogo 2D. Programação. Unity.

ABSTRACT

The world is constantly evolving at all times, and technologies have advanced in such a way that today it is common for people to have the “world” in the palm of their hands, an example of which are cell phones, and there is no way to look to the future without imagine the advances and ways of using the internet and computing devices, and this also interferes with the professions of the “future” that are already emerging and growing and the tendency is to grow even more and are directly linked to programming. Within this prerogative the work of completion of course the creation of a 2D game for computer in order to show the programming languages in a different way associated with the gameplay of the games. In addition, the job market has sought more and more professionals in the area, so it is very important to attract new interested young people. Based on that, it was developed within the unity platform, which is a game engine, together with C # programming through the visual code editor, a 2D game that mixes fun, with logic concepts and programming languages, in order to stimulate interest in this area. There are several applications today with the same purpose in different areas of teaching and the trend is that with all advances it will increase more and more.

Keywords: 2D game. Programming. Unity.

Sumario

1	INTRODUÇÃO	13
1.1	CONTEXTUALIZAÇÃO	14
1.2	PROBLEMA DE PESQUISA	14
1.3	HIPÓTESE	14
1.4	OBJETIVO	14
1.5	OBJETIVOS ESPECÍFICOS	14
1.5	JUSTIFICATIVA	15
2	ESTADO DA ARTE	16
3	REFERENCIAL TEORICO	17
3.1	MERCADO DE TRABALHO	17
3.2	MODERNIZAÇÃO DA SALA DE AULA	18
3.3	O ENSINO DE LÓGICA DE PROGRAMAÇÃO	19
3.4	UNITY	20
3.5	ASSETS	21
3.6	SCRIPTING	21
3.7	VISUAL STUDIO CODE	22
3.8	LINGUAGEM DE PROGRAMAÇÃO	23
3.8.1	LINGUAGEM DE PROGRAMAÇÃO C++	23
3.8.2	LINGUAGEM DE PROGRAMAÇÃO C#	23
3.9	ENGENHARIA DE SOFTWARE	24
3.10	UML : LINGUAGEM UNIFICADA DE MODELAGEM	24
3.10.1	DIAGRAMAS NA UML	25
4	METODOLOGIA	27
4.1	TIPO DE METODOLOGIA QUANTO AOS OBJETIVOS	28
4.2	TIPO DE PESQUISA QUANTO À ABORDAGEM	28
4.3	TIPOS DE PESQUISA QUANTO AOS PROCEDIMENTOS	28
4.4	ETAPAS DA PESQUISA	29
4.4.1	FLUXOGRAMA METODOLOGICO	30
4.5	INSTRUMENTOS DE PESQUISA	30
5	DESENVOLVIMENTO	31
5.1	PROJETO DE ENGENHARIA DE REQUISITOS	31
5.1.1	DIAGRAMA DE CASO DE USO	32
5.1.2	DIAGRAMA DE CASO DE USO DO FRONT END DETALHADO	33
5.1.3	DIAGRAMA DE CLASSE	34
5.1.4	DIAGRAMA DE TRANSIÇÃO DE ESTADOS	35
5.2	IMPLEMENTAÇÃO DO APLICATIVO	36
5.2.3	CONTEXTUALIZAÇÃO	37
5.2.3	OBJETIVO	37
5.2.4	CENARIOS	37

5.2.5 ANIMAÇÕES.....	44
5.2.6 PREFABS.....	45
5.2.7 SCRIPTS.....	46
5.2.7.1 PLAYER.....	48
5.2.7.2 GAMERCONTROLLER	51
5.2.7.3 INIMIGOS	52
5.2.8 LINGUAGENS.....	54
5.2.9 BUILD.....	56
6 RESULTADOS E SIMULAÇÃO.....	58
7 CONSIDERAÇÕES FINAIS	63
7.1 SUGESTÃO PARA TRABALHOS FUTUROS	64
8 REFERENCIA	65
9 ANEXOS:.....	68

1 INTRODUÇÃO

O mundo está em constante evolução a todo momento, e as tecnologias avançaram de tamanha forma que hoje é comum as pessoas terem o “mundo” na palma de suas mãos, um exemplo disso são os celulares, mas para isso ser possível é necessário vários fatores e um dos principais é a programação que já está tão presente na nossa realidade que quase não a percebemos, mas ela está em televisões, computadores, celulares, carros, máquinas industriais, relógios, etc, e como esses constantes avanços a necessidade na área de programação tende a crescer mais e mais.

“Levantamentos de 2015 do Institute for the Future nos dá uma dica: o trabalho desempenhado pelas pessoas serão endossados pelas rotinas computadorizadas, logo, o trabalho desempenhado pelo trabalhador precisa ser mais agregador, e menos rotineiro ou automatizado.”(Santos,2018).

E não tem como olhar para o futuro sem imaginar nos avanços e formas de usar a internet e os dispositivos computacionais, e isso interfere também nas profissões do “futuro” que já estão surgindo e crescendo e a tendência é crescer ainda mais e estão diretamente ligadas a programação, como por exemplo temos: desenvolvedor de software, analista de big data, especialista em segurança da informação, que são bons exemplos de como as coisas estão mudando.

Desta forma fica evidente a necessidade de novos profissionais, porem o grau de dificuldade nesta área é bastante elevado, principalmente se for levar em consideração o lógica de programação que é um requisito básico. Segundo Pereira (2013), a Lógica de Programação deveria andar junto com outras disciplinas do ensino básico, tais como Biologia, Química e Física. Neste contexto, o ensino de programação para crianças poderia desenvolver o pensamento computacional e passos lógicos para a resolução automatizada de problemas.

1.1 CONTEXTUALIZAÇÃO

Muitas vezes o ensino das linguagens de programação é visto como chato e maçante o que acarreta no desinteresse nesta área, então desenvolver novas formas de apresentar as linguagens é algo que pode diminuir essa mística criada. Analisando o contexto será desenvolvido um jogo 2D, que busca mostrar as estruturas básicas da linguagem de programação C++ e seus funcionamentos junto a jogabilidade do jogo.

1.2 PROBLEMA DE PESQUISA

É inegável dizer que as crianças e jovens são o “futuro”, e em um futuro que vai estar bastante atrelado às tecnologias computacionais e suas programações, mas como desenvolver um jogo 2d com a finalidade de apresentar conceitos básicos da linguagem de programação C++?

1.3 HIPÓTESE

A proposta é a criação de um jogo, que possibilite ao jogador ter um contato, “informal” as linguagens de programação, através de breves tutoriais sobre seus funcionamentos e exemplos interativos de códigos prontos.

1.4 OBJETIVO

Desenvolver um jogo 2d que apresente conceitos básicos no âmbito da linguagem de programação C++

1.5 OBJETIVOS ESPECÍFICOS

- Modelar o sistema através dos métodos de Engenharia de Software;
- Definir qual Game Engine utilizar;

- Escolher a linguagem de programação a ser apresentada;
- Implementar as funcionalidades do jogo 2d, cenas, personagens, obstáculos e objetivos da fase;
- Incorporar ao jogo 2d Tutoriais sobre a linguagem de programação escolhida como objeto da presente investigação;
- Testar o jogo a partir do executável gerado;

1.5 JUSTIFICATIVA

É notório que as tecnologias computacionais estão avançando cada vez mais rápido e a demanda por profissionais nesta área cresce por consequência, mesmo com a pandemia a área de TI continuou crescendo, de acordo com uma pesquisa realizada pela Revelo, que é um startup de recrutamento, a área de tecnologia teve um crescimento no período de março a abril de 25% comparado a 2019.

Dentro dessa perspectiva é importante que cada vez mais pessoas entrem na área de programação, ainda tem o fato de alguns estudos mostrarem um taxa de evasão de materiais de programação nos primeiros anos de faculdade, segundo GIRAFFA e MORA (2018), “No que tange a evasão associada aos cursos de Computação, estudos realizados por diversos pesquisadores da área de Educação e Computação e do Ministério da Educação brasileiro(MEC) mostram que os alunos desistem dos cursos logo no primeiro ano do ensino superior. As disciplinas causadoras desta desistência são aquelas associadas ao ensino de Cálculo e de Programação (incluindo-se a disciplina de Algoritmos)”, principalmente porque vários alunos chegam ao ensino superior com déficit em na formação básica e assim um contato mais cedo e de forma mais descontraída pode desmistificar e despertar o interesse na área.

2 ESTADO DA ARTE

Desenvolvimento de Jogos na Plataforma Unity

Esse trabalho visa uma análise do cenário atual do mundo dos jogos, descrevendo a importância dos jogos nos dias de hoje, e, um estudo aprofundado da plataforma Unity, descrevendo como a mesma auxilia na criação dos games, mostrando recursos e funcionalidades. Projeto apresenta o desenvolvimento de um jogo para demonstrar os recursos da plataforma unity para criação de jogos. Através dessa plataforma, é possível reduzir a complexidade no desenvolvimento de jogos. A unity oferece recursos, bibliotecas e componentes que agilizam o processo de desenvolvimento. O projeto faz o uso de algumas ferramentas como o photoshop e o visual Studio, além da própria unity. O autor faz uma descrição da estrutura para criação de um jogo, e mostra uma parte do desenvolvimento. Concluindo por fim que o mercado de jogos se encontra em expansão devido à grande demanda por jogos de vários gêneros. Isso exige pessoas capacitadas e tecnologias que facilitam o desenvolvimento, pois a construção de jogos é um trabalho árduo e complexo que demanda também da criatividade da equipe de desenvolvimento.

Dinâmicas com App Inventor no Apoio ao Aprendizado e no Ensino de Programação

O artigo “Dinâmicas com App Inventor no Apoio ao Aprendizado e no Ensino de Programação”, por Ribeiro, Manso e Borges (2016), Discute maneiras de incentivar o uso do App Inventor como uma ferramenta para ensinar programação e apoiar a aprendizagem escolar, e o avalia em muitos aspectos. Um método de utilização da ferramenta é proposto e os alunos são apresentados à primeira etapa da programação. App Inventor é uma ferramenta de programação baseada em blocos que permite que usuários, mesmo que iniciantes em programação ou usuários comuns, possam desenvolver aplicativos totalmente funcionais para a plataforma de dispositivos móveis Android [MIT App Inventor, 2016]. O objetivo deste projeto foi propor uma dinâmica de aplicação para o App Inventor e analisar o impacto da integração da lógica de programação no suporte à aprendizagem. A dinâmica de desenvolvimento de aplicativos móveis foi realizada com base no App Inventor, com uma turma de alunos do ensino médio de quatorze a dezessete idade, em uma

classe de 40 alunos. Ao final da dinâmica, os jovens devem responder, de forma anônima um questionário, para que se analise os resultados obtidos. As dinâmicas foram realizadas e um questionário respondido pelos participantes. Com os dados dos questionários, pode-se concluir que o App Inventor é uma plataforma que oferece oportunidades de uso em processos de aprendizagem, dando o mesmo tempo para os participantes, uma nova visão de programação e uma carreira na área de Informática.

3 REFERENCIAL TEORICO

3.1 MERCADO DE TRABALHO

A modernização tem influenciado em tudo ao longo dos anos e o mercado de trabalho não foge a essa regra. Se formos olhar a 10 anos atrás, muitas das profissões que estão em alta hoje nem mesmo existiam, como por exemplo marketing digital, cientista de dados, desenvolvedor Salesforce, entre outras.

“Os novos empregos que estão sendo criados exigem e demandam criatividade, habilidades, analíticas, matemáticas e digitais, robótica, programação. Vão se destacar os profissionais que entreguem o que as máquinas não conseguem fazer análises e questionamentos. Vão desaparecer as funções de buscar e organizar dados. Hoje, alguns call centers são atendidos por robôs e bancos totalmente digitais.”(MUSSE, 2018).

Então o digital não é uma ameaça, mas sim um transformador, e nesse processo alguns empregos serão destruídos enquanto outros são criados. E a tendência é que todas as áreas sofram essas mudanças, principalmente pela robotização e sistemas de inteligência artificial. E neste cenário os profissionais que iram se destacar, são aqueles que desempenha papéis que as máquinas não conseguiram, como análise e questionar.

Figura 1 – Perspectiva para de mercado.



Fonte: Fórum Econômico Mundial 2018.

3.2 MODERNIZAÇÃO DA SALA DE AULA

Obvio que com todos esses avanços tecnológicos, vai exigir uma qualificação adequado. Isso começa na sala de aula, que vem passando por incontáveis transformações na sala de aula.

Antigamente a sala de aula se resumia, a um professor na frente da lousa com um giz e alunos com seus livros, hoje em dia as coisas mudaram como a utilização dos data show, computadores, celulares, o EAD,etc.

Segundo SOUZA E FRANÇA (2016), Para que isto ocorra, os professores dos cursos de tecnologia da informação estão recorrendo a estratégias de ensino para facilitar a absorção e fixação do conteúdo passado na sala de aula. Silva et al. (2011) comentam que há uma carência destas práticas educacionais alternativas às práticas tradicionais.

O ensino a distancia (EAD) é um ótimo exemplo dessa modernização, já que ela proporciona assistir aulas a partir de um computador ou celular com conexão

com internet, segundo o Google, a sua plataforma de vídeo conferencia no período de abril teve um crescimento de cerca 3 milhões de novos usuários por dia.

Devido principalmente ao surto do COVID-19, mas é um ótimo exemplo de como a educação e as novas tecnologias podem andar lado a lado se ajudando. Então a tecnologia pode e deve ser vista como uma extensão do espaço físico da sala de aula, já que nela possui infinitas possibilidades de se trabalhar os mais variados conteúdos desde aplicativos de pesquisa até jogos.

3.3 O ENSINO DE LÓGICA DE PROGRAMAÇÃO

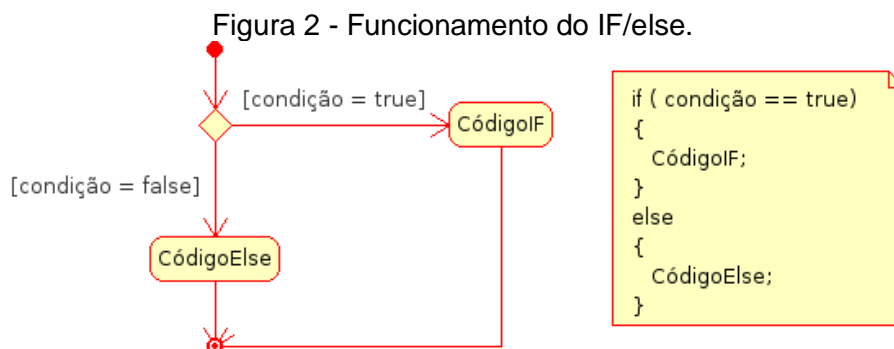
A lógica de programação apesar de sua simplicidade com o processo envolve o uso de dispositivos lógicos como as estruturas condicionais(if/else) e de repetição(for/while), ela é a base de toda e qualquer linguagem de programação que se venha a se aprender, “O que ocorre, na maioria das vezes, é que alunos acabam desistindo do curso de Computação devido às dificuldades encontradas no aprendizado 138 da Lógica de Programação, fazendo com que os mesmos sejam reprovados, diminuindo sua autoestima, gerando uma aversão diante do conteúdo ensinado.”(Garlet et. al., 2018).

“Atualmente, tem-se reconhecido a necessidade de introduzir conceitos computacionais desde as séries iniciais, focando principalmente no ensino da Lógica de Programação, já que a mesma pode auxiliar no desenvolvimento do poder cognitivo das crianças” (ARAÚJO et. al., 2015).

A Sociedade Brasileira de Computação (SBC) entende que é fundamental e estratégico para o Brasil que conteúdos de Computação sejam ministrados na Educação Básica.(SBC, 2019). Porém o trato no ensino da mesma, não deve ser feito da mesma forma que é feita no ensino superior.

Muitos autores mostram que o uso de ferramentas lúdicas tendem a atrair maior atenção por parte destes alunos, pois traz alguns pontos como desafios e recompensas, torna aprendizagem mais prazerosa, facilitando a aprendizagem e o desenvolvimento de suas habilidades. Segundo Romio (2017), jogos educacionais tendem despertar um interesse maior do aluno na sala de aula, já que traz novos

desafios, possibilidades de se aprender errando, busca por mais informações, juntando estímulo e diversão.



Fonte: Igorknop (2009)

3.4UNITY

A Unity é uma plataforma de desenvolvimento de jogos criada pela Unity Technologies, e é denominada como uma Game Engine, ou, motor de jogos. Esta plataforma é utilizada na criação de vários estilos de jogos tanto 2D, quanto 3D. A Unity tem uma loja onde está disponível os mais variados recursos gráficos, como personagens, ambientes, plataformas, animações, etc, disponíveis de forma gratuita ou paga, assim os desenvolvedores podem focar principalmente no funcionamento dos jogos.

A Unity é uma das principais engines de jogos, além de estar presente nas mais diversas plataformas como desktop, navegadores na Internet, iPhone, Android, etc. Alguns exemplos de títulos produzidos nessa engine como: Cities: Skylines, Pokémon GO, Hearthstone: Heroes of Warcraft, Garena Free Fire e vários outros.

A Unity, também, é uma das mais populares engines para jogos, possuindo 770 milhões de jogadores que jogam os títulos criados pela plataforma. A Unity também está presente nas plataformas móveis, atualmente com 1.7 bilhões de dispositivos executando games feitos na Unity. (Silva et al., 2016)

Como suporte, a empresa oferece um ótimo ambiente para quem quer trabalhar com ela, disponibilizando, os mais variados documentações e tutoriais, desde de iniciante aos mais avançados.

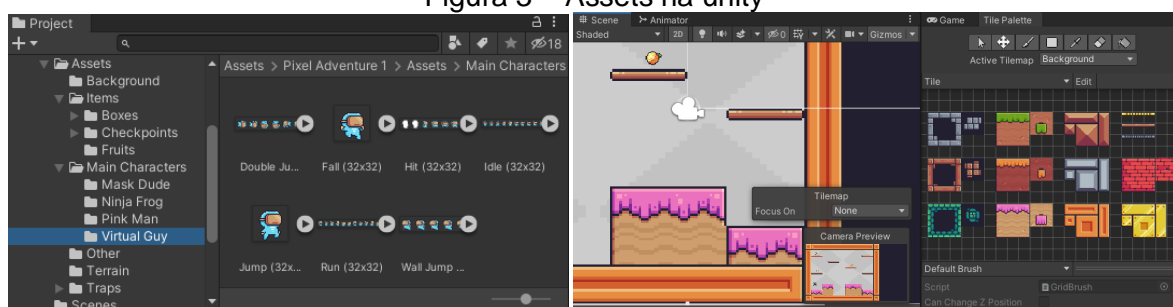
3.5 ASSETS

O desenvolvimento de um jogo deve-se em grande parte a utilização e manuseio dos assets, tradução literal do inglês “ativos”, é nada mais é tudo que compõe o jogo desde textura, objetos, cenários, sons, física, etc. Os assets são criados e editados fora da unity em ferramentas externas especializada para cada componente que posteriormente é importada para o editor de cenas.

“A Unity possui uma forma muito simples e robusta de importação de Assets para dentro de um projeto, bastando que os mesmos sejam “arrastados” para dentro de uma pasta da janela Project.”(PASSOS,2009).

Após a importação dos assets, os recursos ficam disponíveis para serem implementados a cena de forma bastante simples e atrelada a visualização em tempo real que a unity tem, facilita muito o desenvolvimento, ganhado produtividade.

