FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Jogos Digitais

Isaias Lima

Leonardo Moreira

Letícia Galbiati

Victor Fontolan

THE FIRE CODEX

Americana, SP

2017

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Jogos Digitais

Isaias Lima

Leonardo Moreira

Leticia Galbiati

Victor Fontolan

THE FIRE CODEX

Relatório técnico desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Jogos Digitais sob a orientação do(a) Prof.(a) Esp. Gustavo Carvalho Gomes de Abreu.

Americana, SP.
2017

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

L698f LIMA, Isaias da Silva

The fire codex. / Isaias da Silva Lima ; Leonardo Moreira Marchetti ; Letícia Teixeira Galbiati ; Vitor Fontolan. – Americana, 2017.

64f.

Monografia (Curso de Tecnologia em Jogos Digitais) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Esp. Gustavo Carvalho Gomes de Abreu.

1 Jogos eletrônicos I. MARCHETTI, Leonardo Moreira II. GALBIATI, Letícia Teixeira III. FONTOLAN, Vitor IV. ABREU, Gustavo Carvalho Gomes de V. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.6

Isaias Lima
Leonardo Moreira
Letícia Galbiati
Victor Fontolan

THE FIRE CODEX

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais pelo CEETEPS/Faculdade de Tecnologia – Fatec Americana.

Área de concentração: Jogos Digitais.

Americana, 16 de dezembro de 2017

Banca Examinadora:

Gustavo Carvalho Gomes de Abreu (Presidente)
Especialista
CEETEPS/Faculdade de Tecnologia de Americana

Bruno Daniel (Membro)
Mestre
CEETEPS/Faculdade de Tecnologia de Americana

Rafael Moraes (Membro)
Graduado
CEETEPS/Faculdade de Tecnologia de Americana

RESUMO

Considerando nosso percurso como alunos do curso de Jogos Digitais, compreendemos que o desenvolvimento de um jogo completa esse processo ao possibilitar o uso dos conhecimentos e habilidades adquiridos. Assim, apresentamos o processo de desenvolvimento do jogo digital The Fire Codex.

Desenvolvemos um jogo no estilo *puzzle* para plataforma mobile, inspirada na cultura maia e que tem como personagem principal, o Fogo. Em cada capítulo, ilustramos as etapas de cada um dos processos: concepção de arte, level design, mecânica de jogo e áudio.

Assinalamos os desafios enfrentados e o complexo processo de criação, bem como a importância dessa experiência para nossa formação profissional. Esperamos que nossa trajetória auxilie futuros grupos, contribuindo para o crescimento da área.

Palavras Chave: jogos digitais, quebra-cabeças, mobile.

ABSTRACT

Considering our course as students of the Digital Games course, we understand that the development of a game completes this process by making possible the use of acquired knowledge and skills. Thus, we present the process of developing the digital game The Fire Codex.

We developed a puzzle game for the mobile platform, inspired by the Mayan culture and that has as main character, Fire. In each chapter, we illustrate the stages of each process: art design, level design, game mechanics and audio.

We highlight the challenges faced and the complex creation process, as well as the importance of this experience for our professional training. We hope that our trajectory will help future groups, contributing to the growth of the area.

Keywords: *digital games, puzzles, mobile.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

CheckPoints: Ponto de início/reinicio de jogo.

Collider: Colisor, tipo de componente do Unity® que faz com que objetos rígidos colidam.

Cutscenes: É uma sequência em um jogo eletrônico sobre a qual o jogador tem nenhum ou pouco controle, interrompendo a jogabilidade e sendo usada para avançar o enredo, reforçar o desenvolvimento do personagem principal, introduzir personagens inimigos, e providenciar informações de fundo, atmosfera, diálogo, ou pistas.

Download: Ato de baixar um arquivo.

Game Engine: Motor de jogo, conjunto de bibliotecas, disponibilizadas para facilitar o desenvolvimento de um jogo.

HUD: Heads-up display, elementos na tela.

Inputs: Método de entrada de informação.

Joystick: Controlador do jogo, dotado de alavanca para controlar.

Level Design: desenvolvimento do espaço virtual do jogo.

Mobile: Dispositivo móvel ou SmartPhone. Telefone celular com tecnologias avançadas, semelhante à um computador.

Mockup: Conceitos/Protótipos de arte de games

Open Source: Código Aberto é um modelo de desenvolvimento que promove licenciamento livre.

Prefab: É um tipo de Game Object com propriedades e componentes já configurados em cena, que podem ser armazenados e reutilizados.

Puzzle: Quebra cabeça, enigma.

Script: Texto com uma série de instruções escritas para serem seguidas.

Software: Programas eletrônicos.

Sprite: É um objeto gráfico bidimensional que se move numa tela sem deixar traços de sua passagem.

TileMap:

UI: Game Interface.

Update: Atualização.

TileMap: Técnica de construção do mundo do jogo ou mapas de pequeno porte através de imagens em forma de quadrado regular.

1 INTRODUÇÃO

Compreendendo nossa trajetória como alunos do curso de Jogos Digitais da Faculdade de Tecnologia de Americana (FATEC-AM), consideramos que o desenvolvimento de um jogo digital completa esse processo, ao possibilitar o uso dos conhecimentos e habilidades adquiridos durante o curso, bem como o aperfeiçoamento daquelas que existiam antes do ingresso. Dessa forma, nosso Trabalho de Conclusão de Curso (TCC) apresenta a criação do jogo The Fire Codex, sendo que nos próximos capítulos apresentamos o detalhamento desse processo.

1.1. O surgimento da Ideia

No mercado dos jogos digitais constantemente nos deparamos com alguma novidade, seja o surgimento de uma história inovadora, uma mecânica de jogo diferente ou um novo estilo de arte. Quando focalizamos os personagens, percebemos que a maioria dos jogos possui um personagem predefinido, seja uma pessoa, um alienígena, um animal, entre outros. Assim, ao pensar sobre o personagem do nosso jogo, decidimos criar um que fosse diferente e inesperado, um personagem que até então parecia difícil de considerar como algo além do que ele parece ser, como algo além de apenas um elemento.

Dessa forma, escolhemos usar o próprio fogo como um personagem. Logo ele não seria apenas fogo, e sim, o Fogo, com letra maiúscula digno de qualquer substantivo próprio. Ele seria o foco da história, e mais para frente na história, seus companheiros e inimigos também viriam a aparecer.

Depois de decidir o personagem principal, criamos todo o universo ao seu redor e o que ele faria no jogo. Assim, optamos por um jogo de puzzles, no qual o personagem principal Fogo passaria por enigmas e obstáculos nos quais precisaria de suas habilidades para resolvê-los, sendo que sua história seria revelada ao longo do percurso.

1.2. O jogo no Mercado

Os dados da Pesquisa Game Brasil 2017 (Sioux, 2017) indicam que 50,9% da população entrevistada prefere jogo de estratégia, sendo que o público que mais joga tem entre 25 e 34 anos e assinala o entusiasmo ao evoluir pelas fases. Além disso, a pesquisa também aponta a plataforma mobile em primeiro lugar, com 77% de preferência do público.

Considerando esses aspectos, escolhemos o gênero de estratégia e concebemos o The Fire Codex como um jogo que visa garantir uma forma de diversão rápida e de fácil acesso, assim, oferece desafio moderado ao jogador e possibilita a retomada na área em que estava, sem que todo o processo tenha que ser refeito.

1.3. Concorrência de Mercado

Alguns jogos que podem ser vistos como concorrência ao nosso são Limbo (Playdead, 2010), Talbot's Odissey (Studio Miniboss, 2010) e Badland (Frogmind, 2013), pois eles possuem as mecânicas similares ao nosso, além de possuir a ideia de uma fase que não termina, e sim até o final do jogo.

Eles também focam mais na variedade de puzzles e desafios que o jogo oferece, ao invés de uma arte mais realista e elaborada.

Figura 1 - Jogo Badland

Fonte: Badland (Frogmind, 2010)

Figura 2 - Jogo Talbot's Odissey

Fonte: Talbot’s Odyssey (Studio Miniboss, 2010)

Figura 3 - Jogo Limbo

Fonte: Limbo (Playdead, 2010)

2 METODOLOGIA

Nesse capítulo iremos apresentar a história do jogo, as inspirações para a arte, as mecânicas aplicadas, a construção de cenário e os desafios presentes em cada um deles. Além disso, iremos descrever todos os programas utilizados durante o projeto e a justificativa de nossas escolhas.

2.1. Programas utilizados no projeto

Aqui descreveremos cada programa utilizado e o motivo de nossas escolhas.

2.1.1. Unity®

Unity® (Unity Technologies, 2004) é uma *game engine* que visa facilitar o desenvolvimento de jogos para diversas plataformas. Escolhemos essa opção, pois no decorrer do curso essa engine foi mais usada, além de não possuir custo para a exportação de jogos para a plataforma Android.

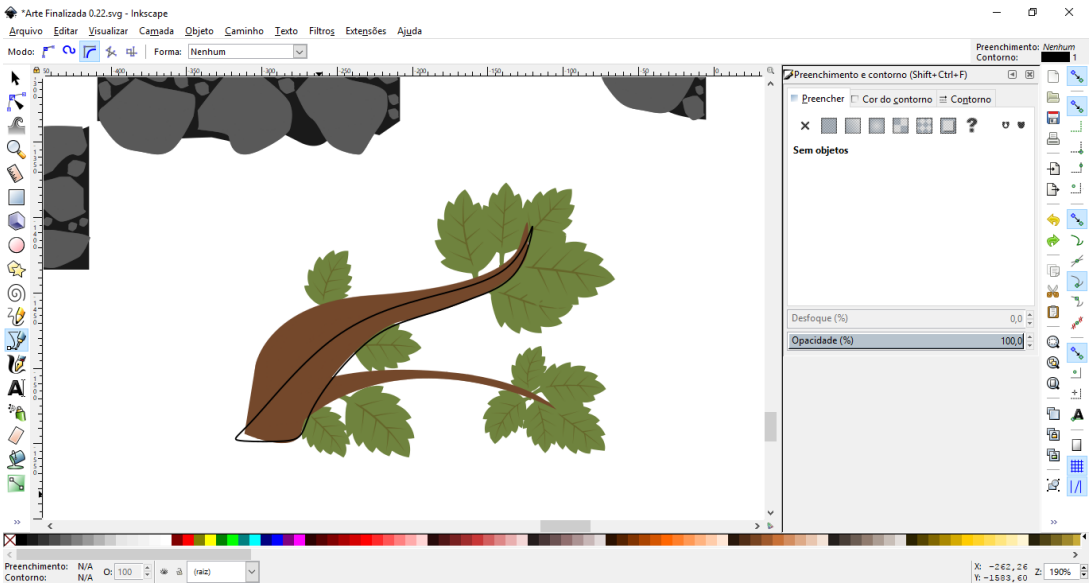
2.1.2. Audacity

O Audacity (Audacityteam, 1999) é um software para criação e edição de áudio. Novamente, foi escolhido devido ao maior contato na disciplina de Animação e Som, além de ser gratuito. Por termos apenas editado os áudios e não os criado, esse software foi satisfatório para nosso projeto.

2.1.3. Inkscape™

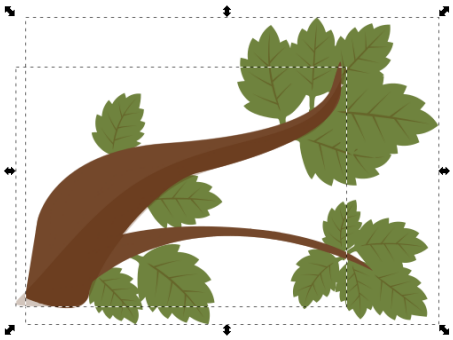
Inkscape™ (Inksape Team, 2003) é um editor de gráficos vetoriais *open source*. Decidimos por esse tipo de imagem, considerando o intuito de que não houvesse perda na qualidade das imagens que compõe a arte do jogo. Utilizando a ferramenta Caneta Bézier do Inkscape™ conseguimos chegar ao resultado desejado (exemplificado nas figuras 4 e 5) devido a sua facilidade no uso e com as propriedades do objeto que o software possuiu de preenchimento, contorno e transparência possibilitando uma gama de variações no resultado final.

