FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Jogos Digitais

João Natan Pascon

Resgate de Zuleide

Americana, SP

2017

FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Tecnologia em Jogos Digitais

João Natan Pascon

Resgate de Zuleide

Relatório técnico desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Jogos Digitais sob a orientação do Prof. Me. Bruno Daniel.

Americana, SP.
2017

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

P287r PASCON, João Natan

Resgate de Zuleide. / João Natan Pascon. – Americana, 2017.

75f.

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

Orientador: Prof. Ms. Bruno Daniel

1 Jogos eletrônicos I. DANIEL, Bruno II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.6

João Natan Pascon

Resgate De Zuleide

Relatório técnico apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Americana, 16 de Dezembro de 2017.

Banca Examinadora:

Bruno Daniel (Presidente)
Mestre
Fatec Americana

Daniele Junqueira Frosoni (Membro)
Especialista
Fatec Americana

Gustavo Carvalho Gomes de Abreu (Membro)
Especialista
Fatec Americana

RESUMO

O projeto de um jogo digital é muito complexo e envolve muitas áreas. É necessária uma combinação de planejamento teórico com trabalho técnico. Visando obter mais experiência nessas duas áreas, foi proposto o desenvolvimento de um jogo para o trabalho de conclusão de curso.

O trabalho apresenta os principais elementos de um jogo, como história, jogabilidade, interface, programação e Level Design. Todos eles foram adaptados para a plataforma mobile e o sistema operacional Android, que mostraram ser a melhor opção para o desenvolvimento, pois é onde se encontra o melhor meio de distribuição de jogos atualmente. Este trabalho apresenta a metodologia utilizada, desde o conceito inicial até o resultado final.

O jogo é do gênero shooter, uma sub-categoria dos jogos de ação com estilo artístico 2D. Ele é similar aos jogos de naves dos anos 90, como Sonic Wings (Super Nintendo – 1994), mas contém algumas diferenças em relação aos outros do seu gênero, como o controle e jogabilidade que foram modificados para darem mais originalidade.

A receptividade das pessoas em relação ao trabalho foi muito positiva. Através de uma pesquisa de opinião muitos classificaram o jogo como bom e relataram sua experiência

Palavras Chave: jogo; *shooter*; *mobile*.

ABSTRACT

The project of a digital game is very complex and involves many areas, there must be a combination of theoretical planning with a great technical work. In order to obtain more knowledge about these two areas, it was proposed the development of a game for the conclusion paper

The work presents the main elements of a game, such as history, gameplay, interface, programming and Level Design. All of them have been adapted to the mobile platform and the Android operating system, which proved to be the best option for development, as it is where the best means of game distribution is currently found. This work presents the methodology used, from the initial concept to the final result.

The game chosen is from the shooter genre a sub category of action games with the 2D artistic style. The game is based on the games of ships of the 90s like Sonic Wings (Super Nintendo - 1994). The game contains some differences from others of its kind, such as control and gameplay that have been modified to give more originality.

The receptivity of people towards work was very positive. Through an opinion survey many rated the game as good and reported their experience

Keywords: *game; shooter; mobile.*

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

Assets: São aditivos/componentes utilizados em projetos.

Boss: Chefe final de uma fase do jogo.

Cooldown: Tempo de recarga de um poder ou ação.

Engine: É um motor para desenvolvimento de jogos.

Foley: Reprodução de efeitos sonoros complementares.

Hp: Quantidade de vida do player ou inimigo.

HUD: *Heads-up display*, objetos que a cena contém.

IDE: Vem do inglês *Integrated Development Environment* que é um sistema de apoio ao desenvolvedor.

Level Design: Parte do desenvolvimento de um jogo que é a criação dos níveis do game.

Open Source: *Softwares* que contém seus códigos em aberto.

Pixel Art: Arte muito utilizada em jogos que tem o conceito de desenhos com principal elemento os pixels.

Power Ups: Itens que dão mais poderes ou aumentam a habilidade do personagem do jogo.

Script: Onde se encontra os códigos fontes do jogo.

Spawn: Local onde objetos são instanciados.

Sprites: Imagem dos objetos.

SUMÁRIO

1. INTRODUÇÃO

O principal mercado atualmente para desenvolvedores *Indies* é o mercado *mobile*, seja pelo grande público que esse mercado provém mas também pela distribuição que é mais facilitada em relação a plataformas mais comuns, como consoles e computadores. Observando isso é de suma importância obter mais conhecimento técnico e operacional para se ter um jogo de sucesso.

Visando aprender mais sobre todos os aspectos que envolvem um *game*, foi proposto a criação de um para plataforma *mobile*, tendo como meta alcançar o máximo nível de conhecimento na realização do trabalho. Então foi escolhido fazer um jogo baseado em um grande sucesso do passado, que seria baseado nos famosos "jogos de naves" dos anos 90. O jogo que tem o gênero de ação foi a escolha mais certa, pois além de trabalhar com algo de grande valor histórico para a cultura dos games ele também contém muitas características que são consideradas importantes para os jogos *mobile*, que são velocidade que o jogo deve ter, o quesito casual demonstrando que ele não precisa ser imersivo e a praticidade de poder parar e continuar quando quiser sem ter grandes perdas na progressão do jogo.

Sendo assim, o objetivo do trabalho é trazer mais experiência na construção de um jogo e também na *engine* em que ele será construído que no caso será utilizado a plataforma **Unity®** (Unity Technologies) com a linguagem de programação C#. Além disso trabalhar outros elementos como, interface, *level design*, HUD e outros aspectos importantes dos games, tudo isso através de um jogo que será disponibilizado para celulares com o sistema operacional Android.

1.1 Sobre o Jogo

O gênero de ação/*shooter* foi escolhido na procura de trabalhar com um gênero de sucesso mas que perderam bastante espaço com a introdução dos jogos 3D no mercado de games, mas que voltaram à tona com o surgimento do *mobile* como plataforma de jogos. O *game* foi inspirado em franquias de sucesso principalmente na época dos 16 bits como *Sonic Wings* (Figura 1)*, Strike Gunner* entre outros.

Figura 1 - Jogo Sonic Wings

Fonte:https://www.gamefaqs.com/saturn/577641-sonic-wings-special/images/219

No gameplay o jogador controla uma nave pela tela com o objetivo de sobreviver perante os obstáculos e derrotar os inimigos que irão aparecer no decorrer da fase. No final de cada irá ter um *boss* como o último desafio. O *player* contará com um número de vidas já pré-determinada, mas poderá conseguir mais no decorrer do jogo. A Fase termina quando ele derrotar o chefe final ou quando ele perder todas as vidas. Nesse caso o jogador terá que começar a fase do zero novamente. No decorrer da fase ele poderá contar com a ajuda de *power ups* que serão liberados para ele conforme derrota os inimigos. Os itens variam desde melhoramento das armas, recuperação de vida ou simples moedas.

A história do jogo é do gênero cômico e conta a aventura de um Senhor que teve a sua querida vaca abduzida por aliens e que sai na caça dos sequestradores afim de resgatar a pequena vaquinha Zuleide. No fim ele acaba tendo descobertas surpreendentes que o ajudam a entender melhor tudo o que se passou no decorrer da aventura.

2. METODOLOGIA

Neste capitulo será descrito o planejamento do jogo, visando explicar as ferramentas utilizadas, e as técnicas empregadas na construção do trabalho.

2.1. Características do Jogo

O jogo contará com um total de seis fases, e cada fase tendo a sua peculiaridade e desafiando o jogador de uma maneira nova. O nível de dificuldade também foi trabalhado em cima das fases. Conforme jogador avança as fases tendem a ficar mais difíceis, sendo assim a primeira entra mais como um aprendizado para o jogo entender as regras do jogo e os controles. O chefe da fase terá a dificuldade dele proporcional a fase que o jogador se encontra. Apenas a fase seis será diferente, pois ela terá apenas o chefe.

As armas do jogador contra os inimigos será como padrão o tiro comum, que sempre estará com ele quando renascer junto de um item especial, que ele terá sempre como padrão. Quando usado o poder especial ele pode recarregar pegando um item deixado pelos inimigos. Pode haver melhoramento da arma quando ele também pegar um item para essa função. O melhoramento de arma pode ser a inclusão de novos tiros, passando de um tiro para três de uma vez por exemplo, ou mesmo aumento do tamanho do tiro.

A arte do jogo é toda feita em 2D, algumas artes de autoria própria mas sendo a sua maioria baixadas da biblioteca de **Assetstore Unity®** (Unity Technologies) da própria *engine*, que disponibiliza *sprites*, componentes entre outras coisas gratuitamente para os desenvolvedores. Em relação ao *background* e imagens de fundo, eles foram obtidas em sites que disponibilizam conteúdos gratuitos e que são citados nos créditos do jogo.

Em relação aos efeitos sonoros a música do *gameplay* e do menu foram pegas diretamente da Assets Store do Unity, já os efeitos sonoros foram baixados de sites gratuitos e alguns outros foram feitos através de *foley* próprio.

2.2. Ferramentas Utilizadas

Para desenvolvimento do jogo foi utilizado a licença gratuita da *engine* **Unity®**, que é uma ferramenta focada para produção de jogos. O programa já proporciona algumas utilidades que ajudam na manipulação dos elementos básicos de um jogo, como as animações, efeitos sonoros e visuais, além de contar com um motor de física já pronto para ser utilizado, ajudando assim em todo o processo.

Na parte artística, foram utilizados alguns *assets* gratuitos da própria ferramenta e também foi utilizado o programa *Open Source* **Piskel** (Piskel, 2017), que é uma ferramenta para parte artística que é mais focado na edição de *pixel arts*, além da criar o programa foi muito utilizado para redimensionar imagens.

Para manipulação de sons e efeitos sonoros foi utilizado o próprio Unity que já proporciona uma ferramenta para essa finalidade, mas também foi utilizado o **Audacity** (Audacity, 2017) que proporciona mais facilidades, como uma ferramenta para excluir ruídos indesejados, diminuir distorções, graves entre outras utilidades.

Em relação a *IDE* para a linguagem de programação, foi utilizada a própria que a *engine* proporciona que é a Mono Developer, que foi utilizada para manipular a linguagem do jogo que foi feito em C#, sendo ela uma linguagem de programação criada pela Microsoft como parte da plataforma .NET e que junto com o JavaScript vem como padrão no Unity.

2.3. Análise de Mercado

A análise de mercado foi feita de forma genérica pois não era esse o foco do trabalho, mas foi tomada como aprendizado para futuros projetos. Ela foi feita analisando meios de distribuição, público alvo, sistema operacional e plataforma. Todos os dados foram retirados do **Pesquisa Game Brasil** (Sioux, 2017) um site especializado que tem o foco em levantamento de números sobre o mercado de games.

2.3.1. Distribuição

O jogo foi divulgado e distribuído gratuitamente pela Play Store, plataforma digital do sistema operacional Android (Figura 2). Essa estratégia não é apenas seguindo a regra do projeto que não pode ter fins lucrativos, mas seria também a estratégia caso fosse visado o lucro, seria uma decisão tomada seguindo a análise do público, que segunda a pesquisa 81% das pessoas baixam apenas jogos gratuitos, e que 77% deles aceitam anúncios publicitário quando o jogo é gratuito, então a ideia seria ganhar em cima da monetização e futuramente da comercialização de itens dentro jogo.

Figura 2 - Android e Play Store

Fonte: https://www.baixarplaystore.net/android/

2.3.2. Público Alvo

Seguindo o gênero e o propósito do jogo o público alvo seria pessoas que na sua infância ou adolescência tenham jogado o estilo de game no seu auge, por volta da década de 90, e aproveitar também o fator nostalgia. Então com essa lógica essas pessoas seriam na sua maioria do sexo masculino e com idade hoje entre 25 a 40 anos. Não apenas por coincidência o maior público dos jogos eletrônicos segunda a pesquisa está entre a faixa etária de 25 a 34 anos, sendo 34,8% dos jogadores.

2.3.3. Plataforma e Sistema Operacional

Segundo a pesquisa conclui-se que os celulares são a segunda plataforma que mais se costuma jogar ficando atrás apenas dos computadores, sendo que quando a questão é qual é a plataforma preferida dessas pessoas o *mobile* segue líder com 34% das preferências. Foi levantado também que 78,2% das pessoas baixem apenas jogos gratuitos.

