Politécnico
Castelo Branco
Escola Superior
de Tecnologia

Framework para gamificação de atividades educativas - jogos - “Supernova QuEST”.

Projeto I

Luís Miguel da Encarnação Salvado, 62006157

Orientador

Fernando Sérgio Rodrigues de Brito da Mota Barbosa

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciado em Informática e Multimédia, realizada sob a orientação científica do Doutor Fernando Sérgio Rodrigues de Brito da Mota Barbosa, do Instituto Politécnico de Castelo Branco.

Janeiro de 2025

Composição do júri

Presidente do júri

Doutor, Carlos Manuel de Oliveira Alves

Professor Adjunto, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco.

Vogais

Doutor, Fernando Sérgio Rodrigues de Brito da Mota Barbosa

Professor Adjunto, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco.

Mestre, Paulo Alexandre Correia da Silva Neves

Professor Adjunto, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco.

Agradecimentos

Aproveito este espaço para agradecer ao orientador deste projeto, o professor Fernando Sérgio Rodrigues de Brito da Mota Barbosa, que sempre mostrou disponibilidade, para marcações de sessões de esclarecimento online, rapidez de resposta nos e-mails e grande profissionalismo ao apoiar-me na construção deste relatório. Sem ele, com certeza que teria sido muito mais difícil.

Outro agradecimento que gostaria de fazer é dirigido à minha companheira Maria João Cerieiro Mendes, que, ao longo dos últimos meses, tem sido a primeira pessoa a fazer a revisão deste documento. Agradeço assim a sua disponibilidade e paciência, tem sido um apoio essencial para que este projeto siga o rumo desejado.

Por fim, gostaria de agradecer a minha família e a todos os colegas de trabalho, que sempre me incentivaram nesta minha aventura de conclusão do grau de licenciado.

Resumo

Cada vez é mais é comum por parte dos docentes a tentativa de gamificação dos seus conteúdos educacionais. No entanto, algumas das estratégias aplicadas para fazer essa metodologia resultar, podem não ter os resultados mais desejados. A pensar nisso, foi criada uma *framework* educacional chamada “GamEducation”, que promete ajudar nesta vertente.

Este projeto, intitulado “Supernova QuEST”, tem como propósito a criação de um videojogo para dispositivos móveis. A tentativa de implementar a gamificação visa, principalmente, captar o interesse e a atenção dos estudantes enquanto estes desfrutam do jogo. Assim, espera-se que o entusiasmo pelo jogo os motive não só a aprender os conteúdos educacionais, mas também a manter a motivação e a desenvolver o gosto pelo estudo. Para garantir a continuidade da motivação nos estudos, os conhecimentos dos jogadores serão postos à prova em momentos específicos durante o jogo, ajudando-os a evoluir ao longo da experiência.

Foram realizados vários estudos neste projeto, um deles foi a análise do estado da arte de videojogos do mesmo gênero, onde foi possível extrair o melhor de cada um deles, para que fosse aplicado na construção do “Supernova QuEST”.

O relatório inclui um documento *GDD*, metodologia muito adotada no que diz respeito ao desenvolvimento de videojogos. Todas as informações necessárias para o desenvolvimento do jogo, estão descritas detalhadamente nesse documento, facilitando o trabalho ao programador.

Por último é apresentado o desenvolvimento de um protótipo, que exigiu várias pesquisas e ajustes para que o resultado obtido estivesse de acordo com o objetivo desejado pelo autor.

Palavras-chave

Framework, GamEducation, Gamificação, Videojogo, Unity

Abstract

Nowadays, it is increasingly common for teachers to attempt to gamify their educational content. However, some of the strategies applied to make this methodology successful may not always yield the most desirable results. With this in mind, an educational framework called "GamEducation" has been created, promising to provide support in this area.

This project, titled "Supernova QuEST", aims to develop a mobile video game. The attempt to implement gamification primarily seeks to capture students' interest and attention while they engage with the game. Thus, it is expected that their enthusiasm for the game will not only motivate them to learn educational content but also help maintain their motivation and foster a love for studying. To ensure continued motivation in their studies, players' knowledge will be tested at specific moments throughout the game, assisting them in progressing during the experience.

Several studies have been conducted as part of this project, one of which involved analysing the state of the art of video games within the same genre. This research allowed the extraction of the best elements from each game, which were then applied to the development of "Supernova QuEST".

The project report includes a GDD (Game Design Document), a widely adopted methodology in video game development. All the necessary information for the game's development is detailed in this document, facilitating the work of the programmers.

Finally, the development of a prototype is presented, which required extensive research and adjustments to ensure that the final result aligned with the objectives set by the author.

Keywords

Framework, GamEducation, Gamification, Video game, Unity

Índice geral

Índice de figuras

Lista de tabelas

Lista de abreviaturas, siglas e acrónimos

GDD - Game Design Document

HUD - Heads-up display

API - Application Programming Interface

1. Introdução

Tendo em conta os tempos em que vivemos, a educação cada vez mais se torna um desafio, tanto para os pais como para os educadores.

Diversos estudos apontam, que o foco e a capacidade de concentração entre os jovens têm vindo a diminuir ao longo dos tempos [1]. Vários são os fatores que são associados a este acontecimento, e a tecnologia é apontada como uma das principais causas. Assim, este projeto visa unir a educação com a tecnologia, mais especificamente os videojogos, muito apreciados pelos mais jovens, que serão o público-alvo deste projeto.

Para este propósito, os ex-alunos João Tiago Alves Dias e Luís Miguel Farinha Mateus, sob a orientação do professor Fernando Sérgio Rodrigues de Brito da Mota Barbosa, desenvolveram uma *framework* "GamEducation" [2], que permite a inclusão de elementos didáticos nos videojogos, de forma a motivar os jovens a estudar.

O desafio deste projeto consiste em desenvolver um videojogo para plataformas móveis, que seja cativante e que integre de forma natural o uso da *framework* previamente construída. Com este propósito, escolheu-se a recriação de um videojogo que se espera servir um público mais homogéneo e que vá de acordo com o "*target"* de idades pretendido, alunos entre os 12 e os 18 anos, que é o "Space Invaders".

O jogo terá o nome de "Supernova QuEST", este nome combina o tema do jogo que será uma aventura espacial ao motivo educacional que o jogo pretende transmitir. O jogador terá de enfrentar vários desafios, como asteroides e naves inimigas, para tal terá à sua disposição, vários tipos de naves e armas. O jogo tem o intuito de ser bastante personalizável, tendo em conta o uso de uma moeda de troca. Esta moeda poderá ser ganha, usando a framework educacional, derrotando inimigos, entre outras formas.

Este projeto, será realizado num dos mais reconhecidos motores de desenvolvimento de videojogos que é o *Unity*. A escolha deste software, deve-se ao conhecimento prévio adquirido pelo autor, em unidades curriculares da Licenciatura em Informática e Multimédia.

1.1 Objetivos

É fundamental definir objetivos nesta fase introdutória do projeto. Assim, os objetivos deste projeto consistem na realização de uma análise prévia de jogos do mesmo gênero, que se encontram atualmente no mercado, seguindo uma estrutura de avaliação personalizada, designada por estado da arte.

Feita esta análise e realizada a escolha dos aspetos positivos a integrar no jogo, o objetivo seguinte é começar a construir os pilares do que será o jogo "Supernova QuEST", a sua história, os seus objetivos, os níveis, a jogabilidade, a estrutura de dificuldade, todos os elementos que farão parte do jogo, as interfaces os sons, entre outros, por forma a tomar o jogo bastante cativante. Tudo isto, será integrado no *GDD (Game Design Document),* que estruturará de forma clara e precisa todos os elementos mencionados anteriormente, e que será um documento fundamental de apoio ao desenvolvimento do jogo.

O último objetivo, será o início da concretização do que foi planeado anteriormente no *GDD*. Para tal, será utilizado o motor de construção de jogos *Unity*. Nesta primeira fase, será realizada uma pequena demonstração do que o jogo poderá vir a ser, onde serão aplicados conhecimentos adquiridos em disciplinas de programação, computação gráfica e aplicações multimédia. Este servirá de base de estudo, onde podem ser aplicados novos conceitos adquiridos pelo autor, será quase como uma sala experimental, onde se pretende obter os melhores resultados para que futuramente o jogo seja concluído com sucesso.

É importante destacar, que um dos objetivos principais, será a inclusão da framework educacional. Para maximizar o seu uso, serão estudadas muitas das técnicas usuais de monetização nos jogos, para que o aluno aumente ou teste os seus conhecimentos em vez de gastar o seu dinheiro. Nesta primeira fase do projeto, serão pensados quais os momentos em que esta fará parte do jogo. Isto envolve que seja feito algum estudo acerca da framework, para tal será feita uma consulta integral aos documentos de relatório desenvolvidos no projeto anterior.

1.2 Estrutura do relatório

O presente documento está dividido em seis capítulos: este primeiro capítulo introdutório, onde é feito um enquadramento geral sobre o projeto, quais são os seus objetivos e a calendarização proposta para a sua conclusão; o segundo capítulo destina-se a uma explicação do que é a framework educacional; o terceiro capítulo, é onde será feita a análise do estado da arte a quatro jogos, que se assemelham ao que será desenvolvido neste projeto, para esse propósito será usada uma estrutura própria de análise; o quarto capítulo inclui o *GDD*, no qual é feita a total pormenorização do que o jogo será e irá conter; o quinto capítulo conta como uma prova de conceito, onde serão descritos os processos mais importantes na criação de um protótipo do jogo "Supernova QuEST"; o sexto capítulo será destinado as conclusões que podem ser retiradas da realização desta fase do projeto e serão elaborados planos de trabalho futuro a ser realizado.

1.3 Calendarização

Neste subcapítulo, será apresentado o diagrama de "*Gantt*". Foi usado este tipo de diagrama por ser o mais indicado para o tipo de projeto que será desenvolvido. Este diagrama permite dividir as várias tarefas do projeto em intervalos temporais que devem ser cumpridos para atingir o objetivo maior, que é a conclusão bem-sucedida desta fase do projeto.

A **Tabela 1**, exemplifica o diagrama usado para o desenvolvimento desta fase primária do projeto.

O primeiro passo, foi a escolha do tipo de jogo que será desenvolvido, discutindo ideias com o orientador Fernando Sérgio Rodrigues de Brito da Mota Barbosa, utilizando técnicas de "brainstorming", para reunir o máximo de ideias possíveis e soluções para possíveis problemas.

Numa segunda fase, foi feita a análise de jogos similares ao que se quer desenvolver. Simultaneamente, iniciou-se o estudo do motor de jogo *Unit*y e da linguagem de programação usada por este motor, que é o "C#".

A tarefa seguinte, seria aplicar esses conhecimentos e começar a construir o protótipo do jogo, intercalando com o início da redação deste relatório.

Tabela 1 - Diagrama de Gantt do Projeto I

Tarefas / Mês	Outubro	Novembro	Dezembro	Janeiro
Escolha do tipo de jogo				
Estado da arte				
Estudo em Unity e C#				
Protótipo				
Relatório / GDD				

2. Framework

O desenvolvimento do videojogo “Supernova QuEST” tem como propósito a utilização de uma *framework* chamada “GamEducation” [2].

Esta *framework* foi criada pelos ex-alunos da Escola Superior de Tecnologia, João Tiago Alves Dias e Luís Miguel Farinha Mateus, sob a orientação do professor doutor Fernando Sérgio Rodrigues de Brito da Mota Barbosa, no âmbito das cadeiras de Projeto I e Projeto II.

A “GamEducation” é uma plataforma de apoio educacional, cujo principal objetivo é a gamificação de atividades educativas, tornando o ensino mais cativante e produtivo para todos.

É dividida em duas áreas de ação: a plataforma web, que serve docentes e programadores, e a plataforma móvel Android, destinada ao uso dos alunos.