Figura 4 - Vetorização do galho



Fonte: Autoral

Figura 5 - Vetorização do galho finalizado



Fonte: Autoral

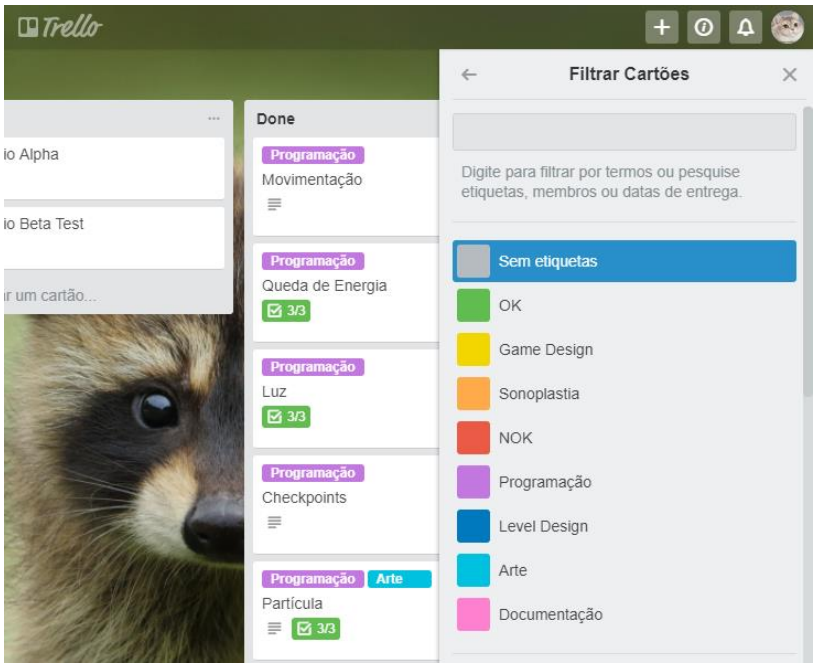
2.1.4. Trello

O Trello (Trello Inc, 2014) é uma ferramenta utilizada para organizar as tarefas do projeto, facilitando a visualização do que tem que ser feito, do que já está pronto e o que está em andamento. Ele possui etiquetas que categorizam as funções, no nosso caso, por área de desenvolvimento, nas quais cada cor corresponde a um tipo de função.

Para sabermos o que já estava completo e facilitar ainda mais o controle, colocamos duas etiquetas (OK e NOK), assim conseguiríamos separar as tarefas

que já estavam feitas (OK) das que ainda não tinham sido concluídas (NOK) mais rapidamente, como mostra a Figura 6.

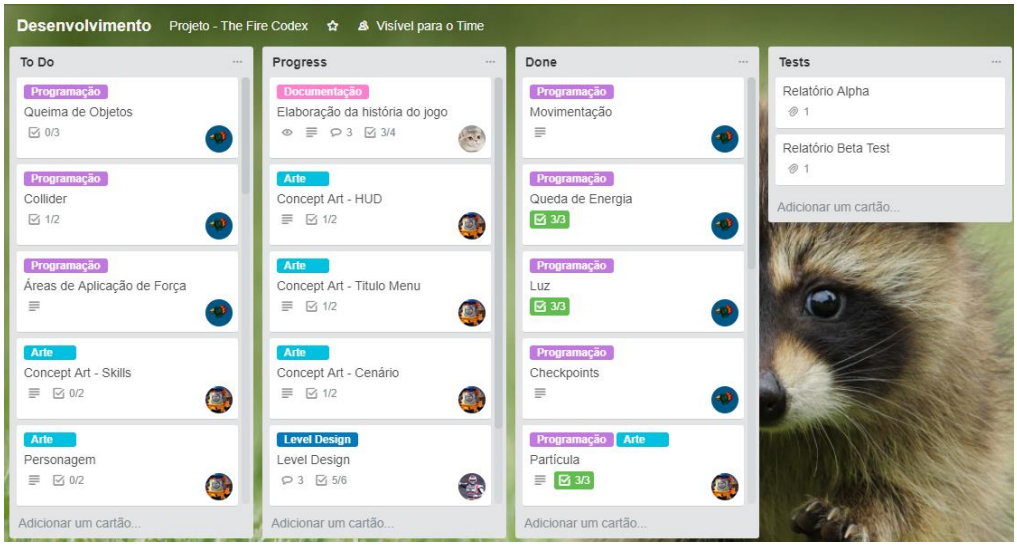
Figura 6 - Exemplo de Etiquetas



Fonte: Autoral

Organizamos as listas de tarefas em *To Do* (precisa ser feito), *Progress* (está em andamento), *Done* (o que já foi feito) e *Tests* (para colocar o resultado dos testes), conforme Figura 7.

Figura 7- Classificações das Tarefas



Fonte: Autoral

2.2. Cronograma de trabalho e divisão de tarefas

O cronograma de trabalho foi elaborado conforme mostra a Tabela 1 e a divisão de tarefas é apresentada na Tabela 2.

Tabela 1 - Cronograma de tarefas

ANO	2017							
ATIVIDADE / MÊS	**MAI**	**JUN**	**JUL**	**AGO**	**SET**	**OUT**	**NOV**	**DEZ**
Estruturação do Jogo	x	x	x					
Game Design		x	x					
Desenvolvimento da história do jogo			x	x				
Programação, Design, Level Design, Sonoplastia				x	x	x		
Apresentação da versão alfa do jogo ao orientador						x		
Apresentação da versão beta do jogo ao orientador							x	
Documentação					x	x	x	
Revisão do trabalho							x	
Elaboração da apresentação							x	
Apresentação e defesa pública							x	
Entrega do trabalho escrito								x

Tabela 2 - Distribuição de tarefas

INTEGRANTE DO GRUPO	FUNÇÃO EXERCIDA
Isaias Lima	Concept art, animações, level design e arte do jogo
Leonardo Moreira	Programação, level design e mecânicas
Victor Fontolan	Sonoplastia e level design
Letícia Galbiati	Roteiro e documentação

2.3. Divulgação do jogo

A divulgação do jogo foi realizada através das redes sociais, Facebook e Whatsapp; por recomendações de amigos e divulgação de Youtubers.

2.4. História do jogo

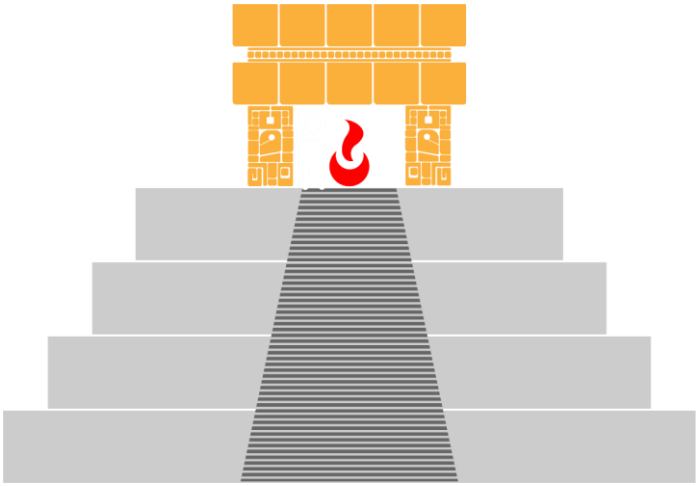
O Deus do Fogo, ao ver a dificuldade dos humanos para sobreviver, resolve compartilhar a sua chama sagrada com eles. Entretanto, um dia sua chama sagrada some do templo, e ele perde seu poder. Indefeso, ele recolhe-se para dentro do templo e lá entra em um sono profundo. A humanidade, sem o elemento fogo, volta a sofrer com o frio.

No jogo, o jogador irá controlar uma chama, tendo o objetivo de descobrir o que aconteceu com o Fogo, e como recuperá-lo para, em seguida, devolvê-lo a humanidade.

2.4.1. Contando a história

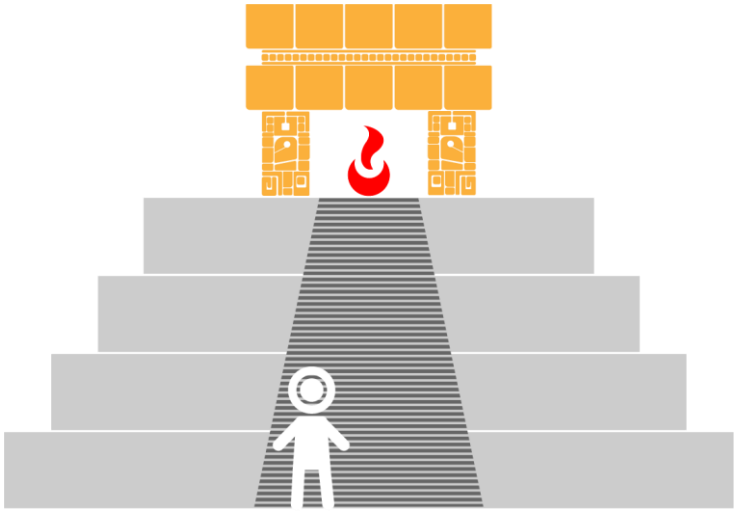
Escolhemos uma maneira diferente de contar a história do jogo, ao invés de colocarmos *cutscenes* ou textos mostrando a história, decidimos apresentá-la através de imagens.O jogo é bem mais visual do que textual, então resolvemos seguir a mesma ideia com a história. Enquanto o jogador percorre os cenários do jogo, poderá aparecer alguma imagem inscrita na parede que mostrará uma pista da história do jogo, como pode ser visto na Figura 9.

Figura 8 - Contando a História parte 01



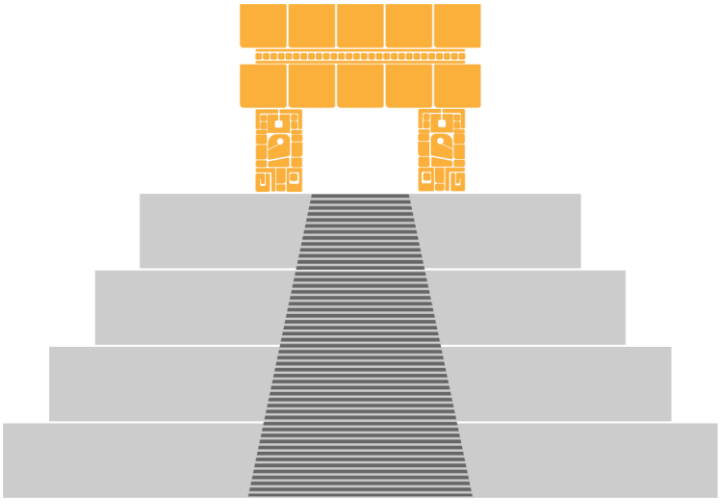
Fonte: Autoral

Figura 9 - Contando a História parte 02



Fonte: Autoral

Figura 10 - Contando a História parte 10



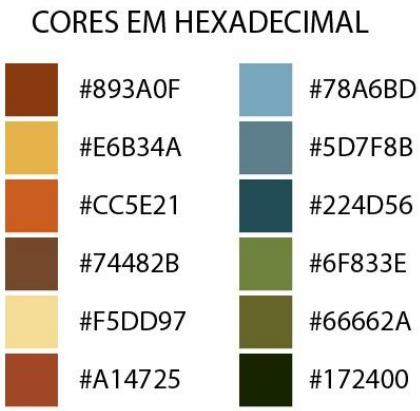
Fonte: Autoral

2.5. Definição de arte, mecânicas, som e level design

2.5.1. Arte do Jogo

A arte do jogo será baseada na cultura maia, assim, utilizamos elementos como as cores adotadas no calendário maia (azul, verde, laranja, amarelo, cinza, marrom e preto, todas com suas variações), objetos (vasos, etc), estátuas, o estilo de construção, etc.

Figura 11 - Paleta de Cores



Fonte: Autoral

Figura 12 - Calendário Maia

Fonte: Calendário Maia retirado do site The Christi Center

Figura 13 - Cena do Filme Apocalypto

Fonte: Imagem retirada do Filme Apocalypto (2007)

Figura 14 – Uncharted Golden Abyss

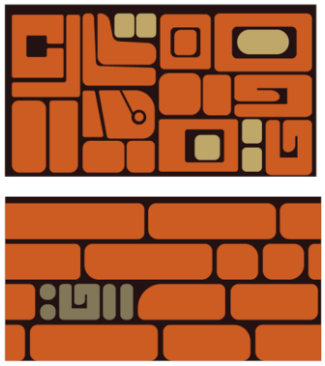
Fonte: Imagem retirada do jogo Uncharted: Golden Abyss (Naughty Dog, 2011)

Figura 15 – MuckUp do Jogo The Legend of Zelda

Fonte: Mockup do jogo The Legend of Zelda por Anders Gullmarsvik

Todo o cenário considerou os padrões existentes nas construções maias e nos seus estilos de desenho, como pode ser visto na Figura 16.