Figura 3 – Assets na unity



Fonte: autor próprio, 2020.

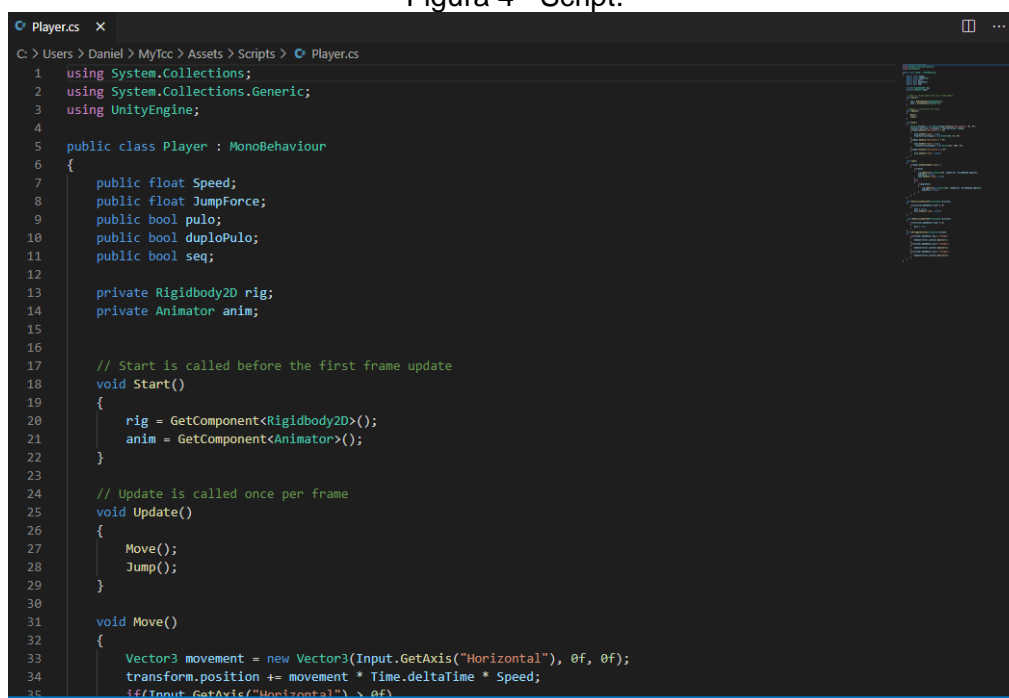
3.6 SCRIPTING

O script é uma parte essencial de todas as aplicações que o desenvolvedor cria na unity, a maioria dessas aplicações precisam do script para responder a entradas do jogador, comandos de movimentação, e para eventos que devem acontecer, tela de “game over”.

Os scripts podem ser usados para manipular quase tudo na cena, desde efeitos gráficos, efeitos sonoros, comportamento de objetos, sistemas de IA.

Na unity os scripts são executados através de uma versão modificada do Mono, que é uma implementação de código aberto para o sistema .NET, onde a mesma permite a implementação nas linguagens javascript, C#, sem que haja qualquer penalidade em qualquer uma das escolhas, e podendo ate usar as mesma no mesmo jogo. Após criados esses scripts eles são anexados a um componente do jogo, game objects.

Figura 4 - Script.



```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Player : MonoBehaviour
6  {
7      public float Speed;
8      public float JumpForce;
9      public bool pulo;
10     public bool duploPulo;
11     public bool seq;
12
13     private Rigidbody2D rig;
14     private Animator anim;
15
16     // Start is called before the first frame update
17     void Start()
18     {
19         rig = GetComponent<Rigidbody2D>();
20         anim = GetComponent<Animator>();
21     }
22
23     // Update is called once per frame
24     void Update()
25     {
26         Move();
27         Jump();
28     }
29
30     void Move()
31     {
32         Vector3 movement = new Vector3(Input.GetAxis("Horizontal"), 0f, 0f);
33         transform.position += movement * Time.deltaTime * Speed;
34         if (Input.GetAxis("Horizontal") > 0f)

```

Fonte: autor próprio, 2020.

3.7 VISUAL STUDIO CODE

O Visual Studio Code ou VSCode, é um editor de código criado pela Microsoft para Windows, Mac e Linux.

Ele atende a uma quantidade enorme de projetos (ASP .NET, Node.js) e oferece suporte para mais de 30 linguagens de programação, como JavaScript, C#, C++, PHP, Java, HTML, R, CSS, SQL, Markdown, TypeScript, LESS, SASS, JSON, XML e Python. Além de gratuito e open source, com seu código disponibilizado no GitHub, e isso permite que você contribua com seu desenvolvimento.

3.8 LINGUAGEM DE PROGRAMAÇÃO

A Programação de Computadores busca que apresenta diversas soluções em forma de códigos, para problemas computacionais, e a linguagem de programação é responsável por isso, que nada mais é que uma sequencia lógica de operações, implementada em códigos que podem ser compilados, através da manipulação de funções de entradas e saídas de dados.

As linguagens de programação possuem uma forma sintática e semântica, sendo respectivamente os padrões de escrita e o significado dos elementos da linguagem(MELO, SOARES, SILVA, 2003).

3.8.1 LINGUAGEM DE PROGRAMAÇÃO C++

A linguagem de programação C++ é uma linguagem de programação de computadores orientada de alto nível, ou seja, mais próxima do que nós seres humanos conseguimos entender, que foi originada a partir da Linguagem C, na década de 1970, na empresa Bell Labs. Em se tratando da linguagem C, a mesma foi ligeiramente modificada e sua adoção foi realizada como um padrão mundial ISO no início da década de 1980.(CASTRO,2018).

A linguagem C++ é uma das mais utilizadas, podendo ser encontrada nas mais variadas aplicações. Essa popularidade se deve, a apresentar varias vantagens quando comparadas com outras linguagens de alto nível, principalmente por ser bastante eficiente.

3.8.2 LINGUAGEM DE PROGRAMAÇÃO C#

O Visual C# (ou apenas C#) é uma linguagem de programação da Microsoft projetada para criar aplicações diversas, tanto para Windows, como para a Web, que são executadas no .NET Framework. É uma linguagem simples, moderna, segura quanto a tipos, orientada a objetos e familiar a programadores C, C++ e Java, pois

destas herda várias características. Embora herde características dessas linguagens, C# traz novos recursos e conceitos de programação, tais como indexadores, propriedades e delegates.(Saade,2011).

3.9 ENGENHARIA DE SOFTWARE

Ao se projetar um software, os desenvolvedores buscam realizar um planejamento para auxiliar os levantamentos das funcionalidades do programa, como será seu comportamento, e outras características gerais.

Segundo Ferreira et al.(2018), a Engenharia de Software reúne metodologias, métodos e ferramentas a ser utilizados, desde a compreensão do problema até o momento em que o software desenvolvido deixa de ser operacional, visando resolver problemas do processo de desenvolvimento e do produto de software.

O desenvolvimento de um software é um conjunto de varias etapas, e que para se alcançar bons resultados e dentro de um prazo estipulado, é necessário um bom planejam

3.10 UML : LINGUAGEM UNIFICADA DE MODELAGEM

UML é um sigla do inglês para Unified Modeling Language, que traduzido para o português fica Linguagem Unificada de Modelagem, Segundo OLIVEIRA (2020), a UML é uma metodologia muito utilizada para construir modelos que expliquem as características ou o comportamento de um sistema de software, contribuindo na identificação das características e funcionalidades que este deverá prover, bem como no planejamento de sua construção.

A UML é composto por um total de 13 diferentes diagramas, que se dividem em três tipos, sendo eles: Diagramas Estruturais e Diagramas Comportamentais, sendo que os comportamentais possuem uma subdivisão chamada de Diagramas de Interação.

De acordo com RESENDE(2019), Diagramas Estruturais: O objetivo é a descrição estática de um sistema, como classes, atributos e operações, além de relacionamento entre eles. Diagramas Comportamentais (Dinâmica): Explica o

funcionamento de parte de um sistema ou processo de negócio relacionado a tal explicação. Diagramas de Interação (Física): Diagrama que é considerado um subgrupo dos diagramas comportamentais, porque é normalmente utilizados na representação de interações entre objetos de uma aplicação.

3.10.1 DIAGRAMAS NA UML

Os treze diagramas da UML, são:

1. Diagrama de Classes;
2. Diagrama de Componentes;
3. Diagrama de Pacotes Detalha;
4. Diagrama de Objetos;
5. Diagrama de Estrutura Composta;
6. Diagrama de Perfil;
7. Diagrama de Casos de Uso;
8. Diagrama de Atividades;
9. Diagrama de Transição de Estados;
10. Diagrama de Sequência;
11. Diagrama de Interatividade;
12. Diagrama de Colaboração ou Comunicações;
13. Diagrama de Tempo;

Já quanto aos tipos que cada Diagrama pertence:

Figura 5 - Diagramas Estruturais.

Diagramas
Diagrama de Classes
Diagrama de Componentes
Diagrama de Pacotes
Diagrama de Objetos
Diagrama de Estrutura Composta
Diagrama de Instalação
Diagrama de Perfil

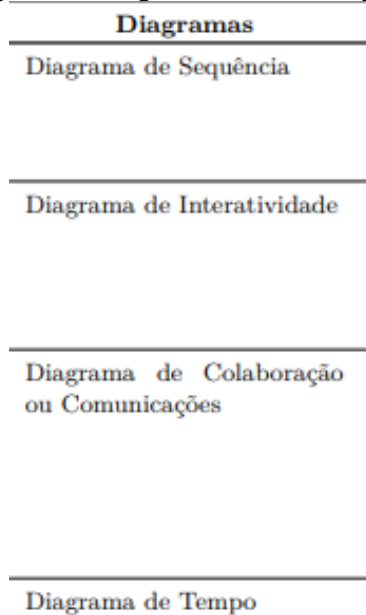
Fonte: Resende, 2019.

Figura 6 - Diagramas Comportamentais.

Diagramas
Diagrama de Casos de Uso
Diagrama de Atividades
Diagrama de Transição de Estados

Fonte: Resende, 2019.

Figura 7- Diagramas de Interação.



Fonte: Resende, 2019

4 METODOLOGIA

Este capítulo apresenta os variados métodos científicos que foram utilizados para estruturar e conduzir a pesquisa.

“...para gerar um tipo de conhecimento especial (científico) que, com certeza, pertence ao mundo das teorias, dos problemas e argumentos justificados. Assim, se constitui o conhecimento científico como: um conjunto 24 · de teorias, doutrinas (idéias, opiniões) formadas sobre determinados assuntos, ora ordenadas e sistematizadas em obras científicas (livros, monografias, dissertações, teses etc.). Tal conhecimento científico deriva das pesquisas, isto é, da resolução de problemas científicos..”(Pereira et al.,2018).

Será abordada seguir a natureza no que se remete a aproximar o autor e o leitor do tema, a qualificação e os procedimentos técnicos utilizados.

4.1 TIPO DE METODOLOGIA QUANTO AOS OBJETIVOS

Quanto ao objetivo este projeto possui uma natureza mista, sendo exploratória e descritiva, pois tem o intuito de apresentar e descrever todos os passos para o desenvolvimento do mesmo

“Este tipo de pesquisa tem como objetivo proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a construir hipóteses.”(GERHARDT et. al., 2019).

Segundo GERHARDT, a pesquisa descritiva é aquela que busca identificar os fatores que auxiliam ou determina o ocorrência de fenômenos. Pesquisas desse tipo podem ser classificadas como experimentais.

4.2 TIPO DE PESQUISA QUANTO À ABORDAGEM

A abordagem do trabalho , é um pesquisa qualitativa, pois visa uma compreensão da importância do uso de tecnologias para alunos e quais utilizar.

Para Minayo (2001), a pesquisa qualitativa trabalha com o universo de significados, motivos, aspirações, crenças, valores e atitudes, o que corresponde a um espaço mais profundo das relações, dos processos e dos fenômenos que não podem ser reduzidos à operacionalização de variáveis. Aplicada inicialmente em estudos de Antropologia e Sociologia, como contraponto à pesquisa quantitativa dominante, tem alargado seu campo de atuação a áreas como a Psicologia e a Educação.

A natureza do trabalho é aplicada, que objetiva gerar conhecimento para aplicações práticas.

4.3 TIPOS DE PESQUISA QUANTO AOS PROCEDIMENTOS

Quanto ao procedimento a pesquisa foi realizada a partir de pesquisa bibliográficas e estudos de caso.

A pesquisa bibliográfica é feita a partir do levantamento de referências teóricas já analisadas, e publicadas por meios escritos e eletrônicos, como livros, artigos científicos, páginas de web sites. Qualquer trabalho científico inicia-se com uma pesquisa bibliográfica, que permite ao pesquisador conhecer o que já se estudou sobre o assunto (GERHARDT et. al., 2019).

Os estudos de caso são pesquisas qualitativas usadas para fins acadêmicos e podem ser descritos na forma de artigos, monografias, dissertações de mestrado e teses de doutorado.

“Um estudo de caso é uma descrição e análise, a mais detalhada possível, de algum caso que apresente alguma particularidade que o torna especial. Sob o título EC se incluem muitos estudos que forma uma gama de variedades. (Pereira et al.,2018)

4.4 ETAPAS DA PESQUISA

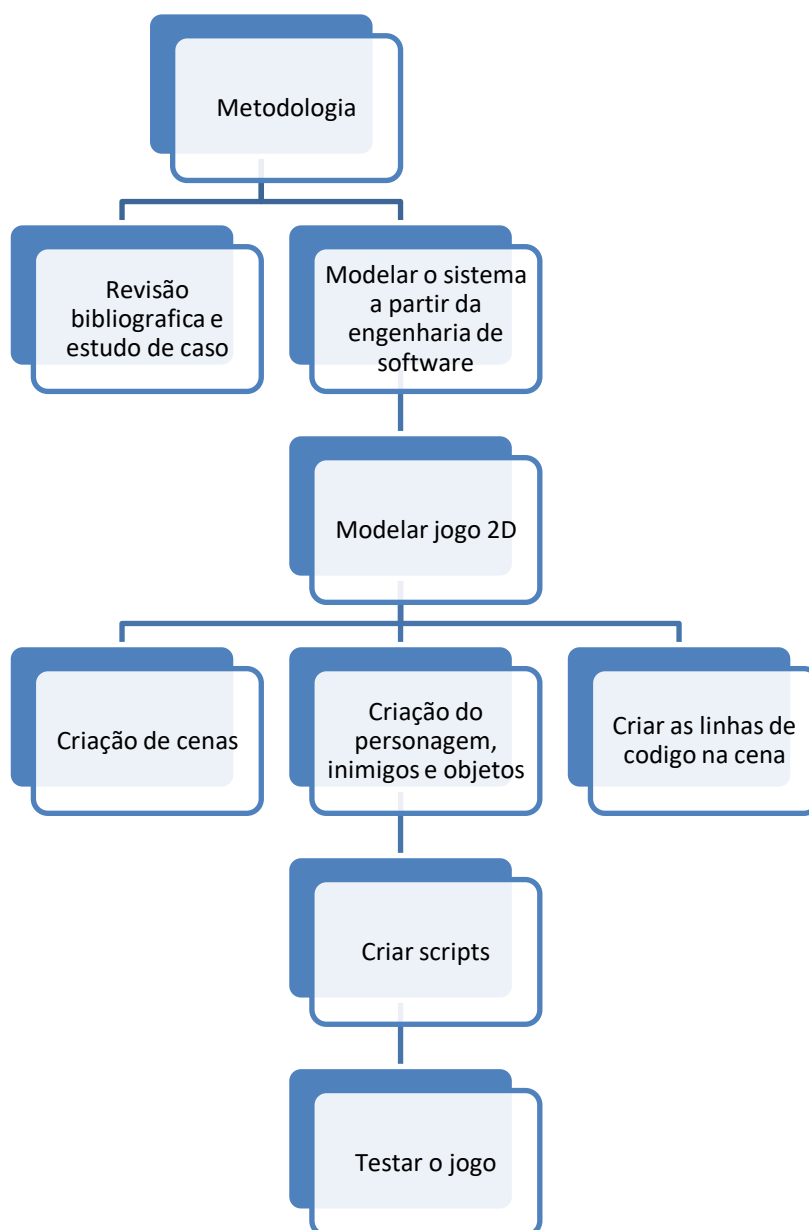
As etapas apresentadas a seguir descrevem como foi realizada a pesquisa.

A escolha do tema se deve ao crescimento da procura por profissionais na área de computação pelo mercado de trabalho, evidenciando a necessidade de cada vez mais pessoas interessadas na área, e utilizado uma alternativa mais lúdica, para apresentar por meio de um jogo as linguagens de programação para jovens.

O planejamento para a pesquisa bibliográfica e o estudo de caso foi entender na área de computação, a necessidade crescente do mercado de trabalho por programadores, e associar isso ao um aplicativo que apresente linguagens de programação de forma mais leve.

Para realizar o estudo e pesquisa foi através das metodologias da engenharia de software, e posteriormente implementada na Unity e Visual Studio Code, utilizando uma linguagem de programação C#.

4.4.1 FLUXOGRAMA METODOLOGICO



4.5 INSTRUMENTOS DE PESQUISA

Através da engenharia de software, usa-se uma metodologia específica, para a partir dela se desenvolva o aplicativo. Para fazer a implementação de um aplicativo é necessário o uso de uma linguagem tão quanto uma IDE, do inglês *Integrated Development Environment*, em português Ambiente de Desenvolvimento Integrado.

As ferramentas de pesquisa foram:

- Notebook Positivo – Intel I5 com sistema operacional Windows 10;
- Unity 3D - Motor de jogos utilizado na criação e edição de cenas e físicas dos personagens e objetos;
- Astah UML – Utilizado para a modelagens dos diagramas;
- Visual Studio Code - É ambiente de desenvolvimento integrado utilizado criação dos scripts;
- Loja de Assets da Unity - Utilizada para download de forma gratuita de todos os componentes visuais do jogo;
- Linguagens de programação utilizadas foram o: C# para confecção dos scripts e o C++ que foi utilizado como metodologia de ensino nas fases;

5 DESENVOLVIMENTO

Para o desenvolvimento do projeto em questão, foi necessário dividir em duas etapas, as cenas e os scripts.

As cenas, é onde todo o jogo será criado: implementação do personagens, monstros, objetos, botões, assim como suas físicas: colisores, gravidade.

Os scripts, é a parte onde será implementada todo o comportamento dos componentes inseridos na cena e suas interações.

A partir disso os diagramas de UML serão criados para facilitar o desenvolvimento, diagramas: de casos de uso, de classes e um de transição de estados do funcionamento do jogo para melhor detalhar as ações.

5.1 PROJETO DE ENGENHARIA DE REQUISITOS

Os requisitos de software são divididos em dois: requisitos funcionais(RF) e requisitos não funcionais(RFN).

Segundo Vinícius e Silva(2010),os requisitos funcionais listados representam as funcionalidades que devem ser suportadas pelo sistema. E os requisitos não funcionais especificam características de comportamento do sistema. Eles são de extrema importância para a obtenção de um sistema com um grau de qualidade satisfatório.

A modelagem do jogo será feita a partir da levantamento e análise de requisitos não funcionais(RFN). No Quadro 1 será apresentada todos os RFN e suas descrições.

Quadro 1 – Requisitos não Funcionais do software.

