Curso Profissional de Técnico de Informática - Sistemas

Trojan.EXE

Prova da Aptidão Profissional

Gonçalo Silva

Oliveira de Azeméis, 18/06/2021

Agrupamento de Escolas Soares Basto

Curso Profissional de Técnico de Informática - Sistemas

Trojan.EXE

Prova da Aptidão Profissional

Professor Acompanhante: Manuel Teixeira

Gonçalo Silva

Oliveira de Azeméis, 18/06/2021

Dedicatória

Em primeiro lugar dedico este trabalho aos meus pais e aos meus irmãos e cunhado que me deram todo seu apoio, motivação e sempre me transmitiram força para ultrapassar cada etapa da minha vida. Dedico este trabalho também aos meus professores que me acompanharam ao longo dos três anos do curso e me transmitiram todos os conhecimentos e sabedoria para ultrapassar todas as minhas dificuldades. Em especial dedico, também, este trabalho aos professores Nuno Veloso e Manuel Teixeira que estiveram sempre disponíveis para ajudarem-me neste meu percurso e ajudaram-me a concluir o curso e o 12 º ano.

… Um grande obrigado a todos que contribuíram para a conclusão desta minha etapa!...

Agradecimentos

Agradeço à direção pedagógica e administrativa da Escola Básica e Secundária Soares Basto, funcionários e a todos os seus formadores a oportunidade que me deram, de poder complementar os meus conhecimentos na área de Técnico de Informática - Sistemas.

Aos meus formadores da área Sociocultural, Cientifica e Técnica. Em especial agradeço ao meu orientador da PAP: Exmo. Manuel Teixeira, e ao meu monitor de estágio Carlos Girão que me ajudaram na realização da minha Prova de Aptidão Profissional. Agradeço a dedicação e empenho demonstrado, por parte de todos, para que eu adquirisse com sucesso as competências necessárias para ser um bom Técnico de Informática – Sistemas.

De seguida agradeço, mais uma vez, ao meu orientador de estágio: Carlos Girão, pela disponibilidade e orientação realizada, em formação em contexto de trabalho, que demonstrou ao longo do mesmo e pela pessoa que foi com o seu profissionalismo demonstrado nas 600 horas de estágio que tive ao longo do curso.

Por fim, agradeço às pessoas que ao longo dos três anos de curso me apoiaram e partilharam comigo os bons e maus momentos e me ajudaram a concluir este percurso da minha formação pessoal, que irá ficar sempre marcado para o resto da minha vida…

Índice

DEDICATÓRIA **3**
AGRADECIMENTOS **4**
ÍNDICE DE ILUSTRAÇÕES **6**
RESUMO **9**
INTRODUÇÃO **10**
PLANO DE DESENVOLVIMENTO **11**
MATERIAIS/RECURSOS NECESSÁRIOS **11**
DESIGN GRÁFICO DO PROJETO **12**
TIPO DE LETRA UTILIZADO **12**
PALETA DE CORES **12**
LOGO E NOME DO PROJETO **13**
BASE DE DADOS **14**
MOCKUPS E ESTRUTURA DO PROJETO **15**
DESENVOLVIMENTO DO PROJETO **18**
Criação da scene MainMenu *18*
Criação da scene Leader Board *20*
Criação da scene Options *22*
Criação da scene Register *25*
Criação da scene Login *29*
Criação da scene Level Selection *34*
Desenvolvimento do jogo *37*
Desenvolvimento do Website *52*
CONCLUSÃO **53**
BIBLIOGRAFIA **54**

Índice de ilustrações

Figura 1 - Linguagem de programação PHP 11
Figura 2 - Compilador Visual Studio Code 11
Figura 3 - Plataforma Unity 11
Figura 4 - Linguagem de programação C# 11
Figura 5 - MySql Workbench 11
Figura 6 - Estilo de letra Carter One 12
Figura 7 - Paleta de cores utilizada 12
Figura 8 - Logo do projeto 13
Figura 9 - Base de Dados 14
Figura 10 – Ecrã de login 15
Figura 11 – Ecrã de registo 15
Figura 12 – Menu Principal 16
Figura 13 – Menu de Opções 16
Figura 14 – Menu de seleção de mapas 17
Figura 15 – Ecrã de jogo 17
Figura 16 - Scene MainMenu 18
Figura 17 - método QuitGame 18
Figura 18 - método Options 19
Figura 19 - método PlayGame 19
Figura 20 - método LeaderBoard 19
Figura 21 - Scene LeaderBoard 20
Figura 22 - código SQL para as melhores pontuações 20
Figura 23 - Objeto para guardar as informações 21
Figura 24 - DTO para guardar as informações 21
Figura 25 - Conversão para uma lista do DTO_UsersScores 21
Figura 26 - foreach para apresentar a informação 21
Figura 27 - Scene Options 22
Figura 28 - método Back 22
Figura 29 - método SetFullscreen 23
Figura 30 - Código para receber as todas resoluções do ecrã 23
Figura 31 - Código para implementar as resoluções na combo box 23
Figura 32 - Código para comparar as resoluções com a resolução atual 24
Figura 33 - Código para implementar a resolução utilizada na combo box como valor predefinido 24
Figura 34 - método SetResolution 24
Figura 35 - Scene Register 25
Figura 36 - código para a criação do WWWForm que vai servir para enviar os dados para o ficheiro php 25
Figura 37 - código para a criação de um WWW que irá servir para fazer a ligação com o localhost e enviar os dados para o ficheiro php 25
Figura 38 - variável $con com código para estabelecer a ligação á base de dados 26
Figura 39 - código para efetuar a verificação da ligação á base de dados 26
Figura 40 - código para a criação das variáveis username e password 26

Figura 41 - variável $namecheckquery com a instrução sql para selecionar os utilizadores com o nome de utilizador igual á variável $username 26
Figura 42 - variável $namecheck com a instrução sql para estabelecer ligação á base de dados 26
Figura 43 - "if" para a verificação de algum utilizador existente na base de dados com o nome igual 27
Figura 44 - variáveis para encriptar a palavra-passe 27
Figura 45 - variável para guardar uma instrução sql de inserção de dados do jogador na base de dados 27
Figura 46 - instrução sql para inserir os dados na base de dados 27
Figura 47 - "if" para a verificação do valor retornado do código php 28
Figura 48 - Método VerifyInputs 28
Figura 49 - Scene Login 29
Figura 50 - variável para guardar uma instrução sql de seleção de dados 29
Figura 51 - variável para guardar uma isntrução sql para fazer ligação a bd e enviar a instrução da variável $namecheckquery 30
Figura 52 - "if" para verificar se há algum utilizador na bd com o username igual ao username passado no login 30
Figura 53 - variáveis para guardar uma instrução sql e para encriptação da palavra-passe 30
Figura 54 - variável para guardar a palavra-passe encriptada 30
Figura 55 - "if" para a verificação da palavra-passe e código para retornar o valor 0 e as "coins" 31
Figura 56 - código SQL para ir buscar a pontuação do jogador nos vários mapas 31
Figura 57 - código PHP para verificar se algum pedido retorna nulo 31
Figura 58 - código SQL para verificar se o utilizador tem posse do primeiro mapa 32
Figura 59 - código SQL para inserir o primeiro mapa na tabela "MapaObtidos" 32
Figura 60 - código PHP para retorno das informações 32
Figura 61 - "if" para a validação do login e para guardar as informações em variáveis 33
Figura 62 - variáveis onde vão ser guardadas as informações 33
Figura 63 - Scene LevelSelection 34
Figura 64 - código SQL para verificar o preço dos mapas 34
Figura 65 - código SQL para verificar os mapas obtidos do utilizador 35
Figura 66 - Pop up com o preço do mapa 35
Figura 67 - código SQL para chamar o procedimento 36
Figura 68 - método Jump 37
Figura 69 - enum dos estados do personagem 37
Figura 70 - switch no método Update 38
Figura 71 - método Awake 38
Figura 72 - código C# para chamar o método "Jump" 39
Figura 73 - Instanciar o cano 40
Figura 74 - Constante PIPE_WIDTH 40
Figura 75 - Constante PIPE_HEAD_HEIGHT 40
Figura 76 - Constante CAMERA_ORTHO_SIZE 40
Figura 77 - "if" para cálculo 41
Figura 78 - "if" para cálculo 41

Figura 79 - Colocar o "pipeHead" na posição "xPosition" 41
Figura 80 - Procedimento para o corpo do cano 41
Figura 81 - Renderizar os obstáculos 42
Figura 82 - Adicionar a uma lista 42
Figura 83 - código para chamar o método "CreatePipe" 42
Figura 84 - método HandlePipeSpwaning 42
Figura 85 - parâmetros necessários para chamar o método 43
Figura 86 - Criar altura aleatória 43
Figura 87 - método Update 43
Figura 88 - ciclo do método HandlePipeMovement 44
Figura 89 - variável para saber se o personagem está a direita do cano 44
Figura 90 - método Move 44
Figura 91 - "if" para contabilizar os obstáculos ultrapassados 45
Figura 92 - código para chamar o método DestroySelf 45
Figura 93 - método DestroySelf 45
Figura 94 - código para armazenar os assests escolhidos 46
Figura 95 - indicar os assests no Unity 46
Figura 96 - Imagem do jogo em execução 46
Figura 97 - criação do objeto "ScoreWindow" no Unity 47
Figura 98 - método Awake 47
Figura 99 - método Update 47
Figura 100 - "if" para apresentar a melhor pontuação 48
Figura 101 - Imagem do jogo em execução 48
Figura 102 - "if" para carregar o mapa selecionado 49
Figura 103 - requisição para verificar a melhor pontuação 49
Figura 104 - código SQL para selecionar a melhor pontuação 50
Figura 105 - Inserção da pontuação 50
Figura 106 - Verificação da pontuação e inserção da mesma 50
Figura 107 - requisição para buscar a melhor pontuação 50
Figura 108 - código SQL para buscar a melhor pontuação do utilizador 51
Figura 109 - requisição para adicionar "coins" 51
Figura 110 - Buscar as "coins" do utilizador 51
Figura 111 - Inserção das "coins" 51
Figura 112 - Website de apoio 52

Resumo

O Trojan.EXE é um videojogo para computador em 2D concebido para que qualquer dispositivo consiga executar e jogar o mesmo, o videojogo é em 2D, onde o utilizador tem que controlar o seu personagem para passar entre vários obstáculos e tentar efetuar a pontuação máxima em diversos mapas que estão disponíveis no videojogo. O videojogo também tem um sistema de compra, esse sistema serve para que o utilizador jogue para ganhar dinheiro virtual e em seguida consiga comprar vários mapas. O videojogo tem um sistema de pontuação, esse sistema de pontuação serve para contabilizar o número de vezes que o utilizador conseguiu passar por entre os obstáculos, a pontuação diferencia de mapa para mapa, ou seja, cada mapa terá a sua própria pontuação. O videojogo também adapta-se a várias resoluções de ecrã, ou seja, é responsivo, dentro do jogo também podemos ativar ou desativar o modo de ecrã inteiro e também podemos diminuir ou aumentar o volume e mudar a resolução. O Trojan.EXE também tem um menu de login, onde o utilizador irá introduzir as suas informações para dar entrada no videojogo e ir buscar os seus dados, caso o utilizador não tenha conta ele poderá efetuar um registo.

Introdução

No âmbito da Prova de Aptidão Profissional (PAP) do Curso Técnico de Informática – Sistemas foi desenvolvido o presente projeto que consiste no desenvolvimento videojogo 2D intitulado “Trojan.EXE”.

A escolha do tema relacionou-se essencialmente com dois aspetos. O primeiro está relacionado com o facto de uma pesquisa de mercado ter revelado não existirem muitos videojogos do tipo do “Trojan.EXE”, pois este projeto é uma combinação de vários videojogos, o primeiro e mais visível é o “FlappyBird”, o segundo “Hill Climb Racing”, o terceiro “Jetpack Joyride” e o quarto “Super Mario World”. E o segundo aspeto com o facto da minha vontade e curiosidade de desenvolver um videojogo, pois desde cedo comecei a jogar vários videojogos e os videojogos em que inspirei-me fizeram parte da minha infância e adolescência.

O “Trojan.EXE” tem como objetivo ser um videojogo 2D onde pus em prática os meus conhecimentos adquiridos no curso de técnico de informática, tais como a programação e a criação de uma base de dados, na realização deste projeto foi necessário aprender uma nova linguagem de programação, essa linguagem foi C# e também foi necessário aprender a utilizar a plataforma Unity, onde foi desenvolvido o videojogo.

Plano de desenvolvimento

Para a realização deste projeto tive que realizar o planeamento do jogo, definição do estilo de arte, plano de produção, level design e as mecânicas. Depois disso terei de realizar a base de dados e seguida terei de realizar o logótipo, criação do background para os menus e a criação do personagem e dos obstáculos. Depois disso poderei avançar para a programação do videojogo, irei realizar o projeto na plataforma Unity na linguagem C#. Depois do jogo estar acabado e polido irei efetuar a realização do site de apoio ao Trojan.EXE.

Materiais/Recursos necessários

Utilização do compilador Visual Studio Code para a criação do código em C#, utilização da plataforma e motor de jogo Unity para efetuar a realização do videojogo, a utilização da linguagem de programação C# para efetuar o código do videojogo e a utilização do MySQL Workbench para a criação da base de dados para o videojogo.