Figura 16 - Estilo das paredes

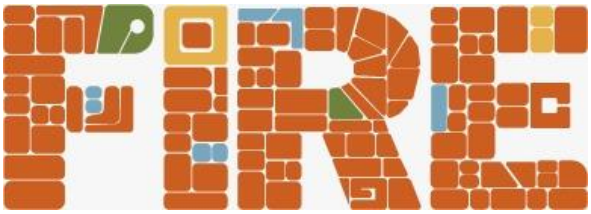


Fonte: Autoral

2.5.1.1. Fonte das letras

Uma parte do título de nosso jogo foi feito exclusivamente por um dos membros da equipe, combinando elementos que já tinha sido utilizado anteriormente para fazer o estilo das paredes do jogo, como mostra a Figura 17.

Figura 17 – Estilo de Fonte do Título I



Fonte: Autoral

A outra parte do título foi elaborada utilizando um estilo de fonte adquirido no site Dafont, no qual escolhemos uma fonte chamada de Stonecross (Figura 13).

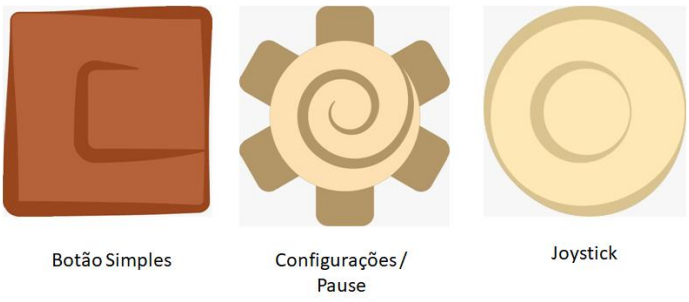
Figura 18 - Estilo de Fonte do Título II

Fonte:

2.5.1.2. Botões do jogo

Os botões do jogo também foram pensados para que mantivessem a concepção de arte do jogo, ou seja, o estilo maia. Como pode ser visto a seguir:

Figura 19 - Botões do Jogo

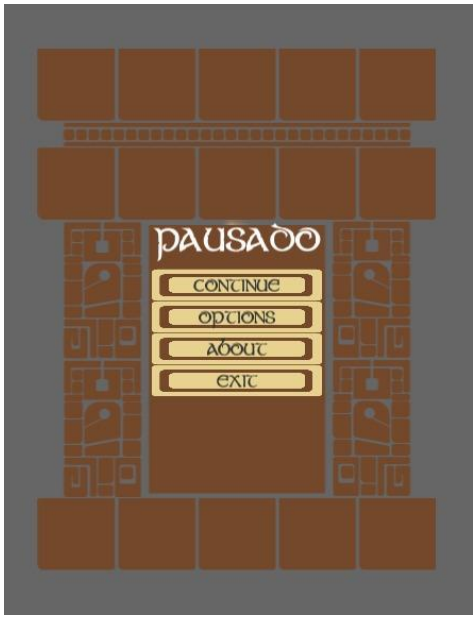


Fonte: Autoral

2.5.1.3. Telas de Menu

Optamos por não ter uma tela inicial, pois o jogo já começará com o personagem, e após uma introdução, o *gameplay* poderá ser iniciado. Entretanto, teremos a tela de *pause* do jogo, a qual se encontra abaixo:

Figura 20 - Tela de Pause

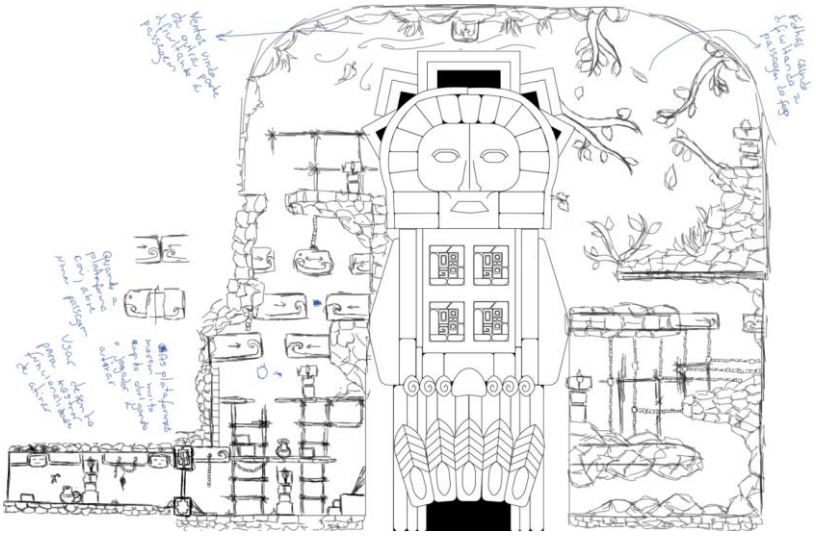


Fonte: Autoral

2.5.2. Level Design

Por ser um jogo de puzzle, o *level design* foi criado para que existissem elementos no caminho que precisassem ser evitados/utilizados e desafios a serem ultrapassados (Figura 21).

Figura 21 - Level Design



Fonte: Autoral

O jogo possui plataformas que ficarão se movimentando, as quais o jogador precisará passar dentro do tempo certo (Figura 22). Além disso, terá objetos como cordas, que precisam ser queimadas para aumentar o tempo de vida do Fogo ou para acionar algum mecanismo (Figura 23).

Figura 22 - Estilo de Plataforma

Fonte: Autoral

Figura 23 - Modelo da corda

Fonte: Autoral

Alguns objetos na cena não poderão ser tocados, pois dessa forma o Fogo irá morrer. Alguns exemplos desses objetos são os galhos de árvore, que por serem mais úmidos apagam o fogo ao contato (Figura 24), e elementos do cenário como as sustentações de madeira que foram criadas para não serem tocadas.

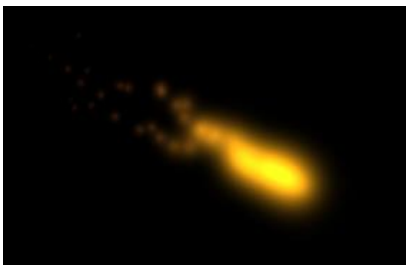
Figura 24 - Modelo da árvore

Fonte: Autoral

2.5.3. Mecânicas do jogo

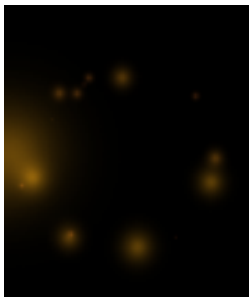
As mecânicas do jogo consistem em movimentação (cima, baixo, esquerda e direita), no qual o personagem principal fica flutuando o tempo todo, mas possui um tempo de vida predeterminado.

Figura 25 - Movimentação do Fogo



Fonte: Autoral

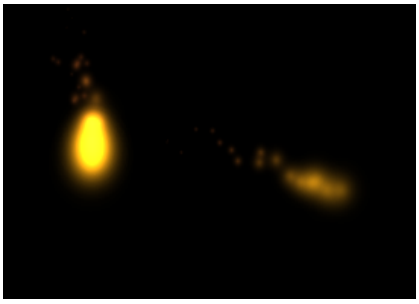
Figura 26 - Morte do Fogo



Fonte: Autoral

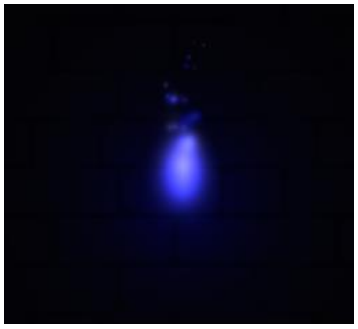
O personagem possui poderes como atirar bola de fogo e aumentar a sua velocidade, para poder passar por plataformas mais facilmente, sendo que a consequência da utilização desses poderes é a diminuição do seu tempo de vida.

Figura 27 - Fogo se movimentando



Fonte: Autoral

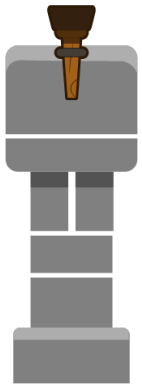
Figura 28 - Fogo aumenta a velocidade e muda de cor



Fonte: Autoral

Além disso, o jogo possui o sistema de *checkpoints* ao longo da fase, assim, o jogo somente será salvo se o jogador alcançar esses locais. Os *checkpoints* são tochas, que ao serem alcançadas pelo personagem também servem como fonte de renovação para o seu brilho, logo, se a chama estiver se extinguindo, ao alcançar a tocha mais próxima, o tempo de vida será renovado, conforme ilustram a Figura 29 e 30.

Figura 29 - Checkpoint desativado



Fonte: Autoral

Figura 30 - Checkpoint ativado

Fonte: Autoral

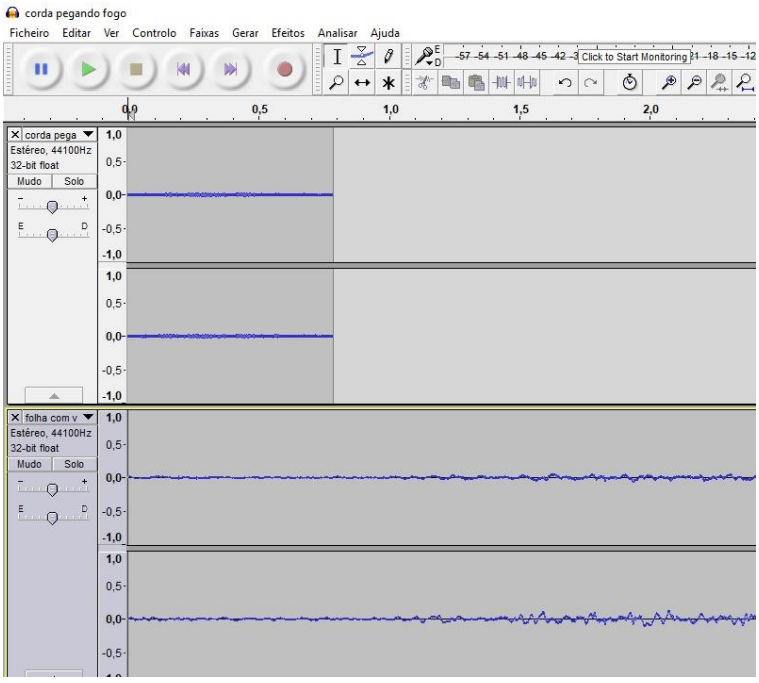
2.5.4. Áudio

O áudio é responsável pela maior parte da imersão que um jogo pode proporcionar, de tal modo, a escolha de todas as músicas e efeitos sonoros foi feita com demasiada atenção. Como não possuímos ninguém em nosso grupo que tenha afinidade com instrumentos musicais, optamos por utilizar o acervo de áudio de bibliotecas públicas, no caso, a biblioteca de áudios e músicas do *YouTube*.

2.5.4.1. Músicas

Em nosso jogo, iremos utilizar apenas a música *Enigmatic*, sendo que obtivemos permissão do músico Ben Sounds para utilizá-lo. Utilizamos o programa Audacity para fazer a mixagem da música (exemplo mostrado na figura 31).

Figura 31 - Edição de áudio no Audacity



Fonte: Autoral

2.5.4.2. Efeitos Sonoros

Buscamos efeitos sonoros para os seguintes eventos: atirar bolas de fogo, cordas queimando, colisão de pedras, fogo ao acender, folhas ao vento, entre outros. Todos os efeitos foram encontrados na biblioteca gratuita do YouTube

.

3 IMPLEMENTAÇÃO

Conforme o cronograma da Tabela 1, nossa implementação foi inicializada após a escolha definitiva do *Game Design* e da História que envolve o jogo, portanto concentrada e focada no jogo definitivo. As implementações dos nossos processos são exemplificadas e descritas na sequência.

3.1. Implementação de Sprite

Realizamos a implementação das Sprites no ambiente da Unity®. Assim, por não utilizarmos animações com base em Sprites, foram necessárias configurações para a conversão das Sprites para TileMap – uma ferramenta nova, disponibilizada pela Unity®. Essa ferramento permite a utilização de múltiplas Sprites como uma só composição, gerando somente um objeto – e configurações específicas para as Sprites utilizadas pelo Sistema de Partículas e pela HUD.

3.2. Controles do Personagem

Foi utilizado scripts, desenvolvidos na linguagem C# pelo grupo, para a realização dos controles do Personagem. Considerando que nossa plataforma de implementação é mobile, tornou-se necessário a criação de uma HUD especial para a interação com o jogo ser mais intuitiva.

Utilizamos uma forma visual de JoyStick (Figura 32) para a realização da interação com os controles de movimentação, e adotamos Botões (Figura 33) para a interação com o Menu de Pause (Figura 34) e as Mecânicas do jogo (Tópico 2.5.3).