NOME DO RFN	DESCRIÇÃO
Iniciar	Iniciar o jogo.
Sair	Fechar o jogo.
Controle de personagem	Comandos para a movimentação do personagem na fase.
Next	Pular as cenas não jogáveis.
Resetar	Recomeçar a mesma fase.

Fonte: autoria própria, 2020.

Com todos os requisitos não funcionais apontados e justificados, a etapa a se seguir é a elaboração do diagramas, os diagramas a serem elaborados serão o de casos de uso e de classes, a fim de facilitar o desenvolvimento.

5.1.1 DIAGRAMA DE CASO DE USO

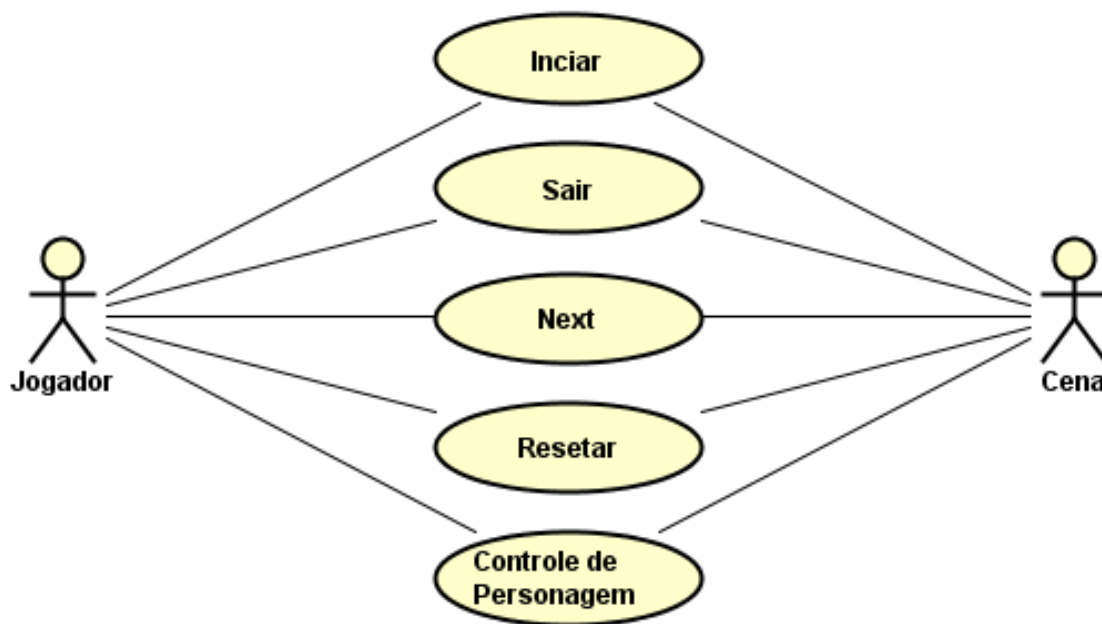
Segundo Resende(2019), o diagrama de casos de uso é voltado à apresentação de funcionalidades e características de um sistema, assim como de

que forma tais elementos se relacionam com usuários e entidades externas envolvidas num determinado processo.

Os atores são os usuários ou outros sistemas que podem executar os cenários, os cenários que definem os casos de uso são representados por elipses, e as linhas de comunicação são elementos que relacionam um ator a um caso de uso, um ator a um ator(generalização), um caso de uso para um caso de uso (*extend* ou *include*).

O diagrama de caso de uso é apresentado na figura 8, correspondente aos requisitos não funcionais.

Figura 8 – Diagrama de caso de uso.



Fonte: autoria própria, 2020.

5.1.2 DIAGRAMA DE CASO DE USO DO FRONT END DETALHADO

Os casos de uso detalhados têm como objetivo o funcionamento detalhado do sistema, apresentado no Quadro 2.

Quadro 2 – Caso de uso detalhado

Nome	Solicitar Cadastro
Descrição Sucinta	Funcionamento do jogo.
Atores	Jogador
Pré-Condições	Não possui
Fluxo Básico	<ol style="list-style-type: none"> 1. Jogador acessa o aplicativo; 2. Na tela de Menu o jogador ira escolher entre iniciar o jogo ou sair; 3. Caso escolha sair, o jogo é fechado; 4. Já se escolher iniciar, sera carregadas telas de tutoriais e falas, com a opção de Next; 5. Prosseguindo com o Next, o jogador seguira ate as fases do jogo, onde alem objetivos e obstáculos na cena, terá as opções de resetar e menu;
Fluxo Alternativo	<ol style="list-style-type: none"> 1. Alternativa ao passo 5: <ul style="list-style-type: none"> • No caso de o jogador morrer, uma tela de Game Over é carregada, com as opções de resetar e menu;

Fonte: autoria própria, 2020.

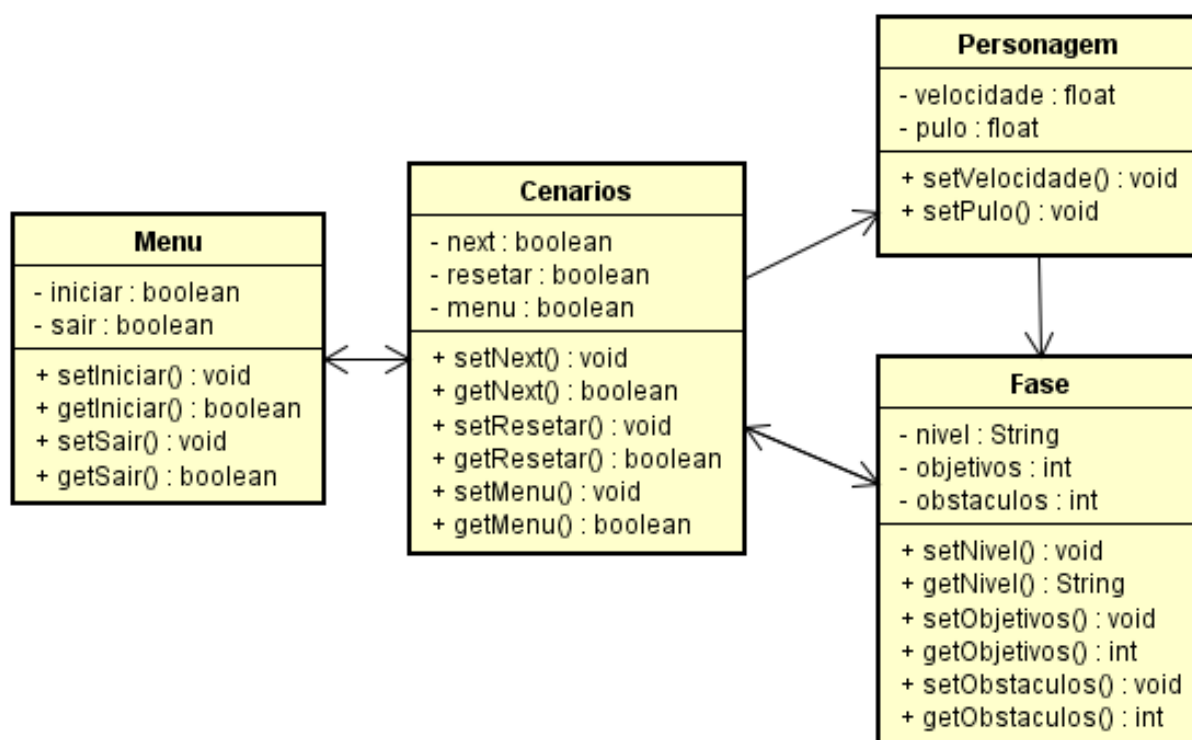
5.1.3 DIAGRAMA DE CLASSE

Segundo Oliveira(2020),o Diagrama de Classes é considerado um dos mais usados e importantes na Engenharia de Software. Através do diagrama de classes é possível visualizar como serão estruturadas as tabelas de um banco de dados e seus relacionamentos. Além disso, este permite visualizar de forma facilitada as

operações que devem ser executadas pelo sistema através das entidades, que apresentam atributos e as operações disponíveis.

Este diagrama define como será a estrutura do jogo. É Importante saber como serão interligados os elementos dessa estrutura, além de saber suas funcionalidades e características. A figura 9 apresenta o diagrama de classes do jogo.

Figura 9 – Diagrama de Classes.

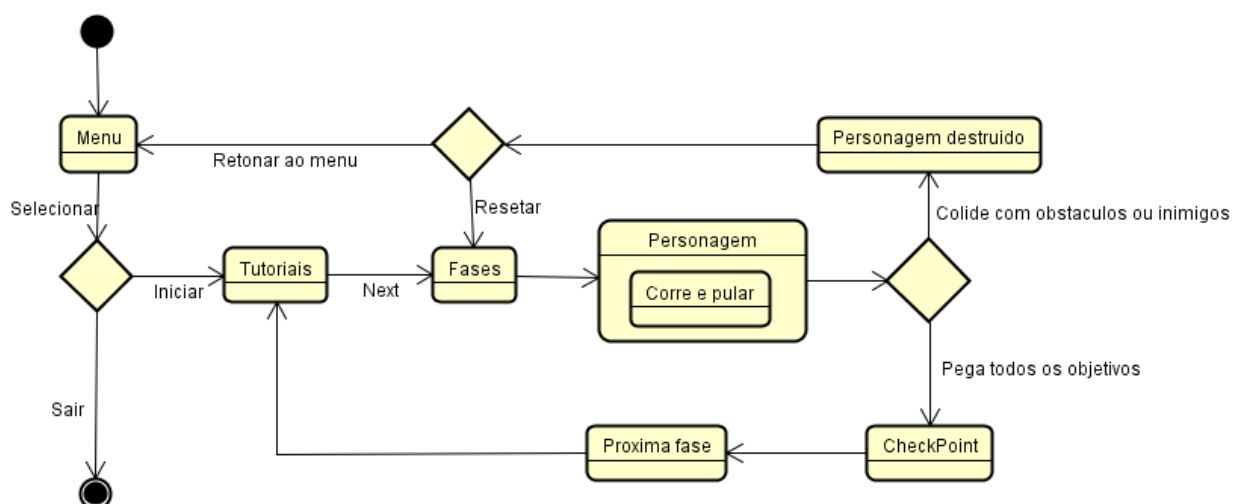


Fonte: autoria própria, 2020.

5.1.4 DIAGRAMA DE TRANSIÇÃO DE ESTADOS

O diagrama de transição de estados é usado para melhor detalhamento do funcionamento do jogo, já que sua função é mostrar os vários objetos que o usuário pode passar, a partir da execução de processos dentro do sistema, como mostrado na figura 10.

Figura 10 – Diagrama de Transição de Estados.



Fonte: autoria própria, 2020.

5.2 IMPLEMENTAÇÃO DO APLICATIVO

Na implementação ira ser mostrado o desenvolvimento do jogo proposto, seguindo a metodologia da engenharia de software e diagramas da UML mostrados.

Para o desenvolvimento foi necessária a instalação da Unity3D, Visual Studio Code e o download de assents.

Quadro 3 – Programas utilizados no desenvolvimento o jogo.

Programa	Valor	Versão atual	Link
Unity3D	Gratuito	2020.1.15f1	https://unity3d.com/pt/get-unity/download
Visual Studio Code	Gratuito	1.51.1	https://code.visualstudio.com/download
C#	Gratuito	1.23.6	Já vem instalado no Visual Studio Code.
Assents	Gratuito	1.0	https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360

		1.0	https://assetstore.unity.com/packages/2d/characters/pixel-adventure-2-155418
--	--	-----	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fonte: autoria própria, 2020.

A implementação é dividida em duas etapas: construção e edição de cenários e físicas, e criação de scripts, porém todo jogo tem uma história e objetivo, para que as fases façam sentido, então na próxima etapa será apresentada a contextualização do jogo.

5.2.3 CONTEXTUALIZAÇÃO

O jogo se passa em um mundo diferente, onde um astronauta de codinome “Comandante” se vê perdido depois que sua nave cai e ele é ejetado, na queda o seu traje que contém um IA (inteligência Artificial) para auxiliar os astronautas, é danificado e perde os dados referentes a entender a fala humana e as operações lógicas mais básicas. Nesse cenário o comandante é obrigado a ensinar de alguma forma a IA as operações perdidas para que ela encontre os destroços da nave e ele consiga se comunicar com a base.

5.2.3 OBJETIVO

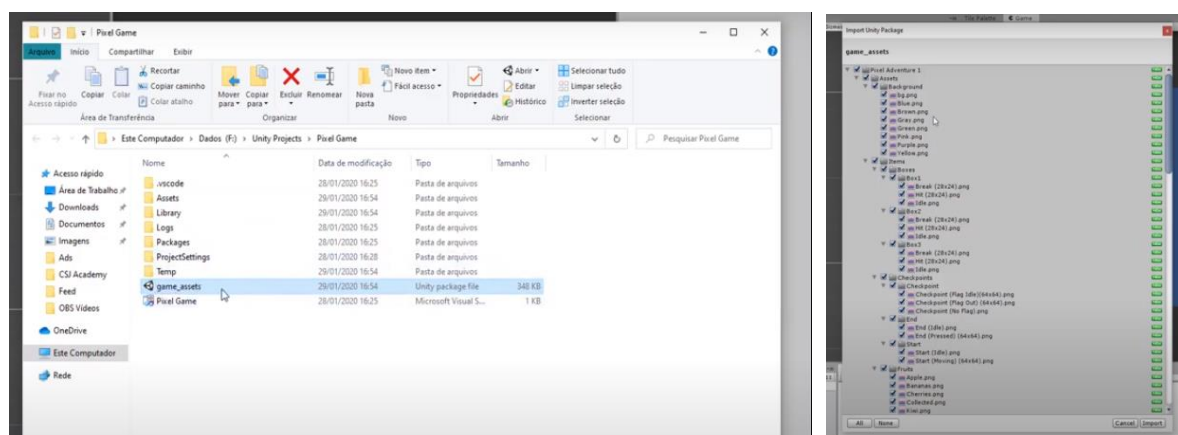
O objetivo do comandante é coletar bolinhas laranjas referentes a cada linha de código em C++, para que a IA consiga recuperar e aprender as operações que foram perdidas, para que eles possam alcançar os destroços.

5.2.4 CENARIOS

Depois de instalado a unity3D e baixado os assets, vem a criação do projeto propriamente dito. Ao criar um novo projeto na unity3D tudo vai estar vazio. Primeira coisa a ser feita é importar os assets para dentro do projeto, que é um processo

simples, após baixar a pasta dos assets é só arrastar ou dar dois cliques no arquivo .unitypackage e todos os assets referentes a ele será importado no projeto. Como na figura 11.

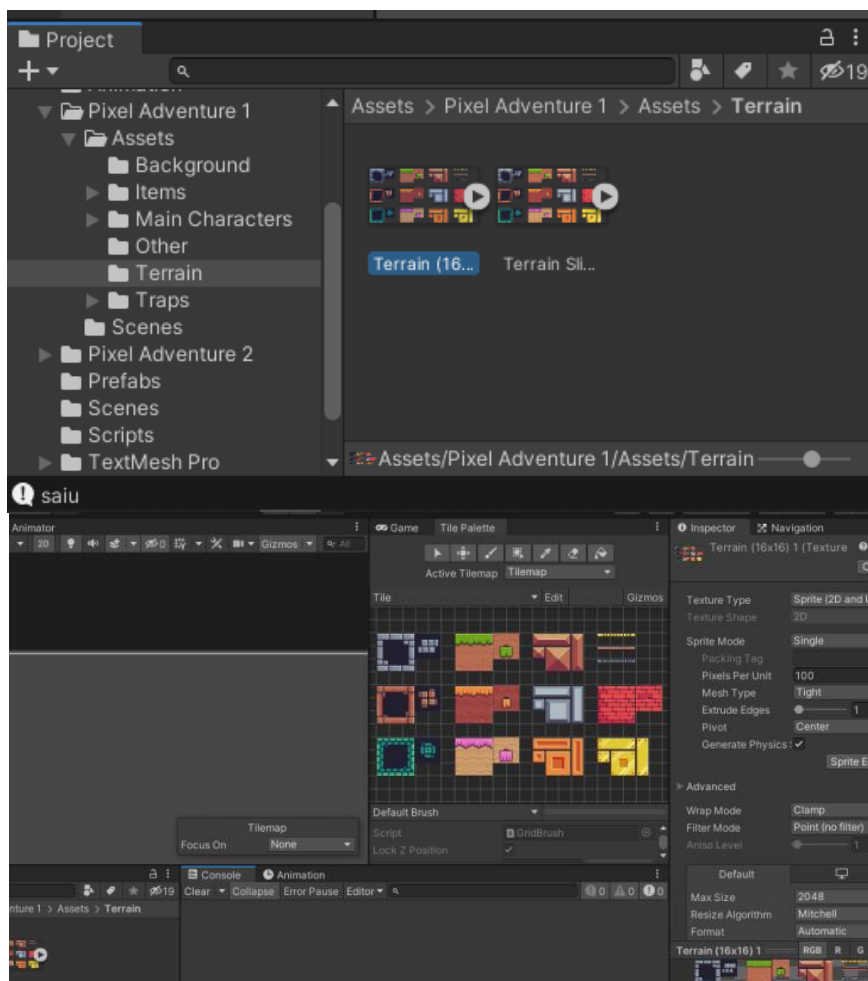
Figura 11 – Importando assets



Fonte: autoria própria, 2020.

O próximo passo é criar o tile map, ou mapa de blocos, para criação do cenário, com o asset já no projeto, vá na pasta referente a ele e segure e arraste para a aba Tile Palette, ou paleta de blocos. Assim como na figura 12.

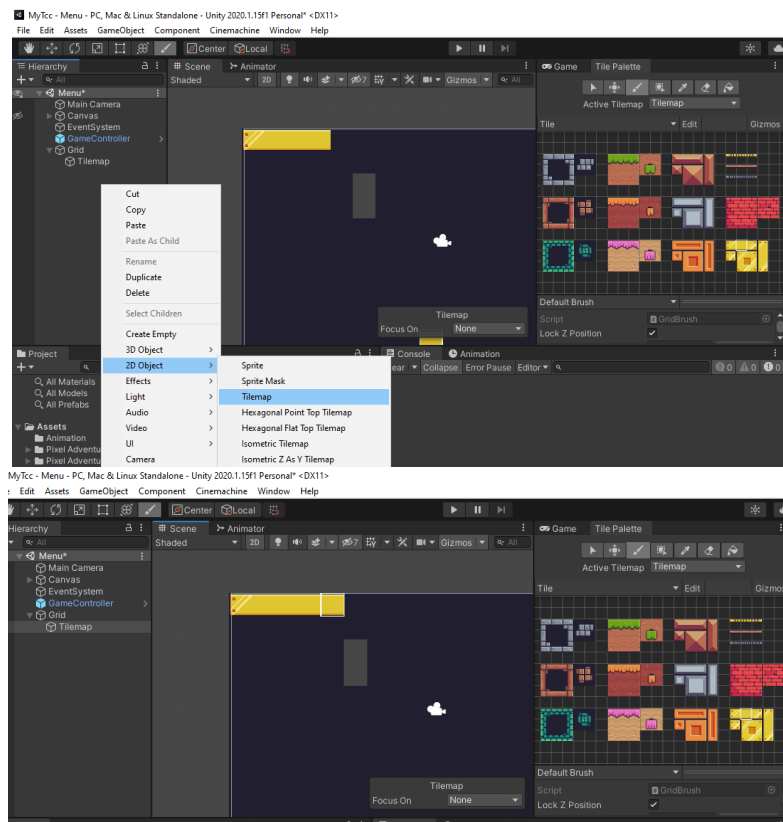
Figura 12 – criando o tile map.