No assunto sistema operacional o Android segue disparado como a escolha mais popular das pessoas com 76,2% do mercado, seu concorrente principal o *iOS* vem em segundo com apenas 10,1%. Esses dados seguem no mesmo patamar quando o assunto é *tablets*.

2.4. Cronograma

O objetivo no começo do trabalho de graduação era fazer um jogo em grupo, mas a realização deste trabalho durou apenas dois meses, isso porque houve alguns problemas no decorrer do projeto, então foi decidido em comum acordo de todos do grupo que seria melhor parar o trabalho já que ficaria difícil a entrega do mesmo no prazo. Então foi decido continuar um trabalho solo, aproveitando a base de um projeto que já tinha sido começado no início do ano.

Será demonstrado a seguir o cronograma montado para a realização do trabalho. A tabela (Tabela 1) a seguir mostrará a divisão das tarefas em meses e quinzenas, e será detalhado mais cada item do planejamento mostrado. Na sequência estão explicadas as atividades de cronograma.

A - Base do Jogo: A base do jogo é o conceito inicial que foi construída estudando e acompanhando vídeo aulas sobre a Unity no canal GameMoviesPro do youtube

B - Planejamento: Essa é a parte onde se definiu para qual plataforma o *game* será distribuído, o gênero do jogo, qual o público alvo, análise de outros jogos no mesmo estilo, história e também o próprio cronograma para as atividades.

C - História: Nessa etapa é onde foi escrito todo o enredo e de como esse enredo foi implementado no *gameplay*, apesar do jogo não ser tão focado na história era necessário manter uma coerência.

D - Desenvolvimento: É a parte técnica do trabalho, aonde terá toda a parte de desenvolvimento e programação. Desenvolvimento dos menus de navegação, sistema de som e áudio, programação do *player*, inimigos, objetos. Montagem da interface, botões, HUD entre outras coisas.

E - Animação e Som: Após o desenvolvimentos dos sistema de som e programação das cenas, essa etapa se destinou a teste de efeitos sonoros e buscas para melodias para o jogo. Além disso teve o encaixe das animações nos objetos e acerto de detalhes dela.

F - Documentação: Etapa para redigir o relatório do projeto que será entregue para a banca na apresentação.

G - Testes: Divulgação do jogo para o público para obter os *feedbacks*.

H - Melhorias e Correções: Após os testes e os dados gerados, essa é a etapa para correções de possíveis bugs e melhorias que foram apresentadas pelo público.

I - Montagem da Apresentação: Esse é o momento para montagem da apresentação do trabalho para a banca e também para treinar a apresentação do que será apresentado.

Tabela 1 - Cronograma

Mês/Tarefas	Fevereiro		Março		Outubro		Novembro		Dezembro	
Quinzenas	1º	2º	1º	2º	1º	2º	1º	2º	1º	2º
Base do Jogo										
Planejamento										
História										
Desenvolvimento										
Animação e Som										
Documentação										
Testes										
Melhorias/Correções										
Entrega										
Montagem da Apresentação										
Apresentação e Entrega do Documento										

Fonte: Próprio Autor

3 IMPLEMENTAÇÃO

Será demonstrando nesse tópico como foi feita as implementações do player, inimigos, sistemas de áudio, entre outros componentes do projeto.

3.1. *Player*

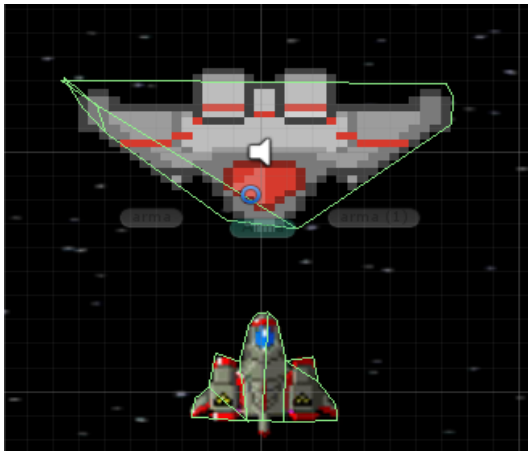
Há muitas coisas que são necessárias levar em consideração quando é preciso implementar o player no jogo, sistemas de colisão, controle, movimentação, e recompensas que ele pode ter.

3.1.1. Sistema De Colisão

Para um objeto ter o sistema de colisão primeiro ele precisa ter *colliders* (Figura 3). O player tem dois tipos de colisão para o seu sistema, uma é a colisão direta com um objeto que é verificado por o método chamado "*OnCollisionEnter2D*" no *script Player.cs* e outro tipo de colisão que ocorre mas nesse caso são colisões que acionam eventos que é verificada pelo método "*OnTriggerEnter2D*". Então quando ocorrer uma batida não importa de qual desses métodos, há uma verificação para ver com o que o *player* colidiu, e a identificação do que foi colidido com ele é feito através da "*tag*" que esse objeto possui, se for uma *tag* com o nome "tiroInimigo", ela caia em um verificação feita através de um *switch* e que nesse caso daria um dano no *player*, se a *tag* for por exemplo "itemEspecial" é adicionado um *power up* ao *player* e assim por diante.

Outro fator interessante é que quando o jogador é destruído ele renasce e durante 3 segundos após o renascimento o jogador permanece invulnerável, isso para não correr o risco dele nascer em um local que já o faça perder novamente, essa invulnerabilidade é feita através de uma simples troca de *layer* no Unity, onde ele é colocado em uma *layer* que não tem colisão com nenhum outro objeto da cena.

Figura 3 – Exemplo de *Colliders*

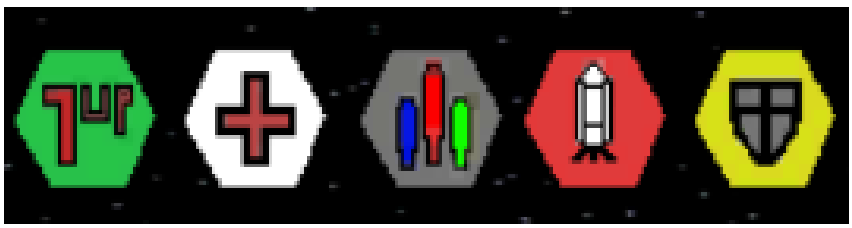


3.1.2. *Power Ups* e Recompensas

Quando o jogador destrói um inimigo ele pode conseguir uma simples moeda ou *power ups*. Quando o jogador colidir com alguma objeto com a *tag* de um *power up* ele recebe o benefício dele, os efeitos dos *power ups* são 5 no total, com seus símbolos demonstrados na Figura 4, e seus efeitos são os seguintes:

- 1 *up* : O Jogador ganha uma vida a mais.
- Recupera vida: A barra de *hp* dele é recuperada.
- Arma: Esse item adiciona mais tiros para o jogador, esse tem duas evoluções.
- Especial: Recarrega o especial do Player.
- Escudo: Forma um escudo ao redor do jogador que absorve os danos.

Figura 4 – Ícone dos Itens



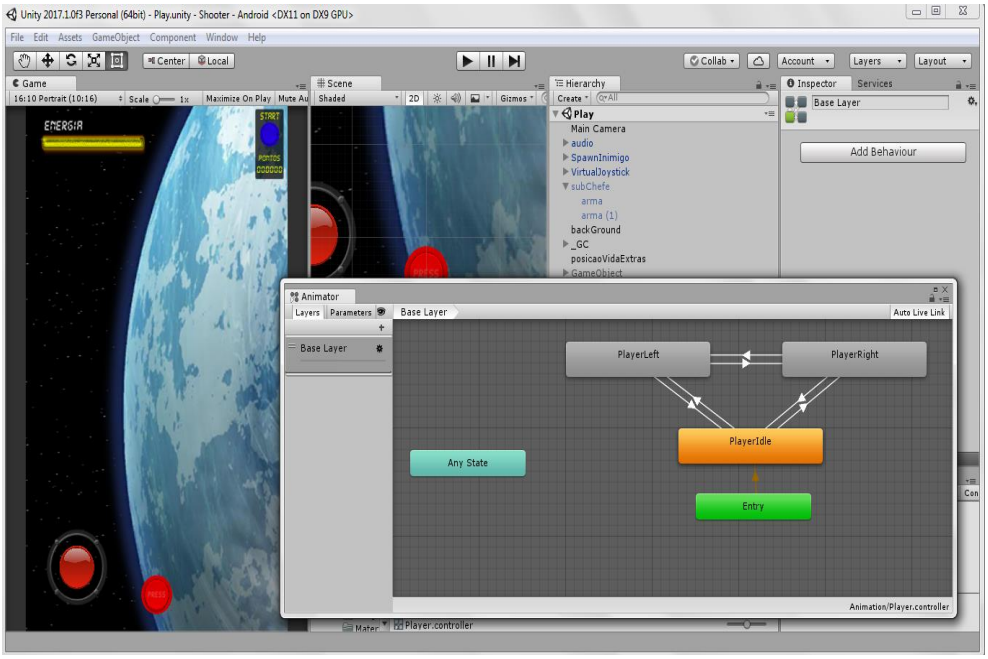
3.1.3. Controles e Jogabilidade

Analisando outros jogos disponíveis para *mobile* nesse mesmo gênero, foi identificado alguns pontos ruins que foram melhorados ou feitos de uma nova maneira para dar um ar de mais originalidade no jogo. Um dos pontos principais era o controle da nave, em sua maioria os jogos deixavam o comando da nave através do “arraste” do dedo pela tela. A ideia é interessante no começo mas conforme o jogo vai acontecendo o dedo do jogador sempre acaba ficando em cima da própria nave ou no meio da tela, dificultando muito a visão do player e a movimentação da própria. Então como forma de melhorar esse quesito e deixar mais original o *game*, foi adicionado uma alavanca no canto esquerdo inferior, deixando assim o dedo do jogador mais preso no canto da tela e não atrapalhando ele no meio do jogo. No começo é preciso um pouco de prática para poder se acostumar, mas depois fica mais divertido.

Outro ponto diferente do jogo em comparação aos seus concorrentes é o quesito do tiro automático da nave. Em sua maioria os jogos deixam a ação do tiro automático deixando para o jogador apenas a ação de movimentação e outros botões como especial e etc. Para muitos entusiastas dos *shooters* antigos ou mesmos jogadores novos essa característica deixa o jogo banal. Então o Resgate de Zuleide traz mais emoção, não só desafiando o reflexo do jogador em relação a movimentação, mas também cobrando ele no quesito de pontaria, deixando ao jogador a obrigação de tomar a decisão no momento certo ou não.

3.1.4. Controle de Animação *Player*

Figura 5 - Esquema Animação *Player*



O jogo não contém muitas animações ficando então a principal com o *player*. O controle de animação é muito simples, como mostrado na Figura 5. Ele é manipulado de acordo com a direção que a nave é movimentada, se a nave estiver indo para a esquerda então o *velocity.x* será negativo assim chamando a animação que contém a *sprite* que tomba a nave para a esquerda. O mesmo caso acontece quando o *velocity.x* for positivo chamando então *sprites* que tem a nave tombada para a direita. Caso o *velocity.x* seja igual zero a nave mantém a posição neutra e com um *sprite* centralizado. Todo esse controle é feito através desse esquema no *script* do *Player.cs*, e através de uma variável *float* que armazena os resultados do *velocity.x*.

3.2. Implementação dos Inimigos

Na implementação de inimigos houve muitos elementos envolvidos. Tem parte de animação, scripts, componente entre outros aspectos que foram importante para o desenvolvimento do jogo,

3.2.1. Inimigos

O jogo contém vários inimigos comuns que são demonstrados na Figura 6, todos se utilizam do mesmo script *enemy.cs*, mas eles possuem alguns aspectos que são parametrizados e que são mudados para variar a função de cada um. Alguns desses elementos parametrizados são o *hp* que pode variar de um até a cinco, a velocidade do tiro, o tempo de *cooldown* de um tiro para o outro, a forma do tiro, já que existem diferentes tipos de tiros e até a movimentação dele, por exemplo ele pode se movimentar apenas no eixo “X” ou então no eixo “Y”, isso depende também do lugar no qual ele vai ser instanciado e com o script também foi possível adicionar um tempo de curva para ele mudar de posição aleatoriamente.

Figura 6 - Inimigos

3.2.2. Chefes

Os chefes foram construídos de modo que cada um tivesse a suas peculiaridades, e de modo que ele desafiasse as diferentes habilidades do jogador. Alguns ele precisaria ter um bom controle de movimentação. Em outros casos ele deveria ser bom de mira e assim por diante. Para isso cada um recebeu ataque, movimentações e imagens diferenciadas (Figura 6) que serão explicadas abaixo.