Figura 5 - MySql Workbench

Figura 4 - Linguagem de programação C#

Figura 3 - Plataforma Unity

Figura 1 - Linguagem de programação PHP

Figura 2 - Compilador Visual Studio Code

Design gráfico do projeto

Tipo de letra utilizado

Carter One

Figura 6 - Estilo de letra Carter One

Paleta de cores

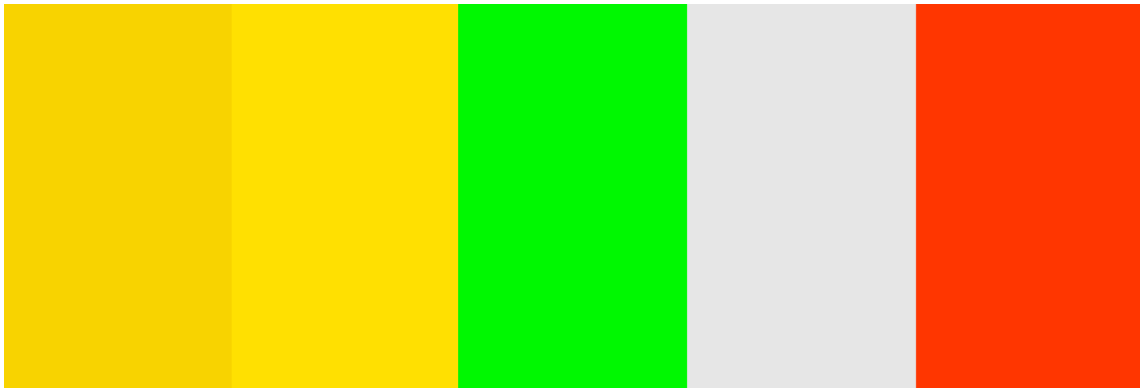

Figura 7 - Paleta de cores utilizada

Logo e nome do projeto

O projeto ficou com o nome "Trojan.EXE" pois queria algum nome relacionado com a parte informática, então decidi optar pelo nome Trojan. O Trojan é um tipo programa malicioso que podem entrar em um computador disfarçado como um programa comum e legítimo.

O nome surgiu devido à história da guerra de Troia e que culminou com a destruição desta. O cavalo de Troia, um grande cavalo de madeira, fora supostamente oferecido como um pedido de paz por parte dos gregos. Sendo um presente para o rei, os troianos levaram o cavalo para dentro das muralhas da cidade.

Durante a noite, quando todos dormiam, este revelou-se uma armadilha e os soldados gregos que se escondiam dentro da estrutura oca de madeira do cavalo saíram e abriram os portões para que todo o exército entrasse e queimasse a cidade.

E como o nome do projeto tinha ficado Trojan pensei nalguma coisa relacionado com um cavalo por causa do cavalo de Troia então depois de muita pesquisa encontrei um artista que faz bastantes trabalhos com desenhos e consegui encontrar um cavalo com assas. Depois bastou-me juntar o nome do projeto ao desenho e criei o logotipo do projeto, como se pode ver na figura abaixo.

Figura 8 - Logo do projeto

Base de dados

Para o projeto Trojan.EXE foi necessário criar uma base de dados composta com 4 tabelas. A primeira tabela é a tabela **"Jogador"**, esta tabela irá servir para que o utilizador possa registar-se no jogo e tenha os seus dados guardados, esta tabela irá armazenar o nome de utilizador, a palavra-passe encriptada e a quantidade de coins que o jogador terá (as coins são o dinheiro virtual do jogo para que o jogador consiga fazer compras de mapas ou fases dentro do Trojan.EXE). A tabela **"Mapas"** irá servir para armazenar os mapas que irão ser implementados dentro do videojogo, esta tabela terá o id, nome do mapa e o seu preço. A tabela **"Score"** irá servir para armazenar a pontuação do jogador nos diferentes mapas, nesta tabela chegará os dados do mapa em que o jogador esteve a jogar, os dados do jogador em questão e a pontuação do mesmo. A última tabela **"MapasObtidos"** servirá para armazenar os mapas comprados pelo utilizador, armazenando assim o id do utilizador e o id do mapa.