Fonte: autoria própria, 2020.

Depois disso é só criar um objeto 2D do tipo tile map para que a cena seja dividida em grids e é só desenhar mesmo, colocando cada bloco onde quiser, até onde sua imaginação levar, criando os mais variados cenários.

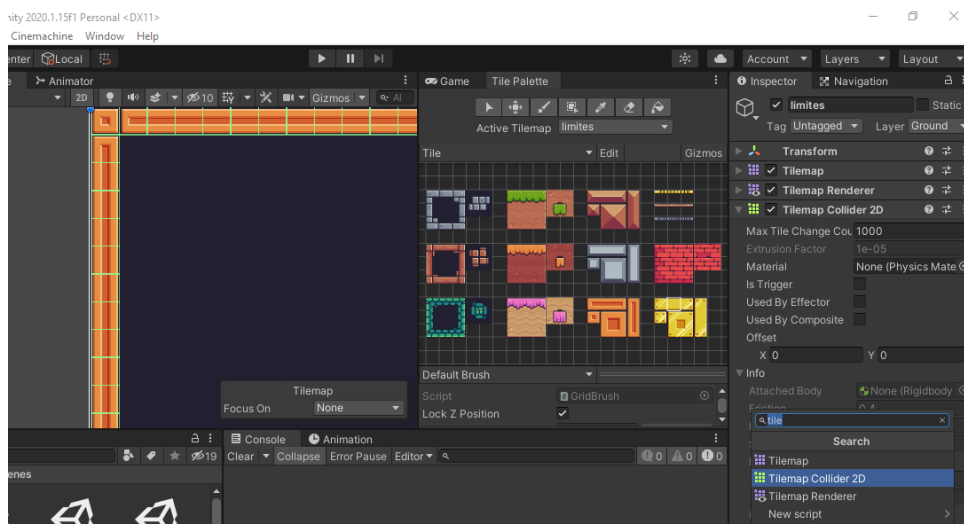
Figura 13 – Colocando blocos na cena.



Fonte: autoria própria, 2020.

Tendo criando o cenário é necessário colocar limites nele, para que quando o jogo iniciar o personagem não cai infinitamente, então se coloca um Tilemap Collider 2D que é um componente já presente no unity3D para identificar limites de blocos de mapa, na figura14 possível ver como fica os colisores em verde claro, também se faz necessário atribuir uma layer, que sera explicada no processo dos scripts.

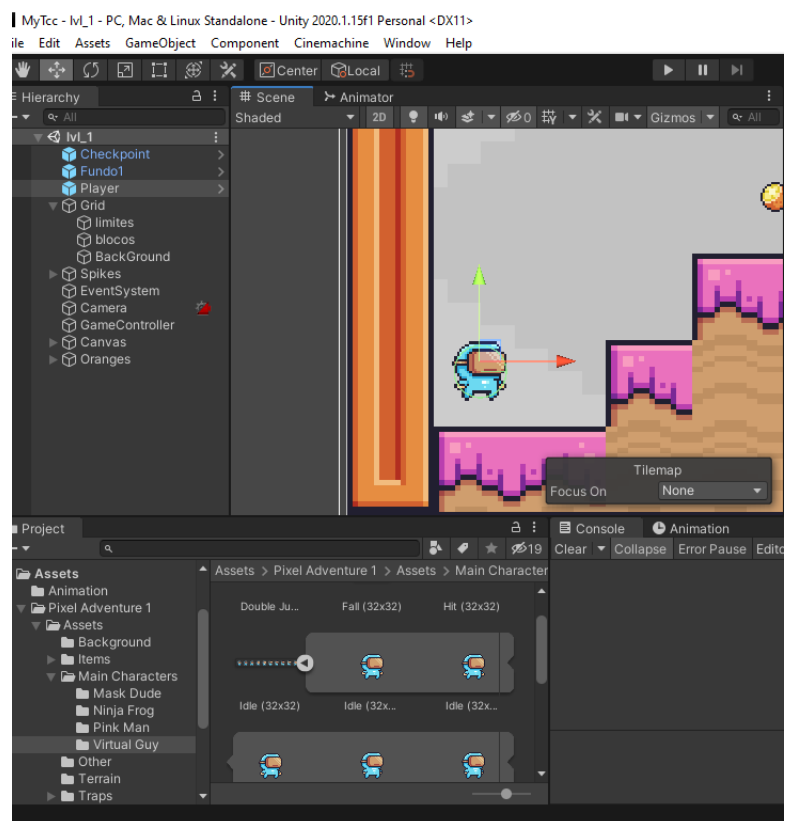
Figura 14 – Inserindo o tilemap Collider 2D.



Fonte: autoria própria, 2020.

Com todo o cenário feito, o próximo passo é a criação do personagem, que é muito parecida com o a dos tilemap (figura15).

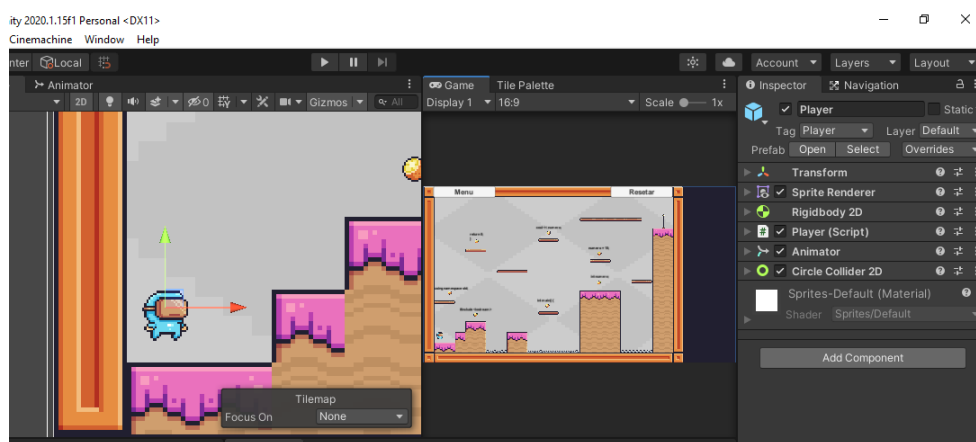
Figura 15 - Criação de personagem.



Fonte: autoria própria, 2020.

A diferença entre eles, além de o personagem já vir pronto, podendo ser alterado o tamanho e a tag de marcação, são os componentes que os compõe. Na figura 16 são mostrados seus componentes: Rigidbody 2D – responsável pela gravidade do boneco, circle Collider 2D – para não haver colisões estranhas com os colisores do cenário, um animator – que é a animação do personagem quando anda ou pula, e um script que é responsável pela forma de interação do personagem com os outros componentes.

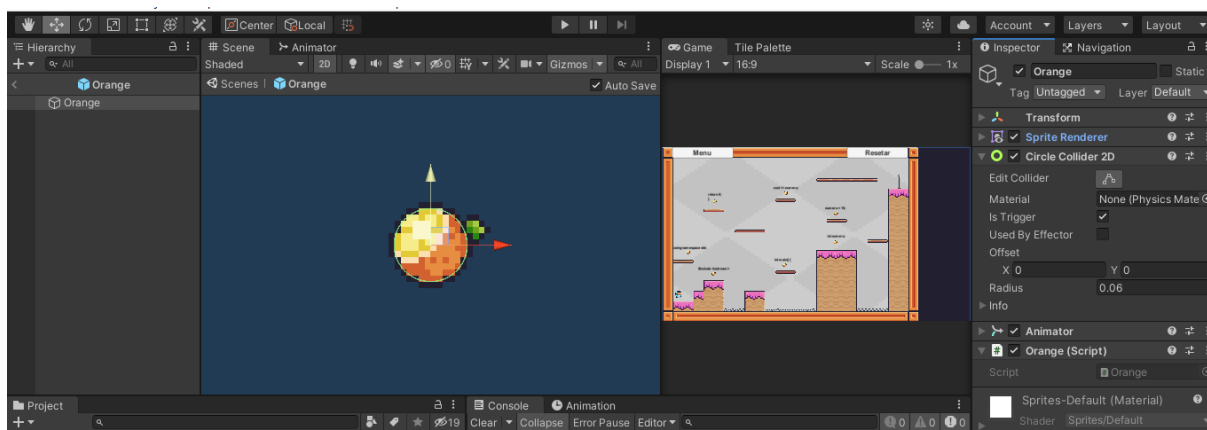
Figura 16 – Criação de personagem.



Fonte: autoria própria, 2020.

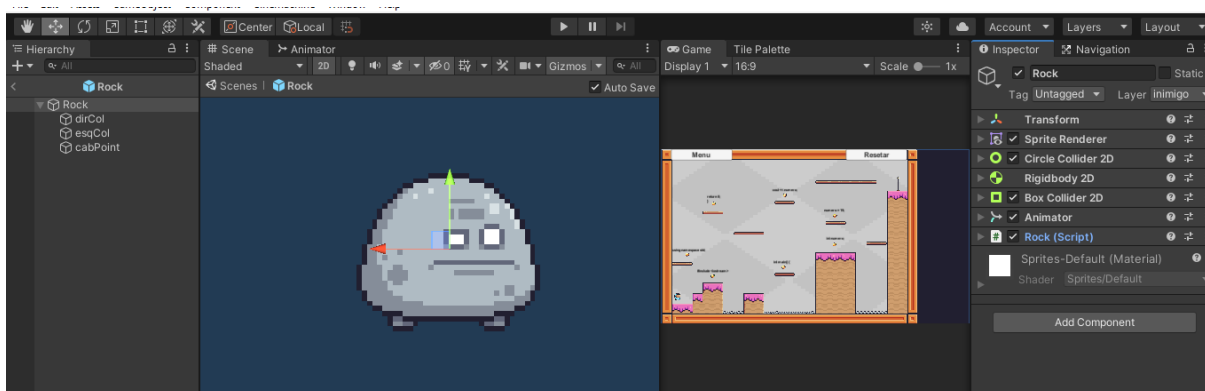
Os outros componentes seguem basicamente a mesma linha de criação, variando só em alguns componentes e suas tags. As figuras 17, 18 e 19 mostram alguns outros componentes de cenário.

Figura 17 – Orange.



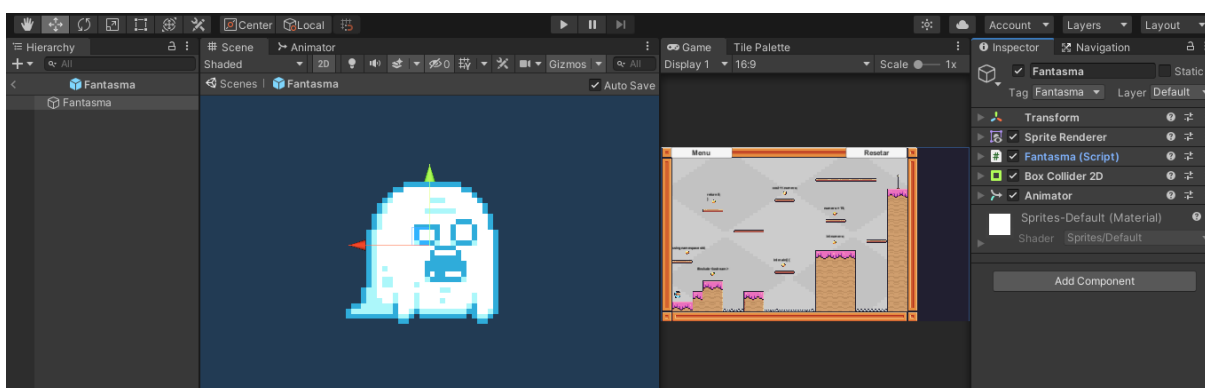
Fonte: autoria própria, 2020.

Figura 18 – Rock.



Fonte: autoria própria, 2020.

Figura 19 – Fantasma.

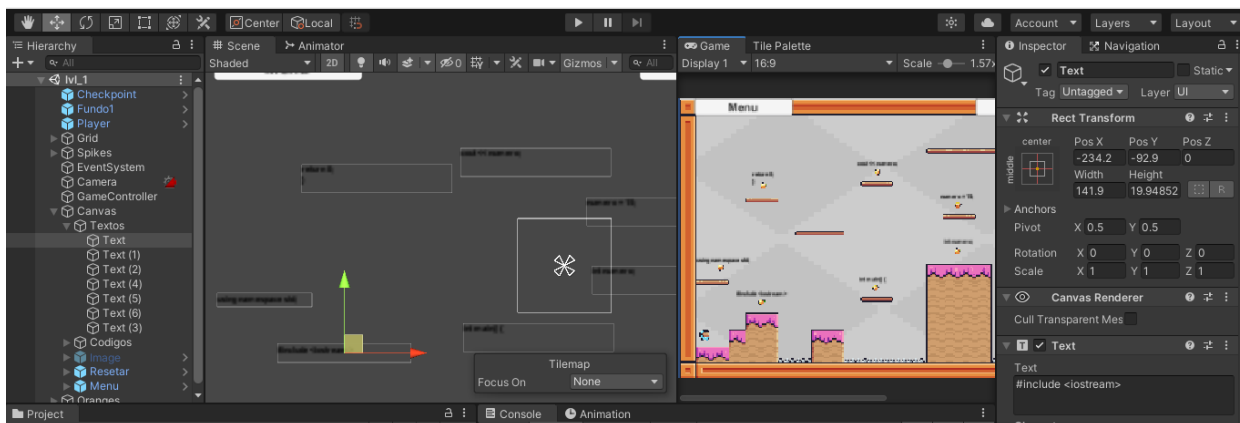


Fonte: autoria própria, 2020.

Para alinhar com a metodologia de jogo educacional, foi inserido textos nas telas, que correspondem a linhas de códigos, que quando consumido a “Orange” referente ao texto ele se apaga do local atual e aparece em outra parte da tela para a escrita do código. Na figura 20, pode se observar os variados linhas de códigos na aba “Scene” , como fica na tela do jogo na aba “Game” e o que esta escrito no primeiro texto .



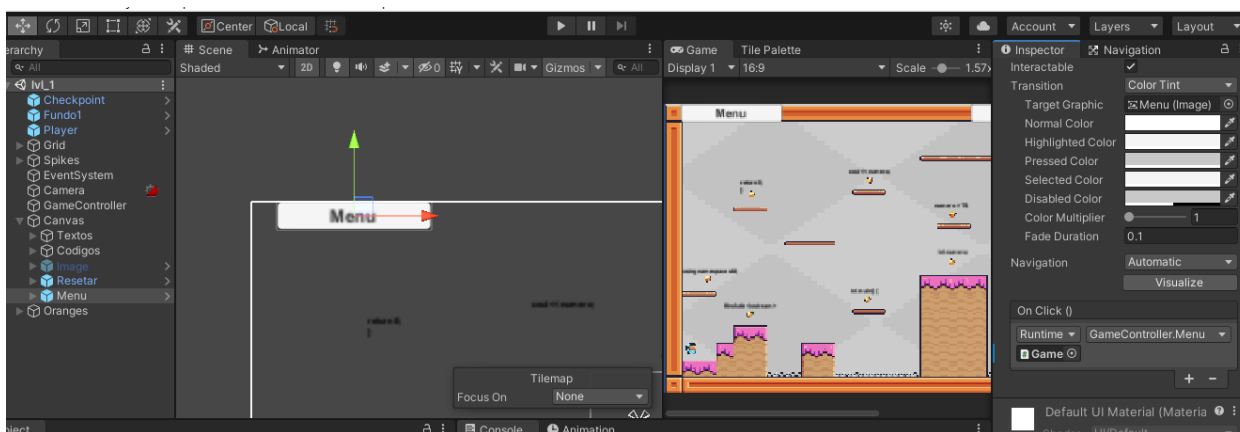
Figura 20 – textos em tela.



Fonte: autoria própria, 2020.

A também os botões: Menu, iniciar, sair, resetar e next, que funcionam com funções provenientes de um script (figura 21).

Figura 21 – Botão de Menu.

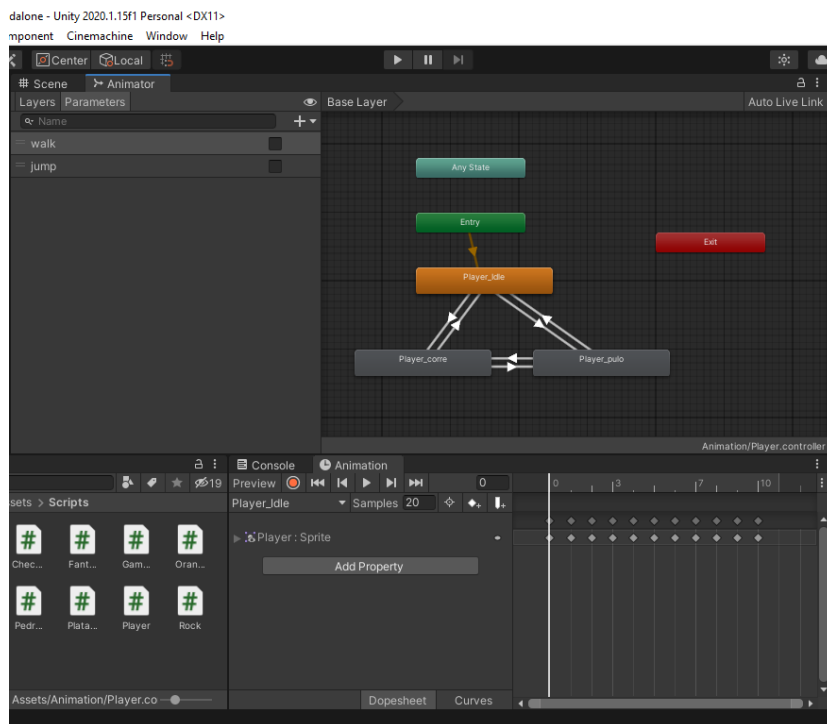


Fonte: autoria própria, 2020

5.2.5 ANIMAÇÕES

As animações estão presentes em alguns componentes, como o próprio personagem e são responsáveis por dar “vida” a eles, trazendo mais realidade a cena. A animação é uma sequência de frames de um objeto que quando passado executada em uma velocidade gera movimento ao objeto.

Figura 22 – Animação do player.



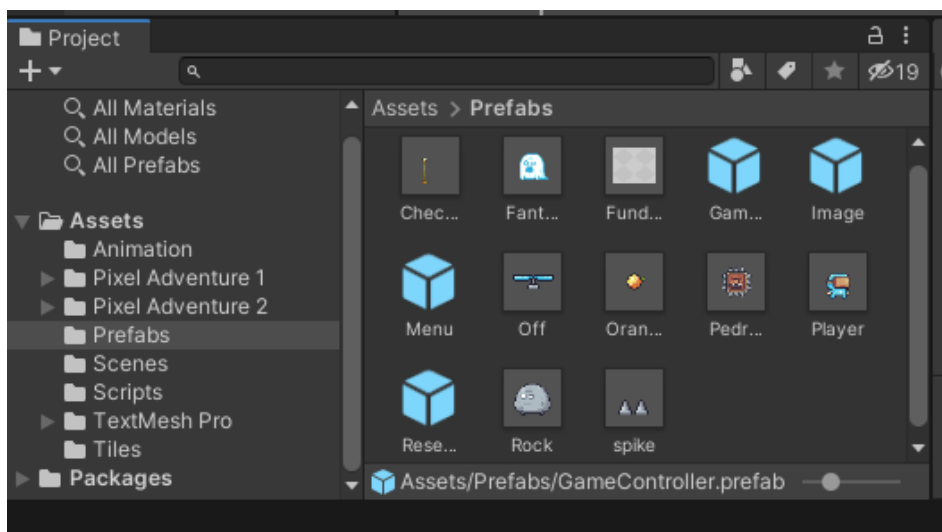
Fonte: autoria própria, 2020.