No âmbito deste projeto, o foco recai na plataforma web, onde, como programador, é possível criar e gerir jogos que servem tanto docentes como alunos. De acordo com o que é descrito no relatório de Projeto I da *framework* “GamEducation”, uma vez criado o jogo e implementada a biblioteca correspondente, será apenas necessário colocar o jogo na plataforma e preencher um formulário para descrever o mesmo aos docentes que pretendam utilizá-lo.

Os alunos, por sua vez, podem aceder à plataforma móvel através de um código de acesso fornecido pelo docente. A partir desse momento, basta escolher um jogo do seu agrado e começar assim o seu progresso educacional.

O objetivo principal deste projeto, é criar um videojogo completo para a plataforma móvel Android, integrando a *framework* “GamEducation” por meio da implementação de uma biblioteca própria exemplificada na documentação da mesma.

Este projeto apresenta-se como uma oportunidade de aliar conhecimentos técnicos de desenvolvimento de videojogos ao objetivo educacional de promover a gamificação no ensino.

Em suma, com a integração desta *framework* “GamEducation”, será possível criar uma experiência motivadora para alunos e docentes, contribuindo para um melhor ambiente de aprendizagem.

3. Estado da arte

Neste capítulo, são apresentadas a pesquisa e a análise de jogos para dispositivos móveis dentro da temática do jogo a desenvolver neste projeto. Procurou-se recolher o máximo de informações sobre quatro jogos que poderiam servir de inspiração na criação do “Supernova QuEST”.

Os jogos em que foi feito o estado da arte são: “Galaxiga – Classic 80s Arcade”, “Galaxy Attack: Space Shooter”, “Retro: Space” e por fim, “Retro Space 3D”.

A informação do estado da arte para os jogos mencionados anteriormente será a seguinte: introdução que inclui uma designação, a idade recomendada e o desenvolvedor; elementos do jogo; interface do jogador; aspetos positivos e por fim os aspetos negativos.

Por último, será feita uma conclusão quanto a ideias finais que se podem retirar desta análise.

3.1 Galaxiga – Classic 80s Arcade

Este jogo foi criado para homenagear os jogos clássicos dos anos 80, como o “Galaxian” de 1979 e o “Galaga” de 1981[3]. É um jogo de tiro espacial, no qual se tem uma experiência de jogo nostálgica, porém com gráficos modernos e atraentes. Oferece a simplicidade e o caráter viciante da jogabilidade do passado, onde o jogador assume o papel de defensor da galáxia em batalhas espaciais épicas. Muito ao estilo destes jogos mais retro, os inimigos aparecem agrupados em ondas de ataque, pelas quais o jogador deve estar pronto para ripostar e eliminar essas ondas de inimigos e invasores, a fim de salvaguardar a sua galáxia.

A idade recomendada para este jogo é a partir dos doze anos e a empresa que o desenvolveu é a ONESOFT GLOBAL PTE. LTD.

3.1.1 Elementos do jogo

O elemento principal do jogo é a nave pilotada pelo jogador.

Podem-se identificar trinta e oito naves distintas, cada uma com uma representação gráfica e um nome únicos. Essas naves possuem níveis de evolução e permitem a combinação de diversas armas e *power-ups*, trazendo ao jogador o lado estratégico do jogo.

Um aspeto adicional relevante é a nave auxiliar, designada como drone, que acompanha a nave principal e ajuda na eliminação dos inimigos. Existem sessenta e quatro opções disponíveis para esta nave, com representação gráfica e nome identificativo únicos. Podem ser usados até três destes drones simultaneamente, combinando armas e ataques.

Os projéteis disparados variam de acordo com a configuração da nave selecionada, apresentando uma ampla diversidade gráfica, comportamental e de níveis de dano causados, incentivando o jogador a jogar para evoluir a nave e otimizar essas características e estratégias.

Adicionalmente, é possível identificar vários tipos de inimigos, que se distinguem pelo seu aspeto visual, comportamental e resistência ao dano. Nos primeiros níveis, não é fácil fazer esta distinção, entre inimigos denominados como “normais”, dos inimigos finais. Contudo, consegue-se identificar que o comportamento passa a ser diferente, pois não surgem em bando de inimigos, mas sim separadamente. Todavia, nos níveis mais avançados, os inimigos finais passam a ser bem identificados, com uma sequência gráfica que indica que ele irá chegar e começar a combater.

Por último, os *power-ups* que podem incrementar a frequência de disparos da nave ou até mesmo mudar o tipo de disparo, em que estes são representados graficamente por retângulos de cor verde néon, para se destacarem do resto dos elementos presentes no ecrã de jogo.

3.1.2 Interface do Jogador

Figura 1 - Captura de ecrã de jogo

Na **Figura 1**, está representada uma captura do ecrã com o jogo em ação, na qual é possível identificar o *HUD, (Heads-up display)* na parte superior do ecrã, onde se pode ver as vidas restantes do jogador e a patente do mesmo no lado esquerdo. Já no lado direito, estão presentes os pontos acumulados até ao momento e um botão de pausa. Na parte central do ecrã, pode-se identificar os elementos principais do jogo, a nave e o seu ajudante, que disparam na direção dos inimigos. Também se pode identificar um projétil vindo dos inimigos, e um *power-up* que surgiu proveniente da eliminação de um dos invasores. No fundo do ecrã, à esquerda, existe um botão que quando pressionado é aplicado um ataque especial, que vai sendo renovado conforme as eliminações feitas durante o progresso do nível.

Figura 2 - Ecrã principal do jogo

Ilustrado na **Figura 2**, está o ecrã principal do jogo.

Na parte superior deste, encontra-se uma barra com todas as recompensas ganhas. Por baixo dessa barra, existe uma segunda, onde é feita a identificação do jogador e o seu nível de experiência do jogo.

Na parte central, existe a possibilidade de selecionar a nave que se deseja usar, assim como o drone, que se quer que acompanhe a nave. É também visível a secção em que se pode combinar várias "*squads*", que no momento da captura do ecrã, estavam bloqueadas. Isso, é verificável através da representação gráfica do cadeado. A "*squad*", permite ao jogador ter para além da nave principal, naves adicionais para conseguir ultrapassar os obstáculos do jogo. Significa que quando a nave principal é eliminada, surge no ecrã a nave da "*squad*" seguinte, e assim por diante. De seguida, existem os botões de seleção de jogo disponíveis, o "*multiplayer*" e o "*single play*". Por último, existe um menu de ícones, onde se pode navegar para outros ecrãs do jogo, como por exemplo, a loja.

Figura 3 - Ecrã de escolha das naves

A **Figura 3**, demostra a escolha das naves, é aqui que se pode fazer a seleção da nave para o jogador e o seu drone. Dispõe da representação gráfica da escolha das mesmas na parte superior do ecrã. No meio do ecrã, é apresentado o nome e o *power* associado à nave selecionada. Aqui, também verificamos que existem dois botões: o de "*equip*" e "*upgrade*" da nave, que promovem a personalização da nave escolhida. Em último, é apresentado o menu de naves, não estão todas disponíveis no início, é com o decorrer do jogo que vão sendo desbloqueadas.

3.1.3 Aspetos positivos

O jogo apresenta gráficos de excelente qualidade, ajustados para qualquer tipo de dispositivos móvel.

Os controlos são intuitivos o que tornam a jogabilidade natural, pois o jogador apenas precisa mover o dedo pelo ecrã, para direcionar a nave, oferecendo uma experiência acessível e divertida. Um pormenor interessante é que não é necessário ter o dedo sobre a nave para a mover, isto é, a nave acompanha o movimento do dedo, mesmo ele estando posicionado em qualquer outra parte do ecrã de jogo.

Com mais de quinhentos níveis disponíveis, nos quais são apresentados inúmeros tipos de inimigos e chefes finais, sobressai-se não apenas como um jogo de tiro espacial, mas também como um título de estratégia, permitindo ao jogador ajustar a sua nave de acordo com suas preferências de jogo.

Existem vários *power-ups* e opções de melhoria de armas, proporcionando uma evolução dinâmica ao longo do jogo.

O jogador, tem a oportunidade de fazer parte de um grupo de jogadores, de forma a competir nas tabelas de classificação e ganhar recompensas.

O jogo, também inclui modos de *multiplayer* que possibilitam empolgantes confrontos competitivos no estilo de um contra um.

3.1.4 Aspetos negativos

Os menus do jogo são muito confusos, contêm muita informação em um só ecrã.

Outro problema, é a quantidade de pop-ups que surgem ao abrir o jogo, com bastante informação sobre a venda de itens do jogo, geralmente mais de dois pop-ups. Este parece ser um número que pode causar irritabilidade a qualquer jogador que pretende apenas chegar ao ecrã onde o jogo efetivamente começa.

O som do jogo é repetitivo entre níveis, o que pode levar o jogador a desativar o som por completo.

3.2 Galaxy Attack: Space Shooter

É um jogo voltado para os fãs de jogos de tiro no espaço, que conta a história de uma galáxia que está sob o ataque de invasores. Galáxia essa composta pelos planetas Terra, Marte, Mercúrio e Júpiter. O objetivo, é eliminar todos os inimigos e com isso, salvar a galáxia da destruição.

A idade recomendada para este jogo é de quatro anos ou mais, e a empresa que desenvolveu este jogo é a GAU TECHNOLOGY LTD [4].

3.2.1 Elementos do jogo

O elemento principal do jogo é a nave pilotada pelo jogador. No total, são sessenta e nove tipos de naves diferentes, com nomes, aparências visuais e tipos de tiros distintos.

Pode-se aumentar o nível de "*skill"* da nave, esse nível de "*skill"* aumenta o dano causado às naves inimigas. Outro aspeto que pode ser incrementado é a "*evolution*" da nave, que apenas adiciona uma estrela a uma barra de estrelas, cujo limite é de seis. Por fim, é dada a opção de "*merge*" de naves, que possibilita fundir duas naves em uma só. Contudo, só é possível em um nível avançado de jogo, o qual não foi alcançado durante os testes, pelo que infelizmente este comportamento não foi possível de observar.

É possível, ter duas "*minishipss*" que seguem o jogador, uma posicionada no lado direito e outra no lado esquerdo da nave principal. Estas disparam na direção do inimigo, ajudando o jogador a ter mais tiros extras na missão de eliminar os inimigos. São setenta as opções disponíveis, sendo que é dada uma opção de tiro especial para estas "*minishipss*", a opção de "*merge*" também está presente nas "*minishipss*", assim como está disponível naves principais. A nave principal tem "*skins*" opcionais, que para além do aspeto visual, aumentam o dano que a nave causa nos inimigos.

Existe a opção de ter uma personagem associada ao nosso jogador, são cinco tipos de personagens que vão sendo desbloqueadas ao longo do jogo, ou que se podem adquirir com a troca de moedas ganhas em jogo. A associação de uma personagem ao nosso jogador, a nível de jogo, apenas aumenta o dano que a nossa nave causa.

Existem três tipos de inimigos, que se dividem pelo nível de dano que conseguem suportar e pelo seu aspeto visual. Contam com os inimigos do tipo inseto, que são mais pequenos que os restantes, têm asas de inseto e não disparam qualquer tipo de projétil na direção do jogador. A nível de movimentação, surgem de forma coordenada no ecrã, seguindo um caminho padrão em circunferências, para no fim, estabilizarem-se no ecrã em formato de grelha, tal como acontecia no jogo "Space Invaders".

O segundo tipo de inimigo é um mais intermédio, que combina o aspeto de inseto com o de robô, estes são um pouco maiores que os anteriores e as suas aparições são esporádicas. Contudo, já têm a capacidade de disparar contra o jogador. O terceiro tipo são os chefes finais, que a nível de tamanho ocupam um terço do ecrã, são anunciados previamente e têm vários tipos de disparo na direção do jogador. Todos eles possuem uma barra de vida que é mostrada quando são atingidos.