Figura 32 - JoyStick

Fonte: Autoral

Figura 33 - Botões de Interação

Fonte: Autoral

Figura 34 - Menu de Pause

Fonte: Autoral

Todas as configurações envolvendo o personagem encontram-se no script Player.cs, Script 1 no Apêndice 1, no qual é possível encontrar todas as interações possíveis com o jogo.

Para as colisões e a movimentação, utilizamos como base scripts desenvolvidos junto à matéria de Física aplicada a jogos digitais, gerando objetos mais próximos dos comportamentos da física do mundo real.

As colisões são importantes para a queima dos objetos em cena e se o personagem Fogo está em condições de morte, configuradas nas funções específicas da Unity® – On Trigger Enter e Exit e On Collision Enter e Exit.

Para organização do script, utilizamos regions para comprimir o código e encontrar funções mais facilmente.Para configuração do JoyStick para mobile, utilizamos o Script InputManager.cs, Script 9 no Apêndice, utilizando conceitos de Matemática aplicada a Jogos Digitais.

3.3. Interações Automáticas

Utilizamos scripts de interação que não precisam de input's do jogador, portanto são administradas automaticamente conforme o fluxo do jogo.

Como base das plataformas que se movimentam tem-se o script MovingPlataform.cs, Script 8 no Apêndice, que consegue tornar objetos em plataformas com movimento, sendo necessário somente configurações de início e parada, facilitando o seu uso em diversas plataformas.

Os scripts como ControlFlame.cs e ControlLight.cs, Scripts 2 e 3 respectivamente no Apêndice, são utilizados para a realização do controle automático do tamanho da Chama do personagem e do tamanho da Luz que o personagem emana, sendo necessário somente a configuração do tempo de vida do personagem e do seu tempo atual.

Para o desenvolvimento desses scripts mais complexos, tomamos como base conceitos abordados na matéria de Programação orientada a objetos, utilizando métodos e parâmetros para o controle de todo o jogo.

3.4. Objetos Inflamáveis e Interativos

No jogo, utilizamos cordas para a interação direta com o personagem Fogo, que consegue queimá-las. Para a configuração de tal foram utilizados os scripts Rope.cs, RopeStart.cs e RopeEnd.cs, Scripts 4, 6 e 5 respectivamente no Apêndice.

Criamos uma função chamada de GenerateRope em RopeStart.cs para o preenchimento automático com Prefabs de Corda de uma distância a outra, diminuindo o tempo para organização das cenas.

Utilizamos como Checkpoint tochas, que foram configuradas com o script Checkpoint.cs, Script 12 no Apêndice. Ao passar pela tocha, ela se acende e marca

a posição no checkpoint do jogador, portanto ao morrer a partir desse ponto, ele retornará ao último checkpoint cruzado.

3.5. UI

Para controlar os painéis no canvas, foi criado um script para realizar a manutenção da tela, chamado de GameUI.cs, Script 10 no Apêndice.Nesse script encontra-se todas as configurações de tela e suas interações.

3.6. Câmera

Para implementação da câmera foi utilizado um recurso da Unity® chamado de Cinematic Camera, que segue o personagem e suaviza os movimentos da câmera.

3.7. Level Design

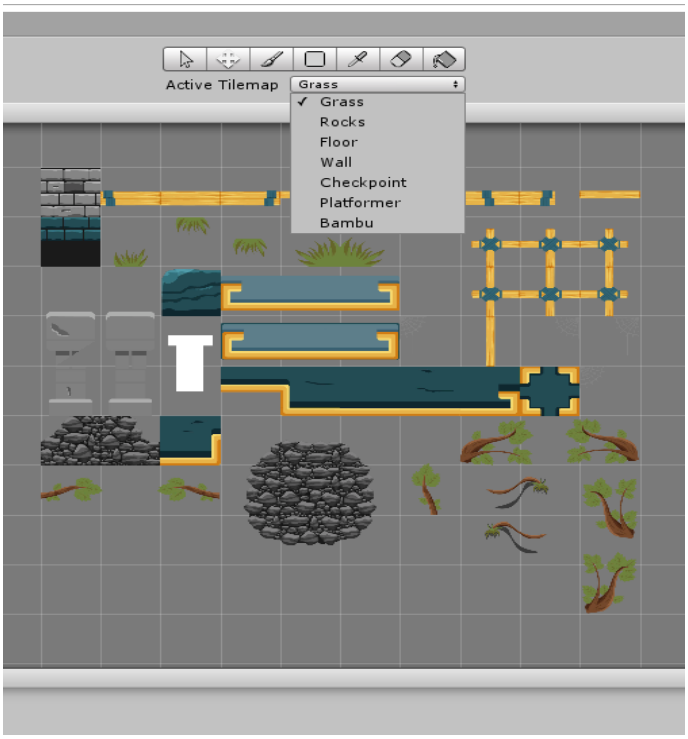
Com o documento de Level Design definido (como explicado em “2.5.2 Level Design”), o processo de implementar as salas da fase no Unity® foi simples.

3.7.1. Design das Salas

O cenário do jogo foi desenvolvido através de um recurso disponível na Unity®, o TileMap, próprio para o desenvolvimento de cenários 2D utilizando um grid com a possibilidade de escolher o tamanho desejado. Em nosso caso, utilizamos 200px por 200px, mantendo a qualidade da arte, e após definir o tamanho padrão do grid, criamos uma paleta de tiles para pintarmos dentro do grid.

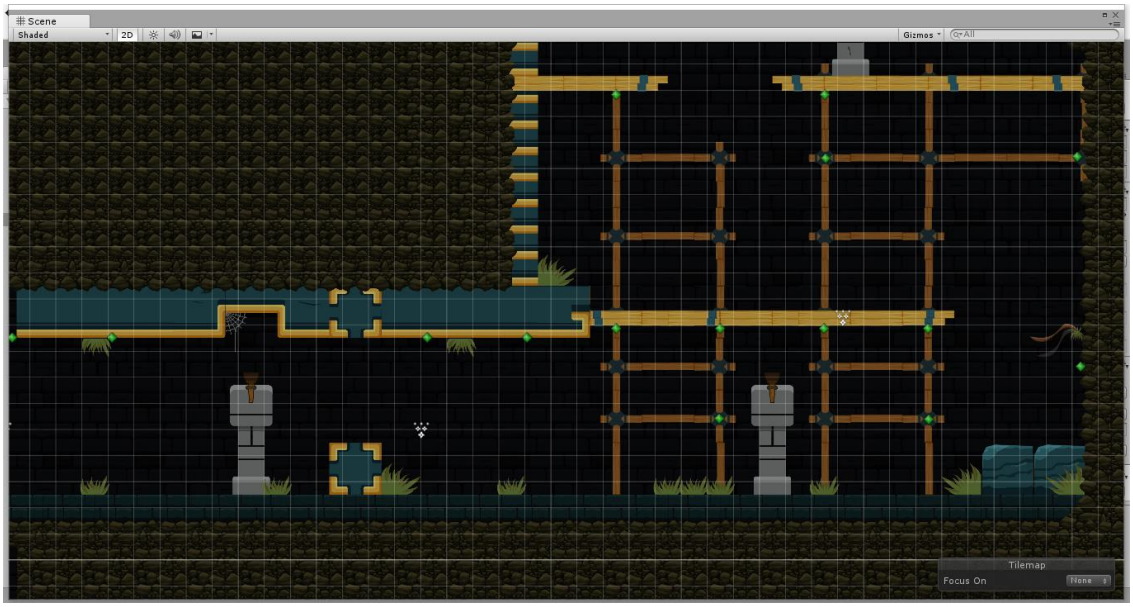
O TileMap também permite a criação de múltiplas layer (camadas) facilitando a montagem do cenário do jogo e dentro da paleta de tiles podemos selecionar qual a layer (camada) que desejamos pintar/preencher com a arte.

Figura 35 - Paleta de Tiles



Fonte: Autoral

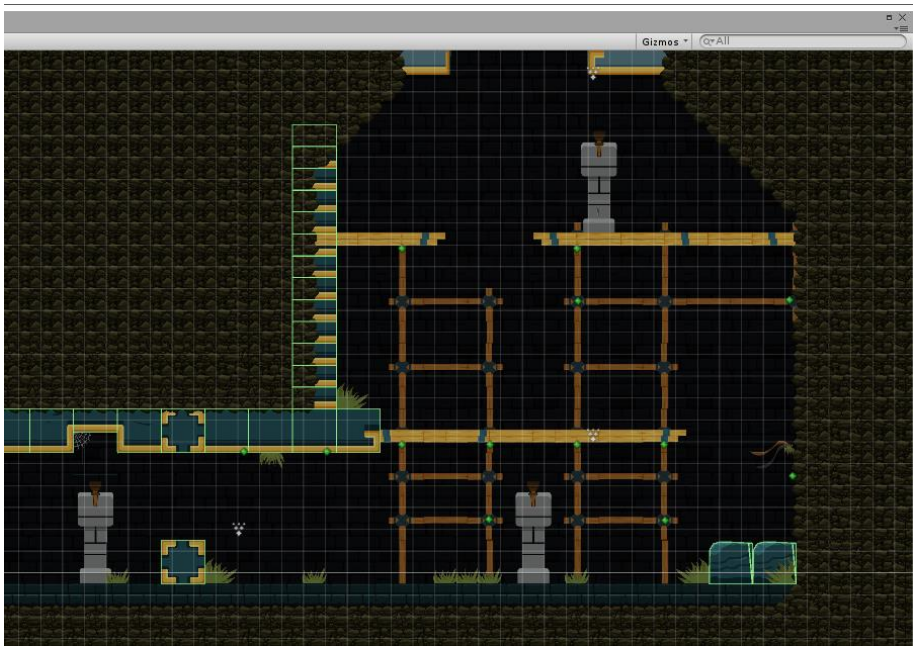
Figura 36 – TileMap preenchido com os tile da paleta de tiles.



Fonte: Autoral

Depois de montado o level design dentro do TileMap, utilizando a janela de Hyerarchy e a janela Inspector, adicionamos as colisões na arte.

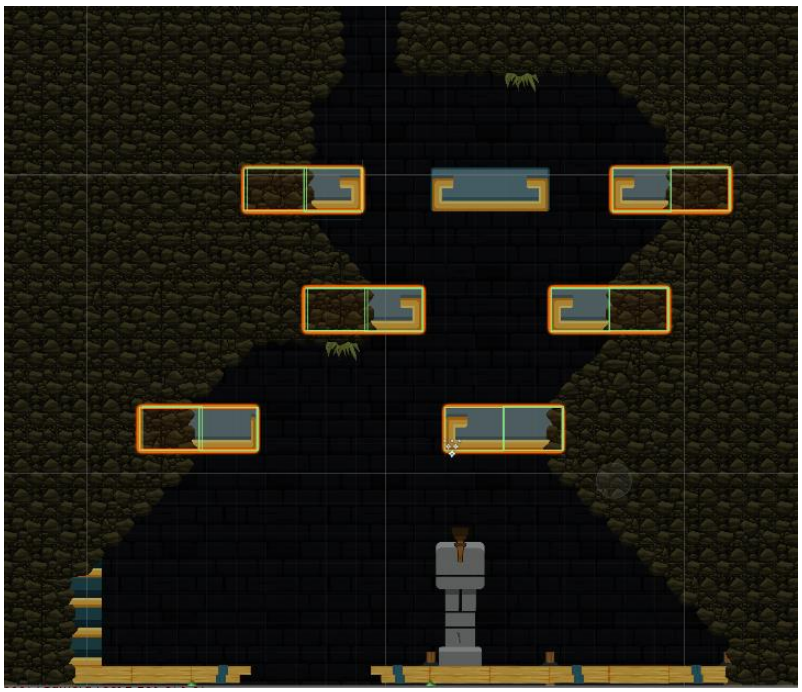
Figura 37 - Colliders inseridos no level design.



Fonte: Autoral

Com o level design montado, adicionamos os desafios do jogo, por exemplo as plataformas que se movem e até mesmo os objetos queimáveis e os checkpoints.

Figura 38 - Inclusão das plataformas de movimentos com collider.



Fonte: Autoral

4 RESULTADOS

O jogo foi testado pelos integrantes do grupo e por colegas de classe durante aulas destinadas a isso e posteriormente, disponibilizado para *download* no Google PlayStore. Todas as versões testadas e analisadas foram de grande importância para verificar e corrigir os eventuais problemas encontrados, além de ajudar na melhoria do jogo em si.