E ela funciona com uma animação inicial padrão, que é executada pelo script e a cada determinada ação o script chama uma animação diferente, no caso do player ele tem três animações: correr, pular e parado.

5.2.6 PREFABS

Os prefabs são muito utilizados na criação de jogos, eles nada mais são que uma componente de cena, como o personagem que fica salvo para ser usado em outras cenas, não precisando criar do zero tudo de novo sempre que começar a criar um cenário diferente (figura 23).

Figura 23 – Prefabs.



Fonte: autoria própria, 2020.

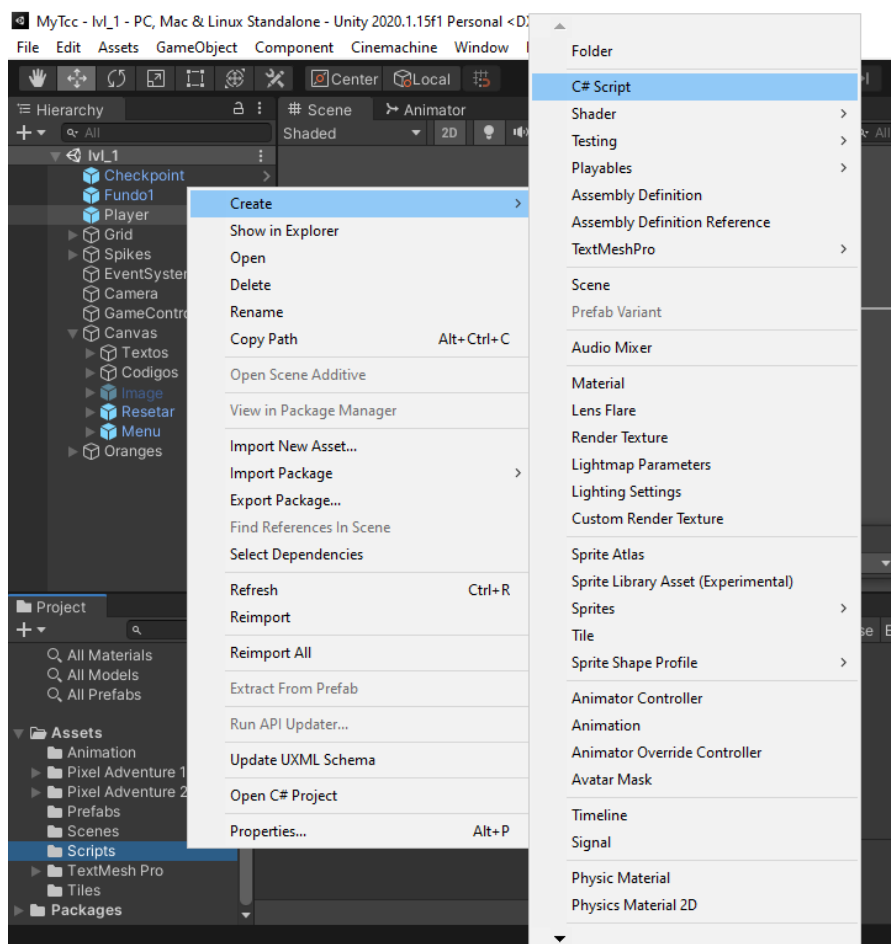
Então sempre que for necessário colocar um componente é só arrasta para a cena e ele será já com as configurações.

5.2.7 SCRIPTS

Os scripts são os responsáveis por como as interações entre os componentes da cena deve ocorrer, por isso ela é uma parte muito importante dos jogos.

Para a criação de um script basta ir: 1 – aba projetos e dentro da pasta assets criar uma nova pasta “Scripts”, 2 – Depois de criar a pasta é só clicar nela com o botão direito, ir em “ Create ” e selecionar “ C# Script ” (figura 24).

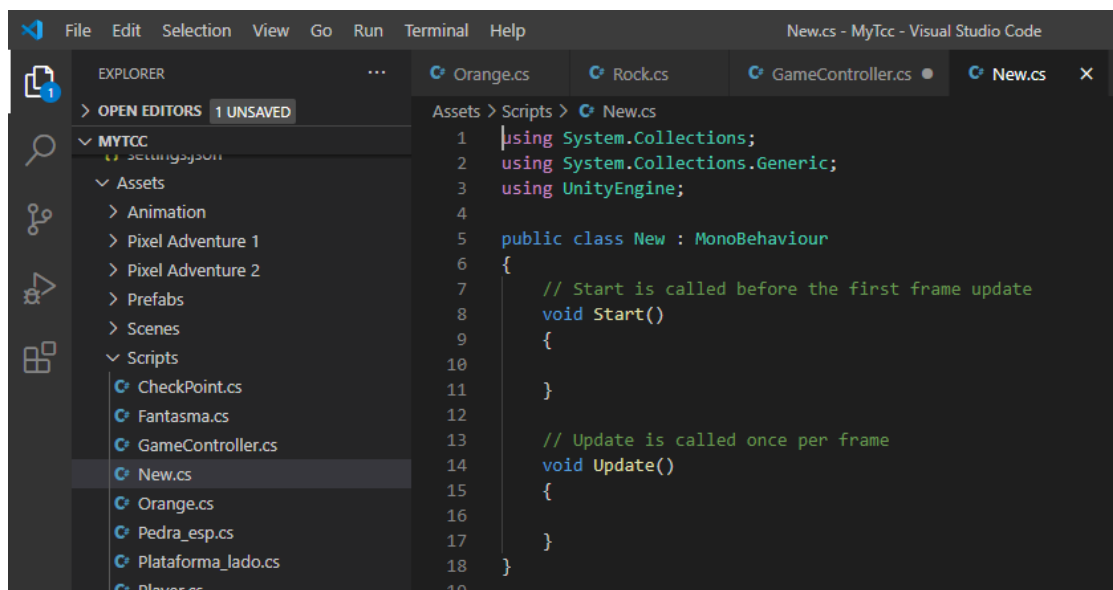
Figura 24 – Criando script.



Fonte: autoria própria, 2020.

Após isso é só atribuir um nome ao script e ele será criado e já pode ser editado no Visual Studio Code. Ao ser criada o script já contém as bibliotecas padrões da unity (“System.Collections”, “System.Collections.Generic”, “UnityEngine”) e com dois métodos padrões, “Void Start” – responsável por dizer qual ação inicial será feita quando o script for acionado pelo início do jogo, e o “Void update” – responsável o que será chamado a cada frame quando o script iniciado (figura 25).

Figura 25 - Script criado.

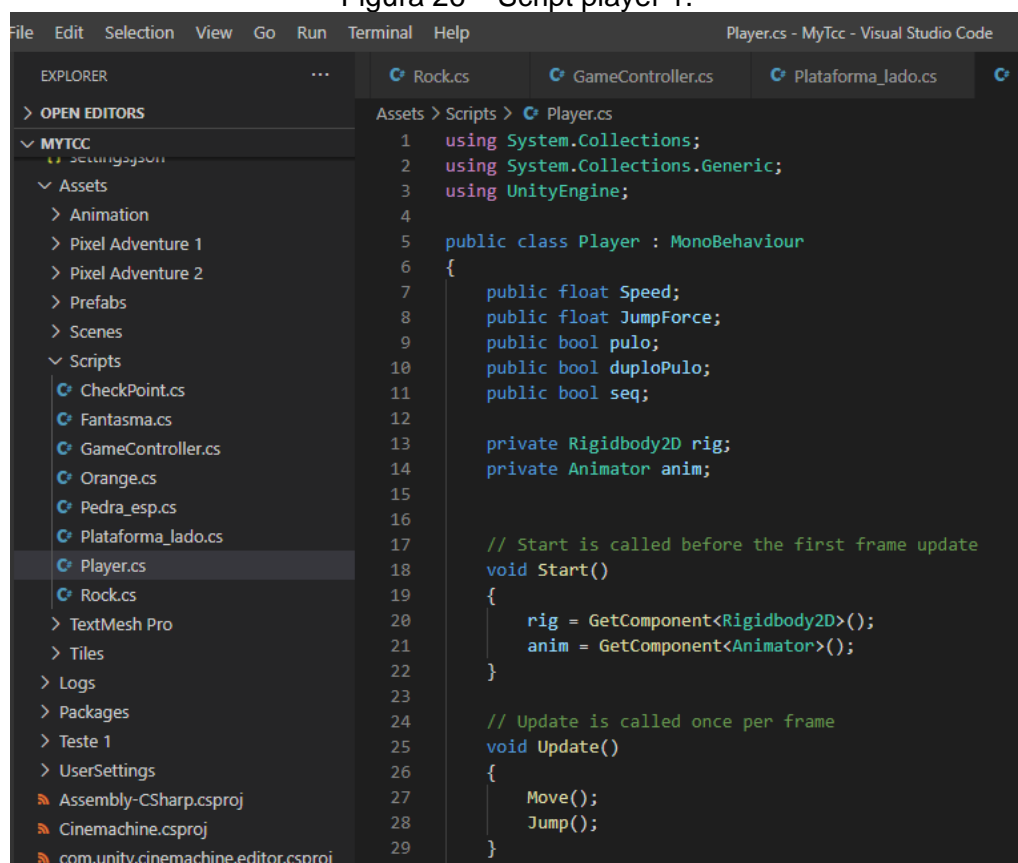


Fonte: autoria própria, 2020.

5.2.7.1 PLAYER

O personagem player possui em seu script, métodos para suas animações, e interações com alguns objetos dos cenários. Na figura 26, mostra suas variáveis, destacando as privadas que são responsáveis pelo gravidade(Linha 13) e animação(linha 14) a ação inicial no método “Strart”, ativando a física e a animação do personagem e método “update” que sempre chama os métodos “move”(Linha 27) e “jump”(Linha 28) .

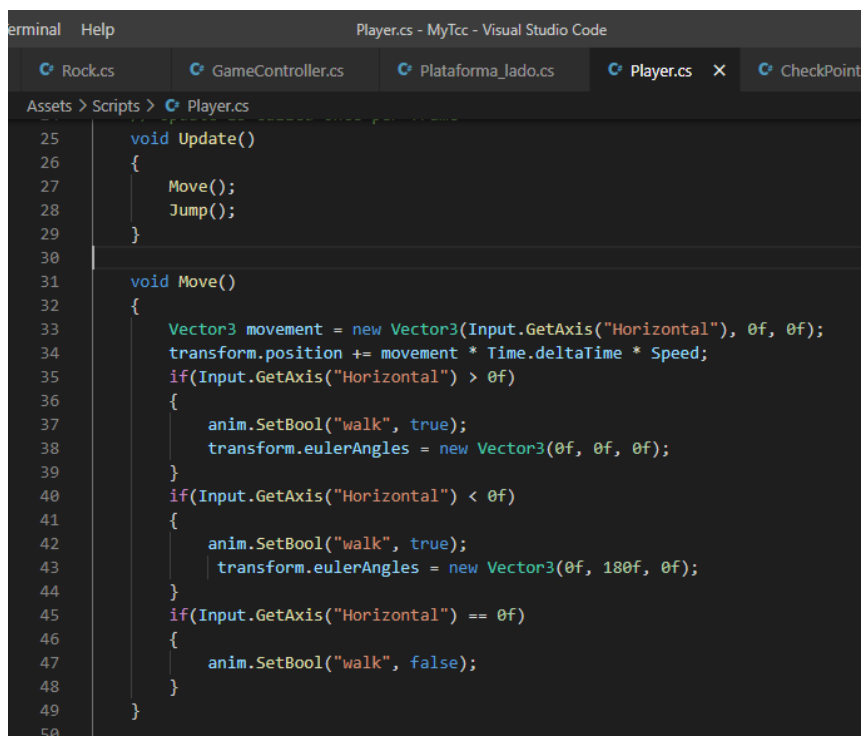
Figura 26 – Script player 1.



Fonte: autoria própria, 2020

Método “Move” funciona a partir entrada de sinal dos direcionais do teclado (Linha 33), a depender do direcional ativado ou se nenhum é ativado, a variável horizontal muda de valor entre 1 - direita(Linha 35), 0- parado(Linha 45) e -1 – esquerda(Linha 40), assim o método sabe em qual direção ir e transformando a posição e ativando a animação correspondente (figura 27), o método “ jump ” é muito parecido com o método “move” com alguns detalhes como pulo duplo que estará disponível na seção de anexo (figura 45).

Figura 27 – Movimentação do player.



```

25 void Update()
26 {
27     Move();
28     Jump();
29 }
30
31 void Move()
32 {
33     Vector3 movement = new Vector3(Input.GetAxis("Horizontal"), 0f, 0f);
34     transform.position += movement * Time.deltaTime * Speed;
35     if(Input.GetAxis("Horizontal") > 0f)
36     {
37         anim.SetBool("walk", true);
38         transform.eulerAngles = new Vector3(0f, 0f, 0f);
39     }
40     if(Input.GetAxis("Horizontal") < 0f)
41     {
42         anim.SetBool("walk", true);
43         transform.eulerAngles = new Vector3(0f, 180f, 0f);
44     }
45     if(Input.GetAxis("Horizontal") == 0f)
46     {
47         anim.SetBool("walk", false);
48     }
49 }
50

```

Fonte: autoria própria, 2020

As colisões dos player são identificadas a partir das tags ou layers dos vários componentes do cenário. Os métodos “OnCollisionEnter2D” – corresponde a um contato que provoca reação em algum dos componentes (Linhas 72 – 90) , “OnCollisionExit2D” – verificar se não está tendo mais contato, utilizado para estabelecer o pulo duplo (Linhas 92 – 98), e o “OnTriggerEnter2D” – é utilizado para acionar ou desacionar um componente ao contato(Linhas 101 – 106), nesse caso para nesse caso utilizado no acionamento e desacionamento chamando um método da Classe GameController dos textos na cena (figura 28).

Figura 28 – Colisão do player com objetos.



```

71
72 void OnCollisionEnter2D(Collision2D collision)
73 {
74     if(collision.gameObject.layer == 8)
75     {
76         pulo = false;
77         anim.SetBool("jump", false);
78     }
79
80     if(collision.gameObject.tag == "spike")
81     {
82         Destroy(gameObject);
83         GameController.instance.GameOver();
84     }
85     if(collision.gameObject.tag == "Fantasma")
86     {
87         Destroy(gameObject);
88         GameController.instance.GameOver();
89     }
90 }
91
92 void OnCollisionExit2D(Collision2D collision)
93 {
94     if(collision.gameObject.layer == 8)
95     {
96         pulo = true;
97     }
98 }
99
100 void OnTriggerEnter2D(Collider2D collider)
101 {
102     if(collider.gameObject.tag == "Orange")
103     {
104         GameController.instance.Aparecer();
105     }
106 }

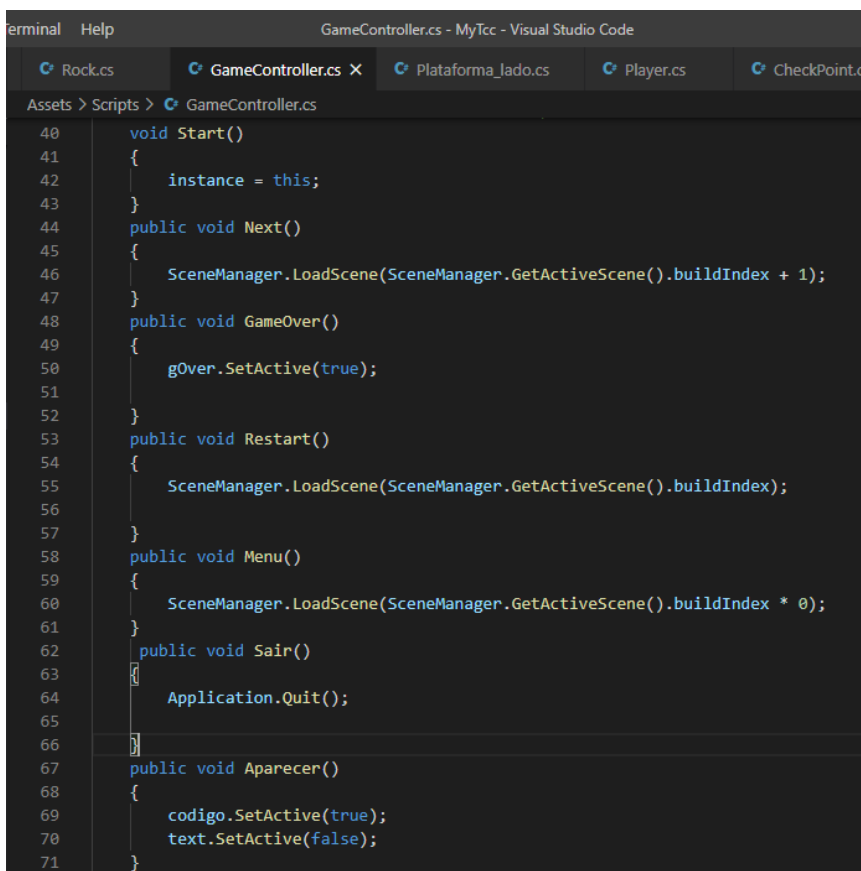
```

Fonte: autoria própria, 2020

5.2.7.2 GAMERCONTROLLER

O GameController é a classe responsável pelas funções presentes nos botões: Next – que pula a cena (Linhas 45 – 47), Resetar – recarregando a mesma cena (Linhas 53 – 57), Menu – redirecionar a tela de menu (Linhas 58 – 61) e Sair – fechar o jogo (Linhas 62 – 66), execução da cena de Game Over quando o player é destruído (Linhas 48 – 52) e executar a função chamada pelo player fazendo uns textos aparecer e outros desaparecer (Linhas 67 – 71), (figura 29).

Figura 29 – Script do GameController.



```

40 void Start()
41 {
42     instance = this;
43 }
44 public void Next()
45 {
46     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
47 }
48 public void GameOver()
49 {
50     gOver.SetActive(true);
51 }
52 }
53 public void Restart()
54 {
55     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
56 }
57 }
58 public void Menu()
59 {
60     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex * 0);
61 }
62 public void Sair()
63 {
64     Application.Quit();
65 }
66 }
67 public void Aparecer()
68 {
69     codigo.SetActive(true);
70     text.SetActive(false);
71 }

```

Fonte: autoria própria, 2020.

5.2.7.3 INIMIGOS

O inimigo que apresenta uma mecânica diferente e precisa ser destacada é o “Rock”, que é uma pedra que anda pela cena. Na figura 30, o “void start” dele é igual ao do player (Linhas 26 – 30), porém seu “void update” é diferente, já que o “rock” sempre se movimenta (Linha 34) e detecta a colisão através de linhas imaginárias, que quando entra em contato com as paredes a partir de uma layer ele muda sua direção e velocidade (Linhas 36 - 42).