Outro elemento que aparece esporadicamente em alguns níveis do jogo, foram os asteroides que se dirigem do canto superior esquerdo para o canto inferior direito, na diagonal, de forma a atingir a nave. São objetos destrutíveis caso sejam atingidos por um disparo do jogador e também podem retirar vidas ao jogador caso embatam no mesmo.

Por último, a nível de *power-ups*, foram identificados dois tipos diferentes, um que aumenta o nível de tiros disparados pela nave do jogador e o outro que troca a nave que o jogador está a utilizar no momento. Verificou-se, que esta troca é feita entre as naves que o jogador já tem desbloqueadas, porém em uma percentagem mais pequena é possível existir a troca por uma nave de um nível superior, o que pode tornar o jogo substancialmente mais fácil para o jogador.

3.2.2 Interface do Jogador

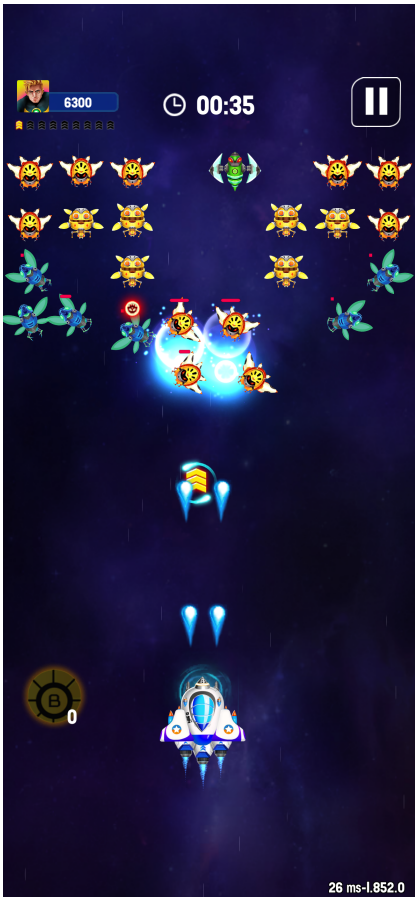


Figura 4 - Ecrã de jogo

A **Figura 4**, apresenta o ecrã de jogo, onde se pode identificar a nave do jogador com um nível de detalhe bastante interessante, os disparos do jogador que facilmente são associados à nave, pelas formas e a cor dos mesmos. Estes disparos são feitos de forma automática e contínua, sem controlo direto do jogador. Pode ser identificado, o elemento *power-up* de incremento de tiros da nave, representado por três bicos de seta amarelos, semelhante às patentes dos militares. Outros elementos identificáveis no ecrã, são os inimigos, de diferentes cores e aspetos, são visíveis três tipos diferentes de inimigo do tipo "inseto", e um do tipo de robô/inseto, de cor verde. É possível também identificar um disparo inimigo, representado pela sua forma circular, com um símbolo no seu interior, de cor laranja fluorescente. A nível de *HUD*, na parte superior do ecrã, está representado o tempo de jogo percorrido ao centro, à esquerda o nível de pontuação do jogador, e à direita a opção de pausar o jogo. À esquerda da nave, existe o botão de ataque especial, que só é desbloqueado quando algumas circunstâncias de jogo se concretizem.

Figura 5 - Ecrã de jogo, com nave especial

Na **Figura 5**, está ilustrada a troca de uma nave “normal” por uma nave “especial”. No momento da captura, a troca já foi realizada. Este aspeto foi documentado, porque ter esta capacidade num nível inicial do jogo parece ser um pouco *overpower*, dado que os disparos deste tipo de nave são direcionados automaticamente para os inimigos, até o nível de dano deste raio laser é muito superior aos dos disparos “normais”, pulverizando quase que instantaneamente o inimigo, podendo tornar o jogo um pouco mais aborrecido para o jogador.

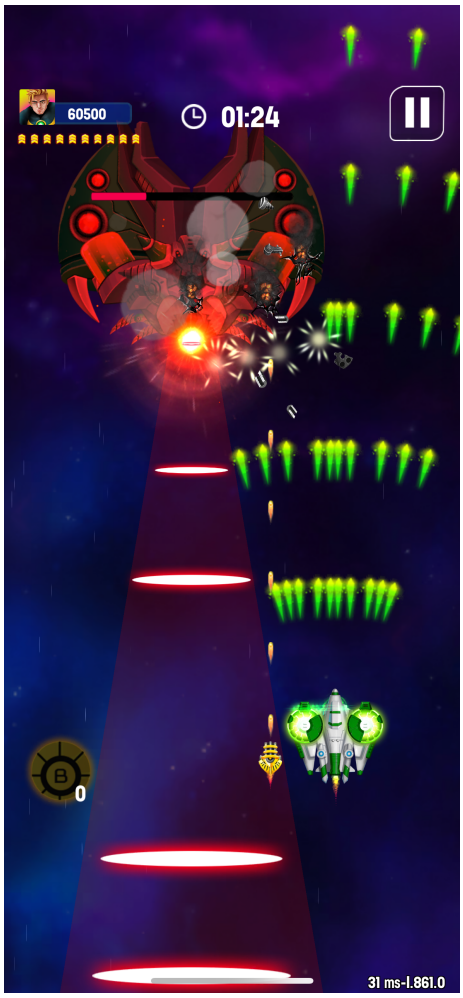


Figura 6 - Luta com inimigo final

Na **Figura 6**, pode-se identificar uma batalha final de nível de jogo, onde se enfrenta um inimigo especial, facilmente identificável pela sua dimensão e distinção na sua representação gráfica. Os projéteis deste tipo de inimigo são exclusivos, na ilustração identifica-se o disparo em cone, de projéteis circulares da cor vermelha. Também, é notável, que a barra de vida deste inimigo é muito superior à dos outros tipos de inimigos.

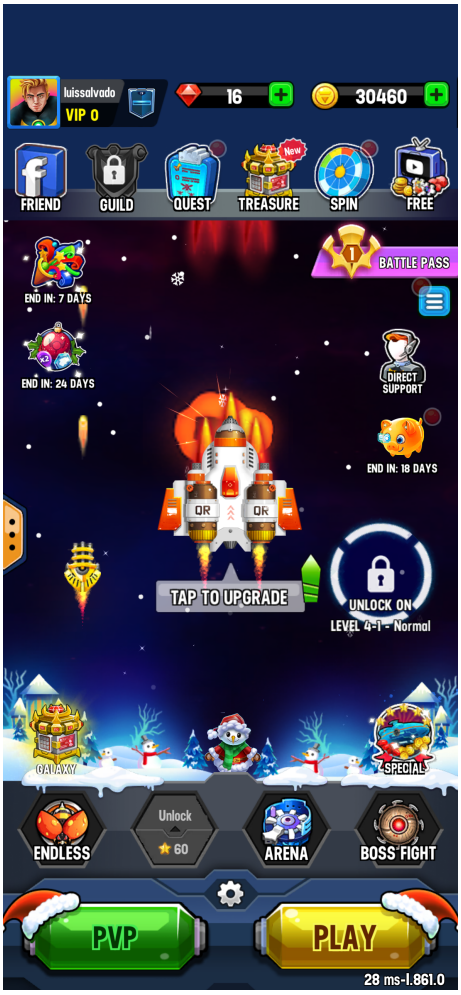


Figura 7 - Ecrã principal do jogo

A **Figura 7**, remete para o ecrã principal do jogo, onde pode-se verificar o excesso de informação anteriormente já mencionado. Desde botões de "*Quest*", "*Treasure*", "*Spin*", "*Direct Support*", "*Tap to Upgrade*", entre outros, o que causa muita poluição visual para o jogador.

É também observável, as quatro variantes de jogo que este oferece, três delas já desbloqueados: o "*Endless*", onde enfrentamos inimigos do tipo inseto de forma contínua e sem tempo limite; a "Arena", em que somos enquadrados num "*ranking*" mundial, no qual consoante os pontos obtidos em jogo, é-nos atribuída uma posição nesse mesmo "*ranking*", criando assim uma competição global no jogo; e por último o "*Boss Fight*", onde somos confrontados apenas com os inimigos especiais de fim de nível.

Na parte inferior do ecrã, pode-se identificar os dois tipos de jogo possíveis, o "*PvP*", ou "*Player versus Player*", e o "*Play*", normalmente denominado "*Single Player".*

Figura 8 - Ecrã de seleção de naves

A **Figura 8**, apresenta o ecrã de seleção de naves. Na parte superior, existem as abas de acesso aos ecrãs de personagens, naves e equipamento. No momento da captura, é a aba de naves que está selecionada. Dispõe da identificação do nome da nave que está selecionada, assim como o seu nível de evolução. Em seguida, verificamos a representação gráfica da mesma e caso seja selecionável as suas "*minishipss*". Na figura, apenas a do lado esquerdo está desbloqueada e selecionada. Na parte central, é identificável a barra com os três botões de "*Skill*", "*Evolution*" e "*Merge*". Na parte inferior, encontra-se as várias naves disponíveis que vão sendo desbloqueadas com ganhos no jogo, seja por troca de moedas ou por visualizações de vídeos publicitários. Já na última barra do ecrã, é mostrado qual o valor de evolução da nave em coins e/ou gemas coletáveis no jogo.

3.2.3 Aspetos positivos

Um dos aspetos que mais gostei no jogo foi o estilo de design utilizado, estilizado e com cores vibrantes.

A opção de troca de condução da nave também me pareceu interessante, o jogador pode optar por conduzir a nave de forma direta, pressionando diretamente em cima da nave e assim movendo-a pelo ecrã, ou pode-se optar pela condução que não necessita, necessariamente, que o dedo do jogador coincida com a nave.

A divisão dos níveis entre quatro planetas parece-me um bom plot para o jogo que se quer desenvolver.

Os combates finais são bastante desafiantes, o disparo de vários tipos de tiros por parte do inimigo final, e a sua movimentação que não se restringe exclusivamente à parte superior do ecrã, tornam este combate muito mais interessante.

A ideia de atribuição de patentes militares ao jogador, consoante a evolução no jogo, é interessante, contudo, pouco explorada nas poucas horas de jogo conseguidas.

Por último, o som e os efeitos sonoros usados no jogo são bastante aceitáveis, existe uma voz que anuncia o nível em que estamos, o que traz uma atenção especial ao jogo, assim como os efeitos sonoros, que são diversos conforme os embates que acontecem no jogo.

3.2.4 Aspetos negativos

Durante o decorrer do jogo, é oferecida a possibilidade de troca por outras naves que não a selecionada pelo jogador. Isso causou alguma confusão, por vezes, a troca é feita para uma nave inferior do que a que se tinha selecionado, o que não faz muito sentido, assim como também pode acontecer a troca para uma nave muito superior da que se tinha escolhido, tornando o jogo monótono.

Os inimigos presentes nos primeiros níveis são muito repetitivos e há pouca interação, apenas ficam parados e agrupados em grelha à espera de serem eliminados. Isto, poderia ser solucionado com a diminuição dos subníveis que cada nível contém, pois só no fim de cada nível é que existe um nível completo com luta contra o inimigo final desse nível.

Do que se pode apurar, o tiro realizado pelos inimigos do tipo robô/inseto é sempre o mesmo e ocorre muito raramente. Poderiam ser adicionados mais tipos de tiro, e que esses ocorressem de forma mais regular.

Muita informação nos ecrãs, deveria haver ecrãs secundários que contivessem estas opções.

3.3 RETRO: Space

Inspirado no jogo clássico "Space Impact", criado pela empresa Nokia e muito divulgado pelo saudoso dispositivo móvel "Nokia 3310", é um jogo de tiro espacial, com uma jogabilidade simples e envolvente. O objetivo é destruir todos os invasores e conquistar o maior número de pontos para alcançar a melhor posição possível na *leaderboard*.