Chefe 1: Ele contém uma movimentação simples na horizontal, e com tiros lentos. Ele continua assim até restar apenas 10 de *hp*, onde ele começa a dificultar soltando tiro mais rápidos. Como é a primeira fase ele desafia mais a questão de movimentação do *player*, pois não há obstáculo desafiando sua mira você consegue acerta-lo sem problemas, só precisa desviar e atacar basicamente.

Chefe 2: O segundo chefe já há mais dificuldades, ele também tem uma movimentação horizontal só que agora há obstáculos que obstruem o tiro do jogador antes de chegar nele, os obstáculos são pequenos meteoros, então além da movimentação agora é testado a mira do *player*.

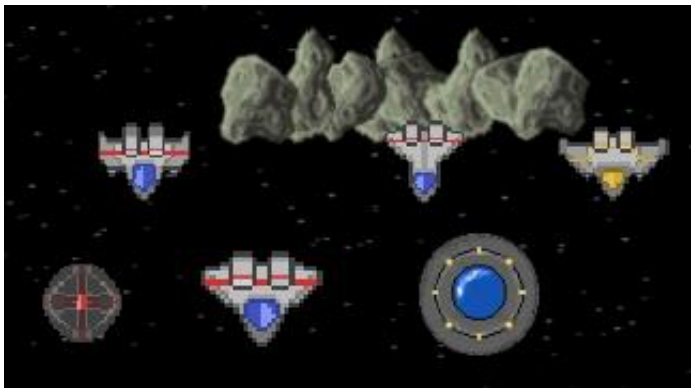
Chefe 3: O terceiro chefe tem uma movimentação diferente, ele não tem ataques diretos, o único ataque dele é a própria movimentação. Ele se movimenta no eixo vertical, e com uma velocidade moderada. Quanto menor seu *hp* mais ele fica veloz.

Chefe 4: Esse talvez o mais desafiador, a movimentação dele é totalmente aleatória. Ele simplesmente bate em um canto e pode ir para qualquer outra direção, além de deixar pequenos tiros durante o caminho. O *player* precisa ter boa movimentação, mira e paciência para saber qual a hora ele deve se aproximar.

Chefe 5: Não tanto desafiador quando o segundo mas é o que contém maior número de *hp*. É mais um teste de resistência para o player continuar sempre atirando. Isso ocorre também porque ele diminui de tamanho conforme seu *hp* vai abaixando, assim dificultando acerta-lo novamente.

Chefe 6: Ele contém três formas distintas, e cada uma tem uma forma de ataque diferente e todas elas são movimentadas através de animação e não script. No primeiro ele tem uma movimentação horizontal e vai soltando bombas durante o percurso. No segundo ele já se movimenta pelas extremidades e solta um ataque que pega todo canto horizontal e vertical em que ele está, então o *player* tem que saber a hora certa que ele vai atirar para não ser pego. Na última fase dele aparecem 3 naves iguais sendo só uma a verdadeira e apenas ela recebe dano. Todas elas ficam e uma animação frenética pela tela confundindo o jogador.

Figura 7 - Chefes do Jogo

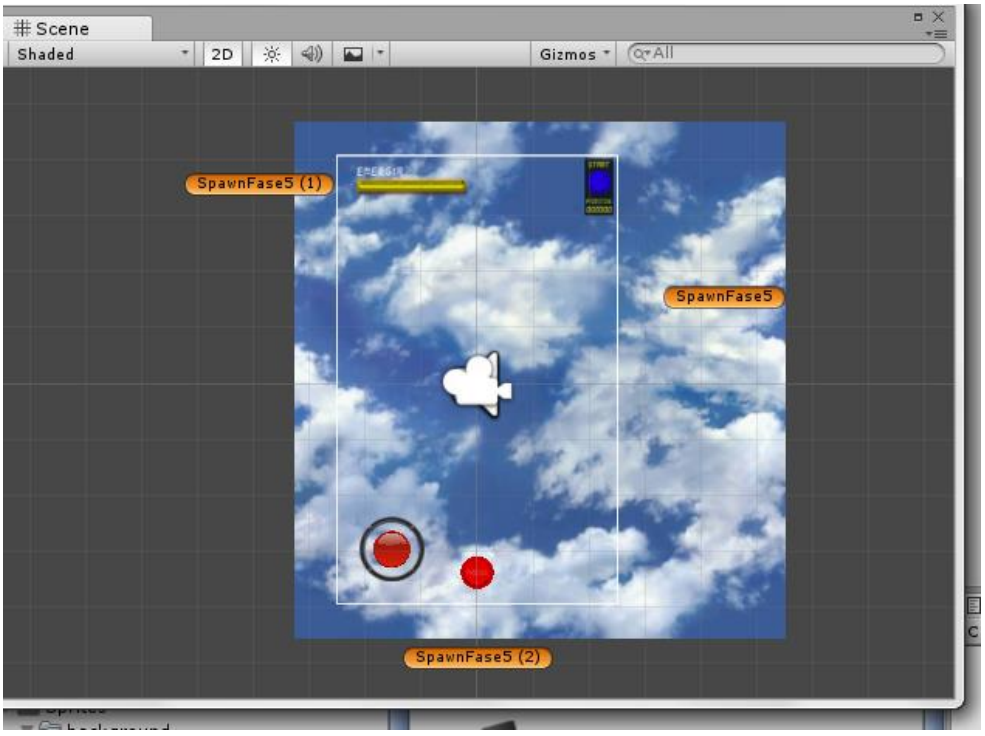


3.2.3. *Spawn* Inimigos

Os inimigos são instanciados na tela através de *spawn* que cada tela contém, os *spawns* ficam nas extremidades das fases, e no jogo a quantidade deles em uma tela varia de acordo com a dificuldade que ela deve apresentar, tem fase que contém dois, tem outras que contém três (Figura 8) e assim por diante.

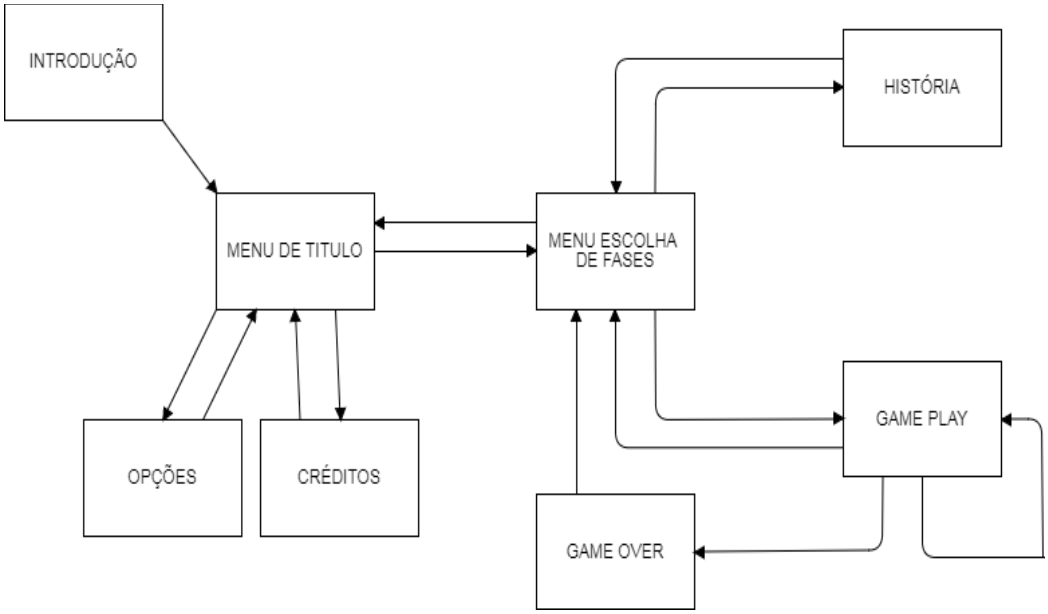
Cada *spawn* tem um controle próprio, tem alguns que ficam com a ajuda de uma animação percorrendo toda a extremidade da fase e soltando os inimigos, tem outros *spawns* que ficam parados em um certo local por exemplo. Muito importante também é o tempo que os inimigos devem aparecer, no começo da fase o tempo é sempre longo, um inimigo instanciado a cada 3 segundos, mas conforme a fase avança a dificuldade deve aumentar e o tempo pode diminuir para 1 segundo ou menos. Na maioria dos casos o objeto que contém o *spawn* fica habilitado durante todo o tempo do jogo e só acaba sendo desabilitado quando o chefe entra em cena, isso para não dificultar ao jogador, e também deixar a atenção dele voltada apenas para o chefe.

Figura 8 - *Spawn* Fase 5



3.3. Fluxograma das Cenas

Figura 9 - Fluxograma das Cenas



A Figura 9 representa a navegação entre as cenas foi feito da maneira mais simples possível, e com o menor número de opções tentando assim facilitar o entendimento do jogador. Na maioria o jogador tem apenas a simples opção de voltar para a cena anterior, mas a casos onde foi necessário adicionar mais navegações como no menu escolha de fases e no *gameplay*, pois existe uma gama maior de possibilidades que podem acontecer. Todos os objetos que fazem as ações de navegação são botões que acompanham um ícone para melhor entendimento, os botões seguem um padrão de cores e ícones que ajudam na identificação da ação que ele vai produzir.

3.4. Sistema de Controle de Fases e História

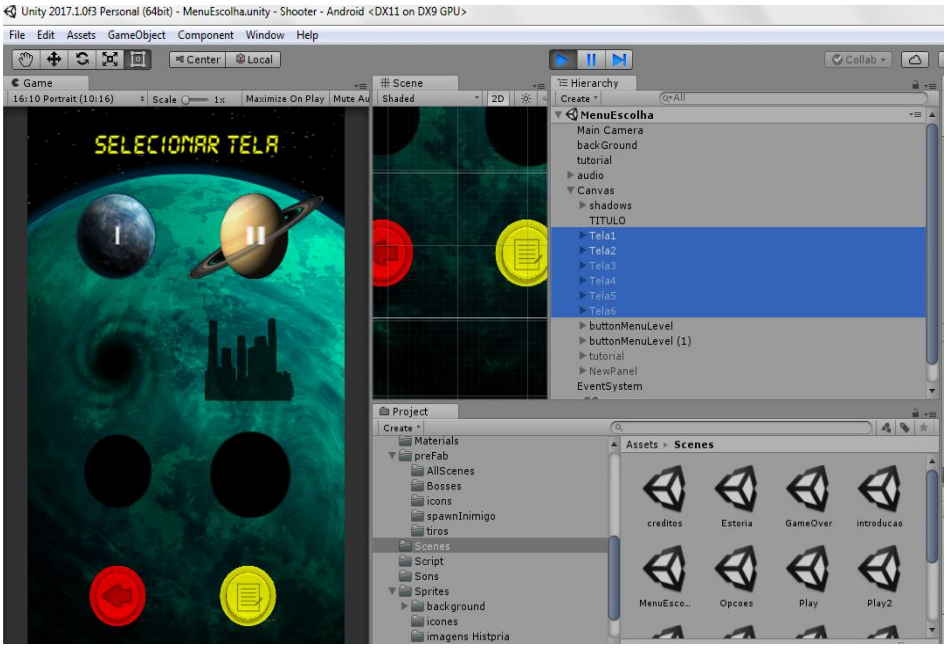
A tela de fases foi feita de forma bem simples mas de maneira funcional. Ele contém 6 botões que quando acionados levam o jogador para a tela selecionada. As telas começarão desativadas, sendo a primeira a única disponível e as telas serão liberadas conforme o jogador supere a anterior. Os botões das telas terão um formato

específico (Figura 10) que remete ao que acontecerá nela ou o local que ela se passará.

O mapa de história foi feito da mesma maneira, a única diferença é que ele conterá 7 trechos da história sendo a introdução já liberada para o jogador, e outros trecos liberados conforme ele avança no jogo. Ele também será acionado por botões e que nesse caso vai acionar um espécie de modal que contém um texto e uma imagem ilustrativa.

O sistemas que faz o gerenciamento dessas duas telas é o mesmo no caso menuEscolha.cs. No começo da cena todos os botões começam desabilitados, então ele recebe um número através de um sistema de salvamento próprio do Unity que é o componente *PlayPreference*, esse número vem em relação a última fase que foi jogada, caso o player tenha terminado a fase esse número é incrementado se não ele continua o mesmo. Então através desse número é feito uma verificação que vai habilitando quais fases devem ser liberadas para o jogador.