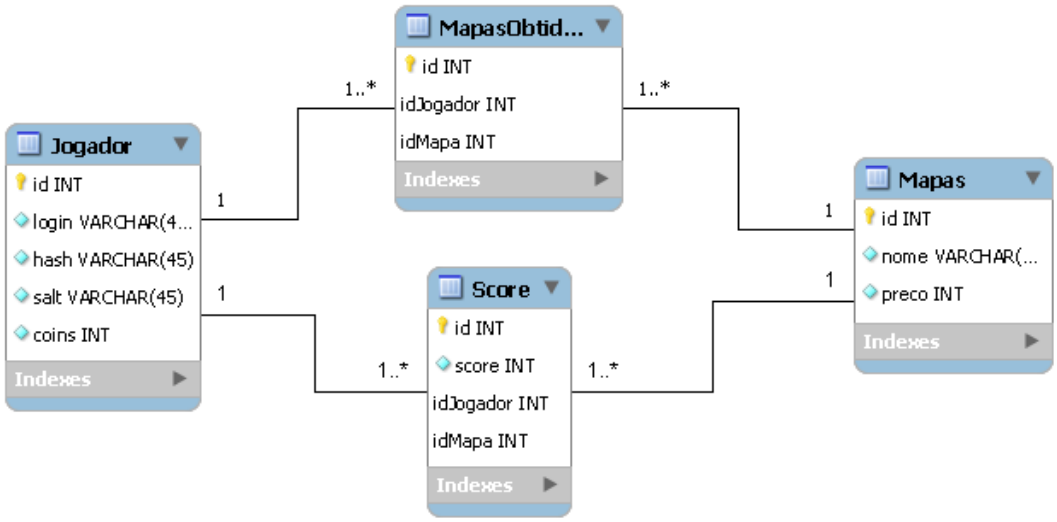

Figura 9 - Base de Dados

Mockups e estrutura do projeto

Ecrã de login

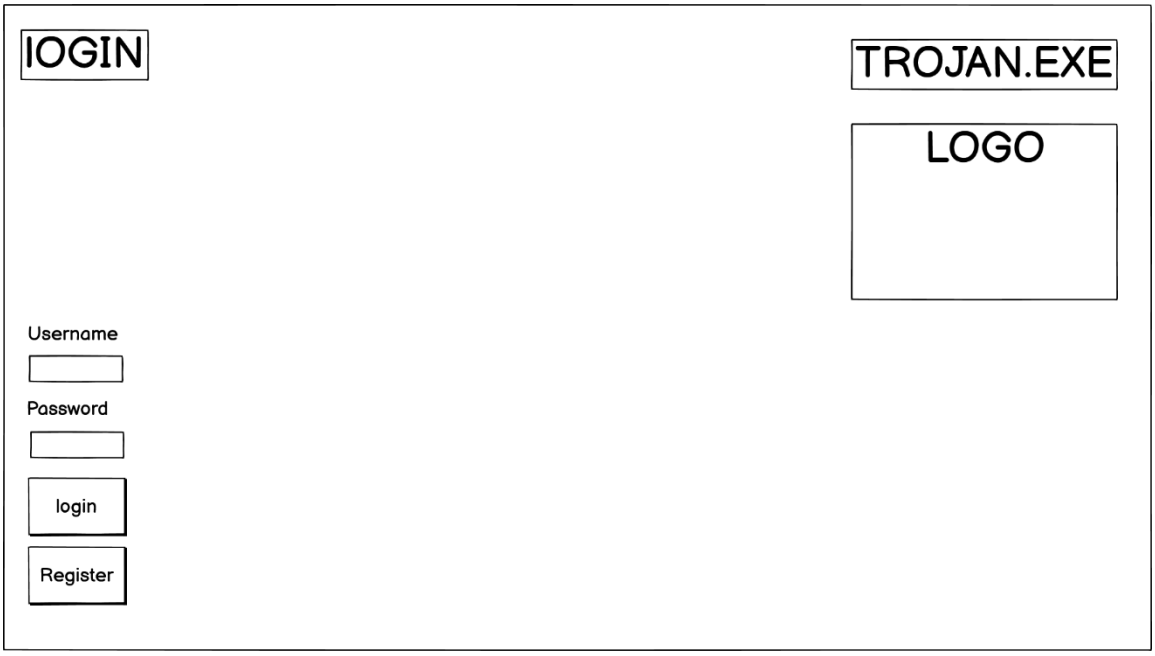

Figura 10 – Ecrã de login

Ecrã de registo

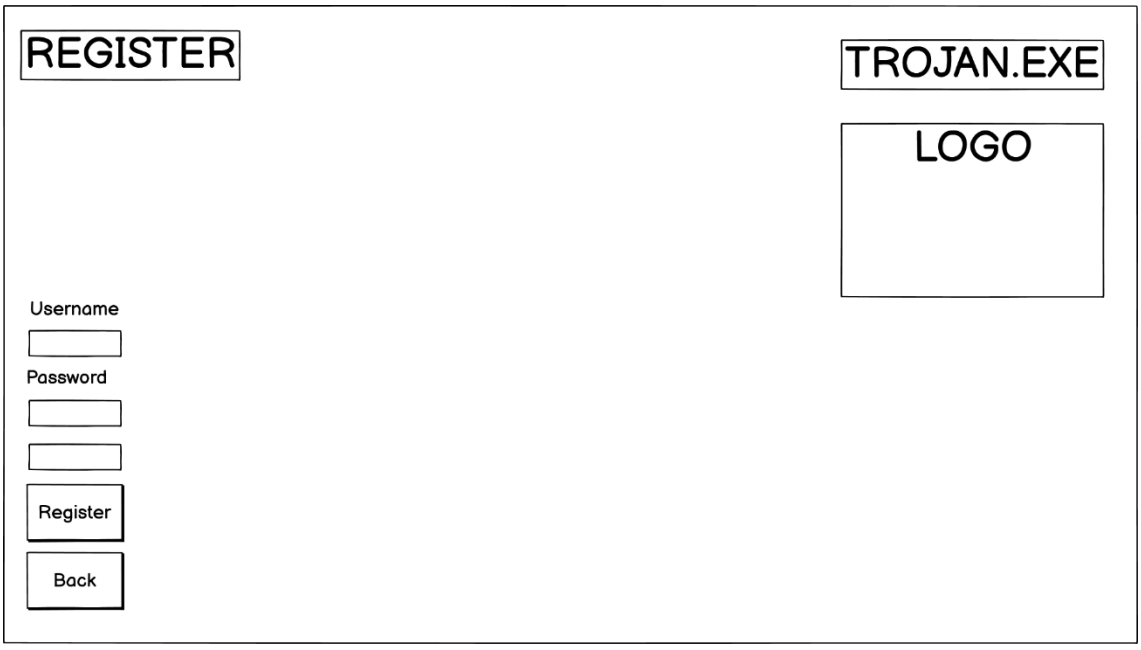

Figura 11 – Ecrã de registo

Menu principal

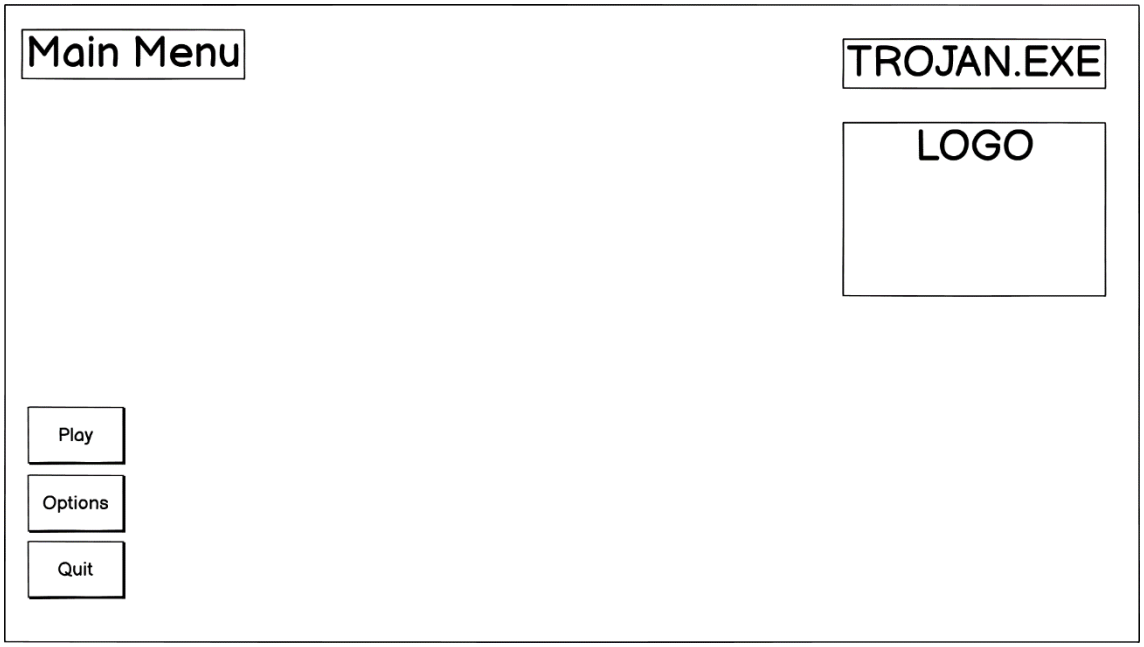

Figura 12 – Menu Principal

Menu de opções

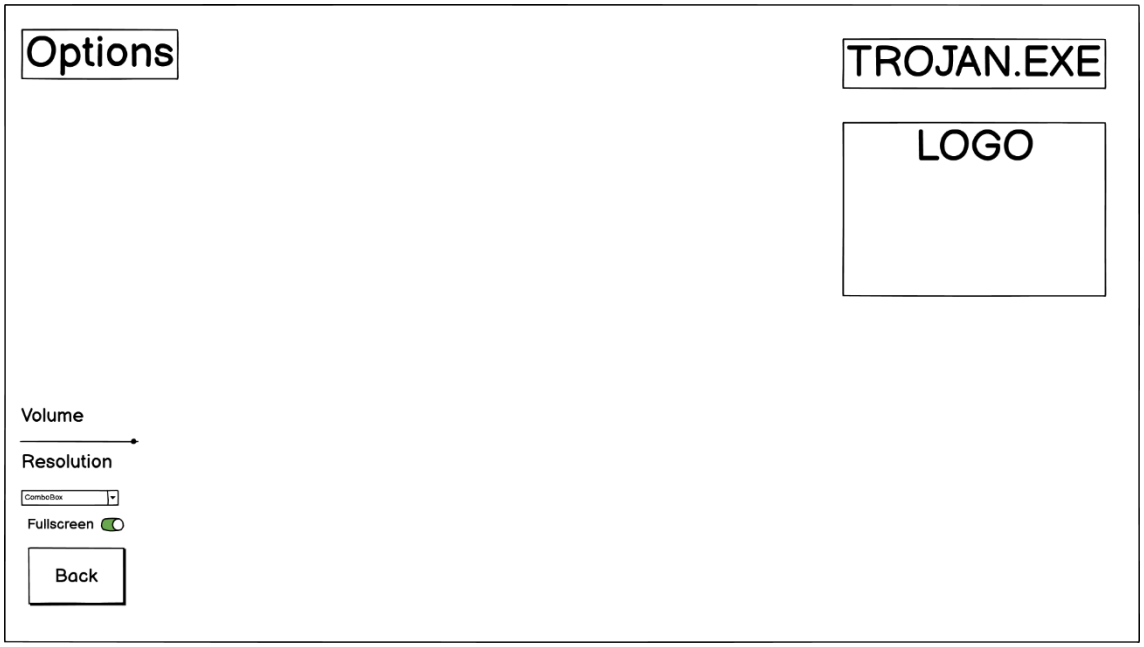

Figura 13 – Menu de Opções

Menu de seleção de mapas

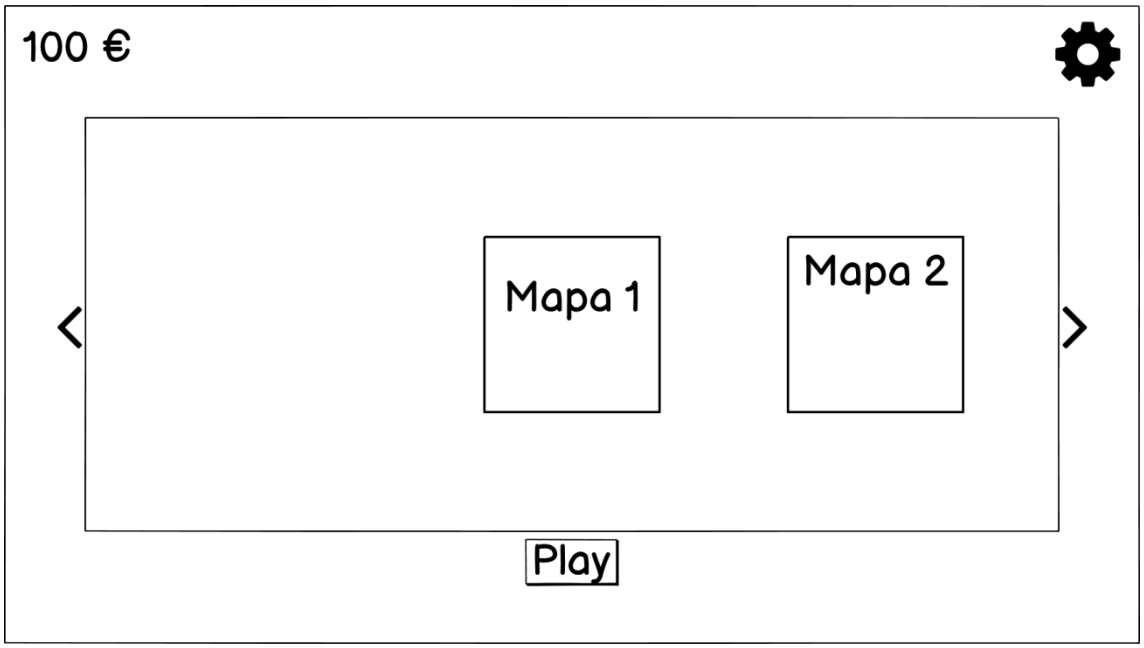

Figura 14 – Menu de seleção de mapas

Ecrã de jogo

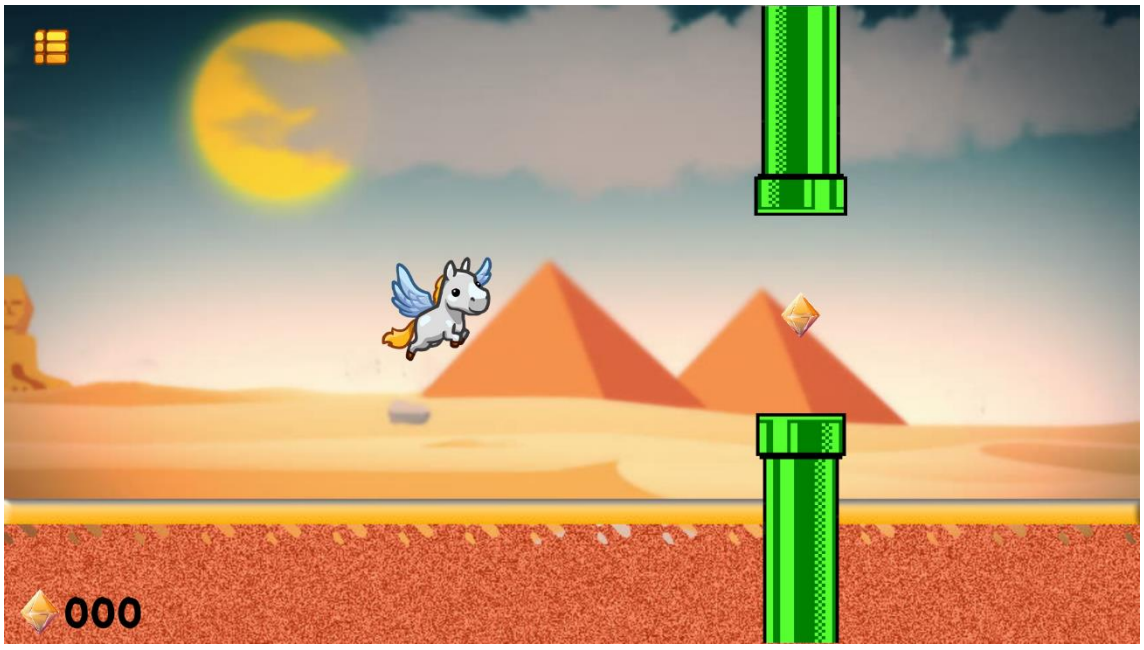

Figura 15 – Ecrã de jogo

Desenvolvimento do projeto

Criação da scene MainMenu

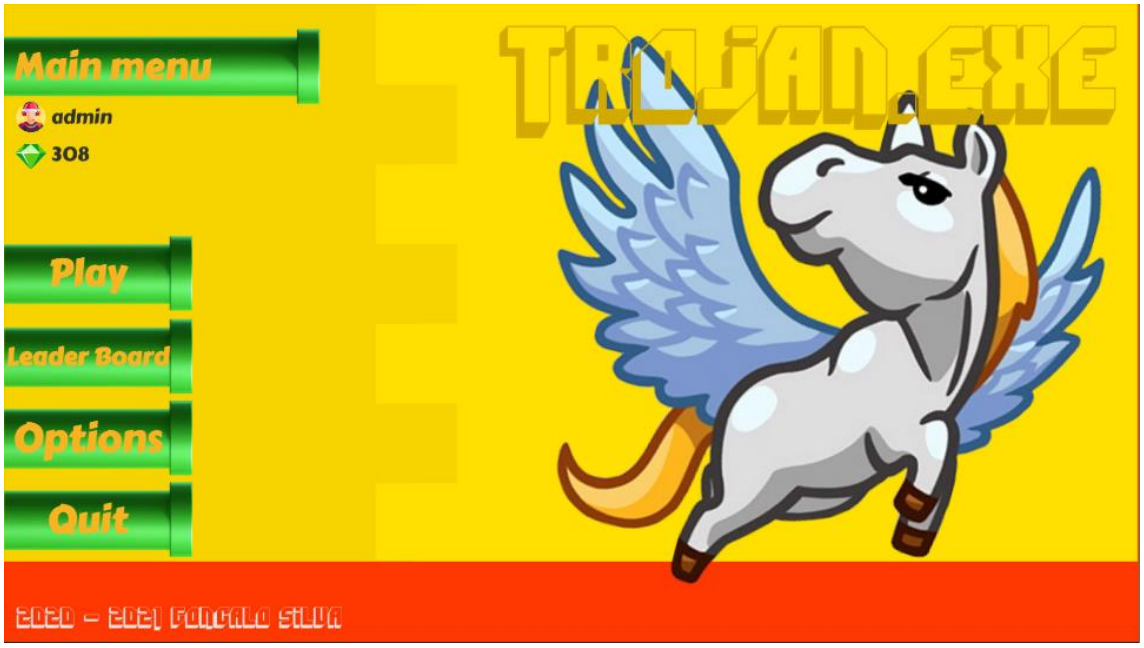

Figura 16 - Scene MainMenu

O desenvolvimento desta PAP começou pela criação de um novo projeto na plataforma "Unity", de seguida pela criação de uma nova "scene" (uma "scene" é como se tratasse de uma cena de teatro e cada "scene" tem as suas propriedades ou irá fazer alguma coisa de diferente que as outras não fazem e que o utilizador consegue interagir), a essa "scene" foi atribuído o nome de "MainMenu", esta "scene" vai funcionar como o menu principal do jogo. Na criação do menu principal foi introduzido um "background" com o logo e o nome do projeto e foram criados vários botões como o botão de "Play", "Options" e "Quit", cada botão está associado a um ficheiro de código C# "Menu", neste ficheiro de código estão todos os métodos necessários para o funcionamento dos botões:

O método "QuitGame" está associado ao botão "Quit", o método faz com que o utilizador consiga sair e fechar o videojogo.

```
public void QuitGame()
{

    Debug.Log("Quit");
    Application.Quit();

}
```

Figura 17 - método QuitGame

O método “Options” está associado ao botão “Options”, o método faz com que o utilizador consiga ir para o menu de opções.

```
public void Options()
{

    SceneManager.LoadScene(1);

}
```

Figura 18 - método Options

O método “PlayGame” está associado ao botão “Play”, o método faz com que utilizador consiga ir para o menu de selecção de mapas.

```
public void PlayGame()
{

    SceneManager.LoadScene(2);

}
```

Figura 19 - método PlayGame

O método “LeaderBoard” está associado ao botão “Leader Board”, o método faz com que o utilizador consiga ir para o visionamento da tabela de melhores pontuações.

```
public void LeaderBoard()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
}
```

Figura 20 - método LeaderBoard

Criação da scene Leader Board

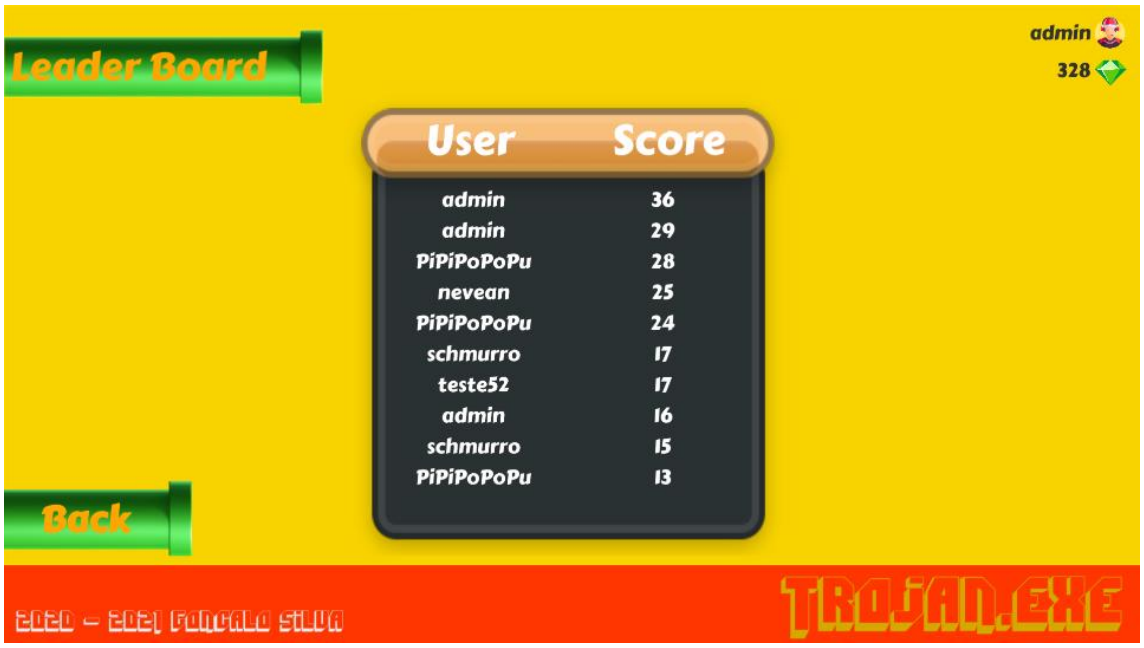

Figura 21 - Scene LeaderBoard

De seguida efetuei a realização da "scene" da visualização da tabela das melhores pontuações do jogo, esta "scene" foi designada como "Leader Board" e irá servir para o utilizador visualizar as melhores pontuações, isto irá servir para trazer algum tipo de competitividade ao jogo. Primeiramente foi criado um "background" próprio para o "Leader Board". Depois da criação do "background", foi criado um "Panel", onde o "background" irá ficar anexado. De seguida foi criado os campos de textos onde irá ser apresentada a informação e o botão para a opção "Back".