4.1. Protótipo

A primeira versão do jogo ficou disponível apenas para os desenvolvedores do grupo. Com essa versão, apenas era apresentada a movimentação do personagem, algumas artes iniciais (que foram trocadas posteriormente) e a primeira mecânica que era o momento onde o personagem alcança uma tocha e salva um *checkpoint*.

Figura 39 - Checkpoint e artes iniciais.

Fonte: Autoral

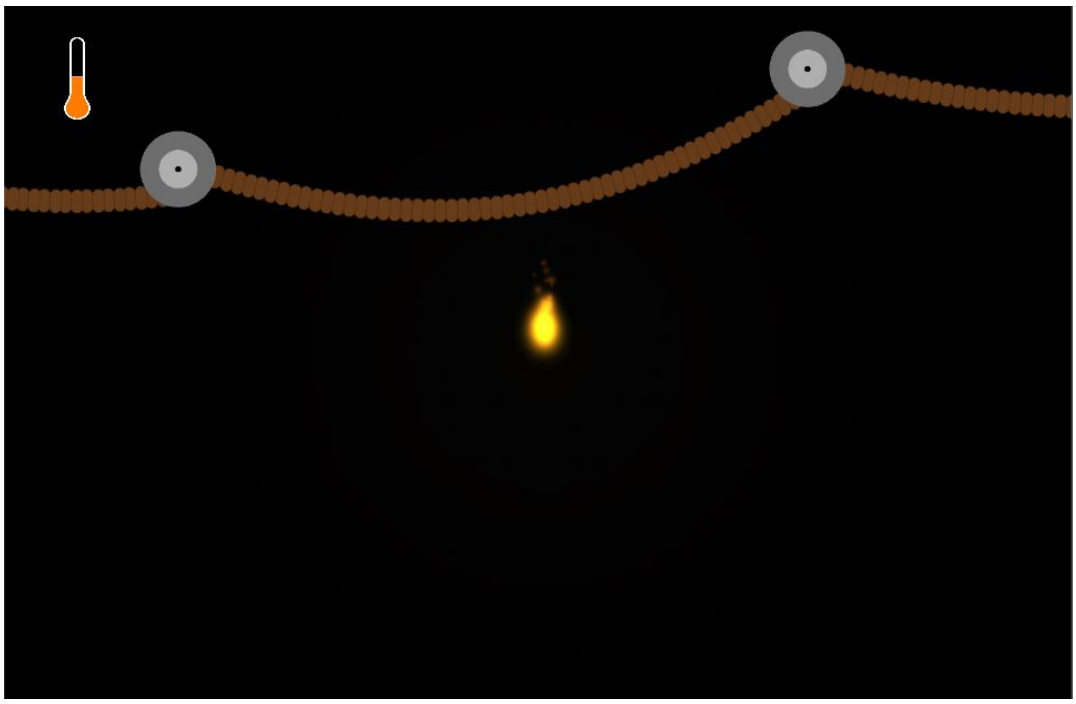
4.2. Versão Alpha

A versão alpha do jogo (Figura 39) continuou sendo testada apenas pelos desenvolvedores do projeto, porém, através de uma apresentação na matéria de Jogos Digitais para Consoles, o jogo foi demonstrado ao público presente na classe. Essa versão já apresentava objetos inflamáveis que forneciam energia ao fogo e serviam de interação com o ambiente.

Outro item adicionado foi uma barra de vida do personagem, apresentada em formato de *timer*, assim mostrava que com o tempo o personagem ia perdendo força gradualmente até se apagar. Antes dessa alteração, o jogador podia passar por qualquer desafio sem se importar com o tempo gasto no percurso.

Essa apresentação foi muito importante, pois foi um momento no qual as pessoas puderam opinar e mostrar uma visão de quem está de fora do projeto, sugerindo melhorias e apontando possíveis falhas.

Figura 40 - Timer implementado e objetos inflamáveis.



Fonte: Autoral

4.3. Versão Beta

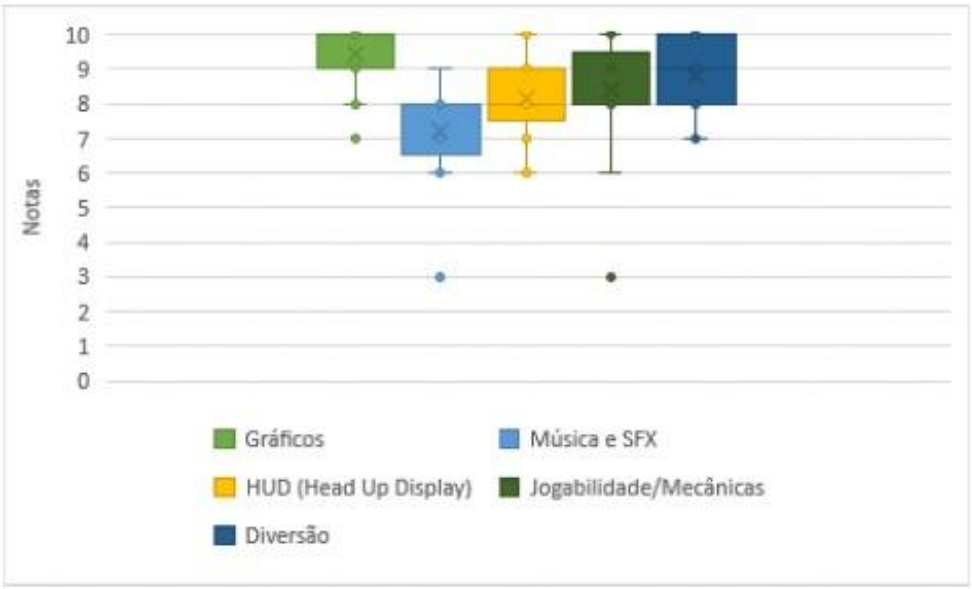
A versão beta foi a primeira versão disponível para ser jogada pelo público. Primeiramente disponibilizada em uma aula de Jogos Digitais para Consoles, onde qualquer aluno ali presente podia jogar e notar a evolução das mecânicas e jogabilidade, além de darem notas para pontos específicos do jogo, conforme mostra figura 41.

Essa versão também foi a primeira a ser disponibilizada no Google PlayStore, sendono início disponibilizada apenas para amigos através de um link compartilhado. Na versão Beta, a maioria das artes foram polidas e modificadas

para se adequar melhor ao tema do jogo. Também foi adicionada a música de fundo e ainda em versão beta através de um update, outros efeitos sonoros.

Uma grande adição nessa versão foi a mecânica de plataformas que se movimentam, trazendo mais um desafio ao decorrer da fase.

Figura 41 - Notas dadas pelos alunos que testaram o Beta.



Fonte: Autoral

4.4. Google Playstore

O jogo foi oficialmente adicionado na Google PlayStore em 23 de novembro de 2017, em uma conta disponibilizada gratuitamente pelo professor Gustavo Carvalho Gomes de Abreu da FATEC Americana.

Como o link para download do jogo foi compartilhado apenas com um pequeno grupo escolhido pelos desenvolvedores, não foi realizado um levantamento de downloads e desinstalações do jogo. O link para download do jogo se encontra nas referencias bibliográficas.

Figura 42 - Jogo Finalizado 01

Fonte: Autoral

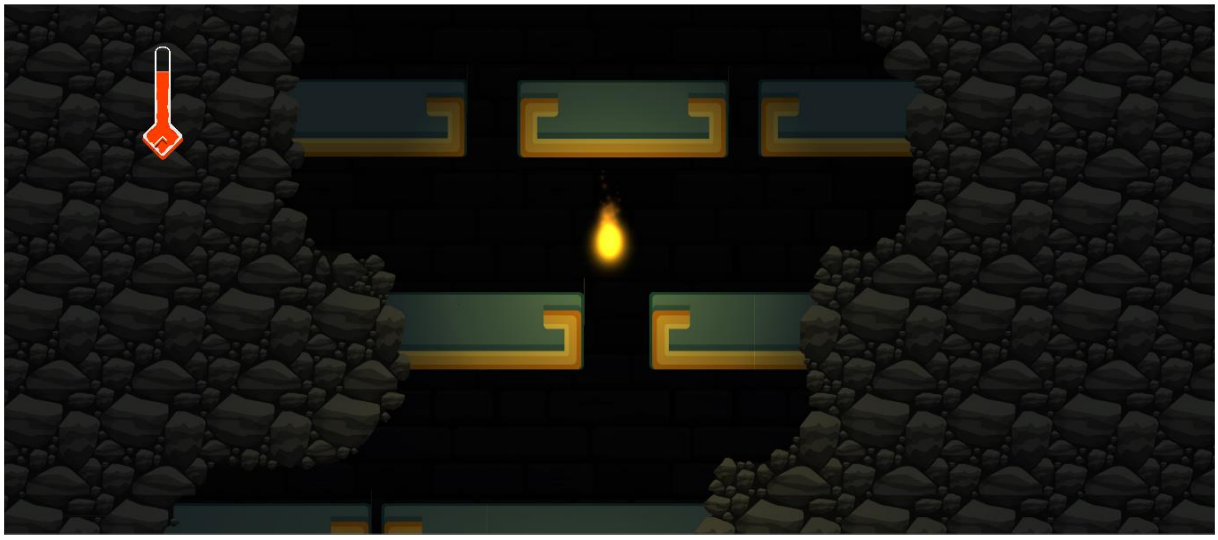
Figura 43 - Jogo Finalizado 02

Fonte: Autoral

Figura 44 - Jogo Finalizado 03

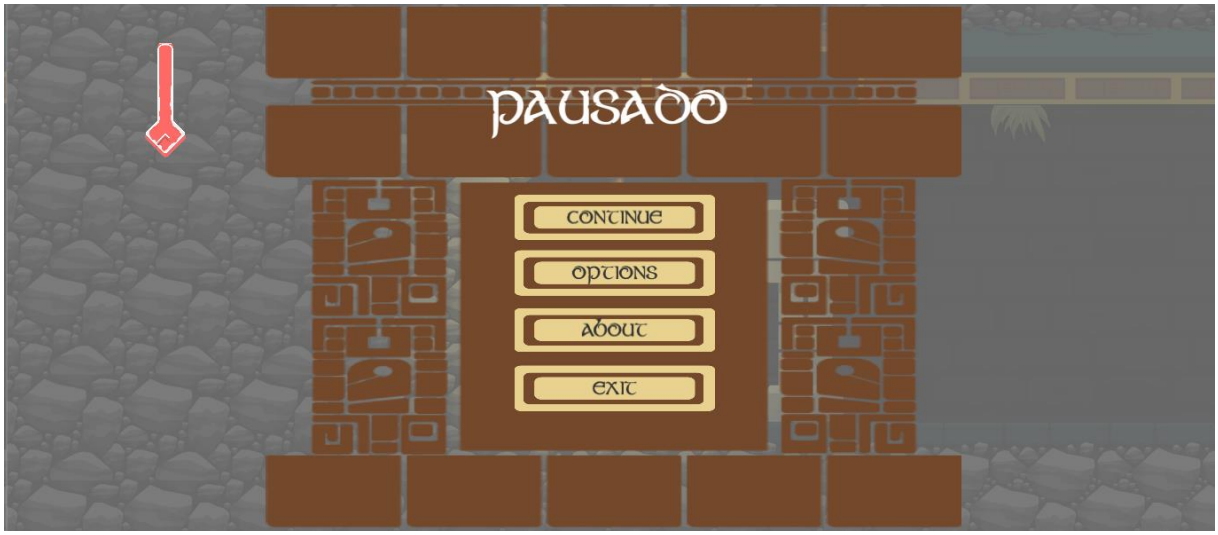
Fonte: Autoral

Figura 45 - Jogo Finalizado 05



Fonte: Autoral

Figura 46 – Jogo Finalizado: Menu de Pause



Fonte: Autoral

5 CONSIDERAÇÕES FINAIS

No começo, criar um jogo do zero parecia uma ideia um tanto quanto assustadora e talvez preocupante, ainda mais se considerarmos o tempo que teríamos para tal feito. Logo de início, várias ideias surgiam e precisavam ser revistas pelo grupo, justamente devido ao nosso fator limitante, o tempo.

Quanto mais chegávamos perto da data que precisaríamos publicar o jogo, maior a preocupação se iria realmente dar tempo. Durante todo o desenvolvimento foi preciso testar as ideias, ver o que realmente funcionaria, o que não ficou bom e analisar se teríamos tempo para melhorias. As aulas de Jogos para Consoles ajudaram muito nesse quesito, pois mostrávamos o jogo para o público e eles já nos davam o *feedback* do que estava bom e até ideias sobre como melhorar, além é claro, de elogios pelo que tínhamos feito.