Figura 10 – Botões de Ativação das Fases



3.5. Sistema de Áudio

O sistema de áudio foi dividido em duas partes. Uma é a música tema que fica de *background* e outro os efeitos sonoros que ele contém, que engloba som de tiro, explosão, alertas entre outros. Os dois utilizam o mesmo script Audio.cs mas são

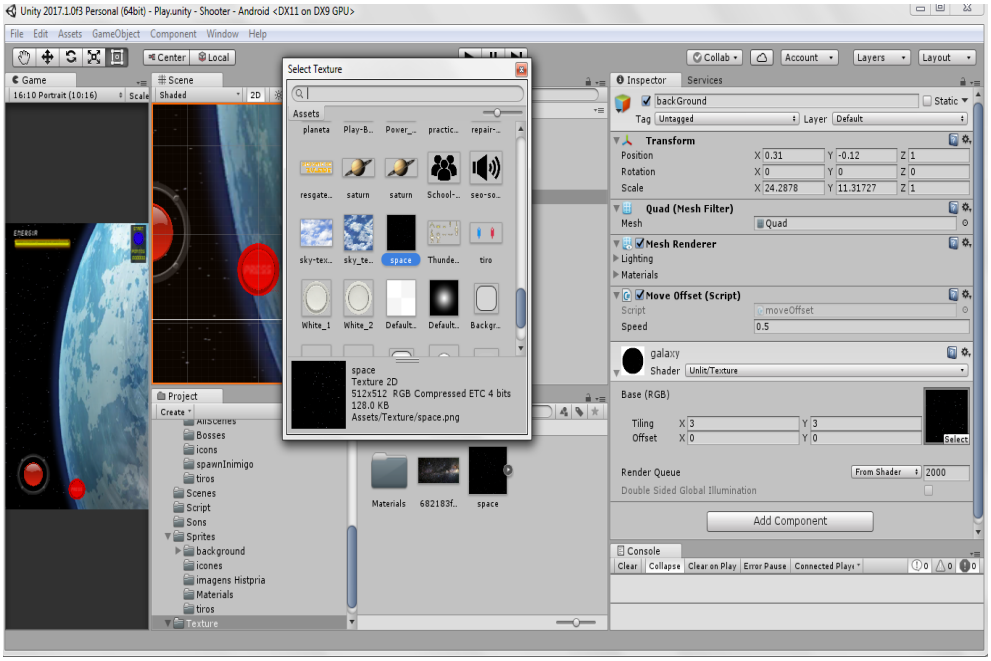
independentes um do outro, ou seja, o jogador pode optar por desligar a música e ouvir apenas os efeitos sonoros assim como o contrário. O sistema é bem simples, existe um *array* que recebe esses sons e dentro do script há uma verificação se eles devem ou não estar ligados, caso o jogador opte por desligar o som que ele escolheu seja da música ou dos efeitos os áudios recebem zero como parâmetro para a entidade de volume desse som, e caso ele esteja ligado ele recebe o valor de cem que é o valor máximo.

No caso a música de background sempre é acionada sem nenhuma dependência. Já os efeitos sonoros ficam em métodos que são chamados através dos objetos que contenham esse efeito, por exemplo o som de explosão só ocorre quando uma nave é explodida aí dentro do script da nave tem uma chamada para o método que toca esse som.

3.6. *Background* e *Parallax*

O *background* do jogo foi feito utilizando a técnica do *parallax*. Esse conceito consiste em movimentar o background do jogo em uma velocidade menor do que o *player* na tela se movimenta dando assim uma sensação mais real de movimentação e também de profundidade para o jogador. A implementação é bem simples, só é necessário criar um material do tipo elipse e adicional uma textura nele como exemplificado na Figura 11, porém essa textura tem que ser contínua, o começo dela deve se encaixar com o seu final, pois senão fica uma imagem quebrada e sem nexo. Depois é só adicionar um simples script no objeto fazendo ele se movimentar, no caso o movimento foi feito no eixo “Y”, mas pode ser adicionado no eixo “X” também. Pode-se controlar também o seu *tiling* que é basicamente o zoom que essa textura vai ter.

Figura 11 - Objeto de *Background*



3.7. Interface

Sendo um jogo *mobile* a construção da interface foi feita de maneira bem objetiva, seja nos menus como no *gameplay* (HUD do jogo). Foi utilizado de botões para fazer a transição das fases, como padrão foi adotado um botão estilo de *arcade* até para remeter um pouco da origem e dar mais sensação do estilo de fliperama ao jogo. Para não poluir visualmente as telas foi utilizado ícones (Figura 12) nos botões para exemplificar quais seriam as suas ações.

Figura 12 – Ícones dos botões

Foi utilizado do mesmo conceito na tela de pause do *gameplay,* apenas botões com ícones contendo significados bem exemplificados, como seta apontando para

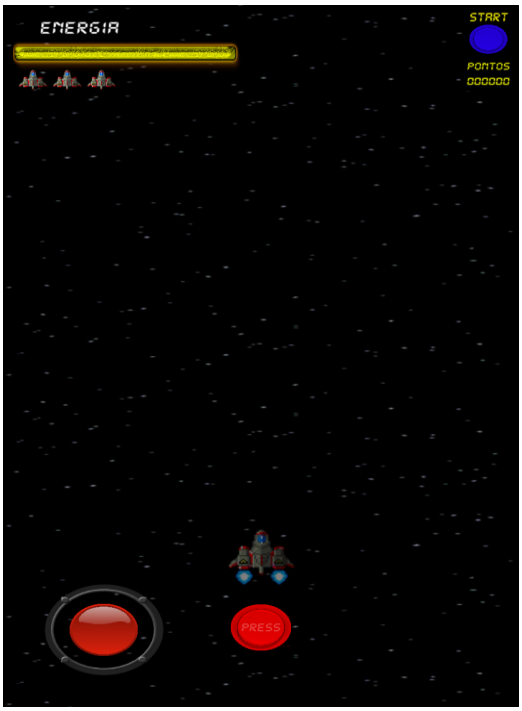
sair, botão com símbolo de play para continuar o jogo além dos botões de que controlam o áudio com símbolo de alto-falante e outro de microfone.

3.8. HUD do Jogo

A HUD do game seguiu o modelo de *less-is-more* (menos é mais) apresentando apenas o que é essencial para o jogador, pois como o game é um gênero de *shooter* ele tem muita movimentação e requer muito reflexo do jogador, então é necessário o maior campo de visão possível para o player.

Os dados que são apresentados para o jogador na tela de *gameplay* conforme a Figura 13 são a quantidade de vidas, que é representada por pequenos ícones da nave que ficam ao lado da barra de vida, e a quantidade desses ícones representam o número de vidas do *player*. A barra de vida fica no canto superior mostrando a quantidade de *life* acompanhada da palavra “energia”, o campo de visão que ela ocupa é sempre o mesmo, a demonstração da quantidade de vida é através das cores, não há um encurtamento da barra em si, mas sim da cor amarela que representa a vida. A pontuação que é apresentada com a palavra “Pontos” seguido dos número de pontuação que o jogador possui e que são atualizados em tempo real na tela, esses pontos são obtidos matando inimigos, pegando moedas ou itens. Por último tem os botões de ações, o *pause* que fica no canto superior direito com uma palavra explicando para que serve, o analógico no canto inferior esquerdo utilizado para movimentar o *player*, botão de ataque “especial” que fica visível apenas quando o jogador possui algum ataque.

Figura 13 - *HUD* do *Gameplay*



3.9. História

Como a plataforma escolhida foi *mobile*, a história se tornou um grande desafio, pois as pessoas procuram no *mobile* em sua maioria jogos casuais que não apresentam muita complexidade ou mesmo aquela imersão que um jogo com uma história mais bem detalhada costuma ter. Então seguindo esse raciocínio foi feito uma história mais curta, simples e bem humorada. Como a história não era o foco principal do jogo, e para não atrapalhar aqueles que não tem interesse ela foi introduzida como um adicional no jogo, o jogador não terá a obrigação de conhecê-la para usufruir do *game*, nem mesmo precisará esperar ou pular alguma parte que ele não ache interessante, ou não queira saber naquele momento.

A história fica em uma cena diferente em que o jogador terá acesso através do menu de fases, ela terá um total de sete trechos, sendo o primeiro uma introdução já liberada, o restante dos trechos serão disponibilizados conforme o jogador passa as fases. Junto com os textos das histórias será mostrado também imagens que ilustram a situação que o texto descreve (Figura 14), dando assim mais criatividade e tentando

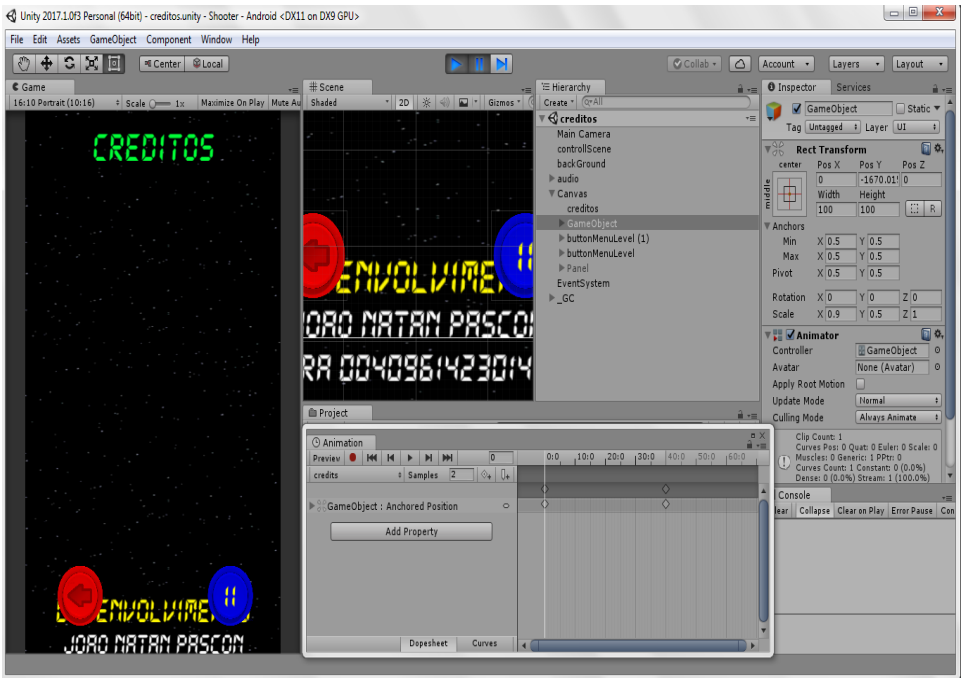
manter o jogador mais interessado nela, pois talvez ele possa não se interessar pelo conteúdo textual do próximo trecho mas pode ficar curioso e ver como será a próxima ilustração.

Figura 14 - Trecho da história e ilustração

3.10. Créditos

Como foi muito utilizado *assets* entre outros componentes no quesito artístico a tela de créditos poderia ficar muito poluída e inadequada então uma saída simples para isso foi a utilização de créditos no estilo de filmes (Figura 15). Foi feita uma simples animação que vai mostrando todos os créditos, e se pessoa quiser ver mais detalhadamente aquele crédito foi colocado um botão que pausa a animação para ela pode pesquisar mais sobre os conteúdos do jogo.

Figura 15 - Implementação Créditos



3.11. *Level Design*

O *level design* do game foi construído baseado no esquema mostrado por Jeannie Novak (2010), que é uma progressão de dificuldade no estilo “plano” onde ele mantém o mesmo grau de dificuldade de um nível para o outro. Isso não quer dizer que o jogo fique menos desafiador porque a proposta é de cada fase desafiar o jogador de uma maneira diferente. Então pode acontecer do jogador achar a fase número dois mais difícil que a número cinco por exemplo, mas isso ocorre porque o ele tem mais habilidade naquilo que a fase cinco cobra do que a segunda fase por exemplo.

4 RESULTADOS

Será apresentado agora alguns dos resultados obtidos tanto no jogo como as opiniões coletadas através de uma pesquisa realizada com o público que experimentou o jogo.

4.1. Tela de introdução

A tela de introdução (Figura 16) tem apenas o papel de demonstrar o logo da Fatec e a uma mensagem representando a sua finalidade.