Para fazer com que a informação seja apresentada nos campos de texto, foi necessário fazer um pedido á base de dados. Para isso foi necessário criar um ficheiro php onde foi feito um pedido sql para buscar as dez melhores pontuações do jogo em todos os mapas.

```
$query = "SELECT j.login as login, s.score as score
          FROM Score s
          join Jogador j on s.idJogador = j.id
          order by s.score desc
          LIMIT 10";
```

Figura 22 - código SQL para as melhores pontuações

De seguida os dados foram guardados num objeto.

```
while($row = mysqli_fetch_assoc($resultsArray)) {
    array_push($result, (object)[
        'login' => $row["login"],
        'score' => $row["score"]
    ]);
}
```

Figura 23 - Objeto para guardar as informações

De seguida depois de os dados estarem guardados num objeto e estarem a ser retornados, foi necessário criar um "DTO", ou seja, um "script" onde irá ser armazenado os dados do objeto.

```
2 references
public class DTO_UsersScores
{
    1 reference
    public string login { get; set; }
    1 reference
    public int score { get; set; }
}
```

Figura 24 - DTO para guardar as informações

Neste caso irão ser guardados o "login" e o "score" do utilizador.

De seguida no "script Leader Board" foi necessário converter o resultado do php para uma lista do "DTO" anteriormente criado.

```
List<DTO_UsersScores> usersScores = JsonConvert.DeserializeObject<List<DTO_UsersScores>>(www.text);
```

Figura 25 - Conversão para uma lista do DTO_UsersScores

Depois bastou dizer que a informação de "login" ia para o campo de texto do nome de utilizador na tabela e a informação de "score" ia para o campo de texto da pontuação.

```
foreach (var item in usersScores)
{
    leaderBoardUserDisplay.text += item.login + "\n";
    leaderBoardScoreDisplay.text += item.score + "\n";
}
```

Figura 26 - foreach para apresentar a informação

Criação da scene Options

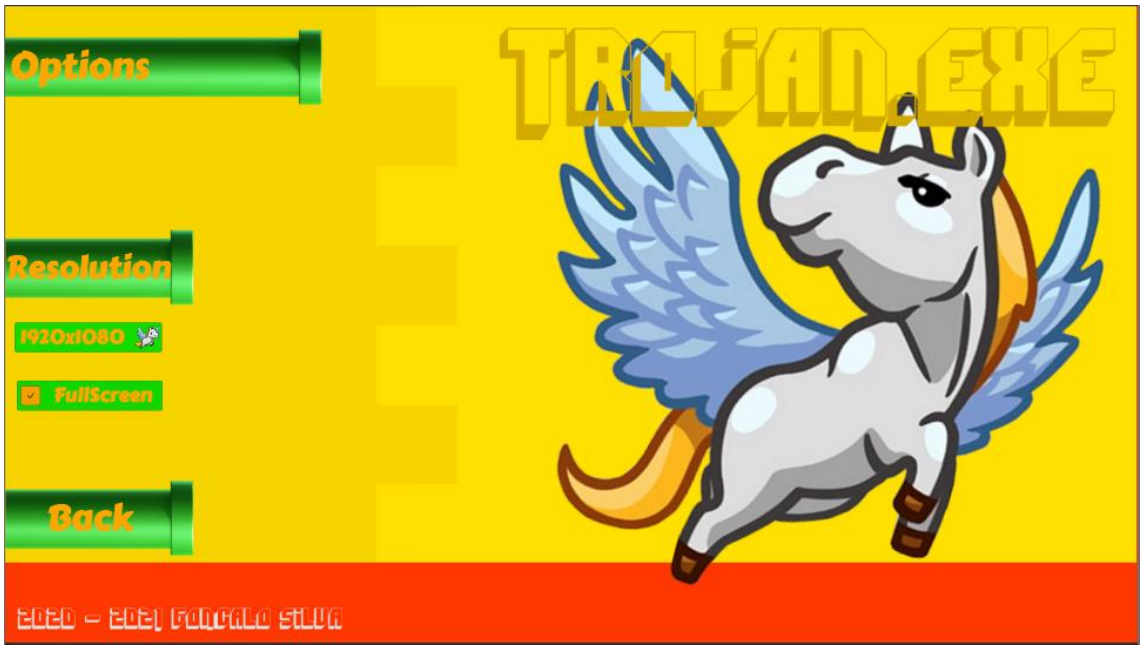

Figura 27 - Scene Options

De seguida efetuei a realização da "scene" do menu de opções, esta "scene" foi designada como "Options" e irá servir para o utilizador controlar o volume do jogo, a resolução e o modo de ecrã inteiro. Primeiramente foi criado um "background" próprio para o menu de opções. Depois da criação do "background", foi criado um "Panel", onde o "background" irá ficar anexado. De seguida foi criado os campos de textos, o botão para a opção "Back", um "slider" para o volume, uma "combo box" para a alteração da resolução e uma "check box" para ativar ou desativar o modo de ecrã inteiro, de seguida foi necessário fazer a programação dos respetivos itens mencionados anteriormente.

Para que o botão "Back" funcionasse corretamente foi necessário criar um ficheiro de código C# com a designação de "OptionsMenu", neste ficheiro de código foi adicionado o método "Back".

O método "Back" está associado ao botão "Back", este método serve para quando o utilizador pressionar o botão "Back" o jogo carregue a "scene" do menu principal, para isso o código irá receber o número da "scene" atual, ou seja, da "scene" "Options" e irá subtrair por 1, ou seja, o código irá fazer o cálculo 1-1 que será 0, e este 0 é o número da "scene" do menu principal.

```
public void Back()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
}
```

Figura 28 - método Back

Na “scene” “OptionsMenu” foi efetuado a criação da opção do modo de ecrã inteiro, para que a “check box” do modo de ecrã inteiro funcionasse corretamente foi necessário criar um método no ficheiro de código C# “OptionsMenu”, o método foi designado como “SetFullscreen”.

O método “SetFullscren” associado a “check box” anteriormente mencionada serve para ativar ou desativar o modo de ecrã inteiro do jogo, para isso foi necessário passar uma variável “isFullscreen” do tipo “bool”, esta variável irá passar o valor da “check box”, se ela estiver com valor “true” o modo de ecrã inteiro irá ser ativado, se o valor for “false”, o modo de ecrã inteiro irá ficar desativado.

```
public void SetFullscreen(bool isFullscreen)
{
    Screen.fullScreen = isFullscreen;
}
```

Figura 29 - método SetFullscreen

Para que a “combo box” dos diferentes tipos de resolução funcionasse corretamente foi necessário criar dois métodos no ficheiro de código C# “OptionsMenu”, o método “Start” que irá ser chamado logo que a “scene” do menu de opções seja carregada e irá servir para o programa conseguir identificar todas as resoluções possíveis, para isso foi necessário criar uma variável global do tipo “array”, na variável global irão ser armazenadas todas as resoluções possíveis.

```
resolutions = Screen.resolutions;
```

Figura 30 - Código para receber as todas resoluções do ecrã

De seguida foi necessário transformar o “array” numa “List” de “strings”, para isso foi criado uma “List” de “strings” designada options, de seguida foi necessário converter cada resolução adquirida no “array” numa “string” designada “option”, foi necessário usar um “for” para converter todas as resoluções do “array”,depois disso é necessário colocar as várias resoluções convertidas em “string” na “List” criada anteriormente.

```
List<string> options = new  List<string>();

for(int i=0; i < resolutions.Length; i++)
{
    string option = resolutions[i].width + "x" + resolutions[i].height;
    options.Add(option);

}
```

Figura 31 - Código para implementar as resoluções na combo box

De seguida para que a "combo box" apresente sempre o valor da resolução atual, foi necessário criar uma variável onde vamos armazenar a resolução que está a ser utilizada no momento, de seguida foi necessário criar um "if" dentro do "for" anteriormente criado, para comparar a resolução que está a ser adicionada na "List" com a resolução atual do ecrã do dispositivo, se a resolução for a correta ela será armazenada na variável anteriormente criada e implementada na "combo box" como o valor predefinido.

```
if(resolutions[i].width == Screen.currentResolution.width
&& resolutions[i].height == Screen.currentResolution.height)
{
    currentResolutionIndex = i;
}
```

Figura 32 - Código para comparar as resoluções com a resolução atual

```
resolutionsDropdown.value = currentResolutionIndex;
```

Figura 33 - Código para implementar a resolução utilizada na combo box como valor predefinido

Para alterar a resolução foi necessário criar um novo método "void" designado como "SetResolution" que irá ter como parâmetro uma variável do tipo inteiro. Dentro do método criado foi necessário criar uma nova variável do tipo "Resolution" onde vai ser igual ao parâmetro do método, de seguida foi necessário fazer um "SetResolution" com a largura e o comprimento da resolução atual e com a "Screen.fullScreen" que irá ter o valor "true" ou "false" dependendo da "check box" se estiver selecionada ou não.

```
public void SetResolution(int resolutionIndex)
{

    Resolution resolution = resolutions[resolutionIndex];
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);

}
```

Figura 34 - método SetResolution

Criação da scene Register

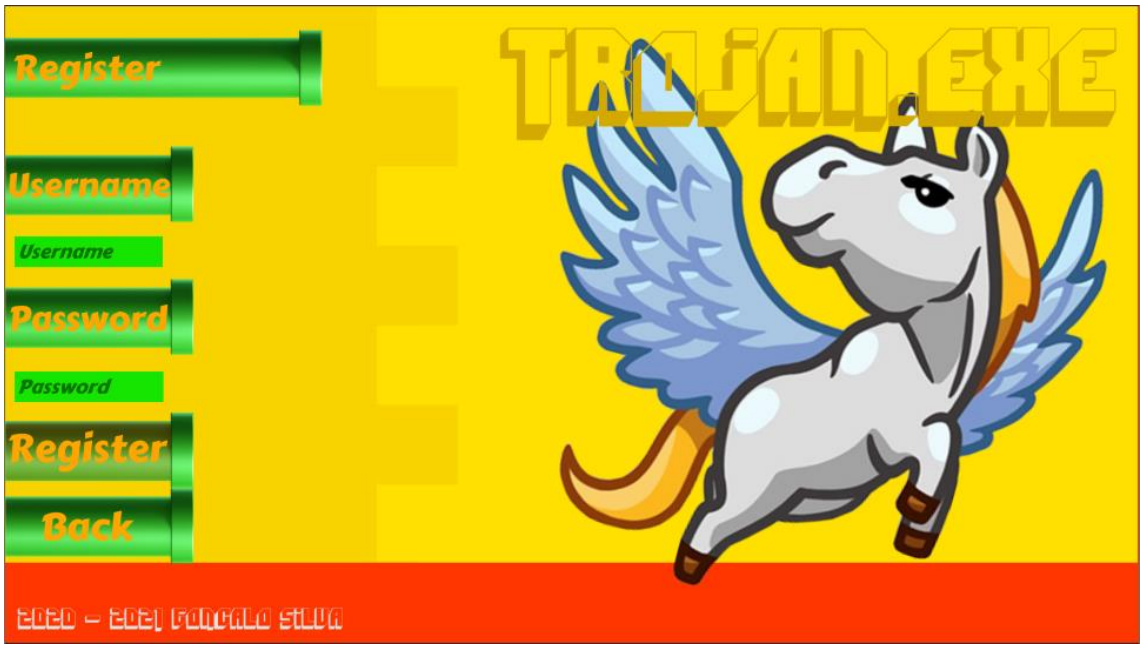

Figura 35 - Scene Register

Depois da criação das "scenes" "Options" e "MainMenu", foi necessário criar duas novas "scenes", uma seria para a efetuação dos registos dos jogadores na base de dados e a outra seria para efetuar o login dos jogadores. Comecei pela criação da "scene" "Register" onde o utilizador poderá efetuar o seu registo no jogo, guardando o seu nome de utilizador e a sua palavra-passe. A "scene" necessitou de dois "fields inputs", ou seja, campos para que o utilizador conseguisse escrever o seu nome de utilizador e palavra-passe. De seguida foi criado um ficheiro c#, esse ficheiro serviu para estabelecer ligação com um ficheiro php nomeado como "register.php", o ficheiro c# vai enviar a informação obtida nos dois "field inputs" e enviar para o ficheiro php.