Estamos satisfeitos com o resultado de nosso jogo, entretanto, algumas ideias que possuímos para a continuação não puderam ser implementadas ainda, por exemplo, adicionar uma transformação no elemento fogo, para o elemento gelo ou até mesmo água. Entretanto, a ideia é continuar desenvolvendo o jogo e adicionando atualizações com mecânicas novas.

Podemos dizer que realizar esse projeto foi, de longe, bem enriquecedor. Somente durante o projeto percebemos como é difícil gerenciar tantas coisas ao mesmo tempo, como tudo pode dar certo e com a mesma facilidade dar errado, e que a corrida contra o relógio é a maior motivação que alguém pode ter. Para alguém que pretende trabalhar nessa área, realizar esse trabalho mostrou como funciona de verdade um projeto de jogo, e isso, foi essencial para nosso crescimento como profissionais.

REFERENCIAS BIBLIOGRAFICAS

Apocalypto. 2007. Disponível em: <http://unrealitymag.com/movies/mel-gibsons-apocalypto-a-masterpiece-in-need-of-a-revival/>. Acesso em: 19 de dezembro de 2017.

Audacity. Disponível em: <http://www.audacityteam.org/download/>. Acesso em: 19 de dezembro de 2017.

Badland. Frogmind, 2013.

BenSound. Disponível em: <https://www.bensound.com/royalty-free-music/track/enigmatic/> . Acesso em: 29 de setembro de 2017.

Calendário Maia. Disponível em: <http://christicenter.org/2013/02/ancient-aztec-perspective-on-death-and-afterlife/>. Acesso em: 18 de dezembro de 2017.

Fonte Stonecross. Da Font. Disponível em: <https://www.dafont.com/pt/stonecross.font>. Acesso em: 19 de dezembro de 2017.

Inkscape. Disponível em: <https://inkscape.org/pt-br/baixar/>. Acesso em: 19 de dezembro de 2017.

Limbo. Playdead, 2010.

Pesquisa Game Brasil 2017. SIOUX. Disponível em: <https://www.pesquisagame-brasil.com.br/>. Acesso em: 21 de novembro de 2017.

Talbot's Odyssey. Studio Mini Boss, 2010.

The Fire Codex. Lit Dreams, 2017. Disponível em: <https://play.google.com/store/apps/details?id=com.Fatec.TheFireCodex>. Acesso em: 19 de dezembro de 2017.

Trello. Disponível em: <https://trello.com/>. Acesso em: 19 de dezembro de 2017.

Uncharted: Golden Abyss. Naughty Dog, 2011. Disponível em: <https://www.playstation.com/en-us/games/uncharted-golden-abyss-psvita/>. Acesso em: 19 de dezembro de 2017.

Unity. Disponível em: <https://unity3d.com/pt>. Acesso em: 19 de dezembro de 2017.

Youtube. Disponível em: <https://www.youtube.com/audiolibrary/soundeffects/>. Acesso em: 25 de setembro de 2017.

Zelda Mockup. Anders Gullmarsvik. The Legend of Zelda. Disponível em: <https://twitter.com/itchabop/status/921085474399191042>. Acesso em: 19 de dezembro de 2017.

APENDICE 1 – CÓDIGO FONTE

Script 1 – Player.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player : MonoBehaviour
{
    private Animator _Animator;
    private Rigidbody2D _Rigidbody2D;
    private ControlFlame _Flame;
    private ControlLight _Light;
    private ControlCollider _Col;
    private InputManager _Inp;

    private int vPlayer = 4,                    //Velocidade Player
                vFireBall = 300, movSide = 1,    //Velocidade e Lado de Arremesso de
Bola de Fogo
                cCont = 0, tCont = 20;          //Count para Reset de Colisão

    private bool isInsideTorch = false,                          //Está    dentro    da
Tocha
                 tRight = false, tLeft = false,                //Colisões em ambos os
lados
                 canFireBall = true, canFast = true;      //Keys


    private  float  materialValue  =  1f,  fireballValue  =  3f,  fastValue  =  0.15f,
//Controle de Energia
                  sizeMaxF    =    4.5f,    sizeMaxL    =    10,    sizeMaxC    =    0.7f;
//Tamanho máximo para Fogo e Luz

    private string tRTag = string.Empty, tLTag = string.Empty;  //Tag para checar na
colisão de ambos os lados

    public float timeTotal = 45, timeCurrent = 0;   //Tempo de Vida

    public  bool  animDeath  =  false,  isFast  =  false,  canInteract  =  true,  isAndroid  =
false, returned = false,    //Status do Player
                FireBallInput       =       false,       FastInput       =       false;
//Inputs pelos Botões em Tela

    public Vector3 pcheck = Vector3.zero;  //Controle do Checkpoint

    void Start()
    {
        _Animator = GetComponent<Animator>();
        _Rigidbody2D = GetComponent<Rigidbody2D>();
        _Flame = GetComponentInChildren<ControlFlame>();
        _Light = GetComponentInChildren<ControlLight>();
        _Col = new ControlCollider { _Collider = GetComponent<CircleCollider2D>() };

        if (!returned)
        {
            pcheck = transform.position;
        }

        _Col.Initialize();
```

```
#if UNITY_ANDROID
             isAndroid = true;

             _Inp = GameObject.FindObjectOfType<InputManager>();

#endif

    }

    void Update()
    {

        #region Reset

        if (Input.GetKeyDown(KeyCode.R))
        {
            Death();
        }

        #endregion

        #region Throw FireBall

        if ((Input.GetAxisRaw("Fire1") > 0 || FireBallInput) && canFireBall &&
canInteract)
        {
            GameObject g = Instantiate(Resources.Load("Prefabs/Objects/FireBall"), new
Vector2(transform.position.x,     transform.position.y),     Quaternion.identity)     as
GameObject;
            Rigidbody2D r = g.GetComponent<Rigidbody2D>();
            if (_Rigidbody2D.velocity != Vector2.zero)
            {
                r.AddForce(_Rigidbody2D.velocity.normalized * vFireBall);
            }
            else r.AddForce(new Vector2(vFireBall * movSide, g.transform.position.y));

            if (!isInsideTorch) timeCurrent += fireballValue;

            canFireBall = false;
            FireBallInput = false;
        }

        if (Input.GetAxisRaw("Fire1") == 0 && !canFireBall) canFireBall = true;

        #endregion

        #region Fast

        if ((Input.GetAxisRaw("Fire2") > 0 || FastInput) && canFast && canInteract)
        {
            isFast = true;

            _Flame.ChangeColor(1);
            _Light.ChangeColor(1);

            vPlayer = 8;

            canFast = false;
        }

        if (Input.GetAxisRaw("Fire2") == 0 && !FastInput && !canFast && canInteract)
        {
```

```
            isFast = false;
            canFast = true;

            _Flame.ChangeColor();
            _Light.ChangeColor();

            vPlayer = 4;

        }

        #endregion

        #region Energy Control

        if (canInteract)
        {
            if (!isInsideTorch)
            {
                timeCurrent += Time.deltaTime;

                if(!canFast) timeCurrent += fastValue;
            }

            _Flame.EnergyControl(Percentage(), true);
            _Light.EnergyControl(Percentage(), true);
            _Col.EnergyControl(Percentage(), true);

        }

        #endregion

        #region Reset Collision

        if (tLeft || tRight)
        {
            CheckCollisionCount();
            cCont++;
        }

        #endregion

        #region Death

        if (timeCurrent >= timeTotal)
        {
            Death();
        }

        /*if (animDeath && _Animator.GetBool("Player_isDead"))
        {
            _Animator.SetBool("Player_isDead", false);
            animDeath = false;
            canInteract = true;
        }*/

        #endregion

    }

    public void FixedUpdate()
    {
```

```
        #region Movement

        if (canInteract)
        {
            if (!isAndroid)
            {
                _Rigidbody2D.velocity = new Vector2(Input.GetAxis("Horizontal") * 
vPlayer, Input.GetAxis("Vertical") * vPlayer);

                //Lado de Arremesso de Bola de Fogo
                if (Input.GetAxis("Horizontal") < 0)
                    movSide = -1;
                if (Input.GetAxis("Horizontal") > 0)
                    movSide = 1;
            }
            else
            {

                _Rigidbody2D.velocity = new Vector2(_Inp.Axis.x * vPlayer, _Inp.Axis.y 
* vPlayer);

                //Lado de Arremesso de Bola de Fogo
                if (_Inp.Axis.x < 0)
                    movSide = -1;
                if (_Inp.Axis.x > 0)
                    movSide = 1;
            }
        }
        else
            _Rigidbody2D.velocity = Vector2.zero;

        #endregion

    }

    public void CheckCollisionCount()
    {

        if (tLeft && tRight && (tLTag == "MovePlataform" || tRTag == "MovePlataform"))
        {
            cCont = 0;
            tLeft = false;
            tLTag = string.Empty;
            tRight = false;
            tRTag = string.Empty;
            Death();
        }
        else if (cCont == tCont)
        {
            cCont = 0;

            tLeft = false;
            tLTag = string.Empty;
            tRight = false;
            tRTag = string.Empty;
        }

    }

    private void Death()
    {
        timeCurrent = timeTotal;
```

```
        _Animator.SetBool("Player_isDead", true);
        _Flame.Death();
        _Light.Death();
        _Col.Death();
        canInteract = false;
    }

    private float Percentage()
    {
        return timeCurrent / timeTotal;
    }

    void OnTriggerEnter2D(Collider2D col)
    {

        if (col.tag == "Torch")
        {
            timeCurrent = 0;
            isInsideTorch = true;
        }

    }

    void OnTriggerExit2D(Collider2D col)
    {

        if (col.tag == "Torch")
        {
            isInsideTorch = false;
        }

    }

    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.tag == "Material")
        {
            if (timeCurrent > 0) timeCurrent -= materialValue;
        }
        else
        {
            foreach (ContactPoint2D cp in col.contacts)
            {
                Vector2 pos = cp.point - new Vector2(transform.position.x,
transform.position.y);

                if (pos.y >= 0.6f && pos.x > -0.1f && pos.x < 0.1f) Death();
            }

        }
    }

    void OnCollisionStay2D(Collision2D col)
    {
        foreach (ContactPoint2D cp in col.contacts)
        {
            Vector2 pos = cp.point - new Vector2(transform.position.x,
transform.position.y);

            if (pos.x <= 0.5f && pos.x > 0.2)
            {
                tRight = true;
```

```
                tRTag = col.gameObject.tag;
                cCont = 0;
            }
            else if (pos.x >= -0.5f && pos.x < -0.2f)
            {
                tLeft = true;
                tLTag = col.gameObject.tag;
                cCont = 0;
            }
        }
    }

    public class ControlCollider
    {
        public float sizeInitial, sizeMin = 0.3f;

        public CircleCollider2D _Collider { get; set; }

        public void Initialize()
        {
            sizeInitial = _Collider.radius;
        }

        public void EnergyControl(float per, bool percent)
        {
            _Collider.radius = sizeInitial - ((sizeInitial - sizeMin) * per);
        }

        public void EnergyControl(float size)
        {
            _Collider.radius = size;
        }

        /*public void Born()
        {
            _Collider.radius = sizeInitial;
        }*/

        public void Death()
        {
            _Collider.radius = sizeMin;
        }
    }

}
```

Script 2 - ControlFlames.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ControlFlame : MonoBehaviour {

       private ParticleSystem _Particle;

       private float        sizeInitial, sizeMin = 1,
                                lifeTInitial, lifeTMin = 0.4f,
                                radiusInitial, radiusMax = 0.8f,
                                arcSInitial, arcSMax = 0.06f;
```

```
    private ParticleSystem.MinMaxCurve velocityInitial, velocityMin = new
ParticleSystem.MinMaxCurve(0f, 0f);

    private ParticleSystem.MinMaxGradient colorInitial, colorBlue = new
ParticleSystem.MinMaxGradient(new Color(0f,0f,1f), new Color(1f,1f,1f,0f));

    public void Start () {

              _Particle = GetComponent<ParticleSystem> ();

              sizeInitial = _Particle.main.startSize.constant;

              lifeTInitial = _Particle.main.startLifetime.constant;

              radiusInitial = _Particle.shape.radius;

              arcSInitial = _Particle.shape.arcSpread;

              velocityInitial = _Particle.velocityOverLifetime.y;

         colorInitial = _Particle.colorOverLifetime.color;

       }

       public void EnergyControl(float per, bool percentage)
    {
              var main = _Particle.main;

              main.startSize = sizeInitial - ((sizeInitial - sizeMin) * per);
              main.startLifetime = lifeTInitial - ((lifeTInitial - lifeTMin) * per);
    }

       public void EnergyControl(float size)
       {
              var main = _Particle.main;

              main.startSize = size;
       }

    public void ChangeColor(int colorN = 0)
    {
        var color = _Particle.colorOverLifetime;

        switch (colorN)
        {
            case 0:
                color.color = colorInitial;
                break;
            case 1:
                color.color = colorBlue;
                break;
            default:
                color.color = colorInitial;
                break;
        }

    }

       public void Death()
       {
              var main = _Particle.main;
              var shape = _Particle.shape;
```

```
            var velocity = _Particle.velocityOverLifetime;

            shape.arcSpread = arcSMax;
            shape.radius = radiusMax;
            velocity.y = velocityMin;
            main.startSize = sizeMin;
            main.startLifetime = lifeTMin;
    }
}
```