Figura 16 – Menu de Introdução

4.2. Menus do Jogo

O jogo contém uma série de menus de interação com o usuário, cada menu tem o seu objetivo específico, e nesse tópico será explicado cada um deles.

4.2.1. Menu Inicial

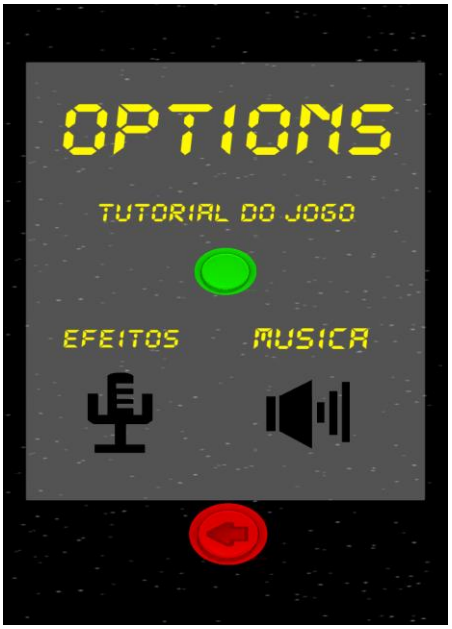
O menu inicial foi feito de maneira bem objetiva, o jogador tem a opção de começar o jogo com um botão de *play* em destaque, e outras duas opções que levam ao menu de *options*, ou a os créditos do jogo. Um botão para ir ao modo história não foi adicionado aí por exemplo para não poluir muito esta tela inicial pois o foco é que o jogador sempre vá direto para começar o *game* e também deixar a logo e nome do jogo em destaque (Figura 17).

Figura 17 - Menu Inicial

4.2.2. Menu de Opções

A tela de opções (Figura 18) é bem simples, ela contém apenas um tutorial que o jogador pode acessar e o controle sobre a música e os efeitos sonoros do jogo, mas esses controles também podem ser alterados pelo menu de *pause* do *gameplay*. Opções não contém a escolha de nível de dificuldade porque essa quesito foi trabalhado nas fases na qual o jogador vai progredindo.

Figura 18 - Menu Opções



4.2.3. Menu de Seleção Fases

A Figura 19 apresenta como ficou a tela de seleção de fases. Na seleção de fases aparecerão todas as fases do jogo. No caso as fases que ainda não estiverem liberadas ficarão com uma sombra indicando para o usuário que ainda há fases para serem liberadas.

Figura 19 - Menu Seleção de Fases

4.2.4. Menu de História

O menu de história é bem similar ao menu de fases. Ele contém botões que quando acionados liberam um modal com o trecho da história. Esse caso ele não contém sombras pois todos os botões são iguais, só muda os números que cada um contém.

Figura 20 - Menu Seleção de História

4.2.5. Menu de Pause do *Gameplay*

Menu de *pause* (Figura 21) como citado possui também a possibilidade de controle dos efeitos sonoros e da música que podem ser habilitados ou desabilitados sem precisar ir ao menu de opções, e conta com os botões padrões de um *gameplay* de fases, que é um botão de continuar o jogo atual, outro para recomeçar a fase atual e outro para voltar ao menu de seleção de fases, caso ele queira trocar a fase atual. O tamanho do menu foi pensado levando em consideração deixar visível as informações do *player*, como quantidade de vidas, porcentagem da barra de *hp* e a pontuação, pois assim não foi necessário incluir tais informações também nessa tela de menu.

Figura 21 - Menu *Start Gameplay*

4.3. Tutorial

Foi acrescentado um simples tutorial (Figura 22) para o jogador. Ele é um modal que é mostrado quando o jogador entra na tela de seleção de fases. Nele é descrito como funcionam as ações da nave, fala sobre a possibilidade de *power ups* e também sobre como acessar o modo história. Esse tutorial fica disponível no menu de opções para quando o jogador quiser ter acesso novamente a ele.

Figura 22 - Tutorial

4.4. Game Over

A Figura 23 representa a cena de *game over* que é bem simples, contém apenas uma imagem acompanhada da frase e um botão para o jogador voltar ao menu de seleção de fases.

Figura 23 - *Game Over*

4.5. Telas do *Gameplay*

Aqui está o resultado de como ficou as telas de *gameplay* do jogo, mostrando o seu *background*, alguns inimigos que ela contém, entre outros aspectos.

4.5.1. Fase 1

Contém um background mais comum do universo e um objeto representando a terra do lado esquerdo (Figura 24). A terra fica girando em seu próprio para dar um sensação de movimentação ao jogador.

Figura 24 - Fase 1

4.5.2. Fase 2

Tem o mesmo *background* da fase anterior, a principal diferença desta em relação às outras fase são os meteoros (Figura 25) que apenas essa contém.

Figura 25 - Fase 2

4.5.3. Fase 3

Fase número três contém um *background* único (Figura 26). O objetivo é acompanhar a história, que no caso fala que o personagem foi puxado por um buraco negro e dentro dele há um lugar novo.

Figura 26 - Fase 3

4.5.4. Fase 4

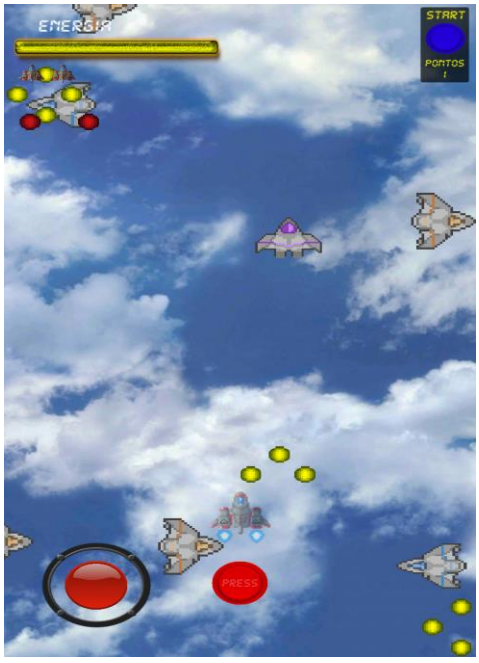
Acompanhando a história o *background* dessa fase representa uma cidade (Figura 27), que segundo a história é onde Seu José está invadindo. No caso as luzes fazem o papel de farol na tentativa de encontrar invasores.

Figura 27 - Fase 4

4.5.5. Fase 5

Fase 5 é para representar a fuga de Seu José da cidade após resgatar a sua vaca. Além de um *background* novo, ele é a única que contém inimigos que aparecem do lado oposto da fase (Figura 28).

Figura 28 - Fase 5



4.5.6. Fase 6

Na última fase (Figura 29) a terra volta como destaque no *background* isso porque foi a tentativa de mostrar paga o jogador que o personagem conseguiu voltar até sua casa, mas que foi surpreendido no final e terá uma última provação enfrentando o chefe final.

Figura 29 - Fase 6

4.6. Efeitos do *Player*

Esses são alguns efeitos que acontecem com o player quando ele pega *power ups*, no caso quando pega um item de arma ou de escudo (Figura 30) ou quando ele usa o especial (Figura 31).

Figura 30 - Efeitos *Power Ups*

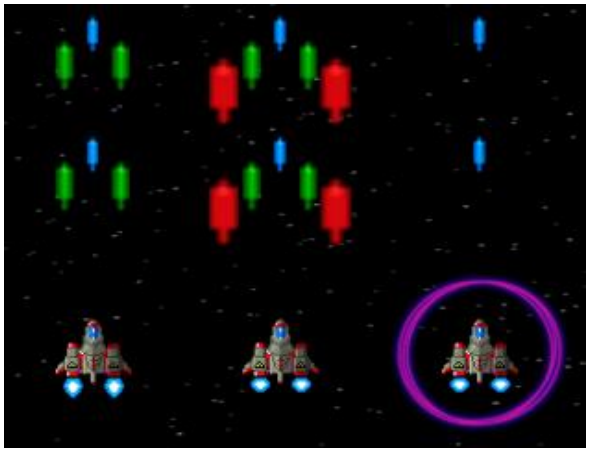
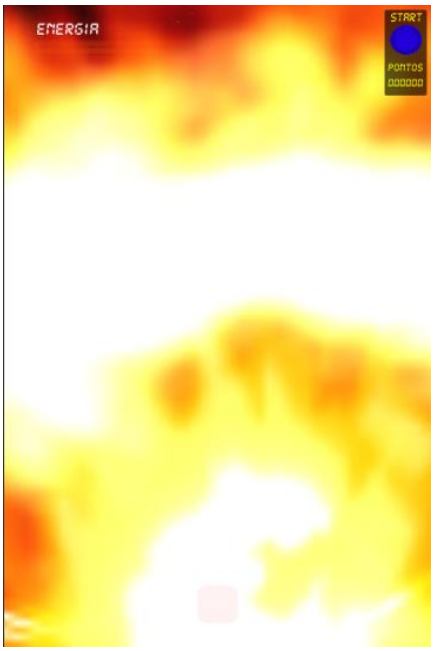


Figura 31 - Efeito Especial

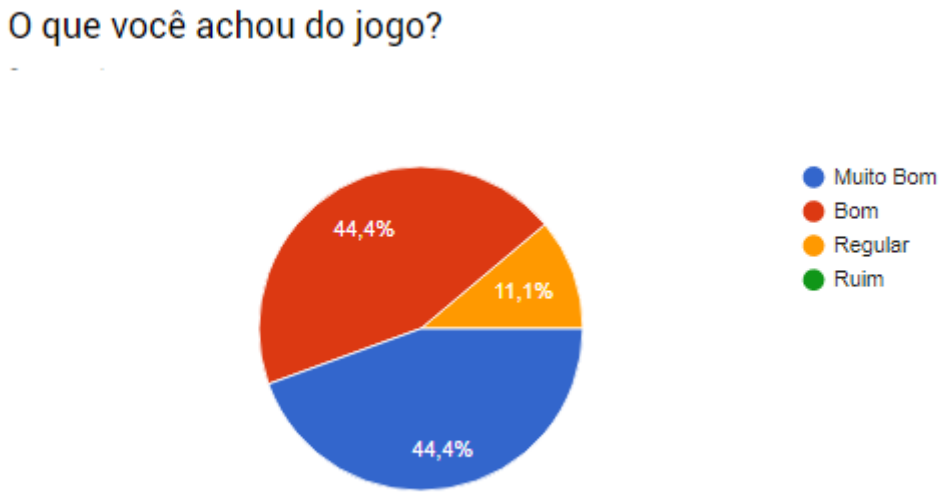


4.7. Pesquisa de Opinião Sobre o Jogo

A pesquisa teve um total de 7 perguntas que envolviam opiniões sobre o jogo, história, controles, mercado para o jogo e terminando com um seção para comentários, críticas e sugestões. Ela foi realizada através de um formulário do **Google Forms** (Google, 2017), e ficou disponível entre os dias 13/11 até o dia 20/11. Foram obtidas no total 9 repostas.

Pergunta 1: Simples e direta para saber a qualidade do jogo, e o resultado foi satisfatória sendo quase 90% aprovando o jogo (Figura 32).

Figura 32 - Pergunta 1

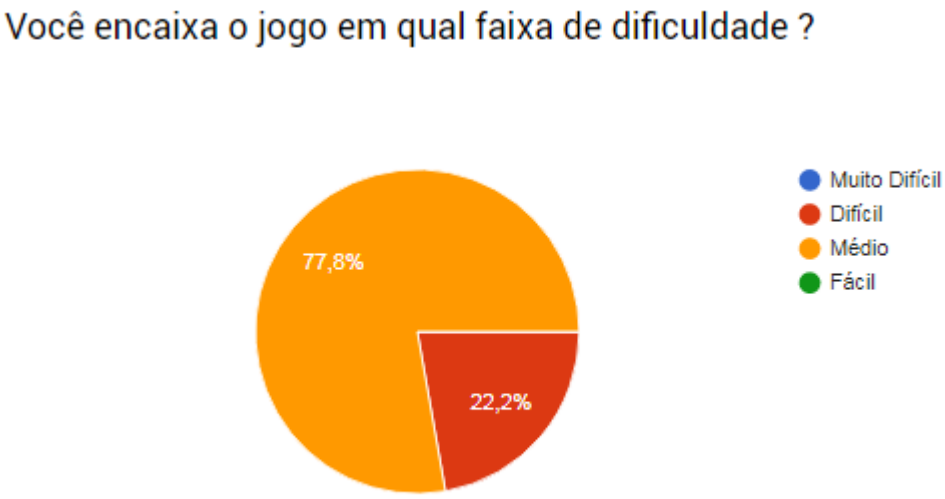


Pergunta 2: Talvez a questão mas polêmica do jogo que seria o controle, e o resultado foi que ninguém reprovou o analógico mas preferia ter o poder de decisão sobre o qual usar, então deixando em aberto uma futura melhoria (Figura 33).