Figura 30 – Script 1 Rock.

```

26 void Start()
27 {
28     rig = GetComponent<Rigidbody2D>();
29     anim = GetComponent<Animator>();
30 }
31 // Update is called once per frame
32 void Update()
33 {
34     rig.velocity = new Vector2(Speed, rig.velocity.y);
35
36     colliding = Physics2D.Linecast(dirCol.position, esqCol.position, layer);
37     if(colliding)
38     {
39         transform.localScale = new Vector2(transform.localScale.x * -1f, transform.localScale.y);
40         Speed *= -1f;
41     }
42 }

```

Fonte: autoria própria, 2020.

Quando “Rock” entra em contato com o player, duas coisas podem acontecer: 1- se o player pular sobre sua cabeça, o “Rock” para, aciona animação dele sumindo e ele é destruído (Linhas 50 – 61), 2- o player colide com uma de suas laterais e então o player é destruído chamando a função do GameController Game Over (Linhas 62 – 67).

Figura 31 – Colisão do “Rock”.

```

Assets > Scripts > Rock.cs
40     Speed *= -1f;
41 }
42 }
43
44 bool playerDes = false;
45
46 void OnCollisionEnter2D(Collision2D col)
47 {
48     if(col.gameObject.tag == "Player")
49     {
50         float height = col.contacts[0].point.y - cabPoint.position.y;
51         if(height >= 0 && !playerDes)
52         {
53             col.gameObject.GetComponent<Rigidbody2D>().AddForce(Vector2.up * 20, ForceMode2D.Impulse);
54             Speed = 0;
55             anim.SetTrigger("die");
56             boxCollider2D.enabled = false;
57             circleCollider2D.enabled = false;
58
59             rig.bodyType = RigidbodyType2D.Static;
60
61             Destroy(gameObject, 0.1f);
62         }
63         else
64         {
65             playerDes = true;
66             GameController.instance.GameOver();
67             Destroy(col.gameObject);
68         }
69     }
70 }
71

```

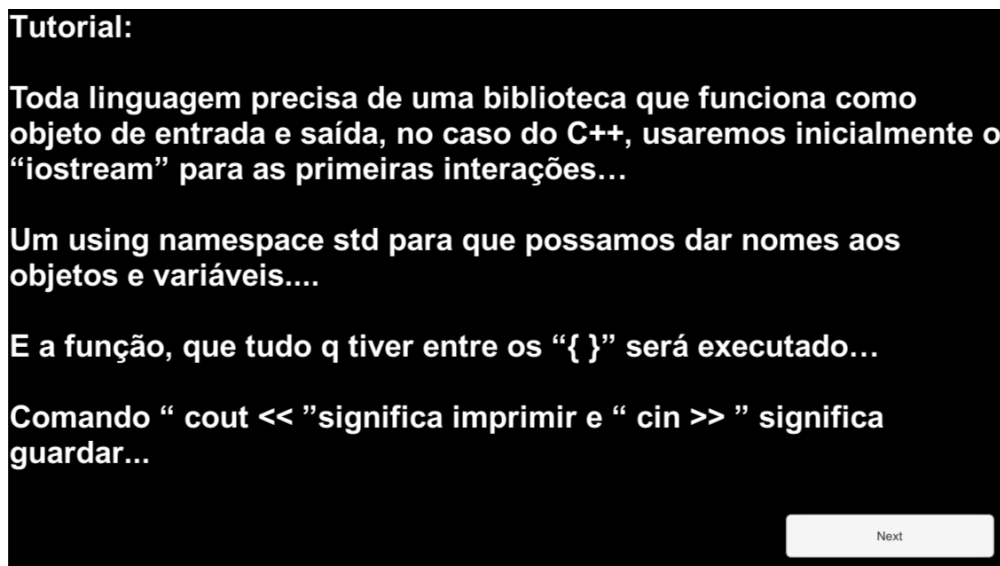
Fonte: autoria própria, 2020

Os demais componentes tem métodos muito parecidos, se diferenciando em pouca coisa, porem para melhor acompanhamento se disponibilizada as figuras dos outros scripts na seção de anexos (figuras 46, 47, 48, 49) .

5.2.8 LINGUAGENS

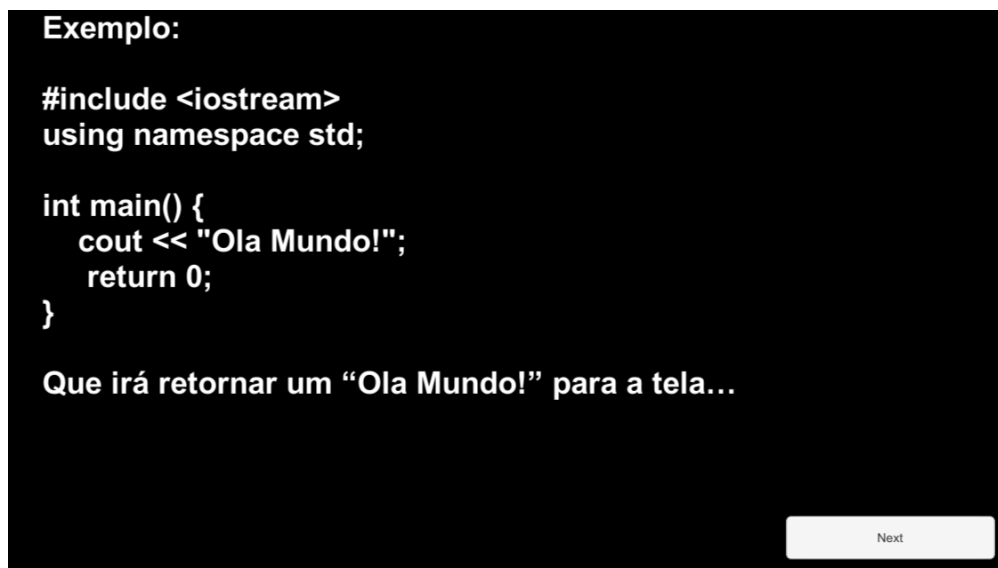
A aplicabilidade do jogo junto a engenharia da computação, é a presença das linguagens de programação presentes nas cenas, através de tutorias, para apresentar o funcionamento de determinada operação (figuras 32 e 33), quanto nas cenas já que o objetivo do jogo é coletar as “Oranges” que cada uma representa um trecho de código e a medida que são coletados(figura 34), os códigos são escritos em uma espécie de “editor de texto” na tela do jogo, e quando todos são consumidos o player pode passar de fase (figura 35).

Figura 32 – Tutorial 1.



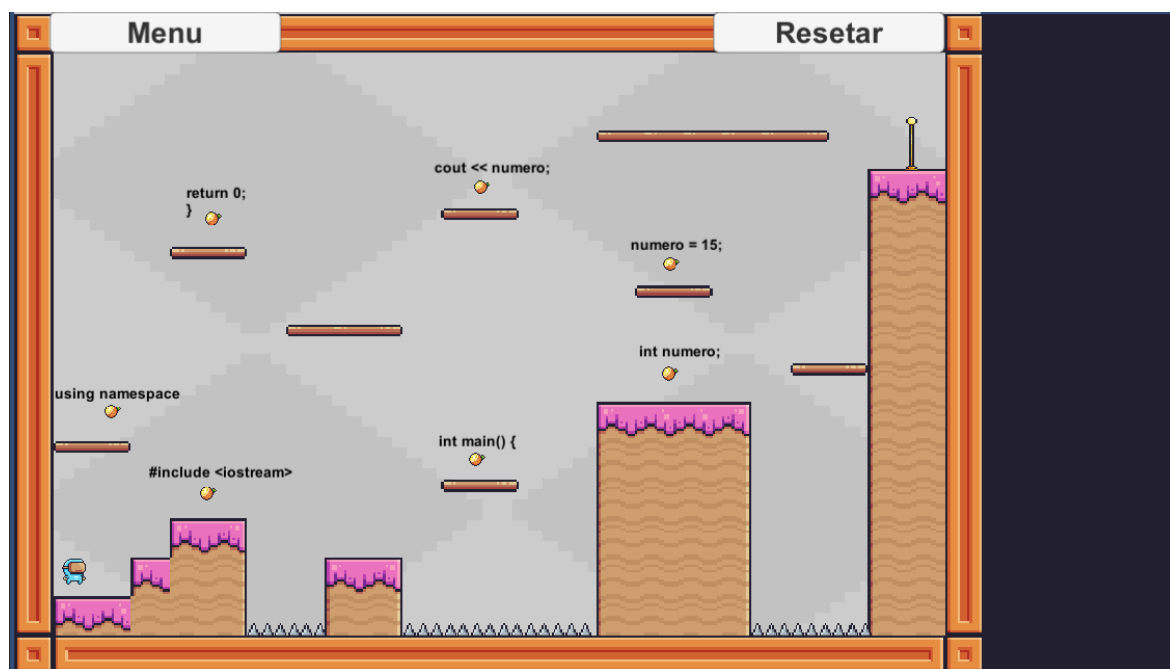
Fonte: autoria própria, 2020

Figura 33 – Tutorial 2



Fonte: autoria própria, 2020

Figura 34 – Fase 2.



Fonte: autoria própria, 2020.

Figura 35 – Fase 2 Incompleta.



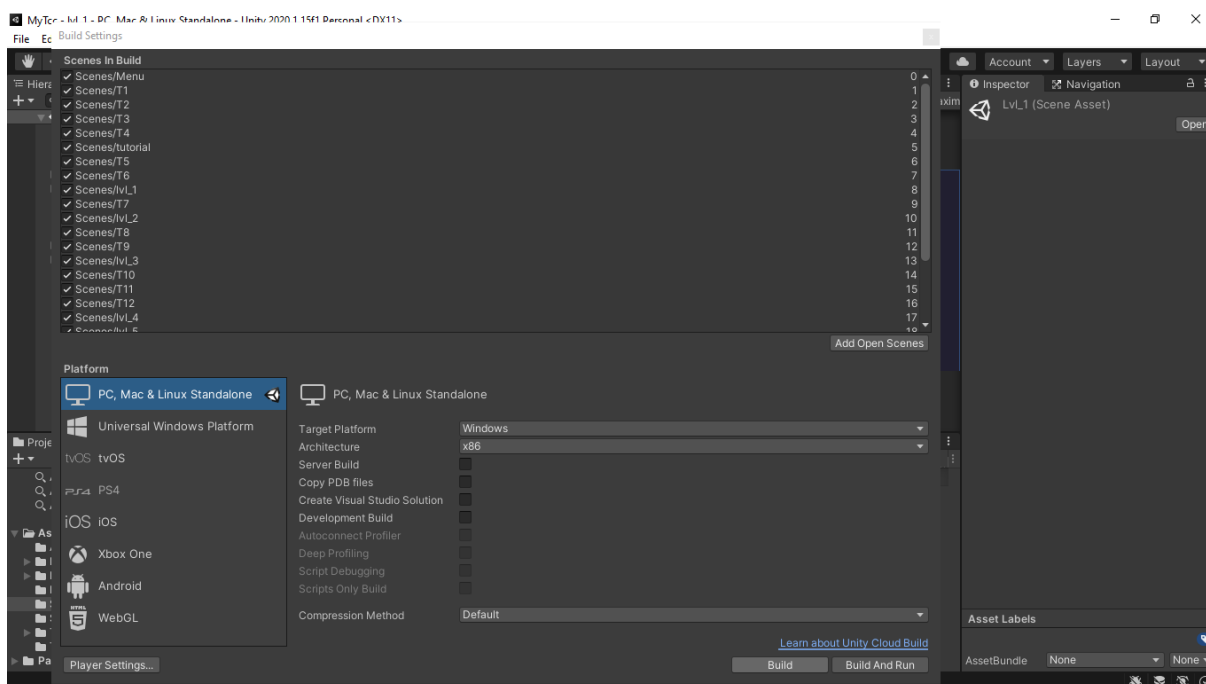
Fonte: autoria própria, 2020.

A utilização de linguagem C++, foi devido a sua popularidade e por muitas vezes ser usada como porta de entrada, pra quem começa a programar, além disso muitas outras linguagens que vieram depois dela, usa muito de suas estruturas como base, o que facilita a migração para outras linguagens.

5.2.9 BUILD

Ao finalizar o projeto, todas as cenas criadas elas tem que ser carregadas e salvas na build e organizadas em ordem, assim que todas as cenas estejam adicionadas é a plataforma e clicar em build embaixo na tela e criar um pasta para o jogo e pronto, o executável do jogo será criado (figura 36).

Figura 36 – Build do jogo.

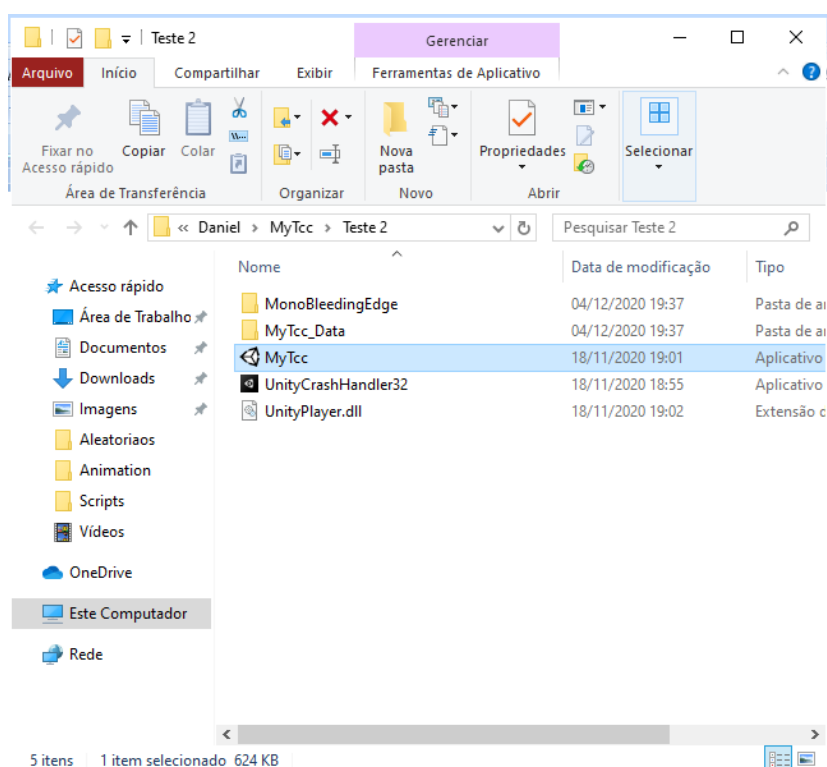


Fonte: autoria própria, 2020.

6 RESULTADOS E SIMULAÇÃO

Nesta seção, serão apresentados os resultados do desenvolvimento do jogo, o primeiro passo foi a criação de um arquivo executável (Figura 37), para a realização dos testes.

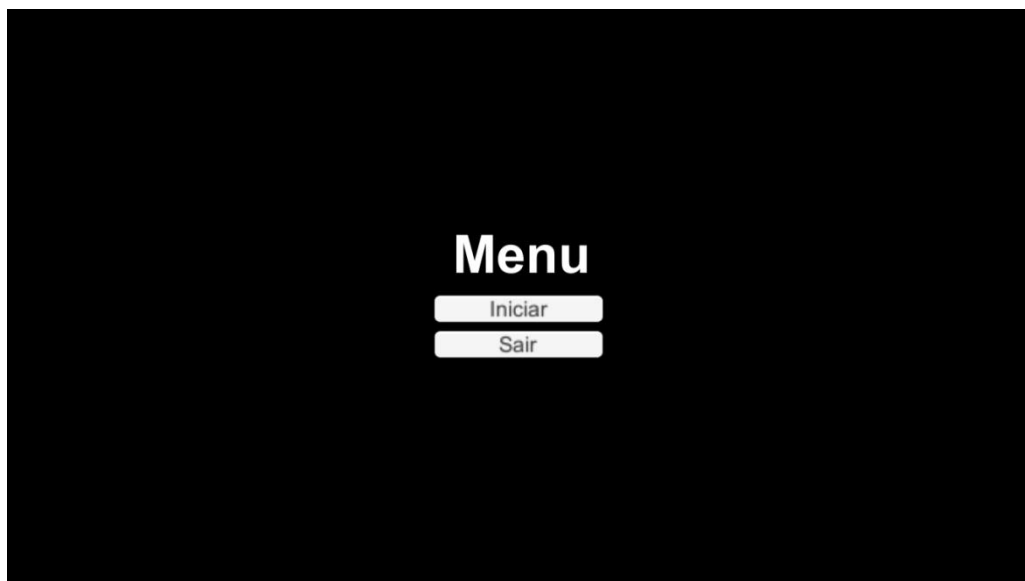
Figura 37 – Executável do jogo.



Fonte: autoria própria, 2020.

Após a criação do executável, o jogo é inicializado aparecendo a primeira tela que será o menu inicial (figura 38).

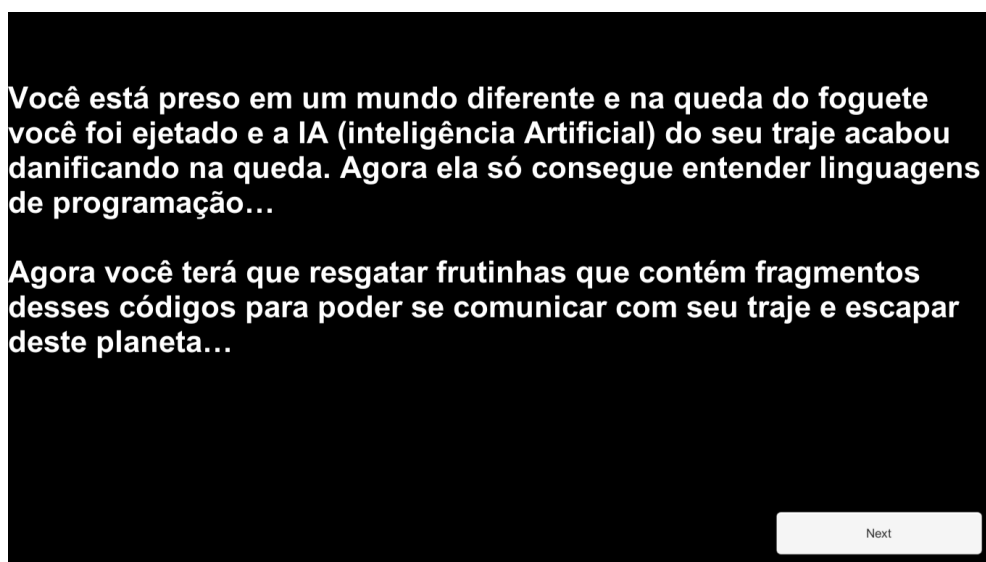
Figura 38– Menu.



Fonte: autoria própria, 2020.

No menu inicial, a 2 opções como mostrado na figura 38, “Iniciar” e “Sair”, onde ao clicar em “Iniciar” o jogador é encaminhado para a tela de historia do jogo (figura 39), e se clicar em “Sair” o jogo é fechado.

Figura 39 – Apresentação da Historia.



Fonte: autoria própria, 2020.

A seguir o jogador só terá a opção “Next” que será usada para avançar as telas para outras telas de historia ou telas de tutorias do funcionamento da linguagem C++ e nas mesmas também só apareceram a opção de “Next” (figuras 40,41 e 42).*

Figura 40 – Tutorial 1: Como funciona o C++.