Script 3 – ControlLight.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ControlLight : MonoBehaviour {

    private Light _Light;

    private float sizeInitial, sizeMin = 3;

    private Color colorInitial = new Color(1f, 0.884f, 0.31f), colorBlue = new
Color(0f, 0f, 1f);

    public void Start () {
        _Light = GetComponent<Light>();
        sizeInitial = _Light.range;
        colorInitial = _Light.color;
    }

    public void EnergyControl(float per, bool percent)
    {
        _Light.range = sizeInitial - ((sizeInitial - sizeMin) * per);
    }

    public void EnergyControl(float size)
    {
        _Light.range = size;
    }

    public void ChangeColor(int colorN = 0)
    {
        switch (colorN)
        {
            case 0:
                _Light.color = colorInitial;
                break;
            case 1:
                _Light.color = colorBlue;
                break;
            default:
                _Light.color = colorInitial;
                break;
        }

    }

    public void Death()
    {
        _Light.range = sizeMin;
    }
}
```

Script 4 – Rope.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rope : MonoBehaviour
{

    private Rope _Previous, _Next;
    private Animator _Animator;

    private bool previous = false, next = false;
    public GameObject PreviousRope { get; set; }
    public GameObject NextRope { get; set; }

    public bool isBurning = false;
    public int count = 0;

    void Start()
    {
        if (_Previous != null)
        {
            previous = true;
            _Previous = PreviousRope.GetComponent<Rope>();
        }

        if (_Previous != null)
        {
            next = true;
            _Next = NextRope.GetComponent<Rope>();
        }
        _Animator = GetComponent<Animator>();

    }

    void Burning()
    {
        _Animator.SetBool("Rope_isBurning", true);
    }

    void Death()
    {
        Destroy(gameObject);

    }

    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.transform.tag == "Player")
        {
            Death();
        }

        if (col.transform.tag == "FireBall")
        {
            Burning();
        }
    }
```

```
}
```

Script 5 – RopeEnd.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RopeEnd : MonoBehaviour {

	private float distanceFromChainEnd = -0.3f;

	public void ConnectRopeEnd(Rigidbody2D end){

		HingeJoint2D joint = gameObject.AddComponent<HingeJoint2D> ();

		joint.autoConfigureConnectedAnchor = false;

		joint.connectedBody = end;

		joint.anchor = Vector2.zero;

		joint.connectedAnchor = new Vector2 (0f, -distanceFromChainEnd);
	}
}
```

Script 6 – RopeStart.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RopeStart : MonoBehaviour {

	private Rigidbody2D lastConnection;
	private GameObject _Rope;

    private bool hEnd = true;
	private int qtdRopes = 10;

    public RopeEnd end;
    public int qtdRopesP = 0;

    void Start () {

		_Rope = Resources.Load ("Prefabs/Objects/Rope") as GameObject;
		SpriteRenderer sprite = _Rope.GetComponent<SpriteRenderer> ();

		lastConnection = GetComponent<Rigidbody2D>();

        if (end == null)
        {
            hEnd = false;

            qtdRopes = qtdRopesP;
        }
        else
```

```
        {
                        float     s     =     (sprite.bounds.size.x     /     2)     +
_Rope.GetComponent<HingeJoint2D>().connectedAnchor.magnitude;

            float v = (end.transform.position - transform.position).magnitude;

                        qtdRopes += qtdRopesP;

            qtdRopes += Mathf.RoundToInt(v / s);
         }

         GenerateRope();
       }

       void GenerateRope(){

              Rigidbody2D previousRB = lastConnection;

         Vector3 tPosition = transform.position;

         for (int i = 0; i < qtdRopes; i++) {

            GameObject link = new GameObject();

                        if (hEnd) {
                               Vector3    endPosition    =    end.transform.position    -
transform.position;
                               link  =  Instantiate  (_Rope,  new  Vector3  (tPosition.x  +
((endPosition.x / qtdRopes) * i), tPosition.y + ((endPosition.y / qtdRopes) * i),
tPosition.z), Quaternion.identity, transform);

                        }
                        else   link   =   Instantiate(_Rope,   new   Vector3(tPosition.x,
tPosition.y - (0.5f * i) , tPosition.z), Quaternion.identity, transform);

            HingeJoint2D joint = link.GetComponent<HingeJoint2D> ();

                        joint.connectedBody = previousRB;

                        if (i < (qtdRopes - 1)) {
                               previousRB = link.GetComponent<Rigidbody2D> ();
                        } else if (hEnd) {
                               end.ConnectRopeEnd (link.GetComponent<Rigidbody2D> ());
                        }
              }

       }
}
```

Script 7 – Parallaxing.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Parallaxing : MonoBehaviour {

       public  Transform[]  backgrounds;  //  Array  (list)  of  all  the  back-end
foregrounds to be parallaxed.
       private float[] parallaxScales; // The proportion of the camera's movimente
```

```
to move  the backgrounds by.
	public float smoothing = 1f;	// How smooth the parallax is going to be. Make sure to set this above 0.

	private Transform cam;			// Reference to main cameras transfom.
	private Vector3 previousCamPos; // The positon of the Camera in the previous frame.

	void Awake(){
		// Set up camera the reference.
		cam = Camera.main.transform;
	}
	// Use this for initialization
	void Start () {
		// The previous frame had the current frame's camera positon.
		previousCamPos = cam.position;
		parallaxScales = new float[backgrounds.Length];

		// Asigning coresponding parallaxScales
		for (int i = 0; i < backgrounds.Length; i++) {
			parallaxScales [i] = backgrounds [i].position.z * -1;
		}
	}

	// Update is called once per frame
	void Update () {
		// For each Background
		for (int i = 0; i < backgrounds.Length; i++) {
			// The parallax is the opposite of the camera movement because the previous frame multiplied by scale
			float parallax = (previousCamPos.x - cam.position.x) * parallaxScales[i];

			// Set a target x position which is the current position plus the parallax
			float backgroundTargetPositionX = backgrounds[i].position.x + parallax;

			// Create a target position which is the backgorund's current position with it's target x position
			Vector3 backgroundTargetPosition = new Vector3 (backgroundTargetPositionX, backgrounds[i].position.y, backgrounds[i].position.z);

			// Fade between current position and the target position using lerp
			backgrounds[i].position = Vector3.Lerp (backgrounds[i].position, backgroundTargetPosition, smoothing * Time.deltaTime);
		}

		// Set the previousCamPos to the Camera's position at the and of the frame
		previousCamPos = cam.position;
	}
}
```

Script 8 – MovingPlataform.cs

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;

public class MovingPlataform : MonoBehaviour {

       public GameObject _closePlataform;
    private SpriteRenderer iSprite, fSprite;
    public Vector3 finalPosition;

       private Vector3 initialPosition, difPosition;

    private float initialIDirection, initialVDirection;

       private bool closeP = false, indo = false, volta = false;
       private float iVelo = 1.8f, vVelo = 0.4f;


       void Start () {

        iSprite = GetComponent<SpriteRenderer>();

             if (_closePlataform != null) {
            if (_closePlataform.GetComponent<SpriteRenderer>() != null)
            {
                closeP = true;

                finalPosition = _closePlataform.transform.position;

                fSprite = _closePlataform.GetComponent<SpriteRenderer>();
            }
            else finalPosition = _closePlataform.transform.position;
        }

        initialPosition = transform.position;

        finalPosition = _closePlataform.transform.position;

        initialIDirection       =       GetDirection(GetDifPosition(initialPosition,
finalPosition));

        initialVDirection        =        GetDirection(GetDifPosition(finalPosition,
initialPosition));

    }

       void Update () {

        if   ((GetDirection(GetDifPosition(transform.position,   finalPosition))   ==
initialIDirection) && (!indo) && (!volta)) {
            indo = true;
        }

        if   ((GetDirection(GetDifPosition(transform.position,   finalPosition))   !=
initialIDirection) && (indo) && (!volta))
        {
            indo = false;
            volta = true;
        }

        if  ((GetDirection(GetDifPosition(transform.position,  initialPosition))  !=
initialVDirection) && (volta) && (!indo))
        {
            indo = true;
```

```
            volta = false;
        }

        if (indo)
        {
            transform.Translate(GetDifPosition(initialPosition,   finalPosition)   *
Time.deltaTime * iVelo);
        }
        else if (volta)
        {
            transform.Translate(GetDifPosition(finalPosition,   initialPosition)   *
Time.deltaTime * vVelo);
        }

    }

    private Vector3 GetDifPosition(Vector3 iPosition, Vector3 fPosition)
    {

        if (fPosition.x > iPosition.x)
        {
            iPosition.x += iSprite.bounds.size.x / 2;
            if (closeP) fPosition.x -= fSprite.bounds.size.x / 2;
        }
        else if (fPosition.x < iPosition.x)
        {
            iPosition.x -= iSprite.bounds.size.x / 2;
            if (closeP) fPosition.x += fSprite.bounds.size.x / 2;
        }

        if (fPosition.y > iPosition.y)
        {
            iPosition.y += iSprite.bounds.size.y / 2;
            if (closeP) fPosition.y -= fSprite.bounds.size.y / 2;
        }
        else if (fPosition.y < iPosition.y)
        {
            iPosition.y -= iSprite.bounds.size.y / 2;
            if (closeP) fPosition.y += fSprite.bounds.size.y / 2;
        }

        return fPosition - iPosition;

    }

    private float GetDirection(Vector3 dPosition)
    {
        return Mathf.Atan2(dPosition.y, dPosition.x);
    }
}
```

Script 9 – InputManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public   class   InputManager   :   MonoBehaviour,   IDragHandler,   IPointerUpHandler,
```

```
IPointerDownHandler{

	private Image m_JoyImage;

	private Image m_BgJoyImage;

	[SerializeField]
	private bool m_UseVirtualJoystick = false;

	public Vector3 Axis { get; private set; }

	private void Start(){

		m_BgJoyImage = GetComponent<Image> ();
		m_JoyImage = transform.GetChild(0).GetComponent<Image> ();
		Axis = Vector3.zero;

		#if UNITY_ANDROID
		m_UseVirtualJoystick = true;
		#endif


		m_BgJoyImage.gameObject.SetActive (m_UseVirtualJoystick);
		m_JoyImage.gameObject.SetActive (m_UseVirtualJoystick);
	}

	public void OnDrag(PointerEventData ped){

		Vector2 position = Vector2.zero;

		if 		(RectTransformUtility.ScreenPointToLocalPointInRectangle
(m_BgJoyImage.rectTransform, ped.position, ped.pressEventCamera, out position)) {

			position.x 		= 		(position.x 		/
m_BgJoyImage.rectTransform.sizeDelta.x);
			position.y 		= 		(position.y 		/
m_BgJoyImage.rectTransform.sizeDelta.y);

			float x = (m_BgJoyImage.rectTransform.pivot.x == 1) ? position.x
* 2 + 1 : position.x * 2 - 1;
			float y = (m_BgJoyImage.rectTransform.pivot.y == 1) ? position.y
* 2 + 1 : position.y * 2 - 1;

			Axis = new Vector3 (x, y, 0.0f);
			Axis = (Axis.magnitude > 1) ? Axis.normalized : Axis;

			m_JoyImage.rectTransform.anchoredPosition = new Vector3 (Axis.x
*	(m_BgJoyImage.rectTransform.sizeDelta.x	/	3.0f),	Axis.y	*
(m_BgJoyImage.rectTransform.sizeDelta.y / 3.0f));

		}

	}

	public void OnPointerDown(PointerEventData ped){
		OnDrag (ped);
	}

	public void OnPointerUp(PointerEventData ped){
		Axis = Vector3.zero;
		m_JoyImage.rectTransform.anchoredPosition = Axis;
	}
```