Figura 33 - Pergunta 2

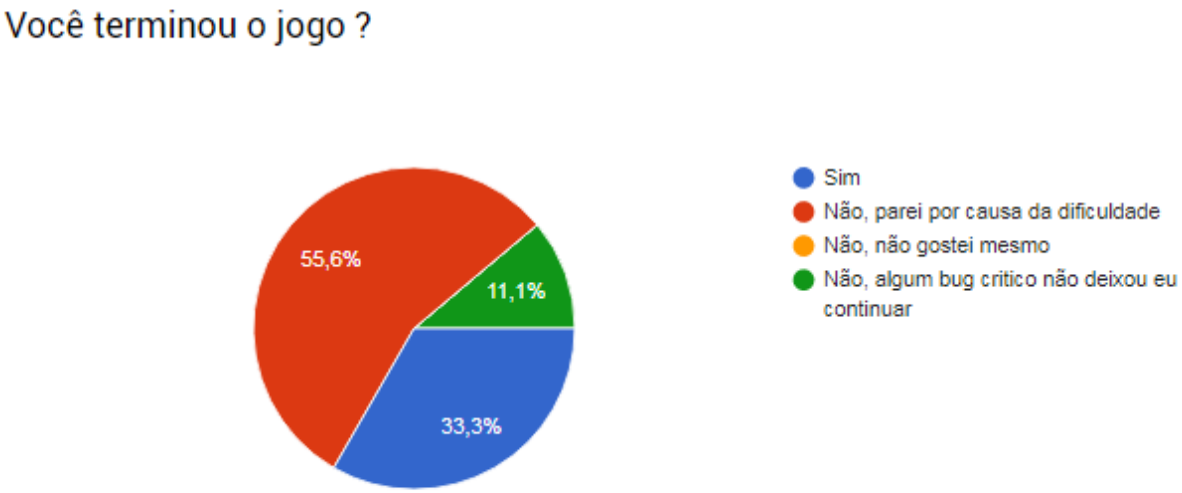
Pergunta 3: A Fim de analisar o fator de *level design* para ver se ele foi bem construído, o resultado foi satisfatória porque ele atinge o jogador mais comum e não apenas o *hardcore*, mas temos um fator curioso que contradiz isso que veremos a seguir (Figura 34).

Figura 34 - Pergunta 3



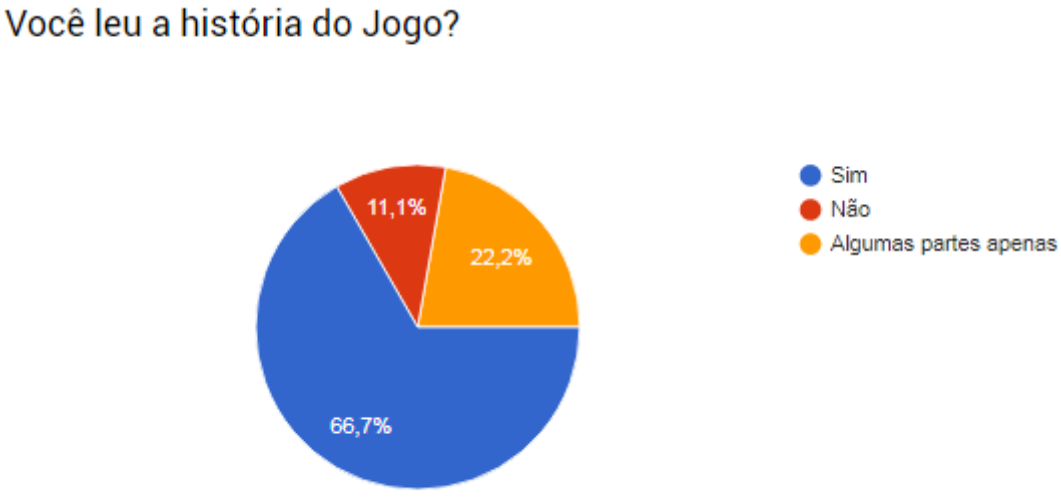
Pergunta 4: Apesar da grande maioria ter colocado o jogo em um nível médio, muitos desistiram exatamente pela dificuldade, levando então uma dúvida sobre o balanceamento de *level design* do jogo, onde até alguma tela o jogo estava muito fácil e em alguma outra tela ele estava impossível de passar, causando então esse desvio. Outro questão é que uma pessoa levantou um problema de bug crítico, que foi reportado pessoalmente e logo corrigido (Figura 35).

Figura 35 - Pergunta 4



Pergunta 5: Outro fator interessante que contradiz próprias pesquisa citadas aqui, que tem um grande número de usuário que se importam sim com a história no celular (Figura 36).

Figura 36 - Pergunta 5



Pergunta 6: Uma pergunta mais estratégica para futuros projetos, condiz com a pesquisa citada nesse trabalho, que a maioria dos usuários aceitam propagandas nos jogos para não precisaram pagar por ele (Figura 37).

Figura 37 - Pergunta 6

Pergunta 7: A única dissertativa que teve o título “Dê sua opinião, sugestão ou crítica sobre o jogo.” Aqui estão algumas das respostas mais relevantes.

“Poderia colocar o modo de tiro automático, já que por várias vezes os dedos ficam na frente da nave.”

“O controle não responde em alguns momentos, e a segunda fase pode ficar bem difícil sem uso de power ups. Tirando isso o jogo está muito bom.”

“Botão de analógico poderia ser maior, às vezes seu dedo sai da área de toque e você perde o controle da nave.”

4.7.1. Avaliação sobre o *Feedback*

Os feedbacks ficaram de acordo com o esperado. A parte mais criticada foi sobre o controle, pois aparentemente alguns tiverem dificuldade com o analógico, o que já era previsto pois era uma nova forma de movimentação. Outro aspecto interessante foi sobre a história, aonde ela teve boa aceitação, o que pode ter ajudado foi o gênero cômico dela, que despertou o interesse do jogador. Em relação ao *Level Design* é uma coisa que precisa ser melhor trabalhada, pois o jogo deve ter um nível de desafio mas ao mesmo tempo não dificultar ao ponto do usuário desistir de jogar.

4.8. Futuras Implementações e Melhorias

Serão apresentadas a seguir algumas melhorias que podem ser feitas levando em consideração a pesquisa e outras implementações extras mas que por conta do tempo não foi possível. Seriam coisas simples mas que por nunca ter feito a parte técnica poderia levar mais tempo e acabaria ficando incompleta, então esse tempo que seria utilizado para isso foi usado para o polimento do jogo em si.

4.8.1. Melhorias em Relação à Pesquisa

A principal melhoria seria a questão do controle, onde deverá ser criado um controle também por movimentação de toque na tela e deixar isso a livre escolha do jogador. Outra questão seria um balanceamento de nível mais bem trabalhado para não ter tantas pessoas sem terminar o jogo.

4.8.2. Implementações Extras

Seria dois tipos de *mini-games*, o primeiro seria uma fase cheia de moedas e com meteoros caindo como obstáculo parecido com a segunda fase do jogo, e o objetivo seria pegar o número máximo de moedas antes de ser destruído. E o segundo seria uma tela com milhares de inimigos e o objetivo seria sobreviver ao maior tempo possível, mas o jogador só poderia usar o tiro básico e nenhum *power-up*. Nos dois casos seriam salvo os recordes dos jogadores para instigar uma disputa entre amigos.

Seria um sistema onde o jogador poderia usar as moedas que ele ganhou no jogo para comprar novos itens, poderia ser *skins* para a nave, novas naves, desabilitar poderes ou mesmo até novas fases secretas ou *mini-games*.

4.9. Link do Jogo

O jogo está disponível na versão beta para dispositivos Android no seguinte endereço:

https://play.google.com/store/apps/details?id=com.Company.ProductName.Space

5. CONSIDERAÇÕES FINAIS

O resultado foi muito satisfatório apesar do pouco tempo de projeto. Foram apenas três meses isso por conta de outro projeto iniciado em grupo mas que não deu certo. O principal objetivo foi alcançado que era obter mais conhecimento técnico em um projeto envolvendo um jogo, não apenas o conhecimento técnico mas também experiência com a *engine* Unity que é uma das mais utilizadas na criação de jogos, abrindo assim caminho para projetos futuros.

Como o trabalho foi uma produção solo nem todas as partes tiveram a atenção necessária. Como o foco pessoal era obter conhecimento na parte de programação o lado artístico acabou ficando sem grande atenção, então a arte ficou mais por conta de *sprites* e imagens baixadas da Asset Store da própria *engine*, que apesar de simples não deixou a desejar e cumpriu o seu papel.

Os feedbacks do público foram de grande importância para entender melhor quais foram os pontos positivos e negativos do trabalho, comentários sobre o controle da nave pro exemplo são de grande valor, pois eles são o ponta pé inicial para o planejamento futuro que irá contém implementações de alguns elementos extras no jogo como *minigames* e também melhoria em relação ao controle e nível de dificuldade.

6. REFERÊNCIAS BIBLIOGRÀFICAS

Pesquisa Game Brasil 2017. Disponível em: <https://www.pesquisagamebrasil.com.br/pesquisagamebrasilgratis>. Acesso em: 02 de Novembro de 2017.

NOVAK, Jeannie – **Desenvolvimento de Games**. 2° ed. São Paulo. Cengage Learning. 2010.

Patrick. Canal de Vídeo Aulas no Youtube **GameMoviesPro**. Disponível em:<https://www.youtube.com/user/GameMoviesPro>. Acesso em: 05 de fevereiro de 2017.

Sonic Wings. Nintendo, Super Nintendo Entertainment System, 1994.

7. APÊNDICE

armaNave.cs - Controla a arma da nave, instância os tiros e verifica os *power ups*, referentes às armas.

```
public class armaNave : MonoBehaviour {
public  GameObject          tiroPreFab;
public  GameObject[]        arms;
public  float               forcaTiro;
public  Audio               scriptAudio;
void Start () {
        scriptAudio = FindObjectOfType (typeof(Audio)) as Audio;
}
void Update () {
        if(Input.GetButtonDown("Fire2") || Input.GetButtonDown("Fire1")){
                atirar ();
        }
}
public void atirar(){
        GameObject tempPrefab = Instantiate (tiroPreFab, transform.position,
transform.rotation) as GameObject;
        tempPrefab.GetComponent<Rigidbody2D> ().AddForce (new Vector2 (0,
forcaTiro));
        scriptAudio.audioLaser ();
}
}
```