Para enviar os dados obtidos nos "fields inputs" foi necessário criar um "WWWForm" que irá servir para enviar os dados obtidos, para um endereço web que neste caso foi utilizado o "localhost" onde se encontram os ficheiros php.

```
WWWForm form = new WWWForm();
form.AddField("name", nameField.text);
form.AddField("password", passwordField.text);
```

Figura 36 - código para a criação do WWWForm que vai servir para enviar os dados para o ficheiro php

```
WWW www = new WWW("http://localhost/Trojan/register.php", form);
yield return www;
```

Figura 37 - código para a criação de um WWW que irá servir para fazer a ligação com o localhost e enviar os dados para o ficheiro php

Depois de estabelecer ligação ao ficheiro php, o mesmo vai estabelecer ligação á base de dados, se a ligação não for concretizada o código devolve 1 e sai.

```
$con = mysqli_connect('localhost', 'root', '', 'trojan');
```

Figura 38 - variável $con com código para estabelecer a ligação á base de dados

```
if(mysqli_connect_errno())
{

    echo "1"; //Erro 1 = conexão falhada
    exit();

}
```

Figura 39 - código para efetuar a verificação da ligação á base de dados

Caso o código prossiga, serão criadas duas variáveis, “username” que vai guardar o nome de utilizador e a varável “password” que vai guardar a palavra-passe.

```
$username = $_POST["name"];
$password = $_POST["password"];
```

Figura 40 - código para a criação das variáveis username e password

Depois vai ser criada uma nova variável “namecheckquery” que vai guardar uma instrução sql para selecionar o nome de utilizador que foi passado pelo utilizador.

```
$namecheckquery = "SELECT login FROM jogador where login = '".$username. "';";
```

Figura 41 - variável $namecheckquery com a instrução sql para selecionar os utilizadores com o nome de utilizador igual á variável $username

De seguida é criada outra variável “namecheck” que vai estabelecer a ligação á base de dados e enviar a instrução da variável “namecheckquery”.

```
$namecheck = mysqli_query($con, $namecheckquery) or die("2: Name check query failed");
```

Figura 42 - variável $namecheck com a instrução sql para estabelecer ligação á base de dados

Depois foi utilizado um “if” para verificar se há algum utilizador na base de dados com aquele nome de utilizador passado anteriormente.

```
if(mysqli_num_rows($namecheck)>0)
{

    echo "3: Name already exits";
    exit();
}
```

Figura 43 - "if" para a verificação de algum utilizador existente na base de dados com o nome igual

De seguida se a condição do “if” for falsa serão criadas duas variáveis, $salt e $hash, que irão servir para encriptar a palavra-passe.

```
$salt = "\$5\$rounds=5000\$" . "steamedhams" . $username . "\$";
$hash = crypt($password, $salt);
```

Figura 44 - variáveis para encriptar a palavra-passe

Depois será criada outra variável, $insertuserquery, que vai guardar uma instrução sql para inserir os dados do jogador na base de dados, depois foi feito um mysqli_query para inserir os dados na base de dados, se a operação falhar irá surgir um erro com a mensagem “4: Falha ao inserir o utilizador”, caso a operação seja bem sucedida o código retornará 0.

```
$insertuserquery = "INSERT INTO jogador (login, hash, salt, coins)
VALUES ('".$username."', '".$hash."', '".$salt."', 0);";
```

Figura 45 - variável para guardar uma instrução sql de inserção de dados do jogador na base de dados

```
mysqli_query($con, $insertuserquery)
or die("4: Falha ao inserir o utilizador");
```

Figura 46 - instrução sql para inserir os dados na base de dados

Se o código retornar 0 o jogo irá para a "scene" de login que será criada depois, caso o valor retornado seja diferente de 0 irá ser apresentada uma mensagem com o erro retornado.

```
if (wwww.text == "0")
{

    Debug.Log("Utilizador criado com sucesso...");
    UnityEngine.SceneManagement.SceneManager.LoadScene(0);

}
else
{
    Debug.Log("Falha ao criar utilizador... Erro #" + wwww.text);
}
```

Figura 47 - "if" para a verificação do valor retornado do código php

No ficheiro c# "Registration", foi também criado um método "VerifyInputs" que vai servir para que o utilizador quando esteja a criar a sua conta tenha pelo menos um nome de utilizador e uma palavra-passe com 5 caracteres.

```
public void VerifyInputs()
{

    submitButton.interactable = (nameField.text.Length >= 5
    && passwordField.text.Length >= 5);

}
```

Figura 48 - Método VerifyInputs

Criação da scene Login

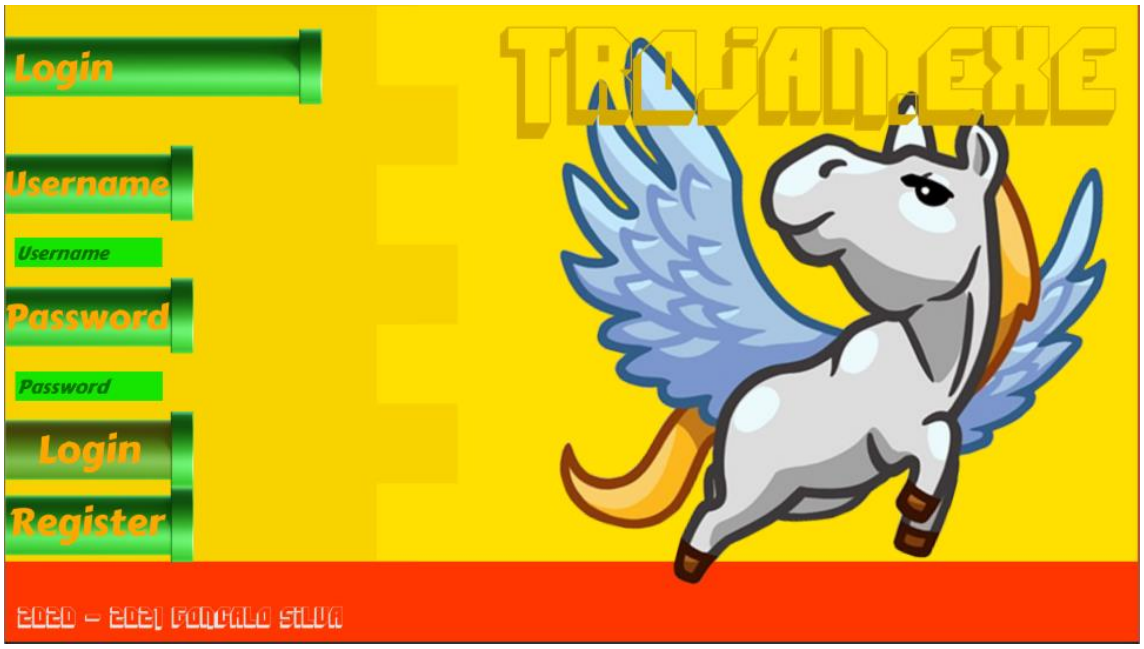

Figura 49 - Scene Login

Depois da criação da "scene" "Register", foi criada a "scene" "Login" para que o utilizador possa efetuar o login e entrar no jogo. A "scene" necessitou de dois "fields inputs", ou seja, campos para que o utilizador conseguisse escrever o seu nome de utilizador e palavra-passe. De seguida foi criado um ficheiro c#, esse ficheiro serviu para estabelecer ligação com um ficheiro php nomeado como "login.php", o ficheiro c# vai enviar a informação obtida nos dois "field inputs" e enviar para o ficheiro php.

Os ficheiros c# e php da "secene" "Login" são bastante semelhantes, a ligação aos ficheiros php é idêntica, mudando apenas o nome do ficheiro, já no ficheiro php a ligação á base de dados é idêntica, mas neste ficheiro há uma variável, "$namecheckquery", que irá guardar uma instrução sql para selecionar o "login" (nome de utilizador), o "salt" e o "hash" (palavra-passe encriptada), e as coins do jogador, onde o nome de utilizador é igual ao nome de utilizador passado pelo utilizador no login.

```
$namecheckquery = "SELECT login, salt, hash, coins
FROM jogador where login = '".$username. "';";
```

Figura 50 - variável para guardar uma instrução sql de seleção de dados

De seguida é criada outra variável “namecheck” que vai estabelecer a ligação á base de dados e enviar a instrução da variável “namecheckquery”.

```
$namecheck = mysqli_query($con, $namecheckquery) or die("2: Name check query failed");
```

Figura 51 - variável para guardar uma isntrução sql para fazer ligação a bd e enviar a instrução da variável $namecheckquery

Depois foi utilizado um “if” para verificar se há algum utilizador na base de dados com aquele nome de utilizador passado anteriormente, se não existir será retornada uma mensagem “5: No user with name, or more than one” e a execução do código acaba.

```
if(mysqli_num_rows($namecheck) != 1)
{
    echo "5: No user with name, or more than one";
    exit();
}
```

Figura 52 - "if" para verificar se há algum utilizador na bd com o username igual ao username passado no login

Caso haja somente um utilizador com aquele nome serão criadas três variáveis, “$logininfo” que irá guardar um comando sql que irá servir para buscar os valores “salt” e “hash” á base de dados, depois foram criadas as duas variáveis restantes, “$salt” e “$hash” que irão guardar os valores do “salt” e do “hash” da base de dados.

```
$logininfo = mysqli_fetch_assoc($namecheck);
$salt = $logininfo["salt"];
$hash = $logininfo["hash"];
```

Figura 53 - variáveis para guardar uma instrução sql e para encriptação da palavra-passe

Depois foi criada outra variável “$loginhash” que irá servir para encriptar a palavra-passe inserida pelo utilizador no login com o “salt”.

```
$loginhash = crypt($password, $salt);
```

Figura 54 - variável para guardar a palavra-passe encriptada

Depois a partir de um “if” foi verificado se o “$loginhash” é diferente do “$hash”, caso sejam diferentes será retornado uma mensagem “6: Incorrect password”.

```php
if($hash != $loginhash)
{
    echo "6: Incorrect password";
    exit();
}
```

Figura 55 - "if" para a verificação da palavra-passe e código para retornar o valor 0 e as "coins"

De seguida serão feitos três pedidos á base dados para ir buscar a pontuação do jogador nos três mapas do jogo.

```php
$scorefirstmapquery = "SELECT score
FROM Score
where idJogador = '".$logininfo["id"]. "' AND idMapa = '1';";

$scoresecondmapquery = "SELECT score
FROM Score
where idJogador = '".$logininfo["id"]. "' AND idMapa = '2';";

$scorethirdmapquery = "SELECT score
FROM Score
where idJogador = '".$logininfo["id"]. "' AND idMapa = '3';";
```

Figura 56 - código SQL para ir buscar a pontuação do jogador nos vários mapas

Caso algum pedido não retorne algum valor foi necessário retornar o valor zero nesse mapa.

```php
if(mysqli_num_rows($scorefirstmap) != 1)
{
 $scorefirstmapinfo["score"] = 0;
}

if(mysqli_num_rows($scoresecondmap) != 1)
{
 $scoresecondmapinfo["score"] = 0;
}

if(mysqli_num_rows($scorethirdmap) != 1)
{
 $scorethirdmapinfo["score"] = 0;
}
```

Figura 57 - código PHP para verificar se algum pedido retorna nulo

De seguida será feito outro pedido á base de dados para verificar se o utiliazador já tem obtido primeiro mapa, o mapa que vem de predefinição.

```
$mapownedcheckquery = "SELECT id
FROM MapasObtidos
where idJogador = '".$logininfo["id"]. "'
AND idMapa = '1';";
```

Figura 58 - código SQL para verificar se o utilizador tem posse do primeiro mapa

Caso não tenha será inserido o mapa na tabela de “MapasObtidos”.

```
if(mysqli_num_rows($mapownedcheck) == 0)
{

 $insertDefaultMap = "INSERT INTO MapasObtidos (idJogador, idMapa)
 VALUES ('".$logininfo["id"]. "', '1');";

 mysqli_query($con, $insertDefaultMap)
 or die("Falha ao inserir mapa");

}
```

Figura 59 - código SQL para inserir o primeiro mapa na tabela "MapaObtidos"

De seguida será retornado as informações das “coins”, do “id”, e das pontuações do utilizador.

```
echo "0\t" . $logininfo["coins"]. "\t". $logininfo["id"]. "\t".
$scorefirstmapinfo["score"]. "\t". $scoresecondmapinfo["score"]. "\t". $scorethirdmapinfo["score"];
```

Figura 60 - código PHP para retorno das informações

Depois da execução do código php terminar, o valor que for retornado para o ficheiro c# irá ser utilizado num "if", caso o valor retornado seja 0 irá ser carregada a "scene" do menu principal e o "username", as "coins", o "id" e as pontuações do ficheiro c# "DBManager" irão ficar a armazernar o montante de dinheiro e o nome de utilizador, este ficheiro c# "DBManager" irá ter esse propósito que será armazenar informações da base de dados para que não seja preciso estar sempre a estabelecer ligação com a mesma. E caso a condição seja falsa irá ser apresentada um mensagem de erro.

```
if (www.text[0] == '0')
{
    DBManager.username = nameField.text;
    DBManager.coins = int.Parse(www.text.Split('\t')[1]);
    DBManager.id = int.Parse(www.text.Split('\t')[2]);
    DBManager.firstMap = int.Parse(www.text.Split('\t')[3]);
    DBManager.secondMap = int.Parse(www.text.Split('\t')[4]);
    DBManager.thirdMap = int.Parse(www.text.Split('\t')[5]);
    SceneManager.LoadScene(2);
}
else
{
    Debug.Log("Erro no login. Erro #" + www.text);
}
```