É constituído por oito níveis, que apresentam vários tipos de inimigos e um fundo distinto, sendo que cada um destes, culmina com um inimigo final diferente.

O jogo funde perfeitamente elementos retrô com gráficos aprimorados, de cores vibrantes e modernas.

Um extra original que foi adicionado ao jogo, foi a introdução de um modo de jogo clássico, transformando o jogo colorido em um jogo monocromático, trazendo a nostalgia de jogar nos dispositivos da "Nokia".

A idade recomendada para este jogo é de quatro anos ou mais, e o desenvolvedor que desenvolveu este jogo é o Samuel Souza [5].

3.3.1 Elementos do jogo

O elemento principal do jogo é a nave controlada pelo jogador. Existem seis naves disponíveis, estas diferem em forma e cor. O tiro é disparado automaticamente pela nave, que apenas dispõe de um tipo de tiro.

Adicionalmente, a nave pode possuir três tipos de tiros especiais, que são recolhidos através de *power-ups*, que aparecem no ecrã quando são eliminados inimigos durante o jogo.

Outro elemento presente no jogo são os inimigos, que surgem em várias formas, cores e com níveis de dano diferentes. Estes, aparecem de forma padronizada ao longo do nível em que o jogador se encontra. Existem apenas dois tipos de inimigos, os "normais" e o inimigo final de cada nível, são oito no total, claramente inspirados no jogo desenvolvido pela "Nokia", que continha o mesmo número de níveis.

O elemento projétil presente no jogo tem a representação mais simples possível, que é apenas um retângulo. O que o difere na identificação de por quem foi disparado, é a sua cor e direção. Quando disparado pelo jogador, é azul e move-se do lado esquerdo para o lado direito do ecrã, quando disparado pelo inimigo, é vermelho e dirige-se na direção contrária ao disparo do jogador.

3.3.2 Interface do Jogador

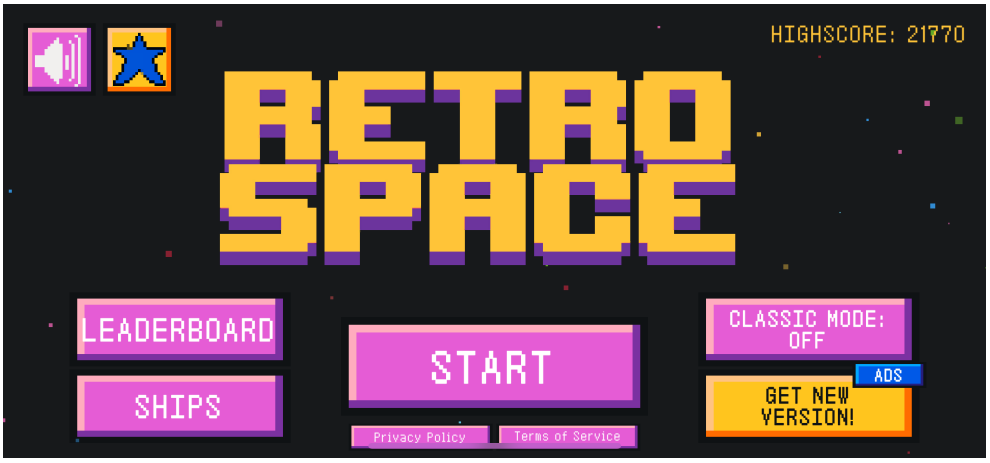


Figura 9 - Ecrã inicial

No ecrã inicial, como ilustrado na **Figura 9**, é mostrado um ecrã simples e de fácil compreensão. Estão presentes oito botões: um que liga ou desliga o som do jogo; um botão "*leaderboard*", para mostrar o ecrã da tabela classificativa mundial do jogo; o botão "*ships*", que exibe as naves que o jogador pode selecionar para jogar; no centro, o botão de "*start*", onde pode-se iniciar o jogo; em baixo desse botão, existem o botão de "*privacy policy*" e o "*terms of service*", que abrem as respetivas páginas no "browser" do dispositivo móvel; no lado direito, o botão "*Classic Mode*", que permite mudar o jogo do modo "normal" para o "clássico"; o botão "*Get New Version*!", abre a loja do dispositivo móvel para a página do segundo jogo desenvolvido pelo criador, este jogo irá ser abordado mais à frente neste documento; por último, no canto superior direito, está presente o registo da maior pontuação alcançada pelo jogador.

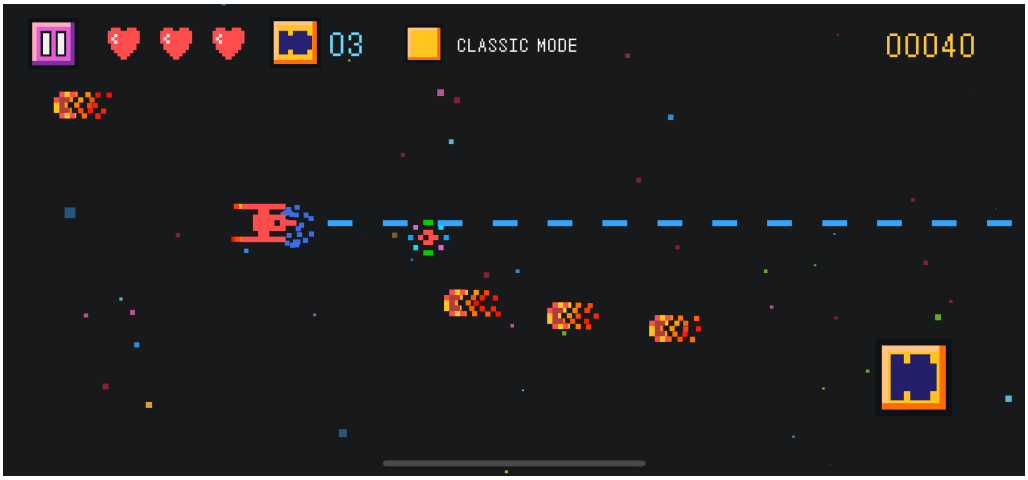


Figura 10 - Ecrã de jogo

A **Figura 10**, mostra o ecrã de ação do jogo, onde pode-se identificar a nave do jogador a vermelho, no momento a efetuar disparos com os projéteis de cor azul, este disparo apresenta uma animação de partículas quando é efetuado. É possível identificar quatro inimigos que se dirigem na direção da nave do jogador, eles podem-se movimentar em três direções: cima, baixo e frente. Na ilustração, existe a representação do *power-up*, graficamente representado por um círculo vermelho e rodeado de partículas de várias cores.

Para além dos vários elementos ativos, pode-se identificar o *HUD* do jogo, posicionado na parte superior do ecrã. Começando da direita para a esquerda, o botão de pausa do jogo, que abre o pop-up de pausa do jogo, onde estão presentes três botões: “*Resume*”, “*Restart*” e “*Menu*”. Em seguida, dispõe-se da representação gráfica, feita por corações, das vidas restantes do jogador. O próximo elemento do *HUD* é composto pelos *power-ups*, o ícone é alterável consoante o *power-up* que está ativo no momento, a representação numérica junto a esse ícone mostra o número de “power-ups” disponíveis para o jogador. Outro elemento interessante no *HUD* é a possibilidade de mudar dinamicamente do modo “normal” de jogo para o modo “clássico”, onde é feito o ajuste gráfico de todos os elementos do jogo. No canto superior direito, é ilustrada a contagem de pontos feita pelo jogador, que é incrementado a cada inimigo eliminado, conforme o valor que esse inimigo representa no jogo.

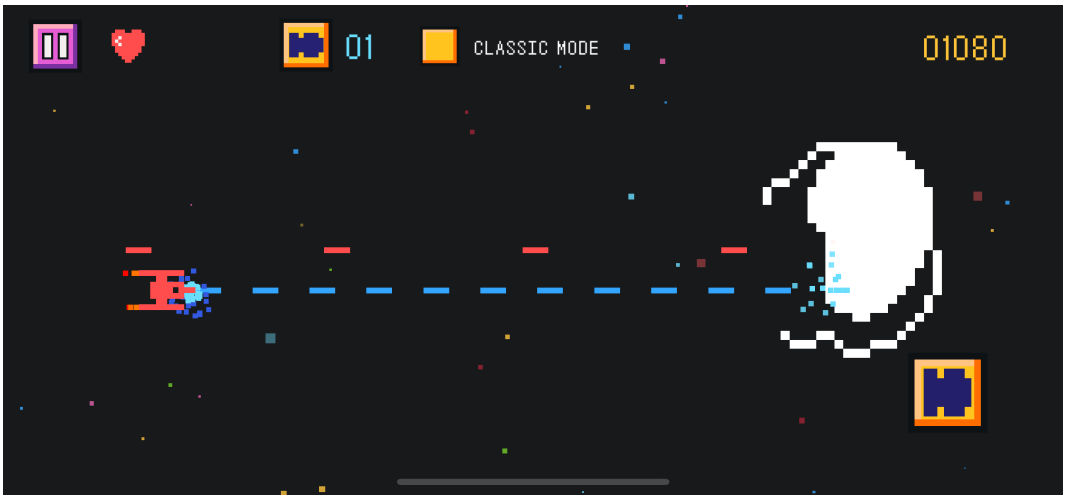


Figura 11 - Luta contra inimigo final de nível

Na **Figura 11**, está ilustrada uma batalha final de nível, onde aparece a nave do jogador e o inimigo final. Pode-se verificar, que este inimigo é muito maior que os inimigos normais e consegue disparar vários projéteis vermelhos na direção do jogador, a fim de eliminá-lo. A animação que foi dada ao inimigo, neste caso representada na ilustração, acontece quando é atingido, o inimigo apresenta a troca da cor original pela silhueta branca alternadamente. É uma técnica interessante, que poderá a ser adotada no “Supernova QuEST”.

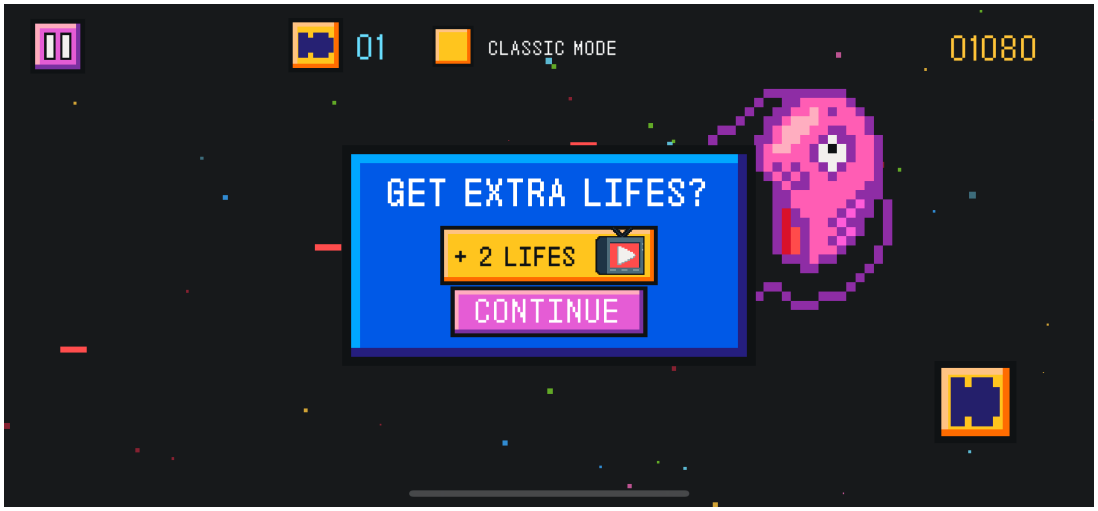


Figura 12 - Pop-up para obtenção de vidas extra

Quando o jogador perde as três vidas disponíveis durante o jogo, é mostrado ao mesmo um pop-up que possibilita obter duas vidas extras se o jogador visualizar um vídeo publicitário como podemos verificar na **Figura 12**. Neste jogo, esta é a única opção para obter vidas extra, pois não existe qualquer tipo de moeda no jogo. Caso selecionado o botão de "*Continue*", é mostrado ao jogador o ecrã de fim de jogo.