_GC.cs - Controla o sistema de transição de telas, HUD entre outros aspectos do *gameplay*.

```
public class _GC : MonoBehaviour {

public Text                 pontos;
public int                  pontuacao;
public  Transform           posicaoVidas;
public  int                 vidasExtras;
public  GameObject          iconeVidas;
```

```
public  GameObject[]        extras;
public  Button              buttonItem;
public  GameObject          itemEspecial;
public  int                 slotEspecial;
public  Transform           spawnPlayer;
public  GameObject          preFabPlayer;
public  GameObject          Inimigo;
public  GameObject          naveChefe;
public  GameObject          subChefe;
public  Inimigo             scriptInimigo;
public  Player              scriptPlayer;
public  Button              ativaTela;
private bool                pause;
public GameObject           panelPause;
public float                tempFase=0, tempFase2=0, tempController;

void Start () {
        pause = true;
        onPause ();
        panelPause.SetActive (false);
        buttonItem.interactable = true;
        Inimigo.SetActive (true);
        VidasExtras (); //crias os icones da via
        scriptInimigo = FindObjectOfType(typeof(Inimigo)) as Inimigo;
        scriptPlayer = FindObjectOfType(typeof(Player)) as Player;
}
void Update () {
        tempController += Time.deltaTime;
        pontos.text = "" + pontuacao.ToString ();
        if (tempFase < tempController) {
                chamaChefe ();
        }
        else if (tempFase2 < tempController) {
                chamaSubChefe ();
        }
        if(Input.GetButtonDown("Fire1")){ativaTela.interactable = true;}
}

void VidasExtras(){
        float posXico;
```

```
                GameObject tempVidas;
                foreach (GameObject v in extras)
                {
                        if (v != null) // verifica quais estao ativos para depois destruir eles
                        {
                                Destroy (v);
                        }
                }
                for (int i = 0; i < vidasExtras; i++)
                {
                        posXico = posicaoVidas.position.x + (0.3f * i);
                        tempVidas = Instantiate (iconeVidas) as GameObject;
                        extras[i] = tempVidas;
                        tempVidas.transform.position = new Vector3 (posXico,
posicaoVidas.position.y, posicaoVidas.position.z);
                }
                StartCoroutine ("renascerEspera");
        }
        public void morreu(){
                vidasExtras--;
                if (vidasExtras >= 0) {
                        VidasExtras ();
                        buttonItem.interactable = true;
                }
                else{
                        PlayerPrefs.SetInt ("pontos", pontuacao);
                        SceneManager.LoadScene ("GameOver");
                }
        }
        public void chamaChefe(){
                naveChefe.SetActive(true); //seta o chefe na tela
        }
        public void chamaSubChefe(){
                subChefe.SetActive(true); //seta o chefe na tela
        }
        public void MenuEscolha(){
                SceneManager.LoadScene ("MenuEscolha");
        }
        IEnumerator renascerEspera(){
                yield return new WaitForSeconds (1);
```

```
                GameObject tempPlayer = Instantiate (preFabPlayer) as GameObject;
                tempPlayer.transform.position = spawnPlayer.position;
                tempPlayer.name = "Player";
                scriptPlayer.GetComponent<SpriteRenderer> ().color = new Color (1, 1, 1,
0.2f);

        }
        public void adicionaVida(){
                vidasExtras++;
                float posXico;
                GameObject tempVidas;
        foreach (GameObject v in extras)
                {
                        if (v != null) // verifica quais estao ativos para depois destruir eles
                        {
                                Destroy (v);
                        }
                }
                for (int i = 0; i < vidasExtras; i++)
                {
                        posXico = posicaoVidas.position.x + (0.3f * i);
                        tempVidas = Instantiate (iconeVidas) as GameObject;
                        extras[i] = tempVidas;
                        tempVidas.transform.position = new Vector3 (posXico,
posicaoVidas.position.y, posicaoVidas.position.z);
                }
        }
        IEnumerator especial(){
                GameObject tempPrefabEx = Instantiate (itemEspecial, transform.position,
transform.rotation) as GameObject;
                yield return new WaitForSeconds (0.7f);
                Destroy (tempPrefabEx);
        }
        public void chamaEspecial(){
                        StartCoroutine ("especial");
                        buttonItem.interactable = false;
        }
        public void onPause(){
                pause = !pause;
                if (!pause) {
```

```
                Time.timeScale = 1;
                panelPause.SetActive (false);
        } else if (pause) {
                Time.timeScale = 0;
                panelPause.SetActive (true);
        }
}
public void RestartSccene(){
        SceneManager.LoadScene (SceneManager.GetActiveScene ().name);
}
public void Opcoes(){
        SceneManager.LoadScene ("Opcoes");
}
public void Creditos(){
        SceneManager.LoadScene ("Creditos");
}
public void     play1(){
        SceneManager.LoadScene ("Play");
}
public void     play2(){
        SceneManager.LoadScene ("Play2");
}
public void play3(){
        SceneManager.LoadScene ("Play3");
}
public void Estoria(){
        SceneManager.LoadScene ("Estoria");
}
public void play4(){
        SceneManager.LoadScene ("Play4");
}
public void play5(){
        SceneManager.LoadScene ("Play5");
}
public void play6(){
        SceneManager.LoadScene ("Play6");
}
public void TelaInicial(){
        SceneManager.LoadScene ("TelaInicial");
}
```

```
    public void creditos(){
        SceneManager.LoadScene ("creditos");
    }
}
```

inimigoChefe.cs - Controla todos os chefes, acontecem algumas variações de um chefe para o outro.

```
public class inimigoChefe : MonoBehaviour {

    private Rigidbody2D         chefeRb;
    public  int                 velocidade;
    public  float               movimentoX, movimentoY;
    public  Transform           limiteLeft, LimiteRight;
    public  int                 HP;
    public  GameObject          prefabExplosao;
    public  GameObject          prefabTiro;
    public  Transform[]         armaPosition;
    public  int                 tempTiro;
    private float               esperaTiro;
    private SpriteRenderer      chefeSR;
    public  GameObject          Spawn;
    private Audio               scriptAudio;
    public  _GC                 script_GC;
    private MenuEscolha         MenuEscolhaScript;
    public  float               tempoParaDano, tempo;
    public  GameObject          panelDanger;
    private int                 proximaFase=1;

    void Start () {
        StartCoroutine ("danger");
        scriptAudio = FindObjectOfType (typeof(Audio)) as Audio;
        script_GC = FindObjectOfType (typeof(_GC)) as _GC;
        MenuEscolhaScript = FindObjectOfType (typeof(MenuEscolha)) as
MenuEscolha;
        chefeRb = GetComponent<Rigidbody2D> ();
        chefeSR = GetComponent<SpriteRenderer> ();
        scriptAudio.audioAlarmeChefe ();
        Spawn.SetActive (false);
```

```
	}
	void Update () {
		tempo += Time.deltaTime;
		if (transform.position.y > 2.65f) {
			chefeRb.velocity = new Vector2 (0, movimentoY * velocidade);
		}
		else
		{
			chefeRb.velocity = new Vector2 (movimentoX * velocidade, 0);
		}
		esperaTiro += Time.deltaTime;
		if (tempTiro <= esperaTiro) {
			esperaTiro = 0;
			StartCoroutine ("atirar");
		}
		if ((transform.position.x - 0.2) > LimiteRight.position.x) {
			chefeRb.transform.position = new Vector2 (LimiteRight.position.x,
transform.position.y);
			movimentoX *= -1;
		}
		if ((transform.position.x + 0.2) < limiteLeft.position.x) {
			chefeRb.transform.position = new Vector2 (limiteLeft.position.x,
transform.position.y);
			movimentoX *= -1;
		}
	}
	void OnTriggerEnter2D(Collider2D col){
		switch (col.gameObject.tag) {
		case "tiro":
			tomarDanoChefe (1);
			break;
		case "especial":
			tomarDanoChefe (5);
			break;
		}
	}
	public void tomarDanoChefe(int dano){
		if (tempo > tempoParaDano) {
			tempo = 0;
			HP -= dano;
```

```
                chefeSR.color = new Color (255, 0, 0);
                StartCoroutine ("pisca");
                if (HP <= 0) {
                        explodirChefe ();
                }
        }
    }
    public void explodirChefe(){
        scriptAudio.audioExplosion ();
        Destroy (this.gameObject);
        Instantiate (prefabExplosao, transform.position, transform.rotation);
        proximaFase = (proximaFase >= PlayerPrefs.GetInt ("fases")) ? 1
:PlayerPrefs.GetInt ("fases");
        PlayerPrefs.SetInt ("fases", proximaFase);
        SceneManager.LoadScene ("MenuEscolha");
    }

    IEnumerator atirar(){
        yield return new WaitForSeconds (1);
        for (int i = 0; i < 3; i++) {
                GameObject tempFabTiro = Instantiate (prefabTiro,
armaPosition [i].position, prefabTiro.transform.rotation) as GameObject;
                tempFabTiro.GetComponent<Rigidbody2D> ().AddForce (new
Vector2 (0, -100));
        }
        if (HP <= 10) {
                for (int i = 0; i < 5; i++) {
                        if (i < 3) {
                                GameObject tempFabTiro = Instantiate (prefabTiro,
armaPosition [i].position, prefabTiro.transform.rotation) as GameObject;
                                tempFabTiro.GetComponent<Rigidbody2D> ().AddForce
(new Vector2 (0, -300));
                        }
                        else {
                                GameObject tempFabTiro = Instantiate (prefabTiro,
armaPosition [i].position, prefabTiro.transform.rotation) as GameObject;
                                tempFabTiro.GetComponent<Rigidbody2D> ().AddForce
(new Vector2 (0, -300));
                        }
                }
```

```
                }
        }
        IEnumerator pisca(){//para piscar a cor quando tomar dano
                yield return new WaitForSeconds (0.2f);
                chefeSR.color = new Color (255, 255, 255);
        }
        IEnumerator danger(){
                panelDanger.SetActive (true);
                yield return new WaitForSeconds (3f);
                panelDanger.SetActive (false);
        }
}
```

enemy.cs - É o sistema dos inimigos, controla a vida, animação, tempo de tiro entre outras coisas.

```
public class Inimigo : MonoBehaviour {

        private  Rigidbody2D        inimigoRb;
        private  Animator           animatorInimigo;
        public  float               velocidade;
        private  int                direcao;
        public  Transform           arma;
        public  GameObject          tiroPreFab;
        public  float               forcaTiroY, forcaTiroX;
        public    int               movimentoX, movimentoY;
        public  float               tempoCurva;
        public  int                 sorteio, chanceCurva;
        private  float              tempTime;
        private  int                rand;
        public  int                 chanceTiro;
        public  int                 tempTiro;
        private float               tempTimeTiro;
        public  int                 HP;
        public  GameObject          explosaoPreFab;
        public  int                 pontosGanhos;
        public  _GC                 script_GC;
        public  GameObject[]        dropPowerup;
        public  int                 chanceDrop;
```

```
private  Audio                scriptAudio;
public   int                  acionadorChefe;
public   GameObject           prefabMoeda;

void Start () {
        script_GC = FindObjectOfType (typeof(_GC)) as _GC;
        scriptAudio = FindObjectOfType (typeof(Audio)) as Audio;
        inimigoRb = GetComponent<Rigidbody2D> ();
        animatorInimigo = GetComponent<Animator> ();
        movimentoY = -1;
}
void Update () {
        tempTime += Time.deltaTime;
        if (tempTime >= tempoCurva)
        {
                tempTime = 0;
                rand = Random.Range (0, 100);
                if (rand <= chanceCurva)
                {
                        rand = Random.Range (0, 100);
                        if (rand < 50) {
                                movimentoX = 1;
                                direcao = -1;
                        }
                        else {
                                movimentoX = -1;
                                direcao = 1;
                        }
                }
                else
                {
                        movimentoX = 0;
                        direcao = 0;
                }
        }
        tempTimeTiro += Time.deltaTime;
        if (tempTimeTiro >= tempTiro)
        {
                tempTimeTiro = 0;
                rand = Random.Range (0, 100);
```

```
                        if (chanceTiro >= rand)
                        {
                                atirar ();
                        }
                }
                inimigoRb.velocity = new Vector2 (movimentoX * velocidade, movimentoY *
velocidade);
        }
        void atirar(){
                GameObject tempPrefab = Instantiate (tiroPreFab, arma.position,
arma.rotation) as GameObject;
                tempPrefab.GetComponent<Rigidbody2D>().AddForce(new
Vector2(forcaTiroX, forcaTiroY));
        }
        void OnTriggerEnter2D(Collider2D col){
                switch (col.gameObject.tag)
                {
                case "tiro":
                        tomarDano(1);
                        break;
                case "especial":
                        Destroy (this.gameObject);
                        explodir ();
                        break;
                case "limiteDown":
                        Destroy (this.gameObject);
                        print ("DESTRUIDO");
                        break;
                }
        }
        void OnCollisionEnter2D(Collision2D col){
                switch (col.gameObject.tag) {
                case "Player":
                        tomarDano (1);
                        break;
                case "naveChefe":
                        tomarDano ((int) HP);
                        break;
                }
        }
```

```
	void tomarDano(int danoTomado){
		HP -= danoTomado;
		if (HP <= 0)
		{
			explodir ();
		}
	}
	void explodir(){
		int i;
		Destroy (this.gameObject);
		Instantiate (explosaoPreFab, transform.position, transform.rotation);
		scriptAudio.audioExplosion ();
		script_GC.pontuacao += pontosGanhos;
		i = Random.Range (0, dropPowerup.Length - 1 );
		sorteio = Random.Range (0, 100);
		if (sorteio <= chanceDrop) {
			GameObject tempDrop = Instantiate (dropPowerup[i],
transform.position, transform.rotation) as GameObject;
		}
		else
		{
			Instantiate (prefabMoeda,
transform.position,prefabMoeda.transform.rotation);
		}
	}
}
```