Tutorial:

Toda linguagem precisa de uma biblioteca que funciona como objeto de entrada e saída, no caso do C++, usaremos inicialmente o “iostream” para as primeiras interações...

Um using namespace std para que possamos dar nomes aos objetos e variáveis....

E a função, que tudo q tiver entre os “{ }” será executado...

Comando “ cout << ”significa imprimir e “ cin >> ” significa guardar...

Next

Fonte: autoria própria, 2020.

Figura 41 – Exemplo de código a ser seguido.

Exemplo:

```
#include <iostream>
using namespace std;

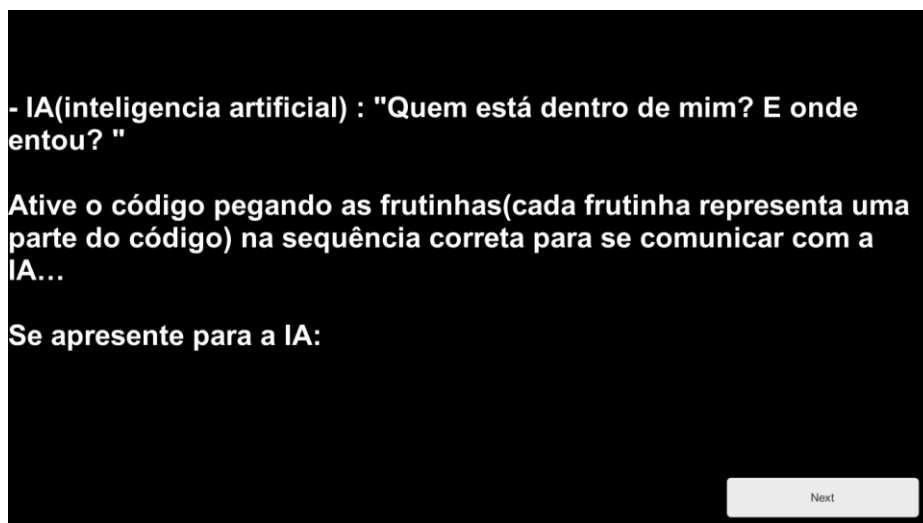
int main() {
    cout << "Ola Mundo!";
    return 0;
}
```

Que irá retornar um “Ola Mundo!” para a tela...

Next

Fonte: autoria própria, 2020.

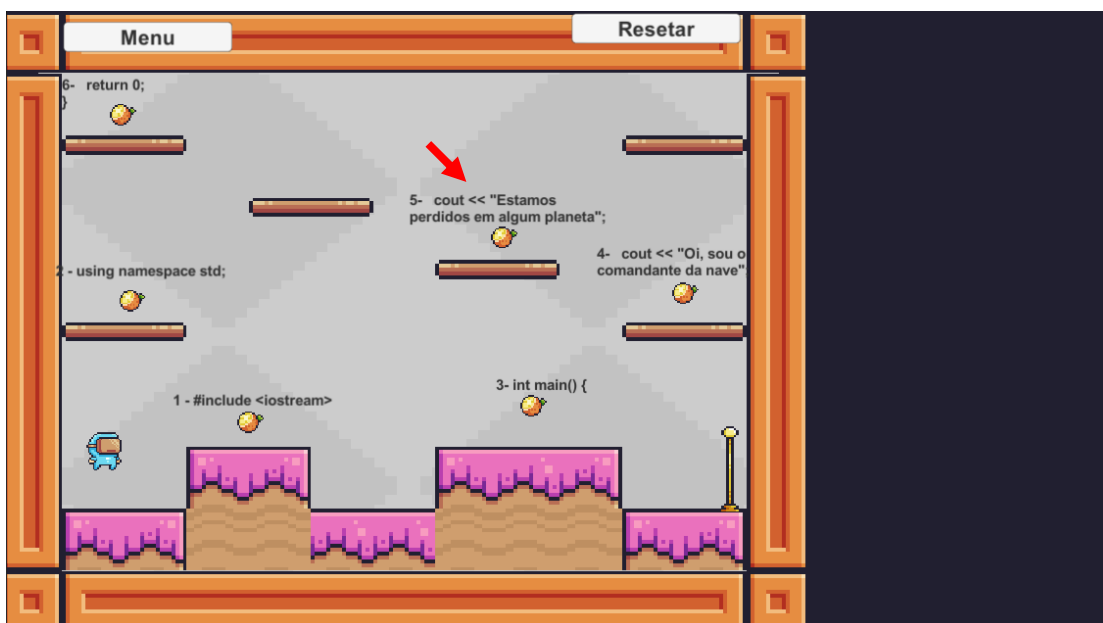
Figura 42 – Dialogo com a IA.



Fonte: autoria própria, 2020.

Após telas de historia e tutorias, começam as fases, que tem o objetivo por meio de linhas de códigos espalhadas pela fase que ao serem coletadas mostram alguma operação lógica referentes a historia e tutorias passados(figura 43).

Figura 43- Fase 1.



Fonte: autoria própria, 2020.

Quando cada linha coletada, a mesma aparece ao lado da tela mostrando como o código esta ficando(figura 44), ate que o mesmo esteja completo e assim possa passar para próxima fase, caso o código não esteja completo o checkpoint não avança a fase.

Figura 44 – Fase 1 Completa.



Fonte: autoria própria, 2020.

O jogo apresenta alguns objetos e inimigos em suas fases para criar uma dinâmica maior do jogador e torná-lo mais difícil (figura 45), onde caso o player colida com algum deles na fase o player morre e a tela de game over aparece, dando as opções de “Menu” e “Resetar” (figura 46).

Figura 45 - Exemplos de inimigos e objetos.

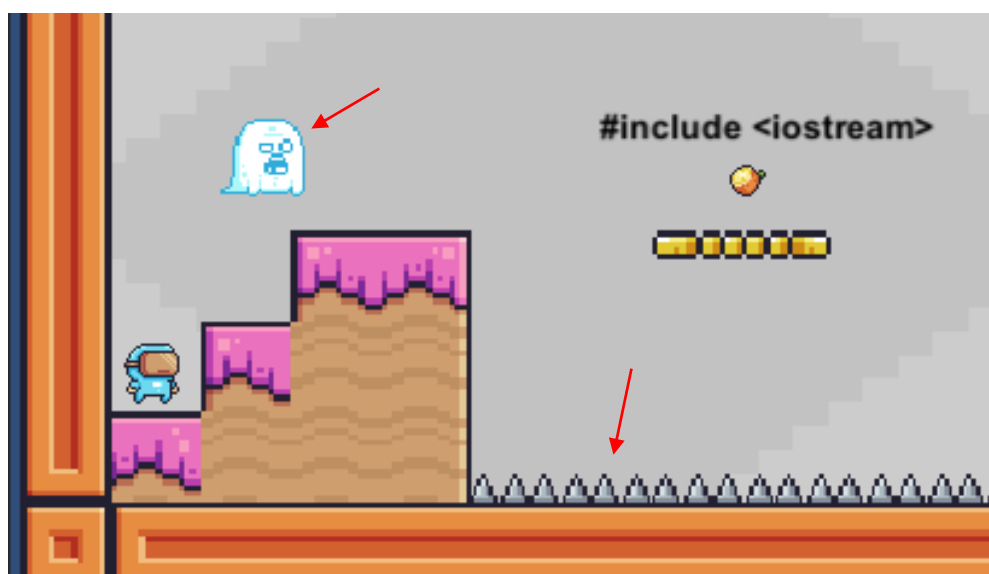
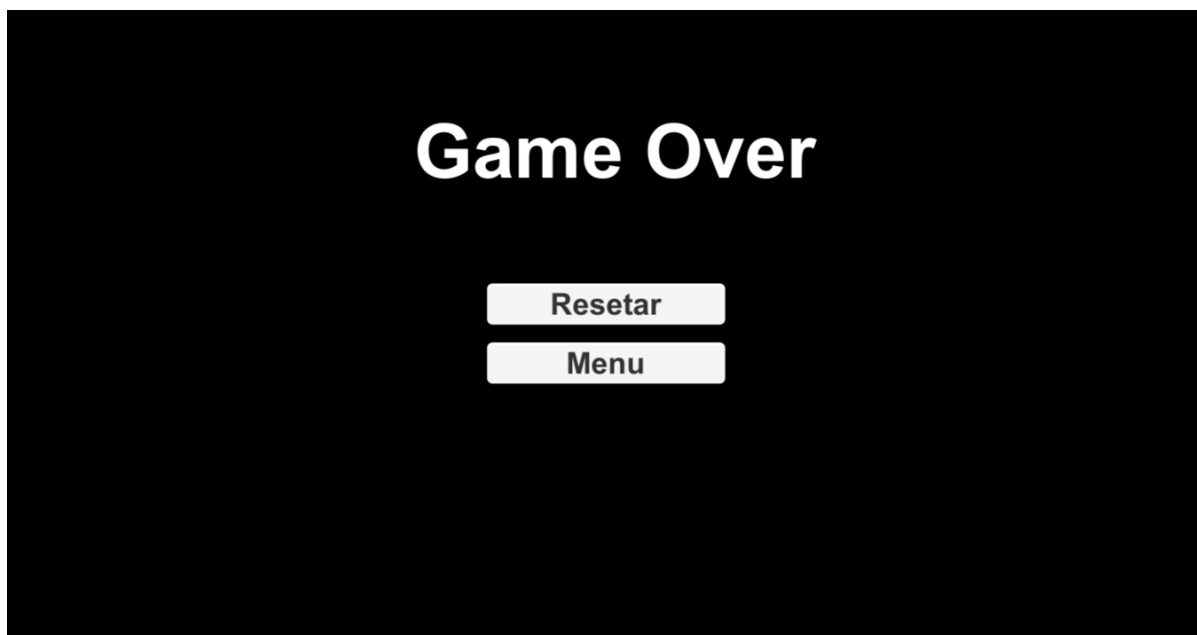


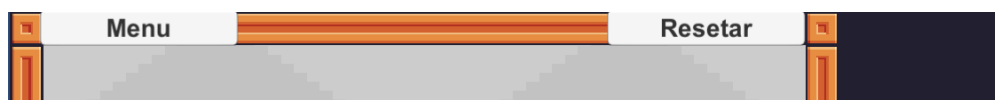
Figura 46– Tela de Game Over.



Fonte: autoria própria, 2020

A presente nas fases também dois botões “Menu” e “Resetar”, onde respectivamente encaminham para o menu inicial do jogo e o outro recomeça a fase(figura 47).

Figura 47- Botões de menu e resetar.



Fonte: autoria própria, 2020

Assim foi possível testar todas as funcionalidades do jogo presentes no jogo, e como elas funcionam a partir da interação do jogador.

7 CONSIDERAÇÕES FINAIS

A demanda do mercado de trabalho por profissionais na área da computação vem aumentando e buscar novas formas de atrair pessoas para a área é essencial. Então jogos no geral são uma ótima escolha, por serem bastante populares e conseguirem mesclar bem as suas funcionalidades com aplicações das mais variadas áreas, inclusive com a engenharia da computação.

Então o desenvolvido do jogo durante este trabalho é a comprovação de que com os conhecimentos adquiridos durante todo o curso de engenharia da computação como as linguagens de programação, interface homem maquina, engenharia de software, é possível se criar aplicações como jogos.

7.1 SUGESTÃO PARA TRABALHOS FUTUROS

Para continuação do desenvolvimento do jogo e trabalhos relacionados futuros, visando aperfeiçoamento:

- Criação de fases maiores e mais códigos;
- Utilização de outras linguagens de programação;
- Criação de banco de dados para salvar progresso, caso expanda o jogo;
- Criação de estatísticas, como tempo de conclusão do jogo;
- Criação de servidores online para disputa por rankings entre os usuários;
- Expansão para outras áreas de conhecimento;

8 REFERENCIA

ARAÚJO, D. C.; RODRIGUES, A. N.; SILVA, C. V. de A.; SOARES, L. S. O Ensino da Computação na Educação Básica Apoiado por Problemas: Práticas de Licenciados em Computação. In: Anais do XXIII WEI (Workshop sobre Educação em Computação) Garanhuns. (2015). Disponível em:

GARLET, Daniela; BIGOLIN, Nara Martini; SILVEIRA, Sidnei Renato. **Ensino de Programação de Computadores na Educação Básica: um estudo de caso.** RESIGeT. (2018). Vol 9. Disponível em: <<http://periodicos.unifacef.com.br/index.php/resiget/article/view/1604/1144>>.

GIRAFFA, Lucia Maria Martins;MORA, Michael da Costa. **EVASÃO NA DISCIPLINA DE ALGORITMO E PROGRAMAÇÃO: UM ESTUDO A PARTIR DOS FATORES INTERVENIENTES NA PERSPECTIVA DO ALUNO.** PUCRS. 2018 . Disponível em : <<https://revistas.utp.ac.pa/index.php/clabes/article/view/888/915>>.

PEREIRA, Leonardo. **Escolas defendem ensino de programação a crianças e adolescentes.** Olhar Digital. (2013). Disponível em: <<https://olhardigital.com.br/noticia/escolas-defendem-ensino-de-programacao-a-criancas-e-adolescentes/35075>>.

RIBEIRO,Juliana P. ; MANSO, Marina A. ; BORGES, Marcos A.F.. **Dinâmicas com App Inventor no Apoio ao Aprendizado e no Ensino de Programação.** UNICAMP. (2016). Disponível em: < <https://www.br-ie.org/pub/index.php/wie/article/view/6645>>

SANTOS, Siméia de Azevedo. **Educação para o futuro: uma abordagem sobre novas possibilidades de carreiras.** Brazilian Applied Science Review. (2018). Disponível em: <<https://www.brazilianjournals.com/index.php/BASR/article/view/645/543>>.

SILVA, Felipe Rodrigues da; LOPES, Vinícius Luiz. **DESENVOLVIMENTO DE JOGOS NA PLATAFORMA UNITY.** UNIFENAS. (2016). Disponível em: <<http://revistas.unifenas.br/index.php/RE3C/article/view/163>>.

SILVEIRA, Denise Tolfo.; GERHARDT, Tatiana Engel. **Métodos de Pesquisa.** (2019). Disponível em <<http://www.ufrgs.br/cursopgdr/downloadsSerie/derad005.pdf>>. Acesso em: 19 jun. 2020.

MUSSI, Celia Regina De Azevedo Ricotta. **AS TRANSFORMAÇÕES DIGITAIS E O MERCADO DE TRABALHO DO FUTURO.** REVISTA CIENTIFICA ICGAP. (2018). v. 1 n. 1. Disponível em: <<http://www.revistaicgap.com.br/index.php/icgap/article/view/5>>.

_____. **C++ Tutorial.** w3schools. Disponível em:
https://www.w3schools.com/cpp/cpp_arrays.asp.

OLIVEIRA, Rodrigo Diego de. **Análise do uso da cor no Diagrama de Classes da Linguagem Unificada de Modelagem (UML).** Revista Brasileira de Design da Informação. São Paulo. (2020). v. 17, n. 1, p. 116–130. Disponível em:
<https://infodesign.emnuvens.com.br/infodesign/article/view/783/469>.

RESENDE, Igor Henrique Correia. **Estudo para a Modelagem de um Sistema Moderno por meio da UML e extensões.** Universidade Federal de Uberlândia. (2019). Disponível em:
<http://repositorio.ufu.br/bitstream/123456789/28179/4/EstudoModelagemSistema.pdf>.

SOUZA, Márcia, FRANÇA, César. **O Sucesso dos Jogos para Ensino de Disciplinas de Engenharia de Software sob a Ótica de uma Teoria Motivacional.** UFRPE. Pernambuco. (2016). Disponível em:
<https://www.br-ie.org/pub/index.php/sbie/article/view/6726/4613>.

FERREIRA, Andressa B.; DE OLIVEIRA, Antônio R. M.; LIMA, Wiliana V.; **Uma Proposta de um Método para a Personalização do Ensino de Engenharia de Software.** Nuevas Ideas en Informática Educativa, Santiago de Chile. (2018). v 14, p. 539 - 544. Disponível em: <http://www.tise.cl/Volumen14/TISE2018/539.pdf>.

CASTRO, Hudson da Silva; BATALHA, Natane Rocha; PONTES, Tainon de Souza; FALCÃO, Manoel Ferreira. **O ENSINO DA LINGUAGEM DE PROGRAMAÇÃO DE COMPUTADORES C++ ATRAVÉS DE UMA OFICINA: UM RELATO DE EXPERIÊNCIA COM ESTUDANTES DO NONO ANO DE UMA ESCOLA ESTADUAL.** Revista de Extensão do IFAM. (2018). v.4, nº 2. Disponível em:
<http://nexus.ifam.edu.br/nexus/index.php/Nexus/article/view/290/139>.

SAADE,Joel. **C# Guia do Programador.** Novatec Editora. (2011). Disponível em:
<https://s3.novatec.com.br/capitulos/capitulo-9788575222539.pdf>.

SILVA, Luis Felipe; VÍNICIUS, Marcus; **SysGa – Sistema Controle de Games.** UNINOVE. (2010). Disponível em: <https://www.doccity.com/pt/sistema-de-controle-de-games/4731266/>.

ROMIO, Tiago; PAIVA, Simone Cristine Mendes; **Kahoot e GoConqr: uso de jogos educacionais para o ensino da matemática**. Scientia cum industria. (2017), v. 5, n. 2, p. 90-94. Disponível em: <<http://www.uces.br/etc/revistas/index.php/scientiacumindustria/article/view/5234/pdf#>>>.

MUSSI, Celia Regina De Azevedo Ricota. **Indústria 4.0 e a revolução digital**. Blasting News Brasil. (2016). Disponível em: <<https://br.blastingnews.com/economia/2016/12/industria-4-0-e-a-revolucao-digital-001309063.html>>.

PASSOS, Erick Baptista; JUNIOR, José Ricardo da Silva; RIBEIRO Fernando Emiliano Cardoso; MOURÃO, Pedro Thiago. **Tutorial: Desenvolvimento de Jogos com Unity 3D**. VIII Brazilian Symposium on Games and Digital Entertainment. Rio de Janeiro. (2009). Disponível em: <<http://www.sbgames.org/papers/sbgames09/computing/tutorialComputing2.pdf>>.

_____. Scripting. Unity Manual. (2019). Disponível em: <<https://docs.unity3d.com/Manual/ScriptingSection.html>>.

9 ANEXOS:

Figura 48 - Método Jump do Personagem.

```

51 void Jump()
52 {
53     if(Input.GetButtonDown("Jump") )
54     {
55         if(!pulo)
56         {
57             rig.AddForce(new Vector2(0f, JumpForce), ForceMode2D.Impulse);
58             duploPulo = true;
59             anim.SetBool("jump", true);
60         }
61         else
62         {
63             if(duploPulo)
64             {
65                 rig.AddForce(new Vector2(0f, JumpForce), ForceMode2D.Impulse);
66                 duploPulo = false;
67             }
68         }
69     }
70 }

```

Fonte: autoria própria, 2020.