Figura 13 - Ecrã de seleção de naves

O ecrã de seleção de naves, demonstrado na **Figura 13**, mostra um pop-up simples, apenas com um título e a disposição das naves numa grelha. Para selecionar a nave que o jogador quer utilizar, basta carregar na nave e de seguida sair do pop-up, clicando na cruz presente no canto superior direito. Como é observável na ilustração, todas a naves contêm formas gráficas simples e não têm nenhum extra a nível de jogo.

Figura 14 - Ecrã de jogo do Space Impact

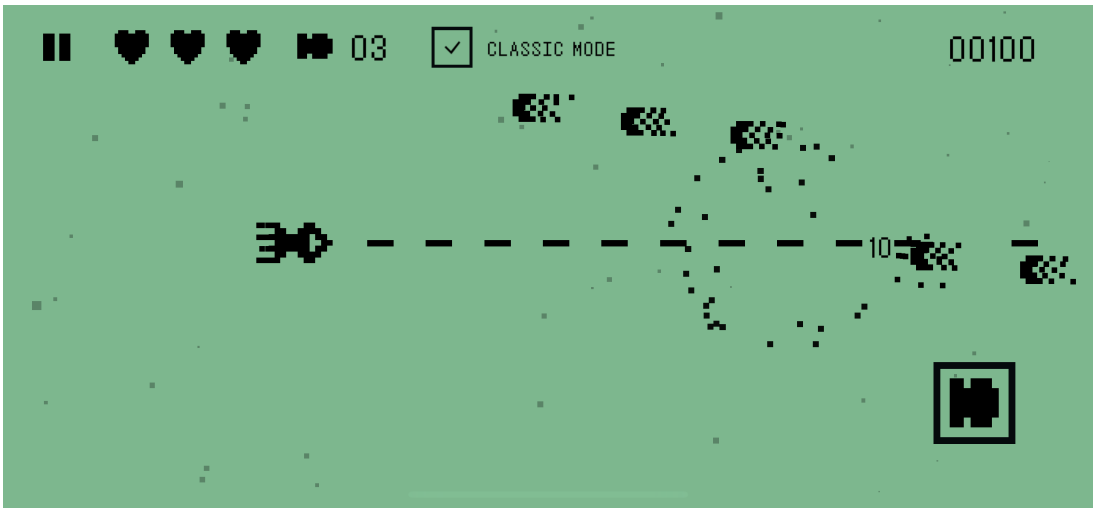


Figura 15 - Ecrã em modo clássico

É possível verificar as semelhanças entre o jogo “Space Impact”, ilustrado na **Figura 14,** e o modo clássico do jogo “RETRO Space”, ilustrado na **Figura 15**. O resultado produzido pelo desenvolvedor é uma bonita homenagem feita ao jogo original e remete o jogador aos anos 2000, quando o jogo saiu para as plataformas móveis da “Nokia”.

3.3.3 Aspetos positivos

Simplicidade nos menus, que são objetivos e concretos, apenas constituídos por botões essenciais ao jogo. Isso pode ser resultado da pouca utilização de publicidade no jogo, estas aparecem apenas quando o jogo é aberto e caso o jogador queira obter vidas extras.

As músicas usadas nos níveis do jogo, tornam o jogo empolgante e têm o pormenor de serem diferentes entre níveis. Os efeitos sonoros também são adequados ao tipo de jogo espacial.

3.3.4 Aspetos negativos

É um jogo que se termina em poucos minutos, é uma soma entre a escassez de níveis, apenas oito, e a capacidade de destruição dos inimigos, tanto os “normais” quanto os “especiais” de fim de nível, são muito previsíveis e têm pouca “inteligência”.

O modo clássico introduzido pelo desenvolvedor é interessante e divertido, contudo, é apenas usado em momentos esporádicos. Depois, o jogador quer retornar ao jogo colorido que é apresentado no modo normal.

3.4 Retro Space 3D

Este jogo é uma evolução do jogo anteriormente analisado o “Retro Space”, produzido pelo mesmo desenvolvedor. Inspirado em jogos clássicos, o objetivo é destruir todos os invasores e alcançar a melhor pontuação possível.

Oferece mais de trinta níveis diferentes, cada um com um inimigo especial de fim de nível distinto.

O design em pixel arte foi o utilizado para este jogo, que oferece gráficos com cores fortes e uma luminosidade bastante interessante e apelativa.

A idade recomendada para este jogo é de nove anos ou mais, e o desenvolvedor que desenvolveu este jogo é o Samuel Souza [6].

3.4.1 Elementos do jogo

Como nos outros jogos analisados, o elemento principal deste jogo é a nave comandada pelo jogador.

Existem dez naves com representações gráficas diferentes, e mais duas extras que são elementos “divertidos”, que podem ser opções para o jogador: uma águia, e a personagem do super-homem. As naves apenas diferem visualmente, não tendo qualquer vantagem em termos de jogabilidade.

A jogabilidade é simples, basta arrastar o dedo na direção que queremos que a nave se mova, e ela move-se, não é necessário que o dedo esteja coincidente com a nave, tal como nos jogos anteriormente analisados.

Os projéteis disparados pela nave, diferem conforme os *improvements*, que são coletados ao longo do jogo. Esses projéteis têm representações gráficas diferentes, dependendo da nave que está a ser utilizada pelo jogador.

Existem doze *improvements* diferentes, são eles: o "*Attack Speed*", que incrementa o dano de ataque dos projéteis disparados pela nave; o "*Damage*", que coloca a percentagem de dano em 100%, fazendo com que cada tiro realizado destrua automaticamente o inimigo; o "*Skill Regeneration*", que acelera o tempo de espera pelo desbloqueio do uso de um *improvement* em 20%; o "*Shield Regeneration*", acelera o tempo de desbloqueio do escudo que protege a nave em 20%; o "*Luck*", que aumenta a geração de itens de *improvements* em 13%, o "*Gems Bonus*", que concede mais 10% de *gems*; o "*Magnetism*", que aumenta o alcance para coletar itens, atribuindo um efeito de magnetismo à nave; o"*Time Increase*", que aumenta o tempo de uso de um item em 10%; o"*Critical*", que dá ao jogador mais 10% de chance de conseguir um tiro crítico nos inimigos; o "*Multiple Shooting*", que oferece ao jogador dois locais de tiro extras na nave; e por fim, os *improvements* "*Start with skill*" e "*Start with shield*", como o nome indica, o jogador começa o nível já com a *skill* ou o *shield* ativo. Estes dois só podem ser adquiridos na loja de *improvements* e não durante o jogo.

Outro elemento importante no jogo são os inimigos, constatou-se que existem dois tipos de inimigos: os "normais", que têm diversas representações visuais e projéteis diferentes, nem todos efetuam disparos, pois alguns deles apenas se movimentam em bando e depois assumem uma posição em grelha no ecrã do jogo.

O outro tipo de inimigo é o especial de final de nível. Existem oito inimigos de fim de nível diferentes, associados a cada um dos capítulos presente no jogo. Pela silhueta, dá para entender que estes são os mesmos presentes no jogo "RETRO Space", porém com uma representação gráfica melhorada.

3.4.2Interface do Jogador

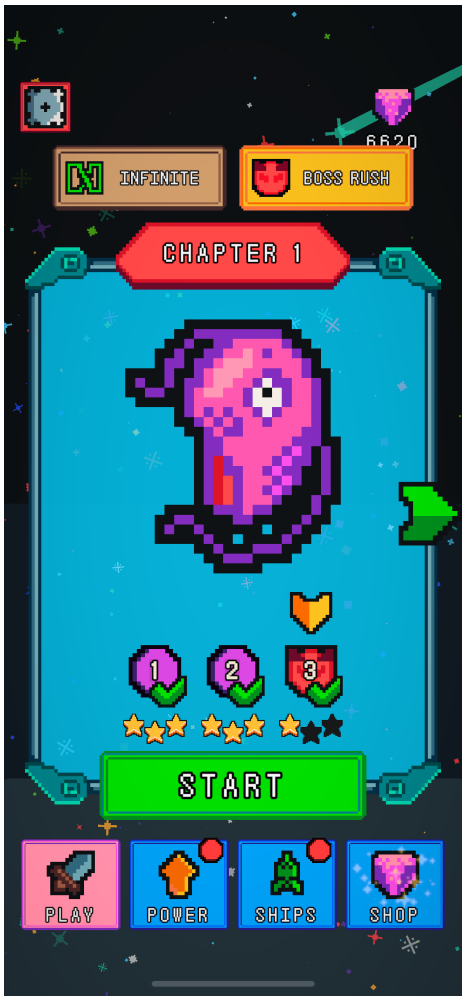


Figura 16 - Ecrã de seleção de capítulos

Na **Figura 16**, é possível observar o ecrã de seleção de capítulos do jogo, no caso o capítulo 1. Este é o ecrã principal do jogo, encontramos no canto superior esquerdo, o botão para aceder às definições do jogo, onde pode-se ativar ou desativar a vibração, a música, os efeitos sonoros, os efeitos da câmera e o texto presente na ação do jogo. Existe também, a hipótese de selecionar o nível de gráficos que desejamos para o jogo, são três as opções disponíveis: baixa, média e alta. Nas definições, é dada a opção de definir a sensibilidade que queremos para o controlo da nave, numa escala de 0 a 5. Já no canto oposto do ecrã, está posicionada a contagem das "*gems*" coletadas ao longo do jogo, estas podem ser trocadas por itens da loja, por naves e ou melhorias. De seguida, dois botões, que representam dois tipos diferentes de jogo: o "*Infinite*" e o "*Boss Rush*", estes dois modos, só se encontram disponíveis, depois de o jogador ter terminado o jogo no modo de história. Na parte central do ecrã, está a identificação do capítulo atual do jogo, com a representação gráfica do inimigo final que o jogador irá enfrentar. Em baixo deste, existem os vários subcapítulos do capítulo principal, pelo qual somos classificados numa escala de três estrelas. A verde, um botão de "*Start*", onde

pode-se iniciar o subcapítulo que desejamos jogar, selecionado em cima. Por último, uma barra com quatro botões: o “*Play*”, que é o que está selecionado na ilustração; “*Power*”, que permite a compra de melhorias associadas ao jogador; o terceiro botão “*Ships*”, remete-nos para o ecrã de seleção de naves; e o quarto e último botão “*Shop*”, direciona o jogador para a loja do jogo, onde se pode adquirir pacotes de naves, entre outros.

De destacar a simplicidade de toda esta interface, que se torna simples e atraente, sem muita informação que pode se tornar desconfortável para o jogador.

Figura 17 - Ecrã do menu de melhorias

Na **Figura 17**, é representado o ecrã das melhorias do jogador, descritas anteriormente, contudo o que se de destaca é a simplicidade deste ecrã, que disponibiliza uma grelha com um *scroll*, onde podem ser encontradas todas as melhorias presentes no jogo, em baixo dessa grelha, existe um retângulo onde é feita uma explicação da melhoria que está selecionada no momento.