Audio.cs - Gerencia todo o sistema de áudio do jogo, tanto a música tema quanto os efeitos sonoros.

```
public class Audio : MonoBehaviour {

	public AudioSource[]   audios;
	public AudioSource     musics;
	public GameObject[]   buttonsAudio;
	public GameObject[]   buttonsMusic;
	public int                 prefAudio, prefMusic;

	void Start () {
```

```
        prefAudio = PlayerPrefs.GetInt ("audioController");
        prefMusic = PlayerPrefs.GetInt ("musicController");
        if (prefAudio == 1)
                enableAudio ();
        else
                disableAudio ();

        if (prefMusic == 1)
                enableMusic ();
        else
                disableMusic ();
}
public void audioLaser(){
        audios [0].enabled = true;
        audios [0].Play ();
}
public void audioExplosion(){
        audios[1].enabled = true;
        audios[1].Play ();
}
public void audioPoweUp(){
        audios[2].enabled = true;
        audios[2].Play ();
}
public void audioAlarmeVida(){
        audios[3].enabled = true;
        audios[3].Play ();
}
public void audioAlarmeChefe(){
        audios[4].enabled = true;
        audios[4].Play ();
}
public void enableAudio(){
        foreach (var item in audios) {
                item.volume = 100f;
        }
        buttonsAudio [0].SetActive (false);
        buttonsAudio [1].SetActive (true);
        PlayerPrefs.SetInt ("audioController", 1);
}
```

```
        public void disableAudio(){
                foreach (var item in audios) {
                        item.volume = 0f;
                }
                buttonsAudio [1].SetActive (false);
                buttonsAudio [0].SetActive (true);
                PlayerPrefs.SetInt ("audioController", 0);
        }
        public void enableMusic(){
                musics.volume = 100f;
                buttonsMusic [0].SetActive (false);
                buttonsMusic [1].SetActive (true);
                PlayerPrefs.SetInt ("musicController", 1);
        }
        public void disableMusic(){
                musics.volume = 0f;
                buttonsMusic [1].SetActive (false);
                buttonsMusic [0].SetActive (true);
                PlayerPrefs.SetInt ("musicController", 0);
        }
}
```

MenuEscolha.cs - Script que faz a verificação de qual fase deve ser liberadas.

```
public class MenuEscolha : MonoBehaviour {

        public   GameObject[]        fases;
        public   int                 fase;

        void Start () {
                fase = PlayerPrefs.GetInt ("fases");
                for (int i = 0; i <= fase; i++) {
                        fases [i].SetActive (true);
                }
        }
}
```

meteoros.cs - Esse script controla o sistema de meteoros da fase 2, *spawn*, recompensa de itens entre outras coisas.

```
public class meteoros : MonoBehaviour {

	public  int 			HP;
	public  GameObject 		prefabexplosao;
	public  int 			contadorMeteor;
	public  SpawnMeteoros 		scriptMeteor;
	public  GameObject[] 		dropItens;
	public  GameObject 		miniMeteors;
	public  GameObject 		meteorTransform;
	public  GameObject[] 		extras;

	void Start () {
		scriptMeteor = FindObjectOfType (typeof(SpawnMeteoros)) as SpawnMeteoros;

	}
	void Update () {
		meteorTransform = GetComponent<GameObject> ();
	}
	void OnTriggerEnter2D(Collider2D col){
		switch (col.gameObject.tag) {
		case "tiro":
			tomarDano(1);
			break;
		case "limiteDown":
			Destroy (this.gameObject);
			scriptMeteor.contadorMeteoro++;
			break;
		case "limiteTop":
			Destroy (this.gameObject);
			break;
		case "limiteRight":
			Destroy (this.gameObject);
			break;
		case "limiteLeft":
			Destroy (this.gameObject);
			break;
		case "especial":
			tomarDano (200);
```

```
                        break;
                }
        }
        public void tomarDano(int dano){
                HP -= dano;
                if(HP <= 0)
                {
                        stantiedNewMeteors ();
                        explodir ();
                        scriptMeteor.contadorMeteoro++;
                        if (this.name == "meteoro3(Clone)"){
                                dropaItem ();
                        }
                }
        }
        public void explodir(){
                Instantiate (prefabexplosao, transform.position, transform.rotation);
                Destroy (this.gameObject);
        }
        public void dropaItem(){
                int i = Random.Range (0, 4);
                Instantiate (dropItens[i], transform.position, transform.rotation);
        }
        public void stantiedNewMeteors(){
                for (int i = 0; i < 3; i++) {
                        GameObject tempDrop = Instantiate (miniMeteors, transform.position,
transform.rotation) as GameObject;
                        transform.position = new Vector3 (transform.position.x + 0.1f,
transform.position.y, transform.position.z);
                }
        }
}
```

moveOffset.cs - Esse é o controle de movimentação do *background*.

```
public class moveOffset : MonoBehaviour {

        private  Material       currentMaterial;
        private  float          offSet;
```

```
	public float		speed;

	void Start () {
		currentMaterial = GetComponent<Renderer> ().material;
	}
	void FixedUpdate () {
		offSet += speed * 0.001f;
		currentMaterial.SetTextureOffset ("_MainTex", new Vector2 (0, offSet));
	}
}
```

OnTrigger.cs - *Script* que destrói os elementos que saem da tela para não sobrecarregar a cena.

```
public class OnTrigger : MonoBehaviour {

	void OnTriggerEnter2D(Collider2D col){
		if (!col.isTrigger)
		{
			Destroy (this.gameObject);
		}
		switch (col.gameObject.tag) {
		case "limiteDown":
			Destroy (this.gameObject);
			break;
		case "limiteTop":
			Destroy (this.gameObject);
			break;
		case "limiteLeft":
			Destroy (this.gameObject);
			break;
		case "limiteRight":
			Destroy (this.gameObject);
			break;
		case "especial":
			Destroy (this.gameObject);
			break;
		}
	}
```

```
}
```

SpawnInimigo.cs - Controla o tempo de *spawn* dos inimigos.

```
public class SpawnInimigo : MonoBehaviour {

	public GameObject[]	onlyEnemy;
	public GameObject[]	muchEnemy;
	public float		tempoSpawn, tempoSpawn2, posXFixed;
	public Transform	limiteEsquerda, limiteDireita;
	public int		i;
	public GameObject	tempPrefab;
	private float		minX, maxX, posX;
	private float		tempTime, tempTime2;

	void Start () {
		minX = limiteEsquerda.position.x;
		maxX = limiteDireita.position.x;
	}
	void Update () {
		tempTime += Time.deltaTime;
		tempTime2 += Time.deltaTime;
		if (tempTime >= tempoSpawn)
		{
			tempTime = 0;
			i = Random.Range (0, 2);
			spawnOnlyEnemeys();
		}
		if (tempTime2 >= tempoSpawn2)
		{
			tempTime2 = 0;
			i = Random.Range (0, 2);
			spwanMuchEnemeys ();
		}
	}
	void spawnOnlyEnemeys(){
		GameObject tempPrefab = Instantiate (onlyEnemy[i]) as GameObject;
		float posX = Random.Range (minX, maxX);
```

```
                tempPrefab.transform.position = new Vector3 (posX, transform.position.y,
transform.position.z);
        }
        void spwanMuchEnemeys(){
                for (int a = 0; a < 8; a++) {
                        tempPrefab = Instantiate (muchEnemy [i]) as GameObject;
                        tempPrefab.transform.position = new Vector3 (posXFixed,
transform.position.y, transform.position.z);
                        posXFixed += 0.5f;
                }
                posXFixed = -1;
        }
        IEnumerator inimigo5(){
                yield return new WaitForSeconds(5);
                tempPrefab = Instantiate (muchEnemy[i]) as GameObject;
                tempPrefab.transform.position = new Vector3 (posX, transform.position.y,
transform.position.z);
        }
}
```

controllCredits.cs - Controla a animação dos créditos.

```
public class controllCredits : MonoBehaviour {

        public GameObject     gameObject;
        private bool          pause=true;

        void Update () {
                if (!pause)
                        Time.timeScale = 0;
                else
                        Time.timeScale = 1;
        }
        public void pauseAnimation(){
                pause = !pause;
        }
}
```

controllLight.cs - Controla as luzes da fase 4.

```
public class controllLight : MonoBehaviour {

	public GameObject[] ligths;
	public GameObject ligthBoss;
	public float timeLights, timeBoss;
	private bool     bossActive=false;

	void Start () {
		disableLigths ();
	}

	void Update () {
		timeLights += Time.deltaTime;
		timeBoss += Time.deltaTime;;
		if(!bossActive){
			ligths [0].SetActive (true);
			ligths [3].SetActive (true);
			ligths [4].SetActive (true);
		if (timeLights > 20)
			ligths [2].SetActive (true);
		if (timeLights > 10) {
			ligths [1].SetActive (true);
		}
		if (timeBoss > 180)
			ligthBoss.SetActive (true);
		}
	}
	public void disableLigths(){
		foreach (var item in ligths) {
			item.SetActive (false);
		}
	}
	public void enableLights(){
		foreach (var item in ligths) {
			item.SetActive (true);
		}
	}
	public void disableBossLigth(){
```

```
            ligthBoss.SetActive (false);
            bossActive = true;
        }
}
```

lights.cs - Complementação para funcionamento das luzes da fase 4.

```
public class lights : MonoBehaviour {

        private   controllLight                 controllLight;
        public    GameObject[]                  spawnEnemys;
        private   GameObject            spawn;
        public   GameObject             boss;

        void Start () {
            controllLight = FindObjectOfType (typeof(controllLight)) as controllLight;
        }
        public void luzAmarela(){
            spawn = spawnEnemys [0];
            StartCoroutine ("controllSpawn");
        }
        public void luzVermelha(){
            spawn = spawnEnemys [1];
            StartCoroutine ("controllSpawn");
        }
        public void luzVerde(){
            spawn = spawnEnemys [2];
            StartCoroutine ("controllSpawn");
        }
        public void luzAzul(){
            boss.SetActive (true);
            controllLight.disableLigths ();
            controllLight.disableBossLigth ();
        }
        IEnumerator controllSpawn(){
            controllLight.disableLigths ();
            spawn.SetActive (true);
            yield return new WaitForSeconds (3f);
            controllLight.enableLights ();
```

```
                spawn.SetActive (false);
        }
}
```

MenuHistory.cs - *Script* de gerenciamento de liberação dos trechos da história.

```
public class MenuHistory : MonoBehaviour {

        public GameObject[]   buttons;
        public GameObject[]   panel;
        public int            fase;

        void Start () {
                fase = PlayerPrefs.GetInt ("fases");
                for (int i = 0; i <= fase; i++) {
                        buttons [i].SetActive (true);
                }
        }
        public void enabledPanel(int x){
                panel [x].SetActive (true);
        }
        public void disablePanel(int x){
                panel [x].SetActive (false);
        }
}
```

VirtualJoystick.cs - *Script* do analógico do controle da nave.

```
public class VirtualJoystick : MonoBehaviour, IDragHandler, IPointerUpHandler,
IPointerDownHandler {

        private         Image           bgImg, joystickImg;
        private         Vector3         inputVector;

        void Start () {
                bgImg = GetComponent<Image> ();
                joystickImg = transform.GetChild (0).GetComponent<Image> ();
```

```
        }
        public void OnDrag(PointerEventData ped)
        {
                Vector2 pos;

        if(RectTransformUtility.ScreenPointToLocalPointInRectangle(bgImg.rectTransform,
ped.position, ped.pressEventCamera, out pos))
                {
                        pos.x = pos.x / bgImg.rectTransform.sizeDelta.x;
                        pos.y = pos.y / bgImg.rectTransform.sizeDelta.y;
                        inputVector = new Vector3(pos.x * 2 - 1, 0, pos.y * 2 - 1);
                        if (inputVector.magnitude > 1)
                {
                        inputVector = inputVector.normalized;
                        }
                        joystickImg.rectTransform.anchoredPosition = new Vector3
(inputVector.x * (bgImg.rectTransform.sizeDelta.x / 3), inputVector.z *
(bgImg.rectTransform.sizeDelta.y / 3));
                }
        }
        public void OnPointerUp(PointerEventData ped)
        {
                inputVector = Vector3.zero;
                joystickImg.rectTransform.anchoredPosition = Vector3.zero;
        }
        public void OnPointerDown(PointerEventData ped)
        {
                OnDrag (ped);
        }
        public float Horizontal()
        {
                return inputVector.x;
        }
        public float Vertical()
        {
                return inputVector.z;
        }
}
```