Figura 61 - "if" para a validação do login e para guardar as informações em variáveis

```
public static class DBManager
{
    public static string username;
    public static int coins;
    public static int id;

    public static int firstMap;
    public static int secondMap;
    public static int thirdMap;
```

Figura 62 - variáveis onde vão ser guardadas as informações

Criação da scene Level Selection

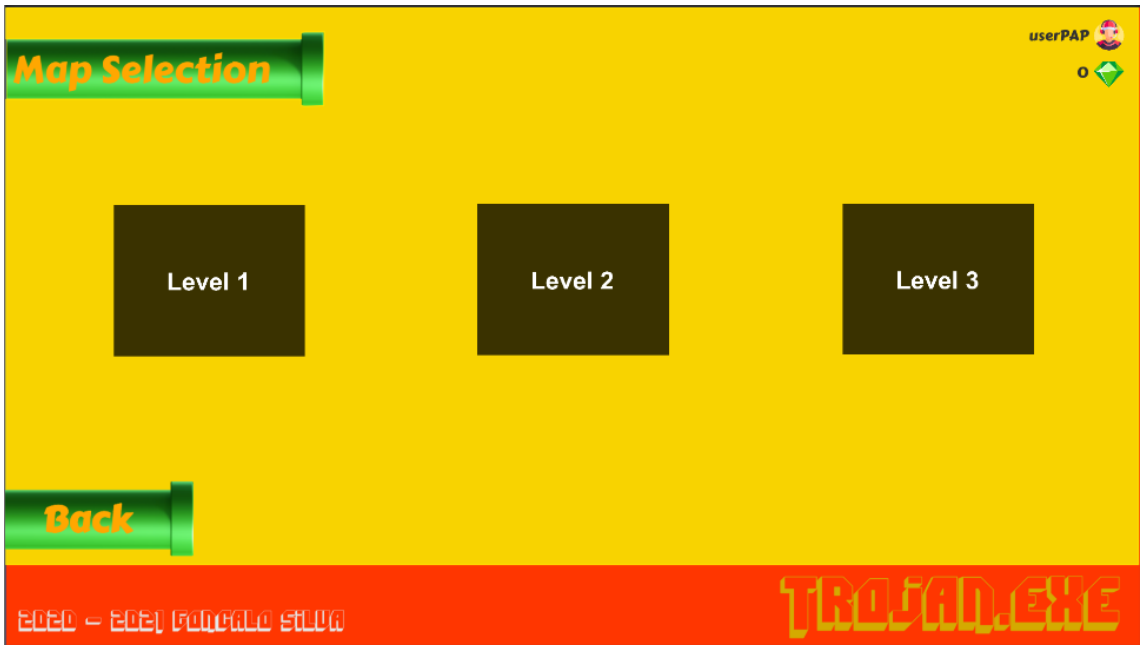

Figura 63 - Scene LevelSelection

A "scene" "LevelSelection" foi desenvolvida para que o utilizador possa escolher o mapa em que vai jogar, caso o tenha, se ainda não tiver obtido o mapa aparecerá uma mensagem com o valor do mapa e se o quer comprar.

Para desenvolver o "Level Selection" foi necessário criar um "script" onde será feito um pedido á base de dados para verificar quais são os mapas já obtidos pelo jogador e os seus preços.

```
$secondMapPriceQuery = "SELECT preco
FROM Mapas
where id = '2';";

$thirdMapPriceQuery = "SELECT preco
FROM Mapas
where id = '3';";
```

Figura 64 - código SQL para verificar o preço dos mapas

```
$firstMapQuery = "SELECT id
FROM MapasObtidos
where idJogador = '".$idUser. "'
and idMapa = '1';";

$secondMapQuery = "SELECT id
FROM MapasObtidos
where idJogador = '".$idUser. "'
and idMapa = '2';";

$thirdMapQuery = "SELECT id
FROM MapasObtidos
where idJogador = '".$idUser. "'
and idMapa = '3';";
```

Figura 65 - código SQL para verificar os mapas obtidos do utilizador

Depois de saber os mapas que o utilizador tem e os preços dos mapas foi necessário realizar um "if" onde caso o utilizador já tenha obtido, o mapa será carregado, caso não tenha aparecerá um "pop up" com o valor do mapa.

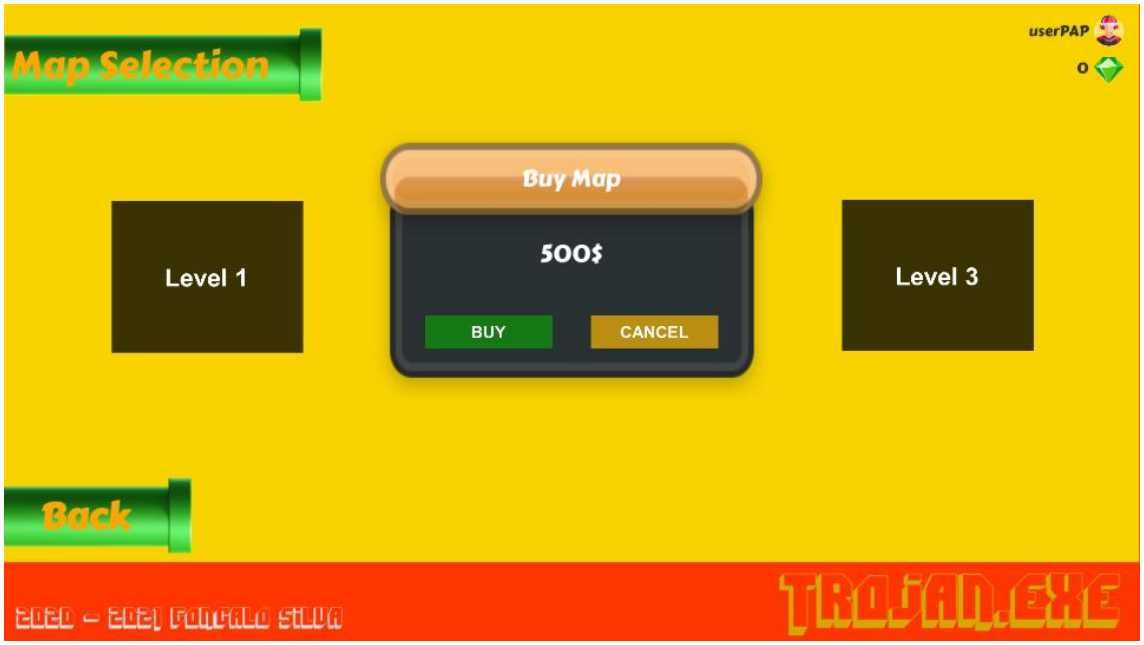

Figura 66 - Pop up com o preço do mapa

E ao clicar no botão “buy” será feito um pedido de um procedimento á base de dados através de um ficheiro php, para verificar se o utilizador tem o dinheiro necessário para a compra do mesmo.

```
$query = "CALL comprarmapa('".$idUser."', '".$idMap."');";
```

Figura 67 - código SQL para chamar o procedimento

Caso o utilizador tenha o dinheiro necessário, o mapa será adicionado aos mapas obtidos do utilizador.

Desenvolvimento do jogo

Script Bird

Para o desenvolvimento do jogo foi necessário dentro do unity, adicionar três componentes, um vai ser o "Rigidbody 2D", este componente serve para que o a personagem tenha gravidade ou seja não vai ser só uma imagem e sim um objeto com gravidade. Outro componente será o "Polygon Collider 2D", ou seja, o personagem já vai poder colidir com objetos que também tenham um "Collider 2D". O outro componente será um script criado que vai ter o nome "Bird", dentro deste script vamos ter vários métodos, como o método "Jump", "Update" e o "`OnTriggerEnter2D`".

O método "Jump" vai servir para o personagem efetuar o salto, para isso foi necessário duas variáveis uma do tipo "Rigidbody2D" e a outra que vai ser um float constante que vai servir como a quantidade de gravidade que será inicializado com o valor 100. No método "Jump" vai ser efetuado um cálculo onde a velocidade da variável do tipo "Rigidbody2D" vai ser igual á quantidade de gravidade a multiplicar por o y do "vector2", ou seja, um vetor 2D.

```
private void Jump(){

    birdRigidbody2D.velocity = Vector2.up * JUMP_AMOUNT;

}
```

Figura 68 - método Jump

De seguida foi criado um "enum" com o nome "State", que vai definir os três estados do personagem, a espera para jogar, a jogar e morto.

```
private enum State
{
    WaitingToStart,
    Playing,
    Dead
}
```

Figura 69 - enum dos estados do personagem

Depois no método “Update” foi necessário colocar um “switch” que por predefinição o primeiro caso é o “WaitingToStart”, ou seja, a espera para jogar, e caso o estado seja este o personagem vai ficar a espera, parado no ecrã de jogo, ate que haja um clique no rato ou no teclado, isto é verificado através de um “if”. Caso haja um clique, o estado do personagem fica “Playing ” e o seu “RigidBodyType2D” fica dinámico, ou seja, o personagem vai ter gravidade, quando iniciamos o jogo o personagem não tem gravidade porque no método “Awake” o “RigidBodyType2D” foi inicializado como estático e o estado como “WaitingToStart”.

```
private void Update()
{
    switch (state)
    {
        default:
        case State.WaitingToStart:
            if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0))
            {
                state = State.Playing;
                birdRigidbody2D.bodyType = RigidbodyType2D.Dynamic;
                Jump();
                if (OnStartedPlaying != null) OnStartedPlaying(this, EventArgs.Empty);
            }
            break;
```

Figura 70 - switch no método Update

De seguida é chamado o método “Jump” e caso o “OnStartedPlaying” que é uma variável do tipo “event” for diferente de vazio, o mesmo fica como “EventArgs.Empty”, assim poderá ser usado no script “Level” que será explicado mais para a frente. E também para se poder chamar métodos deste “script” em outros “scripts” é necessário instanciá-lo.

```
private void Awake()
{
    instance = this;
    birdRigidbody2D = GetComponent<Rigidbody2D>();
    birdRigidbody2D.bodyType = RigidbodyType2D.Static;
    state = State.WaitingToStart;
}
```

Figura 71 - método Awake

No “switch” caso o estado seja “Playing” o programa vai ficar a espera de um clique no rato ou no teclado para chamar o método “Jump”. E caso o estado seja “Dead”, o programa não faz nada.

```
case State.Playing:
    if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0))
    {
        Jump();
    }
    break;
case State.Dead:
    break;
```

Figura 72 - código C# para chamar o método "Jump"

O método “`OnTriggerEnter2D`” vai servir para que quando o personagem entre em contacto comoutro objeto que também tenha o “collider2D” irá fazer com que o personagem fique imóvel e que o “OnDied” que é uma variável do tipo “event” for diferente de vazio, o mesmo fica como “EventArgs.Empty”, assim poderá ser usado no script “Level”.

Script Level

Depois dos scripts do personagem criados, foi necessário criar um objeto vazio no Unity, onde dei o nome de “Level” e criei respetivamente um script e adicionei-o ao objeto “Level”.

Dentro do script correspondente ao objeto “Level”, foi criado um método “Create Pipe” onde vai ser passado duas variáveis do tipo float, uma delas dá-se pelo nome “height”, ou seja, altura e a outra variável é o “xPosition”, ou seja, a posição do cano e uma outra do tipo boolean “createBottom” que será usada para criar o cano de cima caso seja verdadeiro ou o cano de baixo caso seja falso.

Dentro do método “CreatePipe” foi necessário instanciar o cano completo, ou seja, a cabeça e o corpo do cano.

```
Transform pipeHead = Instantiate(GameAssets.GetInstance().pfPipeHead);
Transform pipeBody = Instantiate(GameAssets.GetInstance().pfPipeBody);
```

Figura 73 - Instanciar o cano

Depois de instanciar o cano foi necessário criar uma variável que será uma constante para definir a largura do cano, para que quando o cano apareça venha sempre com a mesma largura.

```
private const float PIPE_WIDTH = 11f;
```

Figura 74 - Constante PIPE_WIDTH

E também foi necessário criar uma constante para definir a altura da cabeça do cano.

```
private const float PIPE_HEAD_HEIGHT = 3.75f;
```

Figura 75 - Constante PIPE_HEAD_HEIGHT

E para finalizar foi necessário uma outra constante para definir os limites da câmera do jogo, ou seja, a câmera do jogo vai de 50 até -50.

```
private const float CAMERA_ORTHO_SIZE = 50f;
```

Figura 76 - Constante CAMERA_ORTHO_SIZE

Depois de criadas estas duas constantes foi necessário criar um "if" onde, caso o "createBottom" for verdadeiro a variável "pipeHeadYPosition", que é uma variável do tipo float criada dentro do método, vai armazenar um cálculo para posicionar a cabeça do cano.

```
if (createBottom)
{
    pipeHeadYPosition = -CAMERA_ORTHO_SIZE + height - PIPE_HEAD_HEIGHT * .5f;
}
```

Figura 77 - "if" para cálculo

Caso o "createBottom" for falso, ele irá fazer o cálculo mas com a variável "CAMERA_ORTHO_SIZE" positiva.

```
else
{
    pipeHeadYPosition = +CAMERA_ORTHO_SIZE - height + PIPE_HEAD_HEIGHT * .5f;
}
```

Figura 78 - "if" para cálculo

Depois de calcular o "pipeHeadYPosition", foi necessário indicar que o "pipeHead" instanciado anteriormente vai ficar na posição "xPosition" e que vai ficar referente ao Y na posição "pipeHeadYPosition".

```
pipeHead.position = new Vector3(xPosition, pipeHeadYPosition);
```

Figura 79 - Colocar o "pipeHead" na posição "xPosition"

Depois foi necessário fazer o mesmo procedimento para o corpo do cano.

```
if (createBottom)
{
    pipeBodyYPosition = -CAMERA_ORTHO_SIZE;
}
else
{
    pipeBodyYPosition = +CAMERA_ORTHO_SIZE;
    pipeBody.localScale = new Vector3(1, -1, 1);
}
pipeBody.position = new Vector3(xPosition, pipeBodyYPosition);
```

Figura 80 - Procedimento para o corpo do cano

Depois de saber onde os canos vão ser criados, foi necessário renderizá-los, pra isso foi necessário utilizar uma variável "pipeBodySpriteRenderer" que vai ser do tipo "SpriteRenderer" e que vai ser igual ao "pipeBody".

```
SpriteRenderer pipeBodySpriteRenderer = pipeBody.GetComponent<SpriteRenderer>();
```

Figura 81 - Renderizar os obstáculos

Depois foi necessário transformar o cano criado num cano do tipo "pipe", isto é, foi criada uma classe com o nome "pipe" onde vai ter vários métodos e nesses métodos.