Figura 49 – Script da Orange.

```

terminal  Help  Orange.cs - MyTcc - Visual Studio Code
Orange.cs x  Rock.cs  GameController.cs  Player.cs
Assets > Scripts > Orange.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Orange : MonoBehaviour
6  {
7      public static int num = 0;
8      // Start is called before the first frame update
9      void Start()
10     {
11         num = 0;
12     }
13
14     // Update is called once per frame
15     void Update()
16     {
17
18     }
19
20     void OnTriggerEnter2D(Collider2D collider)
21     {
22         if(collider.gameObject.tag == "Player")
23         {
24             Destroy(gameObject);
25             num = num + 1;
26         }
27     }
28 }
29

```

Fonte: autoria própria, 2020.

Figura 50 – Script CheckPoint.

```

Assets > Scripts > CheckPoint.cs
9      public string lvlnameatual;
10     public int nume;
11
12     void Update()
13     {
14         nume = Orange.num;
15     }
16
17
18     void OnCollisionEnter2D(Collision2D collision)
19     {
20         if(collision.gameObject.tag == "Player")
21         {
22             if(lvlnameatual == "tutorial" )
23             {
24                 if( nume == 6){
25                     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
26                 }
27             }
28             if(lvlnameatual == "lvl_1" )
29             {
30                 if( nume == 7){
31                     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
32                 }
33             }
34             if(lvlnameatual == "lvl_2" )
35             {
36                 if( nume == 9){
37                     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
38                 }
39             }
40             if(lvlnameatual == "lvl_3" )
41             {
42                 if( nume == 10){
43                     SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

```

Fonte: autoria própria, 2020.

Figura 51 – Script Fantasma.

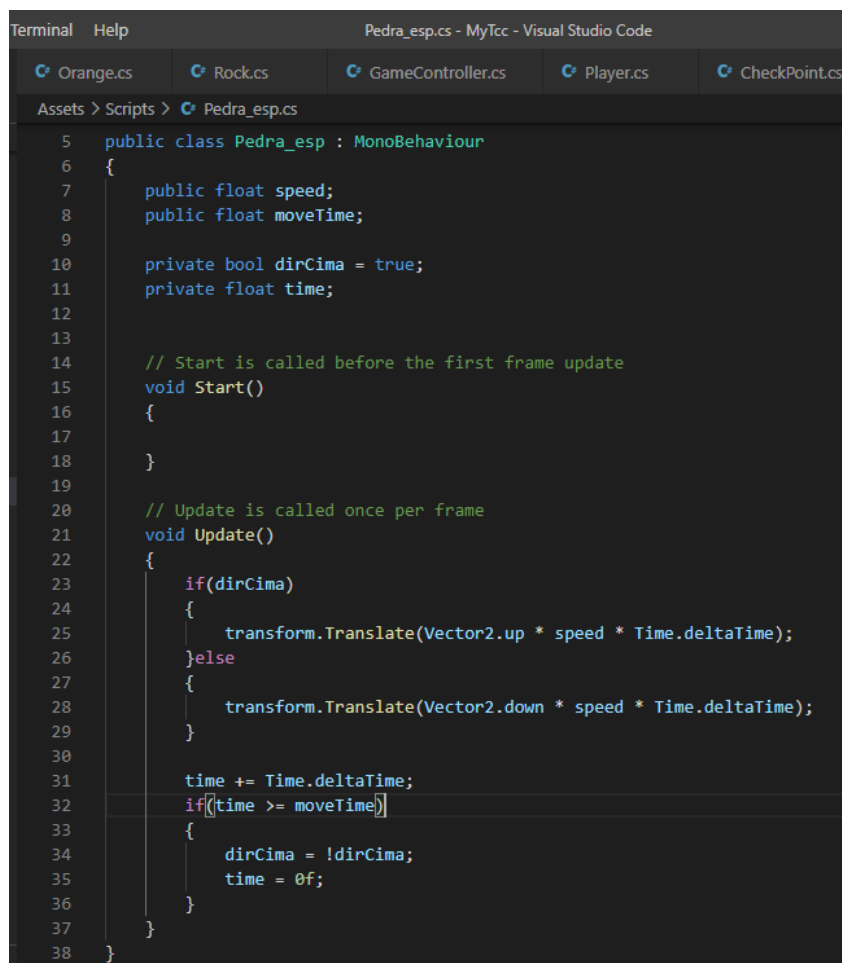
```

Terminal Help fantasma.cs - MyTcc - Visual Studio Code
Orange.cs Rock.cs GameController.cs Player.cs CheckPoint.cs fantasma.cs x Pedra_
Assets > Scripts > fantasma.cs
6      {
7          public float speed;
8          public float moveTime;
9
10         private bool dirRight;
11         private float time;
12
13
14         // Start is called before the first frame update
15         void Start()
16         {
17         }
18
19
20         // Update is called once per frame
21         void Update()
22         {
23             if(dirRight)
24             {
25                 transform.Translate(Vector2.right * speed * Time.deltaTime);
26             }else
27             {
28                 transform.Translate(Vector2.left * speed * Time.deltaTime);
29             }
30
31             time += Time.deltaTime;
32             if(time >= moveTime)
33             {
34                 dirRight = !dirRight;
35                 time = 0f;
36                 transform.localScale = new Vector2(transform.localScale.x * -1f, transform.localScale.y);
37             }
38         }
39     }

```

Fonte: autoria própria, 2020.

Figura 52 – Script da pedra espinhosa.



```

5  public class Pedra_esp : MonoBehaviour
6  {
7      public float speed;
8      public float moveTime;
9
10     private bool dirCima = true;
11     private float time;
12
13     // Start is called before the first frame update
14     void Start()
15     {
16
17     }
18
19     // Update is called once per frame
20     void Update()
21     {
22     {
23         if(dirCima)
24         {
25             transform.Translate(Vector2.up * speed * Time.deltaTime);
26         }else
27         {
28             transform.Translate(Vector2.down * speed * Time.deltaTime);
29         }
30
31         time += Time.deltaTime;
32         if(time >= moveTime)
33         {
34             dirCima = !dirCima;
35             time = 0f;
36         }
37     }
38 }

```

Fonte: autoria própria, 2020.

Figura 53 - Tutorial 5.

Manipular dados através de contas é uma coisa muito importante e como o DEV não conhecia nem mesmo as variáveis, então talvez seja bom aprender também operações matemáticas...

Exemplo:

```
#include <iostream>
using namespace std;
```

```
int main() {
```

```
    int soma ;
    soma = 5 + 10;
    return 0;
```

Next

Fonte: autoria própria, 2020.

Figura 54 - Tutorial 6.

DEV: “ ok, mas e como eu consigo fazer comparações? ”

Tutorial:

Condicionais:

Lógicas de matemática:

Menor que: $a < b$
Menor ou igual a: $a \leq b$
Maior que: $a > b$
Maior ou igual a: $a \geq b$
Igual a: $a == b$

Next

Fonte: autoria própria, 2020.

Figura 55 - Tutorial 7.

Instruções condicionais:

if(se) e else (se nao)

```
int y = 50;  
if (y = 50)  
{  
    cout << "Exatamente igual";  
}
```

Mostre ao DEV o funcionamento de uma condicional...

Next

Fonte: autoria própria, 2020.

Figura 56 - Tutorial 8.

Temos o For também, quando você sabe exatamente quantas vezes deseja percorrer um bloco de código, use o for loop em vez de um while loop.

```
for (Instrução 1; Instrução 2; Instrução 3) {  
  // código a ser executado  
}
```

A instrução 1 é executada (uma vez) antes da execução do bloco de código. A instrução 2 define a condição para executar o bloco de código. A instrução 3 é executada (sempre) após a execução do bloco de código.

Next

Fonte: autoria própria, 2020.

Figura 57 - Tutorial 9.

DEV : “ Mas agora estou me perdendo na hora de comparar as posições, sera que vc sabe um método de deixar organizado todos esses números?

Matrizes são usadas para armazenar vários valores em uma única variável, em vez de declarar variáveis separadas para cada valor.

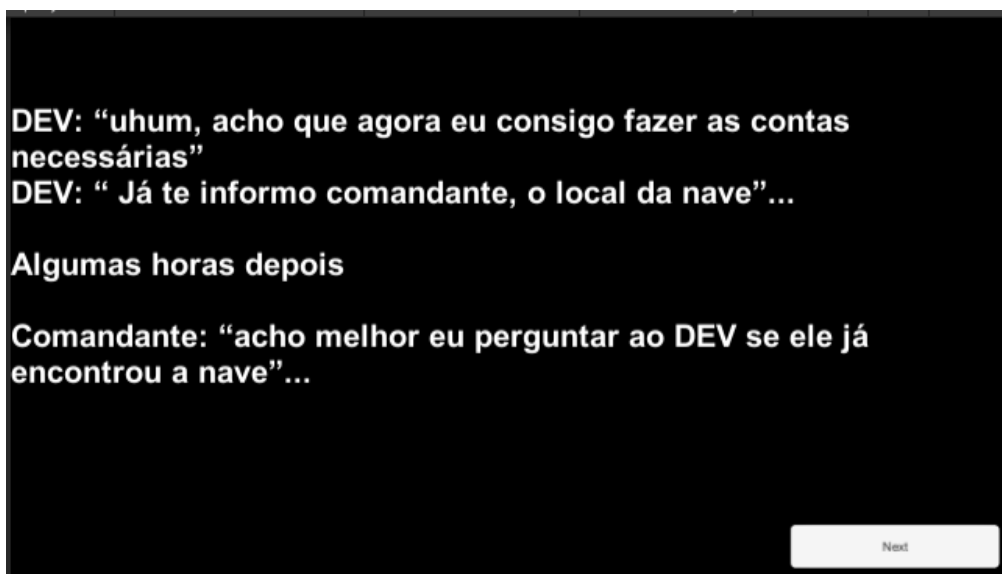
Exemplo:

```
string carros[4] = {"Volvo", "BMW", "Ford", "Mazda"};  
cout << carros[0];  
// ira imprimir Volvo
```

Next

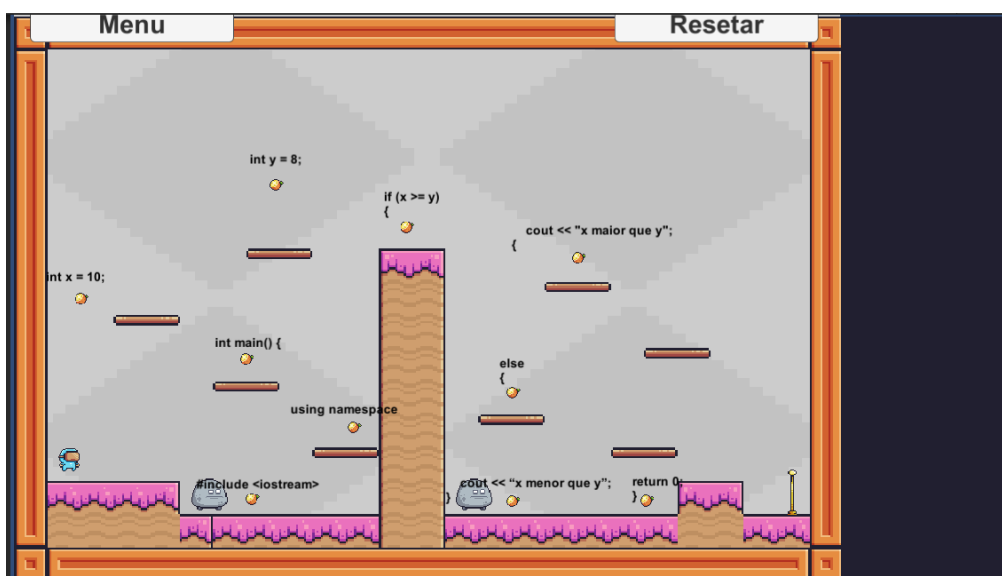
Fonte: autoria própria, 2020.

Figura 58 - Tutorial 10.



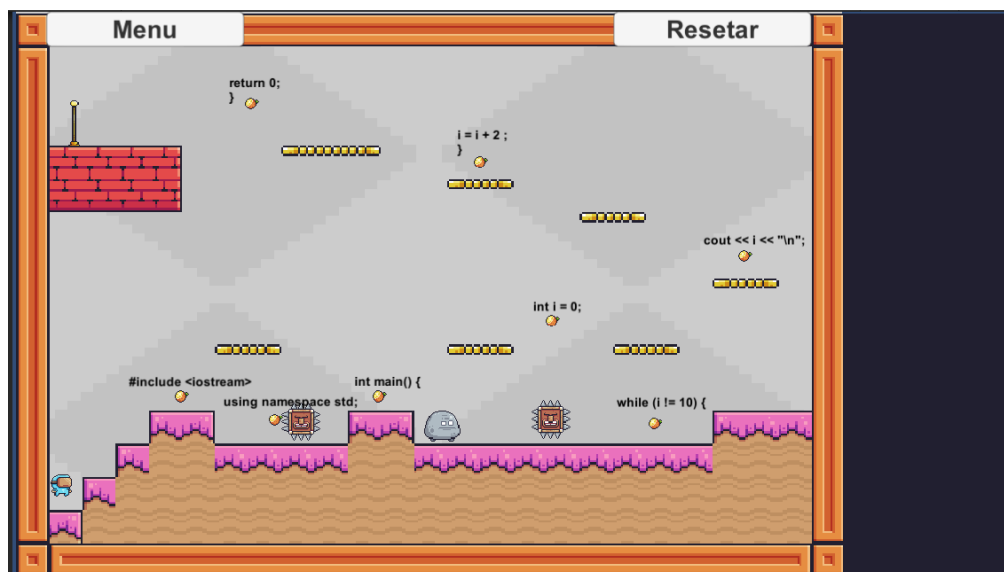
Fonte: autoria própria, 2020.

Figura 59 - Fase 3.



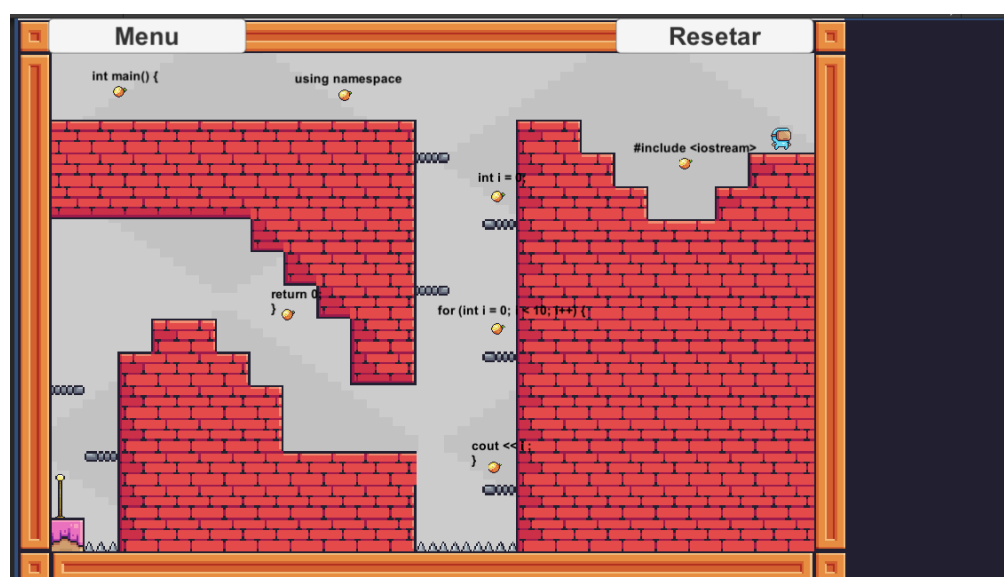
Fonte: autoria própria, 2020.

Figura 60 - Fase 4.



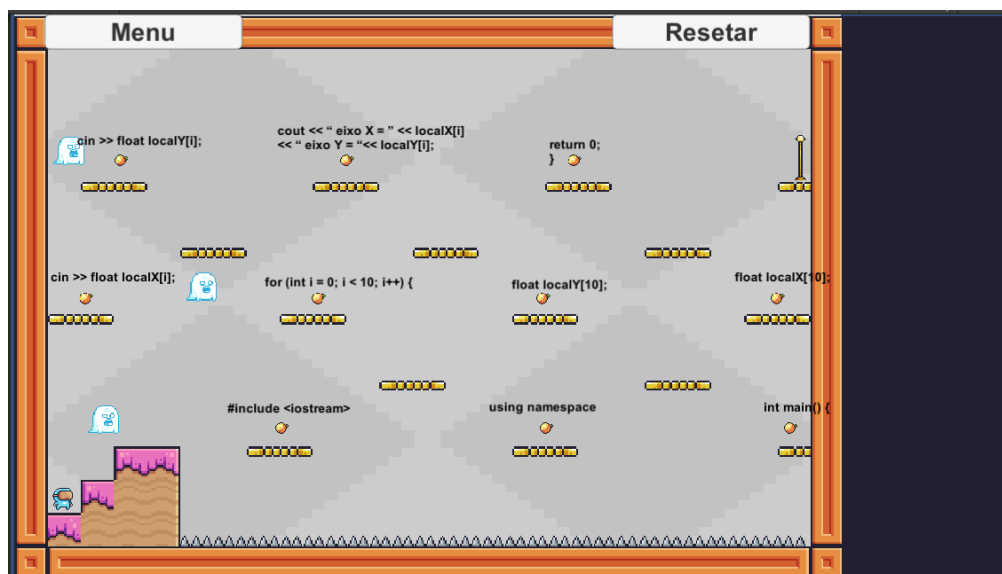
Fonte: autoria própria, 2020.

Figura 61 - Fase 5.



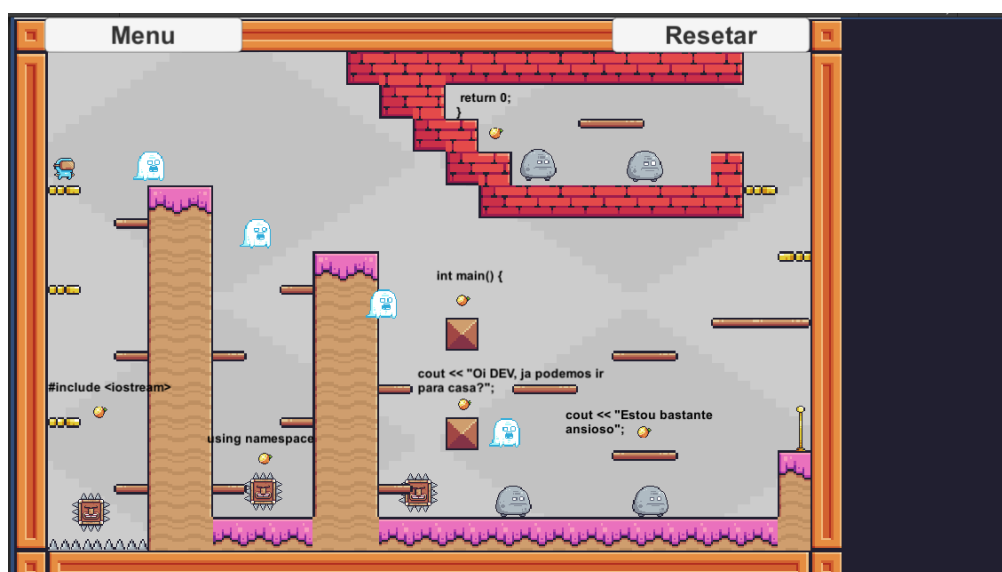
Fonte: autoria própria, 2020.

Figura 62 - Fase 6.



Fonte: autoria própria, 2020.

Figura 63 – Fase 7.



Fonte: autoria própria, 2020.