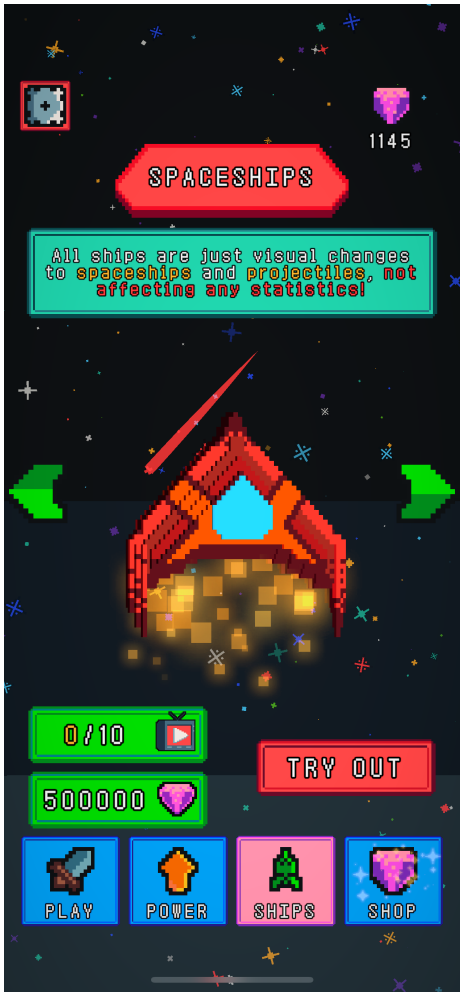


Figura 18 - Ecrã de seleção de naves

A **Figura 18**, é destinada á seleção de naves. Um menu que funciona com navegação horizontal. O jogador pode verificar todas as naves disponíveis do jogo, clicando nas setas verdes à esquerda e à direita do ecrã. A representação gráfica da nave selecionada no momento destaca-se ao centro, permitindo uma visualização mais pormenorizada da mesma.

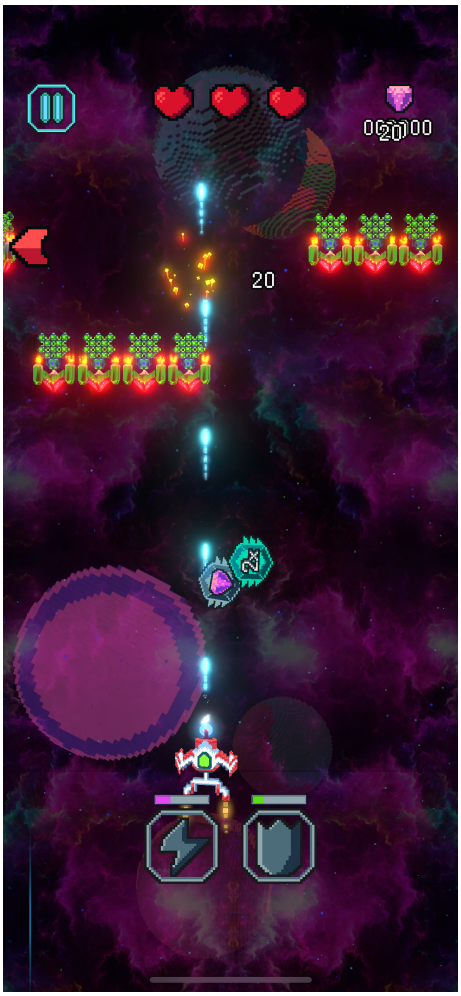


Figura 19 - Ecrã de jogo

A ilustração da **Figura 19** mostra o ecrã de ação do jogo. Na parte superior do ecrã, está presente o *HUD*, onde, no canto superior esquerdo, encontra-se o botão de pausa. Ao centro, as representações gráficas das vidas que o jogador possui, através dos corações. No lado direito, estão as *gems* ganhas ao longo do nível. Verifica-se um primeiro bug neste mesmo item, a numeração fica sobreposta por alguma razão, pode-se observar vários zeros atrás do vinte que está presente na contagem. Presente na ilustração existe um tipo de inimigo, que no caso surgem organizados em filas horizontais, um deles acaba de ser eliminado e onde é possível se verificar a numeração da pontuação obtida por essa ação. Ao centro do ecrã, estão representados dois tipos de *improvements*, que foram criados com a morte de um ou mais inimigos. Na parte inferior do ecrã, existe a representação da nave do jogador, a disparar os projéteis de cor azul e luminosos na direção das naves inimigas. Por último, existem dois botões: um para acionar o ataque especial e outro para ativar o escudo de proteção da nave, estes só se tornam operacionais após alguns segundos de jogo, e quando usados retornam ao seu estado inicial.

Figura 20 - Bug no subcapítulo 2 do capítulo 1

A imagem da **Figura 20**, mostra um outro bug, que pode bloquear o jogador por muito tempo neste nível do jogo. Quando as naves inimigas são eliminadas, elas permanecem no ecrã e o nível não termina. Possivelmente, porque algumas das naves inimigas haviam passado pela ação do jogo e não foram eliminadas, contudo é só uma suposição. Outro bug, que foi detetado e que travou a experiência de teste/análise do videojogo, foi que após terminar a luta com o inimigo final do capítulo 1, com sucesso, o ecrã ficou bloqueado tendo de ser fechado manualmente, isto repetiu-se por várias vezes.

3.4.4 Aspetos positivos

A luminosidade, as cores vibrantes, as animações brilhantes, tornam o jogo bastante atrativo.

O uso correto de momentos de publicidade, antes do jogo iniciar e no fim de cada subcapítulo, ultrapassado ou não com sucesso.

As opções oferecidas ao jogador nas definições, são muitas, e algumas podem mudar a jogabilidade e perceção do jogo, como por exemplo desativar os textos presentes no jogo.

Sons e efeitos sonoros de acordo com o estilo de jogo de tiro espacial.

3.4.5 Aspetos negativos

Primeiro aspeto a ser referido, são os bugs presentes no jogo, que, além de bloquearem a experiência de jogo, diminuíram o entusiasmo de o jogar.

O número de vídeos necessários para desbloquear as naves e melhorias é muito alto e deveria ser ajustado.

O baixo número de capítulos, tornam-no um jogo de curta de duração.

3.5 Ideias finais

Finalizado, o capítulo do estado da arte, vários aspetos dos quatro jogos analisados irão influenciar o jogo que será criado, o “Supernova QuEST”.

Absorvendo os aspetos positivos dos jogos analisados, será incluída a jogabilidade na vertical. Ao contrário do que tinha pensado inicialmente, parece mais natural manter o dispositivo móvel na vertical do que na horizontal, devido à experiência que se obteve quando foi feita esta análise.

Outro aspeto importante que se quer implementar no jogo, é a possibilidade de comandar a nave de forma paralela ao toque do jogador, não sendo necessário clicar em cima da mesma para movê-la, isto estava presente em todos os jogos analisados anteriormente.

Serão priorizados, ecrãs simples e o mais minimalistas possível. Manter os ecrãs sem muita informação é essencial para cativar o jogador, evitando criar poluição visual ao utilizador.

A criação de capítulos e inimigos finais associados aos mesmos, mostra-se bastante interessante, para que se obtenha uma conexão maior do jogador com o jogo.

Para além destas ideias, outras serão aproveitadas e futuramente abordadas neste mesmo documento.

4. GDD - Game Design Document

Neste capítulo, será apresentado o GDD, que também pode ser referido como o "*Game Design Specification*". Designado a descrever o jogo com um nível de especificações aceitáveis, de modo a que não existam ambiguidades na sua implementação. Para tal, foi criada uma estrutura própria, muito inspirada na abordagem de "Tom Sloper" [7], com referências a outras metodologias semelhantes. Assim, este GDD será composto por uma descrição geral do jogo, uma descrição detalhada, que inclui: a história do jogo, o objetivo, os capítulos do jogo, os elementos, a jogabilidade, os *storyboards*, as interfaces do utilizador, a música e os efeitos sonoros.

4.1 Descrição Geral

No jogo "Supernova QuEST", o jogador assume o comando de uma nave espacial equipada com artilharia pesada, para defender a galáxia "Supernova" do ataque dos alienígenas da galáxia vizinha "Horn". A galáxia está sob a tirania desse inimigo voraz, será que o jogador consegue libertá-la para que toda a galáxia possa viver novamente em harmonia e paz?

Este é um jogo 2D de tiro espacial de ação e sobrevivência, em inglês, destinado às plataformas móveis Android, com a idade recomendada de 12 anos ou mais.

O jogador tem à sua disposição várias naves, das quais fará uso para combater os inimigos. Este combate exigirá do jogador, alguma destreza e reflexos rápidos para se manter a salvo. Estes inimigos, são uma raça alienígena que mistura elementos de insetos e robôs, eles estão a fim de continuar o seu ataque aos planetas pertencentes à galáxia "Supernova" (Rosara, Celestia, Aurelion, Violurn e Bronzara) e irão ripostar através do disparo de projéteis na direção da nave do jogador. A fim de consolidar a liberdade do planeta no qual o jogador está presente, haverá uma batalha final com o inimigo que comanda a invasão desse planeta. Este inimigo especial terá dimensões avantajadas e estará dotado de ataques especiais contra a nave do jogador.

Por sua vez, o jogador terá à sua disposição algumas melhorias que são concedidas quando alguns dos alienígenas inimigos são eliminados, tais como a inclusão de mais um canhão de tiro na nave, um escudo de proteção para evitar danos e a destruição repentina da nave, e por fim, a reparação da estrutura da nave danificada durante a jornada de defesa e eliminação dos inimigos.

A aventura desenrola-se toda no espaço, o que faz com que exista a ocorrência de vários acontecimentos aleatórios. Um deles é a presença de asteroides no caminho da nave, de várias dimensões e com velocidades de queda diferentes, dos quais a nave se deve desviar, a fim de evitar danos e continuar a sua jornada de defesa e reconquista da galáxia "Supernova".

4.2 Descrição Detalhada

Este subcapítulo é onde pode ser encontrada toda a informação relativa ao jogo descrita de uma forma pormenorizada. Aqui, podem ser encontrados os seguintes subcapítulos: a história por detrás do jogo; o objetivo principal; os capítulos; a estrutura de dificuldade que o jogador irá enfrentar; a descrição detalhada de todos os elementos presentes, desde naves a inimigos; qual a jogabilidade; os "storyboards" desenhados a mão que descrevem cenas do jogo; a interface; a música e efeitos sonoros; e por último a descrição dos momentos de uso da *framework*.

4.2.1 História do jogo

Numa galáxia muito longínqua do nosso universo, brilhava a galáxia "Supernova", esta galáxia, estava dividida em cinco regiões, cada uma delas identificada pelo seu planeta principal, planetas estes, pacíficos e prósperos. Começando por "Rosara", o planeta rosa, com os seus cristais naturais de cor rosa e púrpura, o seu vizinho, o planeta "Celestia", caracterizado pelas suas gélidas planícies, seguidamente o planeta "Aurelion", famoso pelos seus desertos dourados, o quarto planeta "Violurn", que é dono de uma forte luminosidade própria, de cor roxa, e por fim, o planeta mais longínquo, "Bronzara", coberto de montanhas e conhecido pela sua mineração de "Bronzium". Por muito tempo, os planetas coexistiram em paz, unidos por uma aliança que celebrava o progresso e a harmonia. Essa utopia, foi quebrada quando uma frota sinistra vinda da galáxia vizinha "Horn", invadiu a galáxia "Supernova", liderada por forças alienígenas devastadoras.

Os exércitos da galáxia "Horn" são compostos por criaturas monstruosas, como os "Insetroid_02" e "Insetroid_04", soldados biomecânicos, frutos de uma mistura entre insetos e robôs. Outro dos soldados desta força, são os "Oculon", caracterizados pelos seus enormes olhos que veem para lá do espectável, estes são espiões excecionais. Por último, como soldados, os "Omnivision", evolução do "Oculon", que são estrategistas e bem mais fortes que a sua versão anterior.