Depois bastou adicionar o cano a uma lista, "pipeList".

```
pipeList.Add(pipe);
```

Figura 82 - Adicionar a uma lista

De seguida foi criado um método "CreatGapPipes" para definir o espaçamento entre os canos.

Este método vai chamar o método "CreatePipe" e vai enviar o valor das variáveis necessárias para calcular esse espaçamento.

```
CreatePipe(gapY - gapSize * .5f, xPosition, true);
CreatePipe(CAMERA_ORTHO_SIZE * 2f - gapY - gapSize * .5f, xPosition, false);
```

Figura 83 - código para chamar o método "CreatePipe"

Depois de os obstáculos estarem a serem criados foi necessário criar um novo método "HandlePipeSpawning" para chamar o método "CreateGapPipes". No novo método, foi criado um foi criado um "if" onde este só vai ser executado se o "pipeSpawnTimer", que é uma variável global do tipo float, for meno que zero, para que o código funcionasse, fora do "if" foi necessário efetuar uma linha de código que vai dizer que o "pipeSpawnTimer" vai ser igual ao "pipeSpawnTimer" menos o deltaTime, que é função que calcula o tempo em segundos desde o último frame. Dentro do "if" foi criada uma linha de código onde o "pipeSpawnTimer" vai ser igual "pipeSpawnTimer" mais o "pipeSpawnTimerMax", que também é uma variável global do tipo float onde tem o valor de "1f".

```
private void HandlePipeSpawning()
{

    pipeSpawnTimer -= Time.deltaTime;

    if (pipeSpawnTimer < 0)
    {
        pipeSpawnTimer += pipeSpawnTimerMax;
```

Figura 84 - método HandlePipeSpwaning

Depois disso foram passados alguns parâmetros necessários para chamar o método "CreateGapPipes".

```
float heightEdgeLimit = 10f;
float minHeight = gapSize * .7f;
float totalHeight = CAMERA_ORTHO_SIZE * 2f;
float maxHeight = totalHeight - gapSize * .7f - heightEdgeLimit;
```

Figura 85 - parâmetros necessários para chamar o método

Depois de passados alguns parâmetros foi necessários fazer um "random", ou seja, para que os canos fossem gerados de forma aleatória foi necessário fazer um "random" que vai gerar os canos dentro de um limite passado no código que neste caso foram a altura mínima e máxima.

Depois disso bastou chamar o método "CreateGapPipes", com os parâmetros de altura, que foi gerada anteriormente, depois o tamanho do espaçamento entre os obstáculos e por fim a posição onde os canos vão ser gerados.

```
float height = Random.Range(minHeight, maxHeight);
CreateGapPipes(height, gapSize, PIPE_SPAWN_X_POSITION);
```

Figura 86 - Criar altura aleatória

De seguida foi necessário colocar o método criado no método "Update", que como o nome indica, este método vai estar a ser chamado a cada "frame".

```
private void Update()
{
    if (state == State.Playing)
    {
        HandlePipeMovement();
        HandlePipeSpawning();
        HandleGround();
    }
}
```

Figura 87 - método Update

Depois de os obstáculos estarem a serem gerados foi necessário que eles se movimentassem, para isso foi preciso criar outro método. Este novo método foi apelidado de "HandlePipeMovement". Dentro deste método foi criado um ciclo "for" que vai ser executado enquanto a variável do ciclo for menor do que a "pipeList", ou seja, a lista onde armazenamos os obstáculos no método "CreatePipe".

```
private void HandlePipeMovement()
{

    for (int i = 0; i < pipeList.Count; i++)
    {
```

Figura 88 - ciclo do método HandlePipeMovement

Dentro do ciclo foi necessário armazenar os obstáculos da posição atual da lista numa variável "pipe" que vai ser do tipo "Pipe", este tipo "Pipe" é uma classe que foi necessária criar para conseguir mover, e destruir os obstáculos. Depois de armazenar os obstáculos da posição atual do ciclo, é criada uma variável do tipo "bool", ou seja, vai armazenar verdadeiro ou falso, que vai verificar se a posição do obstáculo encontra-se ao lado direito do personagem e vai armazenar se a condição é verdadeira ou falsa.

```
bool isToTheRightOfBird = pipe.GetXPosition() > BIRD_X_POSITION;
```

Figura 89 - variável para saber se o personagem está a direita do cano

De seguida é chamado o método "Move" da classe "Pipe", este método é bastante simples só vai mover a cabeça e o corpo do cano, ou seja, como os obstáculos foram guardados na variável "pipe" da classe "Pipe" a cabeça e o corpo do cano já foram transformados basta agora no método "Move" incrementar ao "pipeHeadTransform.position" um novo "vector3" onde vai ser indicado para o cano se movimentar para a esquerda, e este "vector3" vai ser multiplicado pelo "PIPE_MOVE_SPEED" que é uma constante com o valor de "50f" que também vai ser multiplicado pelo tempo de conclusão em segundos desde o último "frame".

```
public void Move()
{
    pipeHeadTransform.position += new Vector3(-1, 0, 0) * PIPE_MOVE_SPEED * Time.deltaTime;
    pipeBodyTransform.position += new Vector3(-1, 0, 0) * PIPE_MOVE_SPEED * Time.deltaTime;
}
```

Figura 90 - método Move

Depois de movimentar o cano, foi necessário criar um "if" com a condição de, se o cano está a direita e a posição no eixo do "X" do personagem for menor ou igual a 0, o "pipesPassedCount" que é uma variável global para a contabilização dos pontos, é adicionado mais um valor a essa variável.

```
if (isToTheRightOfBird && pipe.GetXPosition() <= BIRD_X_POSITION) {
    //Pipe passed bird
    pipesPassedCount ++;
```

Figura 91 - "if" para contabilizar os obstáculos ultrapassados

Depois para não haver um número infinito de obstáculos a serem armazenados na memória RAM foi necessário definir um ponto em que os canos são destruídos. Para isso foi necessário usar uma constante que vai ser a posição de destruição dos obstáculos, depois bastou fazer um "if" em que verifica-se se a posição do cano é menor do que a posição de destruição, se for verdadeiro irá ser chamado o método "DestroySelf" da classe "Pipe" que vai destruir o cano.

```
if (pipe.GetXPosition() < PIPE_DESTROY_X_POSITION)
{
    pipe.DestroySelf();
```

Figura 92 - código para chamar o método DestroySelf

```
public void DestroySelf()
{
    Destroy(pipeHeadTransform.gameObject);
    Destroy(pipeBodyTransform.gameObject);
}
```

Figura 93 - método DestroySelf

Depois para finalizar a parte do jogo, foi necessário criar outro "script" que vai ser utilizado para indicarmos no Unity quais são os obstáculos que queremos gerar.

```
public Sprite pipeHeadSprite;
public Transform pfPipeHead;
public Transform pfPipeBody;
```

Figura 94 - código para armazenar os assests escolhidos

Depois no Unity basta indicar quais são as texturas que queremos usar.

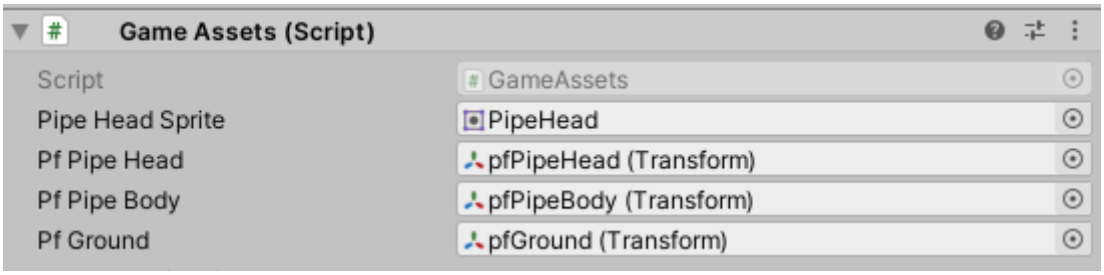

Figura 95 - indicar os assests no Unity

De seguida ao executarmos o jogo, ele teve o seguinte aspeto:

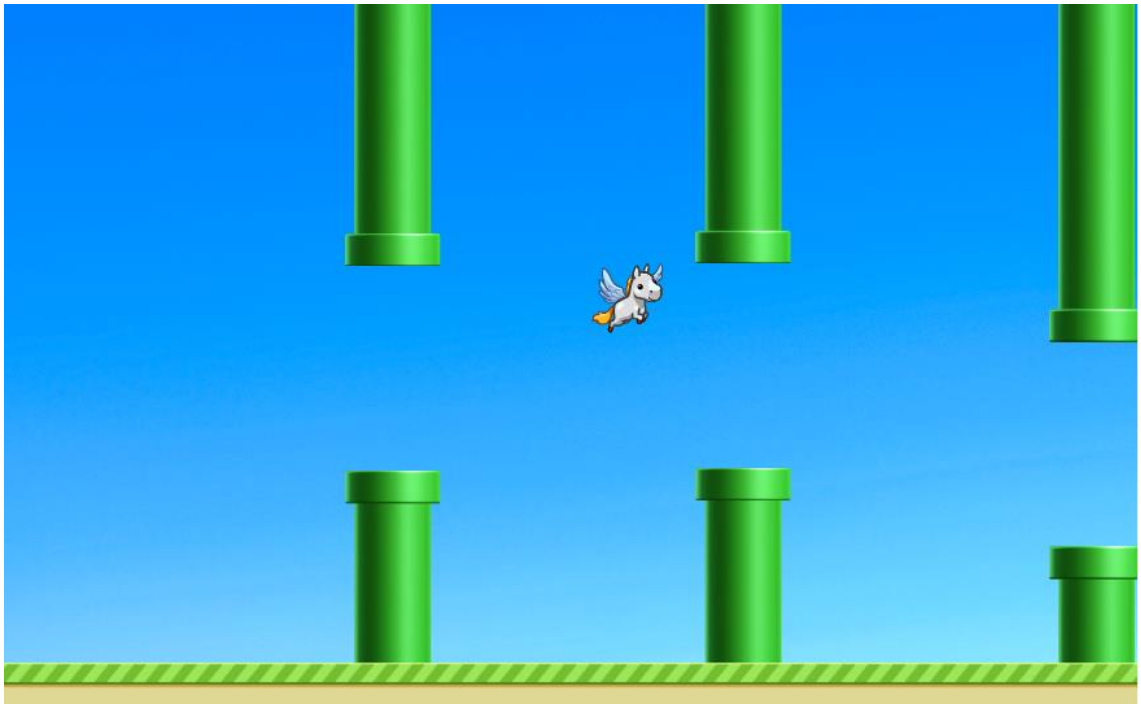

Figura 96 - Imagem do jogo em execução

Depois para fazer com que a pontuação aparecesse no ecrã foi necessário no unity criar um "Canvas" e dentro do mesmo criar um objeto "ScoreWindow" e a esse objeto vai ficar associado um "script" que vai ser criado com o nome "ScoreWindow", dentro do objeto também foi necessário criar dois campos de texto, um para a pontuação atual e outro para a melhor pontuação.