```
}
```

Script 10 – GameUI.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class GameUI : MonoBehaviour
{
    private GameObject[] Buttons = new GameObject[8], Category = new GameObject[3];

       private GameObject Pointer, Pause, PauseFilter, PauseButton, JoyStick,
InputFireBall, InputFast;

    private Player _Player;
    private CameraMov _CameraMov;

    private Image[] images;
    private Image ebFill, ebFillDelay;

       public bool isPaused = false, keyVertical = false, keyEnter = false,
keyHorizontal = false, keyPause = false;

       public bool PausedButton { get; set; }
    public bool buttonEnter { get; set; }
    public int idButton { get; set; }

    private int screen = 0, index = 0;

    private bool fRed = true;
    private float fPoint = 0;

    void Start()
    {
        _Player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();

        _CameraMov                                                                    =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<CameraMov>();

        images = FindObjectsOfType<Image>();

        foreach (Image im in images)
        {
            if (im.name == "EnergyBarFill") ebFill = im;
            if (im.name == "EnergyBarFillDelay") ebFillDelay = im;
        }

        SetGameObjects();

        SetButton(PauseButton, true);

              PausedButton = false;

              idButton = -1;

    }
```

```
    void Update()
    {
        if (_CameraMov.changed)
        {
            _Player = _CameraMov._sPlayer;
            _CameraMov.changed = false;
        }

        #region Pointer

             if ((Input.GetAxisRaw("Pause") > 0 || Input.GetKeyDown(KeyCode.Escape)
|| PausedButton) && !isPaused && !keyPause)
        {
            index = 0;
            screen = 0;
            isPaused = true;
            Time.timeScale = 0;
            _Player.canInteract = false;

                    SetPointer(Pointer, true);
                    Pause.SetActive(true);
                    PauseFilter.SetActive(true);
            Category[screen].SetActive(true);
            SetPointerPosition(Buttons[index].transform.position);
            SetButton(PauseButton, false);

            keyPause = true;
        }

        if (Input.GetAxisRaw("Pause") == 0 && keyPause) keyPause = false;

        if (isPaused)
        {
            if  (Input.GetAxisRaw("Vertical")  ==  0  &&  keyVertical)  keyVertical  =
false;
            if (Input.GetAxisRaw("Horizontal") == 0 && keyHorizontal) keyHorizontal
= false;
            if (Input.GetAxisRaw("Submit") == 0 && keyEnter) keyEnter = false;

            if (screen == 0)
            {
                if (idButton != -1)
                {
                    index = idButton;
                    SetPointerPosition(Buttons[index].transform.position);
                }

                if (Input.GetAxisRaw("Vertical") < 0 && !keyVertical)
                {
                    if (index == 3)
                    {
                        index = 0;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                    else
                    {
                        index++;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                }
                if (Input.GetAxisRaw("Vertical") > 0 && !keyVertical)
                {
```

```
                    if (index == 0)
                    {
                        index = 3;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                    else
                    {
                        index--;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                }

                if ((Input.GetAxisRaw("Submit") > 0 || buttonEnter) && !keyEnter)
                {
                    switch (index)
                    {
                        case 0:
                            Time.timeScale = 1;
                            isPaused = false;
                            _Player.canInteract = true;

                                                SetPointer(Pointer, false);
                            Pause.SetActive(false);
                                                PauseFilter.SetActive(false);
                            Category[screen].SetActive(false);
                            SetButton(PauseButton, true);
                                                PausedButton = false;

                            keyEnter = true;
                            break;

                        case 1:
                            Category[screen].SetActive(false);

                            screen = 1;
                            index = 4;

                            Category[screen].SetActive(true);
                            SetPointerPosition(Buttons[index].transform.position);
                            keyEnter = true;
                            break;

                        case 2:
                            Category[screen].SetActive(false);

                            screen = 2;
                            index = 7;

                            Category[screen].SetActive(true);
                            SetPointerPosition(Buttons[index].transform.position);
                            keyEnter = true;
                            break;

                        case 3:
                            Application.Quit();
                            keyEnter = true;
                            break;
                    }

                    buttonEnter = false;

                }
```

```
                        if      ((Input.GetAxisRaw("Pause")      >      0      ||
Input.GetKeyDown(KeyCode.Escape)) && !keyPause)
                {
                    Time.timeScale = 1;
                    isPaused = false;
                    _Player.canInteract = true;

                                SetPointer(Pointer, false);
                    Pause.SetActive(false);
                                PauseFilter.SetActive(false);
                    Category[screen].SetActive(false);
                    SetButton(PauseButton, true);
                                PausedButton = false;

                    keyPause = true;
                }
            }
            else if (screen == 1)
            {
                if (idButton != -1)
                {
                    index = idButton;
                    SetPointerPosition(Buttons[index].transform.position);
                }

                if (Input.GetAxisRaw("Vertical") < 0 && !keyVertical)
                {
                    if (index == 6)
                    {
                        index = 4;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                    else
                    {
                        index++;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                }
                if (Input.GetAxisRaw("Vertical") > 0 && !keyVertical)
                {
                    if (index == 4)
                    {
                        index = 6;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                    else
                    {
                        index--;
                        SetPointerPosition(Buttons[index].transform.position);
                    }
                }

                if (Input.GetAxisRaw("Horizontal") < 0 && !keyHorizontal)
                {
                    if (index != 6)
                    {
                        SetScrollBarValue(Buttons[index].GetComponent<Scrollbar>(),
-0.1f);
                    }
                }
                if (Input.GetAxisRaw("Horizontal") > 0 && !keyHorizontal)
```

```
                {
                    if (index != 6)
                    {
                        SetScrollBarValue(Buttons[index].GetComponent<Scrollbar>(),
0.1f);
                    }
                }

                            if ((Input.GetAxisRaw("Submit") > 0 || buttonEnter) &&
!keyEnter)
                {
                    if (index == 6)
                    {
                        Category[screen].SetActive(false);

                        screen = 0;
                        index = 0;

                        Category[screen].SetActive(true);
                        SetPointerPosition(Buttons[index].transform.position);
                        keyEnter = true;
                    }

                    buttonEnter = false;

                }

                            if      ((Input.GetAxisRaw("Pause")      >      0      ||
Input.GetKeyDown(KeyCode.Escape)) && !keyPause)
                {
                    Category[screen].SetActive(false);

                    screen = 0;
                    index = 0;

                    Category[screen].SetActive(true);
                    SetPointerPosition(Buttons[index].transform.position);
                    keyPause = true;
                }
            }
            else
            {
                if (idButton != -1)
                {
                    index = idButton;
                    SetPointerPosition(Buttons[index].transform.position);
                }

                if ((Input.GetAxisRaw("Submit") > 0 || buttonEnter) && !keyEnter)
                {
                    if (index == 7)
                    {
                        Category[screen].SetActive(false);

                        screen = 0;
                        index = 0;

                        Category[screen].SetActive(true);
                        SetPointerPosition(Buttons[index].transform.position);
                        keyEnter = true;
                    }
```

```
                    buttonEnter = false;

                }

                        if      ((Input.GetAxisRaw("Pause")      >      0      ||
Input.GetKeyDown(KeyCode.Escape)) && !keyPause)
                {
                    Category[screen].SetActive(false);

                    screen = 0;
                    index = 0;

                    Category[screen].SetActive(true);
                    SetPointerPosition(Buttons[index].transform.position);
                    keyPause = true;
                }
            }
        }

        idButton = -1;

        #endregion
    }

       void FixedUpdate() {
              #region HUD

              float pLife = (_Player.timeCurrent / _Player.timeTotal);

              ebFill.fillAmount = 1 - pLife;
              ebFillDelay.fillAmount       =       Mathf.Lerp(ebFillDelay.fillAmount,
ebFill.fillAmount, 0.07f);

              if (pLife > 0.5f)
              {
                     if (fRed)
                     {
                            ebFill.color = new Color(1, 0, 0);
                            fPoint = pLife;
                     }

                     fRed = false;

                     ebFill.color  =  new  Color(1,  Mathf.Lerp(ebFill.color.g,  fPoint,
0.2f), 0);

                     if (ebFill.color.g > fPoint - 0.1f) fRed = true;

              }
              else ebFill.color = new Color(1, pLife, 0);

              #endregion
       }

    private void SetButton(GameObject button, bool v)
    {
    #if UNITY_ANDROID
        button.SetActive(v);
    #endif
    }

       private void SetPointer(GameObject button, bool v)
```

```
       {
              #if !UNITY_ANDROID
              button.SetActive(v);
              #endif
       }

    private void SetPointerPosition(Vector3 Position)
    {
        Pointer.transform.position  =  new  Vector3(Position.x  -  105,  Position.y,
Position.z);
        keyVertical = true;
    }

    private void SetScrollBarValue(Scrollbar scrollbar, float value)
    {
        scrollbar.value = scrollbar.value + value;
        keyHorizontal = true;
    }

    private void SetGameObjects()
    {
        Buttons[0] = GameObject.Find("p_Continue");
        Buttons[1] = GameObject.Find("p_Options");
        Buttons[2] = GameObject.Find("p_Credits");
        Buttons[3] = GameObject.Find("p_Exit");
        Buttons[4] = GameObject.Find("p_ScrollbarVol");
        Buttons[5] = GameObject.Find("p_ScrollbarGama");
        Buttons[6] = GameObject.Find("p_BackOpt");
        Buttons[7] = GameObject.Find("p_BackCred");

        Category[0] = GameObject.Find("c_Main");
        Category[1] = GameObject.Find("c_Options");
        Category[2] = GameObject.Find("c_Credits");

              JoyStick = GameObject.Find("BgJoyStick");

        InputFireBall = GameObject.Find("p_FireBallButton");
        InputFast = GameObject.Find("p_FastButton");

        Pointer = GameObject.Find("p_Pointer");
              Pause = GameObject.Find("p_Pause");
              PauseFilter = GameObject.Find("p_PauseFilter");
        PauseButton = GameObject.Find("p_PauseButton");

        foreach (GameObject g in Category)
        {
            g.SetActive(false);
        }

              JoyStick.SetActive (false);
              SetButton(JoyStick, true);

        InputFireBall.SetActive(false);
        SetButton(InputFireBall, true);

        InputFast.SetActive(false);
        SetButton(InputFast, true);

        PauseButton.SetActive (false);
              SetButton(PauseButton, true);

              Pointer.SetActive (false);
```

```
            SetPointer(Pointer, true);

        PauseFilter.SetActive(false);
        Pause.SetActive(false);

    }

    public void FireBall()
    {
        _Player.FireBallInput = true;
    }

    public void FastIn()
    {
        _Player.FastInput = true;
    }

    public void FastOut()
    {
        _Player.FastInput = false;
    }

}
```

Script 11 – FireBall.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FireBall : MonoBehaviour {

       private int timeTotal = 2;
       private float timeCurrent = 0;
    private ControlFlame _Flame;
    private ControlLight _Light;
    private Animator _Animator;
    private Rigidbody2D _Rigidbody2D;

    public bool isDead = false;

    private void Start()
    {
        _Animator = GetComponent<Animator>();
        _Flame = GetComponentInChildren<ControlFlame>();
        _Light = GetComponentInChildren<ControlLight>();
        _Rigidbody2D = GetComponent<Rigidbody2D>();
    }

    void Update(){

            timeCurrent += Time.deltaTime;

            if (timeCurrent >= timeTotal) {
                  Death();
            }

        if (isDead)
        {
            Destroy(gameObject);
```

```
        }
       }

    private void Death()
    {
        timeCurrent = timeTotal;
        _Animator.SetBool("FireBall_isDead", true);
        _Flame.Death();
        _Light.Death();
        _Rigidbody2D.velocity = Vector2.zero;
    }

    void OnTriggerEnter2D(Collider2D col)
    {

        if (col.tag != "Player" && col.tag != "FireBall" && col.tag != "Torch")
        {
            Death();
        }

    }
}
```

Script 12 – Checkpoint.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Checkpoint : MonoBehaviour {

       //Variável para animação da tocha
       private Animator _Animator;
    private CameraMov _Camera;

       void Start(){

             _Animator = GetComponent<Animator> ();
        _Camera                                                                    =
GameObject.FindGameObjectWithTag("MainCamera").GetComponent<CameraMov>();
    }

    void OnTriggerEnter2D(Collider2D col){

             //Verifica se Collider é Player ou FireBall
             if (col.transform.tag == "Player") {

           //Ativa o checkpoint
           _Camera._sPlayer.pcheck = transform.position;

                    //Ativa animação da tocha
                    _Animator.SetBool ("Torch_isActive", true);
             }
        else if (col.transform.tag == "FireBall")
        {

            //Ativa o checkpoint
            _Camera._sPlayer.pcheck = transform.position;

            //Ativa animação da tocha
            _Animator.SetBool("Torch_isActive", true);
```

```
        }

      }

}
```