No comando destes soldados, existem os temíveis líderes: "Krakthorn", um brutal mestre de guerra que se apoderou do planeta "Rosara"; "Glacithron", conhecido pelos seus tentáculos gélidos, que facilmente se adaptou ao planeta "Celestia"; os gémeos "Ruckhorn" e "Redhorn", ambos com uma aparência feroz e intimidante, blindados de espinhos e chifres afiados, que ficaram atribuídos aos planetas "Aurelion" e "Violurn", respetivamente. No planeta mais longínquo, "Bronzara", no qual é possível obter o metal mais precioso da galáxia, o "Bronzium", ficou a cargo do líder máximo "Toxicaris", uma criatura envolta em mistério e veneno, cujo único objetivo é transformar a galáxia "Supernova" em um domínio de destruição e poluição.

Para enfrentar essa ameaça, os habitantes da galáxia “Supernova” recrutaram a “Vanguard Estelar”, uma força especial interplanetária dedicada à defesa das galáxias, da qual o jogador fará parte. Estas forças especiais têm em sua posse oito magníficas naves espaciais que o jogador terá a oportunidade de pilotar: “Azure Horizon, Celestial Blue, Inferno Falcon, Phoenix’s Talon, Verdant Flame, Ember Leaf, Midnight Voyager e Shadow Star”.

Com a derrota de “Toxicaris” e as suas tropas, a galáxia “Supernova” retornará a sua paz e prosperidade. Esta será a história de um herói celebrado, uma história de coragem e sacrifício, pela qual os habitantes de “Supernova” ficarão eternamente gratos ao seu herói.

4.2.2 Objetivo

O objetivo do jogo, é salvar a galáxia “Supernova” da sua destruição proveniente do ataque alienígena. Para isso, o jogador deve eliminar todos os inimigos e obstáculos que encontrar pela sua frente. O jogo fica completo quando todos os capítulos forem concluídos com sucesso.

4.2.3 Capítulos

Esta secção tem como objetivo apresentar os capítulos do videojogo, estes capítulos serão os níveis que o jogo irá oferecer, no total serão cinco capítulos.

Foi feita, uma associação entre capítulo e planeta pertencente à galáxia “Supernova”, de forma a tornar o jogo mais relacionável para o jogador.

A dificuldade varia por ordem crescente, ou seja, conforme o progresso feito no jogo, haverá mais espécies de criaturas a enfrentar e o seu nível de dano será maior. É associado um inimigo final a cada um dos capítulos, com níveis de dificuldade diferentes. O capítulo seguinte só é desbloqueado quando o capítulo anterior ficar completo com a destruição do inimigo final.

Os elementos que diferenciam os planetas, para além dos alienígenas que o coabitam, são os fundos em que a ação ocorrerá e a representação do planeta que será usada no menu de seleção de capítulos.

O primeiro capítulo retrata o planeta “Rosara”, um planeta de tons rosa e púrpura devido ao seu solo, que possui cristais naturais dessas cores. O combate ocorre no espaço sombrio, o que nos permite reconhecer o planeta são as suas constelações e os reflexos de luz dos cristais pertencentes ao mesmo.

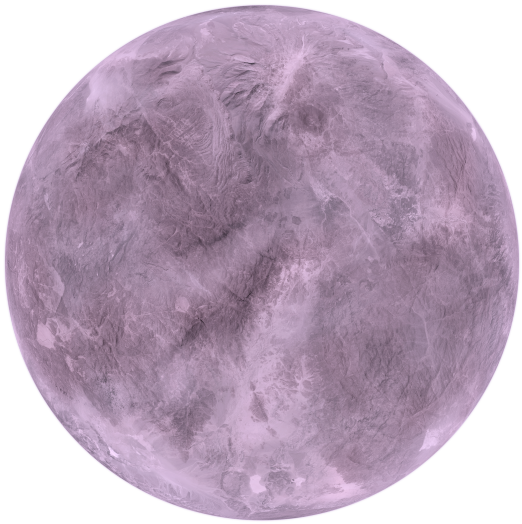


Figura 21 - Ilustração do planeta Rosara

Como ilustrado na **Figura 21**, é possível ver o que será a representação do planeta “Rosara”. Esta representação faz parte do menu de seleção de capítulos.

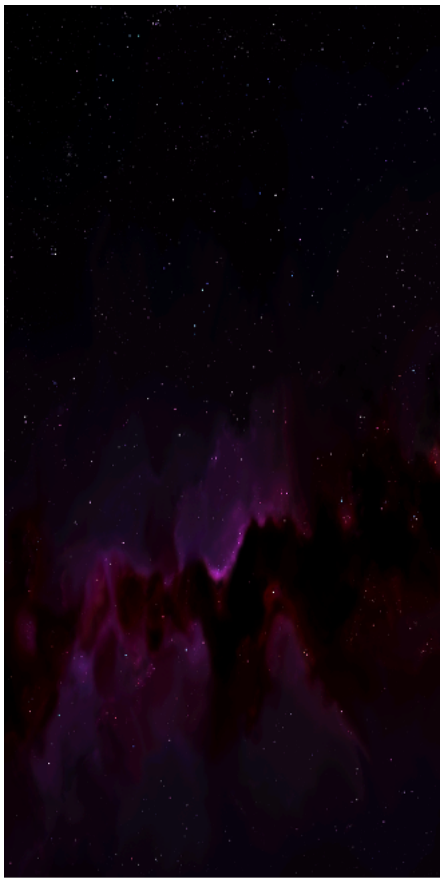


Figura 22 - Fundo do capítulo 1

Na **Figura 22,** está presente o fundo utilizado para representar o espaço do planeta “Rosara”. O ambiente é definido pelo fundo espacial, onde as constelações e os pontos de luz em tons rosados e púrpuras são os elementos que o distinguem.

O segundo capítulo, retrata o planeta “Celestia”, caracterizado pelas suas planícies gélidas e suas belas auroras boreais.

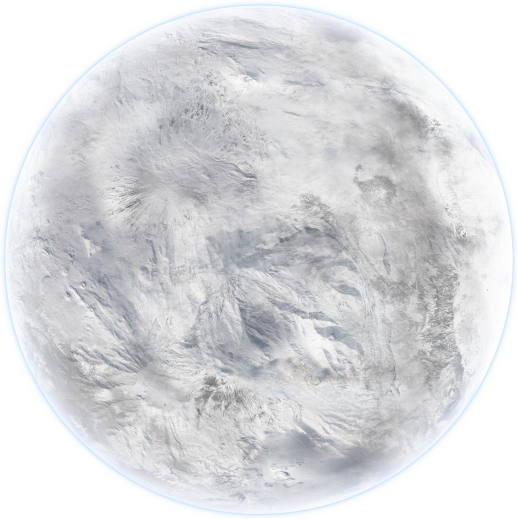


Figura 23 - Ilustração do planeta Celestia

A **Figura 23**, representa o planeta “Celestia”, um planeta de cores brancas e cinzas, esta imagem fará parte do menu de seleção de capítulos do jogo.

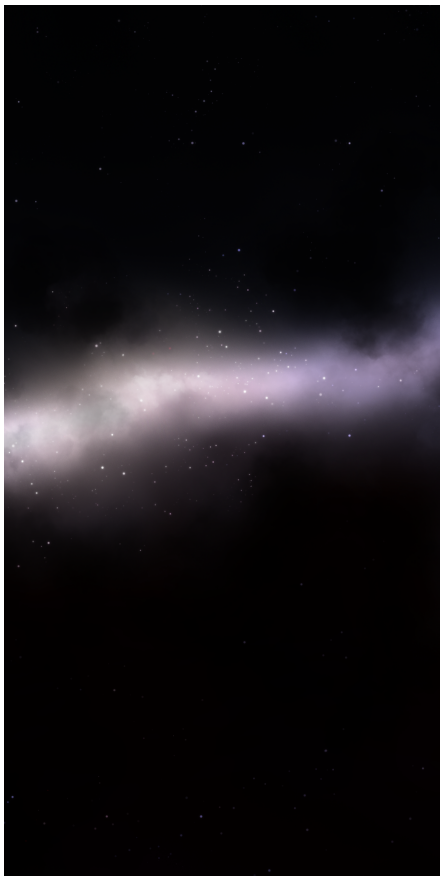


Figura 24 - Fundo usado no capítulo 2

A ilustração da **Figura 24**, define como será o fundo do capítulo 2 relacionado ao planeta “Celestia”. Como descrito, este planeta é envolto por auroras boreais, logo uma representação delas deve estar presente no fundo deste capítulo.

O terceiro capítulo, é protagonizado pela defesa do planeta “Aurelion”, planeta caracterizado pelos seus desertos e neblinas gasosas de tom amarelo.

Figura 25 - Imagem do planeta Aurelion

Na **Figura 25**, é possível visualizar a representação do planeta “Aurelion”, esta será outra das imagens presentes no menu de seleção de capítulos.

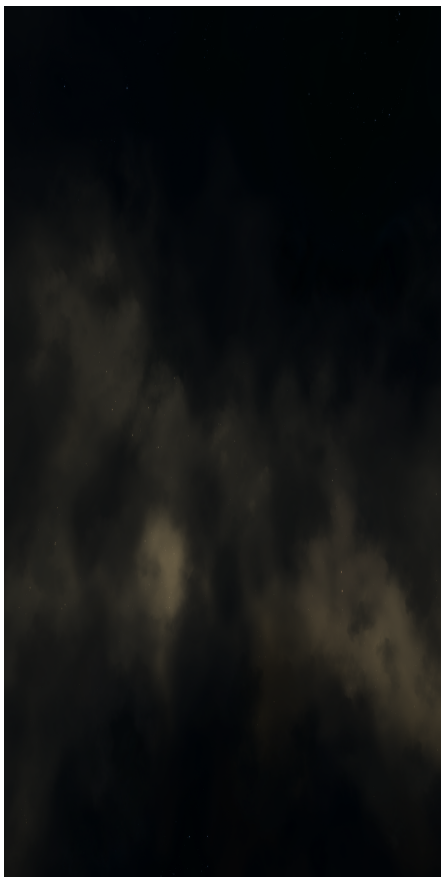


Figura 26 - Fundo do capítulo 3

Na **Figura 26**, é possível verificar o fundo que será usado no capítulo 3, onde são retratadas as neblinas gasosas.

O quarto capítulo passa-se no planeta “Violurn”, conhecido pela sua luminosidade própria em tons roxos, rodeado por várias constelações e gases que refletem o tom do planeta no firmamento.

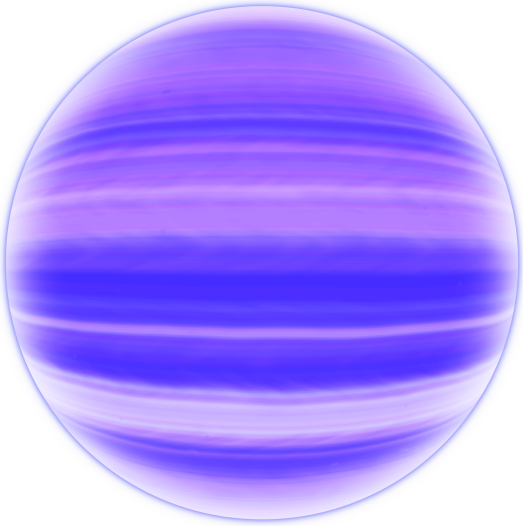


Figura 27 - Representação do planeta Violurn

A **Figura 27**, é a representação do planeta “Violurn”. Como se pode verificar, é um planeta com muita luminosidade e cores vibrantes.