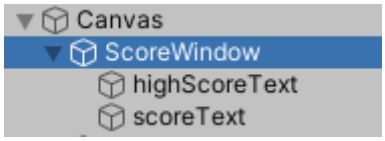

Figura 97 - criação do objeto "ScoreWindow" no Unity

Dentro do "script" criado, foi necessário criar duas variáveis do tipo texto, uma para o "high score" e a outra para o "score", de seguida foi necessário, no método "Awake" dizer que as duas variáveis vão apresentar os valores nos campos de texto criados no Unity.

```
private void Awake()
{
    scoreText = transform.Find("scoreText").GetComponent<Text>();
    highScoreText = transform.Find("highScoreText").GetComponent<Text>();
}
```

Figura 98 - método Awake

De seguida para apresentarmos a pontuação da jogada atual sempre que o utilizador passava por um cano, foi necessário dentro do método "Update" dizer que a variável anteriormente criada vai ser igual ao método "GetPipesPassedCount" criado no "script Level".

```
private void Update()
{
    scoreText.text = Level.GetInstance().GetPipesPassedCount().ToString();
}
```

Figura 99 - método Update

Depois para o “high score” foi necessário, no método “Start”, obter qual seria o mapa em que o utilizador estaria a jogar, para isso foram criados “if´s” onde iram verificar qual é o nome da “scene” atual, depois de verificar a “scene” o programa irá buscar ao “script DBManager” onde vai armazenar os “high scores” de cada mapa e irá apresenta-lo conforme o mapa.

```
if (SceneManager.GetActiveScene().name.ToString() == "Lvl1")
{
    highScoreText.text = "High Score: " + DBManager.firstMap.ToString();
}

if (SceneManager.GetActiveScene().name.ToString() == "Lvl2")
{
    highScoreText.text = "High Score: " + DBManager.secondMap.ToString();
}

if (SceneManager.GetActiveScene().name.ToString() == "Lvl3")
{
    highScoreText.text = "High Score: " + DBManager.thirdMap.ToString();
}
```

Figura 100 - "if" para apresentar a melhor pontuação

De seguida ao executarmos o jogo, ele teve o seguinte aspeto:

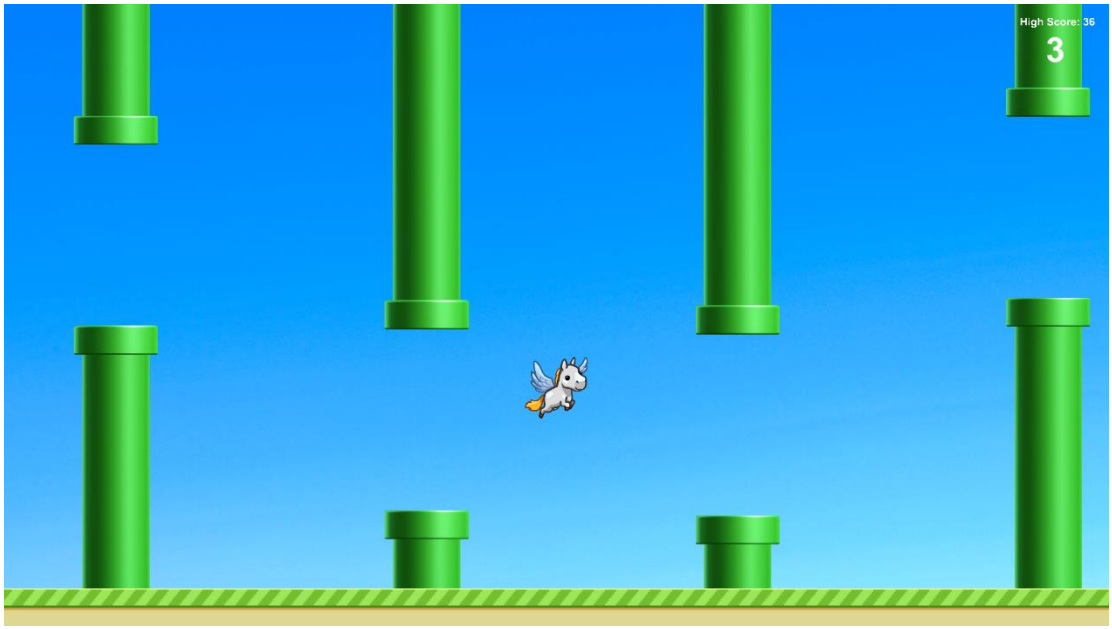

Figura 101 - Imagem do jogo em execução

Depois para quando o personagem morra foi necessário criar um menu de “Game Over”, para que o utilizador possa jogar novamente ou ir para o menu principal. Para isso foi necessário no Unity criar um pequeno menu com dois botões.

De seguida foi necessário criar um “script” referente ao menu de “Game Over”.

Dentro do “script” para o botão de jogar novamente foi necessário no método “Awake” verificar qual é o mapa em que o utilizador está a jogar e depois basta fazer com que o botão recarregue a “scene” que o utilizador estava a jogar.

E para o botão do menu principal, basta carregar a “scene” do menu principal.

```
if(SceneManager.GetActiveScene().name.ToString() == "Lvl1") {
    transform.Find("retryBtn").GetComponent<Button_UI>().ClickFunc = () => { Loader.Load(Loader.Scene.Lvl1); };
}

if (SceneManager.GetActiveScene().name.ToString() == "Lvl2")
{
    transform.Find("retryBtn").GetComponent<Button_UI>().ClickFunc = () => { Loader.Load(Loader.Scene.Lvl2); };
}

if (SceneManager.GetActiveScene().name.ToString() == "Lvl3")
{
    transform.Find("retryBtn").GetComponent<Button_UI>().ClickFunc = () => { Loader.Load(Loader.Scene.Lvl3); };
}

transform.Find("mainMenuBtn").GetComponent<Button_UI>().ClickFunc = () => { Loader.Load(Loader.Scene.Menu); };
```

Figura 102 - "if" para carregar o mapa selecionado

Nesse mesmo “script” também foram feitas três requisições a base de dados, a primeira para verificar se a pontuação que o utilizador fez é maior do que a melhor pontuação do utilizador guardada na base de dados, a segunda para buscar as melhores pontuações dos três mapas e a última para adicionar as “coins” á base de dados.

A requisição que serve para verificar se a pontuação que o utilizador fez é maior do que a melhor pontuação do utilizador guardada na base de dados leva três parâmetros de entrada, o id do utilizador, a pontuação e o mapa. De seguida será feito um pedido ao ficheiro php que está no servidor disponibilizado pela escola.

```
WWWForm form = new WWWForm();
form.AddField("idUser", DBManager.id);
form.AddField("score", scoreText.text);
form.AddField("map", SceneManager.GetActiveScene().name.ToString());
WWW www = new WWW("https://alpha.soaresbasto.pt/~goncalosilva/api/checkHighScore.php", form);
```

Figura 103 - requisição para verificar a melhor pontuação

No ficheiro php foi necessário fazer uma ligação a base de dados e de seguida foi preciso fazer um pedido sql á base de dados para verificar qual é a melhor pontuação guardada.

```
$namecheckquery = "SELECT score
FROM Score where idJogador = '".$idUser."'
AND idMapa = '".$idMap."';";
```

Figura 104 - código SQL para selecionar a melhor pontuação

Caso não haja nenhuma pontuação guardada, a pontuação feita irá ser guardada na base de dados.

```
if(mysqli_num_rows($namecheck) == 0)
{
 $insertuserquery = "INSERT INTO Score (score, idJogador, idMapa)
 VALUES ('".$score. "', '".$idUser. "', '".$idMap. "');";
```

Figura 105 - Inserção da pontuação

Mas se já houver alguma pontuação irá ser verificado se a pontuação atual é maior do que a guardada na base de dados, caso seja irá ser guardada na base de dados.

```
if($highScore < $score){

 $insertuserquery = "UPDATE Score
 SET score = '".$score. "'
 WHERE idJogador = '".$idUser. "'
 AND idMapa = '".$idMap."';";
```

Figura 106 - Verificação da pontuação e inserção da mesma

A seguinte requisição que serve para ir buscar as melhores pontuações nos três mapas existentes no jogo, isto irá servir para que o programa possa estar sempre apresentar as melhores pontuações do utilizador.

Esta requisição leva dois parâmetros de entrada, o id do utilizador e o id do mapa.

```
WWWForm form = new WWWForm();
form.AddField("idUser", DBManager.id);
form.AddField("map", SceneManager.GetActiveScene().name.ToString());
WWW www = new WWW("https://alpha.soaresbasto.pt/~goncalosilva/api/getHighScoresUser.php", form);
```

Figura 107 - requisição para buscar a melhor pontuação

No ficheiro php irá ser feito um pedido sql á base de dados onde vai verificar qual é a melhor pontuação do jogador no id que foi passado anteriormente, de seguida o valor do resultado será retornado e guardado no "script DBManager".

```
$MapQuery = "SELECT score
FROM Score
where idJogador = '".$idUser. "'
and idMapa = '".$idMap. "';";
```

Figura 108 - código SQL para buscar a melhor pontuação do utilizador

A última requisição serve para adicionar as coins obtidas á base de dados, esta requisição leva dois parâmetros de entrada, o id do utilizador e as coins obtidos que são a pontuação obtida pelo jogador.

```
WWWForm form = new WWWForm();
form.AddField("idUser", DBManager.id);
form.AddField("coins", scoreText.text);
WWW www = new WWW("https://alpha.soaresbasto.pt/~goncalosilva/api/addCoins.php", form);
```

Figura 109 - requisição para adicionar "coins"

De seguida no ficheiro php foi necessário fazer um pedido á base de dados onde vai retornar o valor do dinheiro do utilizador e a esse valor é somado o valor obtido.

```
$coinscheckquery = "SELECT coins
FROM Jogador where id = '".$idUser."';";
```

Figura 110 - Buscar as "coins" do utilizador

```
$coins = $coins + $score;

 $insertcoinsquery = "UPDATE Jogador
 SET coins = '".$coins. "'
 WHERE id = '".$idUser. "';";
```

Figura 111 - Inserção das "coins"

E assim o jogo está completamente funcional e pronto para ser jogado.

Desenvolvimento do Website

Depois do desenvolvimento do jogo decidi fazer um website onde as pessoas possam descarregar o jogo e ler algumas funções do jogo e informações sobre mim através do seguinte link: “https://alpha.soaresbasto.pt/~goncalosilva/Trojan/”.

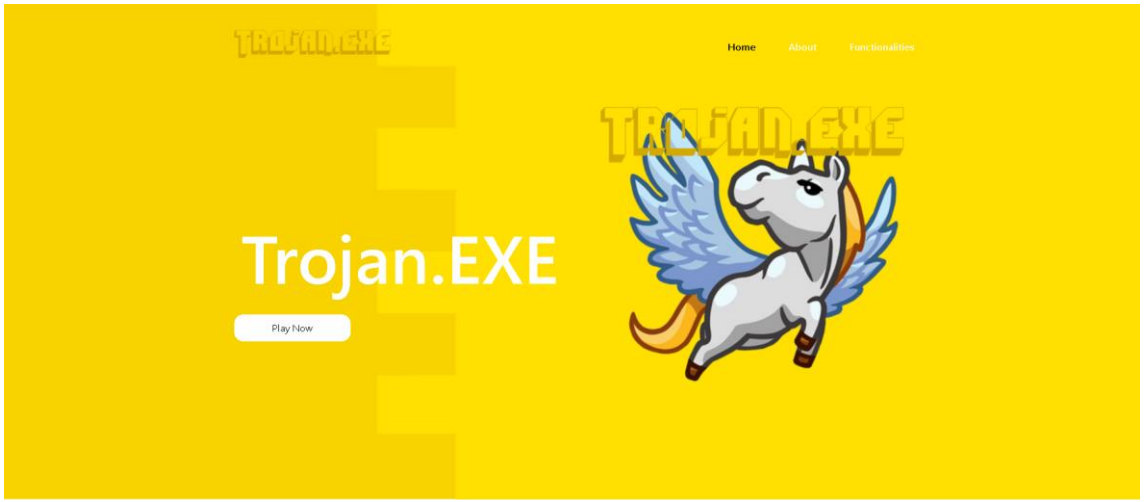

Figura 112 - Website de apoio

Ao clicar no botão “Play Now”, uma cópia do jogo será descarregado e o utilizador conseguirá jogar.

Conclusão

Como comentário final, após concluir todas as etapas do meu projeto, posso afirmar que os objetivos inicialmente traçados foram totalmente atingidos e consequência disso a minha satisfação pela realização do que havia sido proposto a fazer. Não obstante a isso, depois de terminado o trabalho e analisado o seu funcionamento, verifiquei que poderia ter sido criado uma quantidade de melhorias a serem efetuadas, mas de futuro gostaria de concluir essa melhorias e trabalhar mais neste projeto que mais uma vez digo que estou completamente satisfeito com o resultado final.

Bibliografia

Brackeys (Realizador). (2020). How to Program in C# - BASICS (E01) | Disponível em: https://www.youtube.com/watch?v=jGD0vn-QIkg [Filme].

Brackeys (Realizador). (2020). How to Program in C# - Conditions (E03) | Disponível em: https://www.youtube.com/watch?v=u_Qv5IrMUqg&list=RDCMUCYbK_tjZ2OrIZFBvU6CCMiA&index=2 [Filme].

Brackeys (Realizador). (2020). How to Program in C# - Variables (E02) | Disponível em: https://www.youtube.com/watch?v=g-9Jp4dmOBo [Filme].

Games, B. T. (Realizador). (2018). Unity & MySQL Databases, Part 1: Setting Up | Disponível em: https://www.youtube.com/watch?v=SKbY-0zt2VE [Filme].

Games, B. T. (Realizador). (2018). Unity & MySQL Databases, Part 2: Registering Users | Disponível em: https://www.youtube.com/watch?v=4W90-mh70JY [Filme].

Games, B. T. (Realizador). (2018). Unity & MySQL Databases, Part 3: Parsing Server Data | Disponível em: https://www.youtube.com/watch?v=AjBtDvERlyg&t=45s [Filme].

Games, B. T. (Realizador). (2018). Unity & MySQL Databases, Part 4: User Login | Disponível em: https://www.youtube.com/watch?v=NVdjlXgbiMM&t=1204s [Filme].

Imphenzia (Realizador). (2020). LEARN UNITY - The Most BASIC TUTORIAL I'll Ever Make | Disponível em: https://www.youtube.com/watch?v=pwZpJzpE2lQ [Filme].