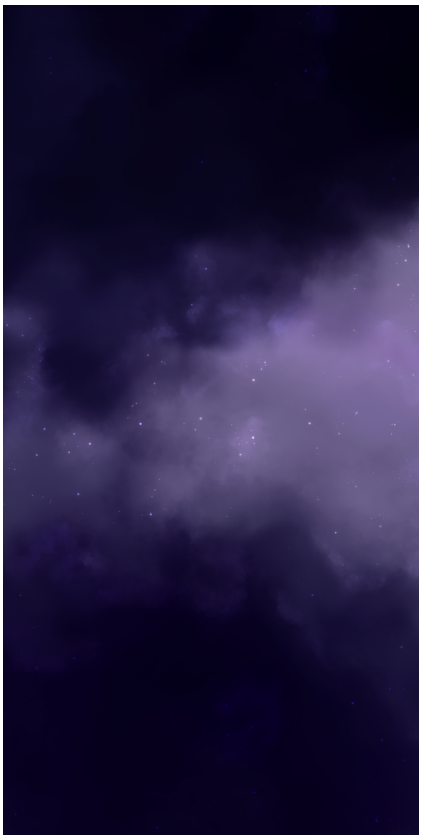


Figura 28 - Fundo do capítulo 4

É possível verificar na **Figura 28** o fundo usado para retratar o espaço do planeta “Violurn”, onde estão presentes algumas das constelações que o rodeiam e os gases de tons roxos que o planeta produz.

O último capítulo ocorre no planeta “Bronzara”, caracterizado pelas suas vastas montanhas de argila, onde é feita a extração de “Bronzium”, um metal muito precioso da galáxia “Supernova”.

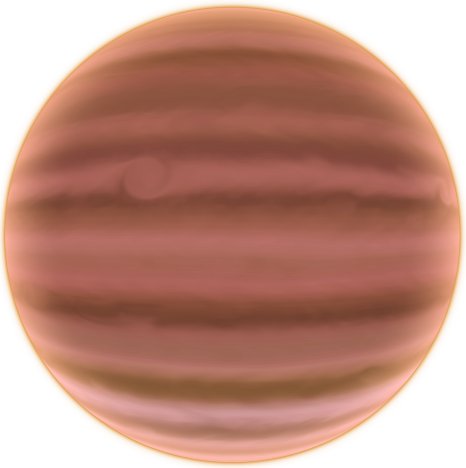


Figura 29 - Ilustração do planeta Bronzara

Na **Figura 29**, mostra a representação do planeta “Bronzara”. Esta será a última ilustração a ser desbloqueada, no menu de capítulos do jogo.

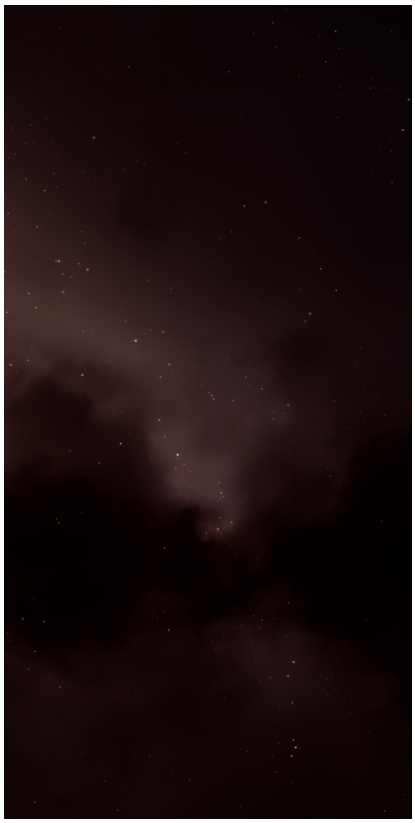


Figura 30 - Fundo do capítulo 5

Na **Figura 30**, é possível ver o fundo usado para representar o planeta "Bronzara", que apresenta as constelações de toda a galáxia "Supernova" e neblinas da mesma cor do planeta.

Os *assets* usados neste subcapítulo "Capítulos", fazem parte de um *package* de *assets* chamado "*2D Space Kit*", com autoria de "Brett Gregory", disponível de forma gratuita na loja de *assets* do *Unity* [8].

4.2.3 Estrutura de dificuldade

A estrutura de dificuldade dos níveis está presente na **Tabela 2**, onde se pode verificar a correspondência entre capítulos, inimigos, inimigo final e o tempo em segundos que o capítulo vai durar, até aparecer o inimigo final.

Por outras palavras, o funcionamento base de todos os capítulos, passa por eliminar o máximo possível de inimigos para acumular moedas e *power-ups*. Para que após um determinado tempo, o inimigo final apareça no ecrã e o jogador tenha de enfrentá-lo. Caso o jogador elimine o inimigo especial, o capítulo fica completo e o jogador é convidado a jogar o capítulo seguinte, caso contrário, se o jogador for eliminado, recomeçará o capítulo desde o tempo inicial de duração do nível sem *power-ups*, e com o "*power*" da nave restaurado.

Por exemplo, no capítulo "Celestia", o jogador irá enfrentar os inimigos "Insetroid_02" e "Insetroid_04", estes surgem de forma aleatória. Quando o nível inicia, a contagem do tempo começa de forma sincronizada, quando este tempo de nível for igual a quatro minutos e vinte segundos, definidos na **Tabela 2**, para este capítulo, os inimigos deixam de ser gerados e surge o inimigo final do capítulo no ecrã de jogo. Este deverá ser eliminado em uma luta final, neste duelo não existe qualquer limite de tempo.

Um dado importante a acrescentar, é que existe outro elemento que não está presente na **Tabela 2**, são os asteroides, pois estes estão presentes em todos os capítulos do jogo e serão gerados com maior ou menor frequência, conforme a ordem crescente de dificuldade dos níveis.

Tabela 2 - Estrutura de dificuldade

Capítulo	Inimigos	Inimigo Final	Tempo(s)
Rosara	Insetroid_02	Krakthorn	200
Celestia	Insetroid_02 + Insetroid_04	Glacithorn	260
Aurelion	Insetroid_04 + Oculon	Ruckhorn	320
Violurn	Insetroid_02 + Oculon + Omnivision	Readhorn	380
Bronzara	Insetroid_02 + Insetroid_04 + Oculon + Omnivision	Toxicaris	440

4.2.4 Elementos

Esta secção tem como principal objetivo, apresentar de forma detalhada todos os elementos presentes no jogo. As naves controladas pelo jogador, os vários tipos de inimigos: os asteroides; o exército de soldados extraterrestres e os seus líderes; os projéteis disparados tanto pelo jogador como pelos inimigos; os três tipos diferentes de *power-ups*: "*shield protection*", "*power boost*" e "*ultra shoot*"; o elemento escudo da nave; e por fim, as moedas usadas na loja do jogo.

4.2.4.1 Naves

O elemento principal do jogo, é a nave pilotada pelo jogador. Esta terá atributos como, "*power*" ou vida, o tipo de projétil disparado, o número de canhões de tiro e o intervalo de tempo em que os projéteis são disparados. Começando com o primeiro atributo, o "*power*", será o sistema de vida da nave, ele determina o quanto de dano a nave irá suportar. Existirá uma barra no topo do ecrã de jogo que ilustra qual o nível atual de "*power*" que a nave possui no momento. O segundo atributo refere-se ao fato de que cada nave terá associado um tipo de projétil, que poderá diferir em forma, cor e dano causado. O terceiro atributo, é o de número de canhões da nave ou pontos de disparo, uma vez que algumas das naves têm a capacidade de disparar de mais que um ponto de tiro. O último atributo, de intervalo de tempo de disparo, define as naves que têm uma cadência de tiro mais rápido, isto é definido em espaços de tempo em milissegundos. Este disparo, é feito automaticamente não sendo necessário qualquer tipo de interação por parte do jogador. A única ação que é requerida do jogador é pilotar a nave com o seu dedo, esquivando-se e derrotando os inimigos. No jogo, são apresentados oito tipos de naves com atributos e histórias diferentes, que são as seguintes:

Figura 31 - Azure Horizon

Como ilustra a **Figura 31**, é possível observar a representação da nave “Azure Horizon”, em tons de azul, preto e metalizados. Esta é a nave inicial do jogo, a única que estará desbloqueada quando ele começa. Terá um nível de “*power*” de 50, o dano do projétil que dispara é de 10, a cor deste projétil será o branco. Tem apenas um canhão de disparo, e o seu tempo entre disparos é de 0,6 segundos.

Figura 32 - Celestial Blue

A **Figura 32**, representa a nave “Celestial Blue”, esta é a evolução da “Azure Horizon”, mantendo as cores, contudo com características estéticas diferentes. Em termos de atributos, o seu “*power*” será de 55, o dano dos projéteis será de 10, este projétil tem a mesma cor do anterior, o branco. Contudo, terá dois canhões de tiro, ao invés da sua antecessora que só tinha um. Por último, o tempo entre disparos será de 0,6 segundos.

Figura 33 - Inferno Falcon

A ilustração da **Figura 33**, retrata a nave “Inferno Falcon”, em tons laranja, vermelho e metalizados. Tem um “*power*” de 60, apenas um canhão de disparo, com um projétil com dano de 15, de cor vermelha. Sendo mais rápida que as versões anteriores, pois demora 0,5 segundos a efetuar o disparo.

Figura 34 - Phoenix's Talon

Como ilustrado na **Figura 34**, pode-se ver a representação da “Phoenix’s Talon”. A evolução da nave “Inferno Falcon”, com as mesmas cores, mas com uma representação gráfica diferente. O “*power*” desta nave será de 65, terá dois canhões de disparo, o dano dos projéteis será de 15, de tom vermelho. O tempo de tiro mantém-se em 0,5 segundos, assim como na sua versão anterior.

Figura 35 - Verdant Flame

A **Figura 35**, exibe da “Verdant Flame”. Esta nave, tem como cores principais o verde, verde-escuro e o laranja. Os seus atributos são os seguintes, o “*power*”, é de 70, tem um canhão de disparo apenas, porém o nível de dano do projétil passa a ser de 20 e a sua cor será a laranja. Por último, o tempo entre disparos é de 0,4 segundos, bem mais rápida que as primeiras versões das suas antecessoras.

Figura 36 - Ember Leaf

A **Figura 36**, apresenta a nave “Ember Leaf”. Esta é a versão evolutiva da “Verdant Flame”, apresenta os mesmos tons, mas com características estéticas diferentes. Quanto aos seus atributos, o nível de “*power*” é de 75 e terá dois canhões de disparo, com projéteis de cor laranja e dano de 20. O tempo de disparo será igual ao da versão anterior, de 0,4 segundos.

Figura 37 - Midnight Voyager

Como ilustrado na **Figura 37**, pode-se ver a nave “Midnight Voyager”. Esta nave é caracterizada por seus tons de verde, verde-escuro e alguns detalhes em laranja. Ela possui um “*power*” de 80, que é o máximo permitido pelo jogo e terá apenas um canhão de tiro. Cada projétil disparado pela nave causa 25 de dano, a cor destes projeteis é azul. Por fim, o tempo de intervalo do disparo automático feito pela nave é de 0,3 segundos, o que é bastante rápido, causando um efeito de metralhadora no tiro da nave.

Figura 38 - Shadow Star

A última nave disponível no jogo está representada na **Figura 38**, de seu nome “Shadow Star”, é a evolução da “Midnight Voyager”, com os mesmos tons da versão anterior, mas características estéticas diferentes. O seu nível de “*power*” será de 80, o número de canhões disponíveis são dois, o dano do projétil é de 25, o projétil tem o mesmo tom da nave anterior. O tempo de intervalo entre disparos será de 0,3 segundos.

Como nota final, acrescenta-se que todas as figuras representadas neste subcapítulo “Naves”, fazem parte de um *package* de *assets* disponível na loja do *Unity,* chamado *“Free Stylized 2D Space Shooter Pack”,* criado por” Larzes” [9].

4.2.4.2 Inimigos

O jogo terá três tipos de inimigos: os asteroides, o exército de alienígenas e os seus líderes.

Existem, duas representações gráficas dos asteroides. Estes, surgem no ecrã em posições aleatórias, caindo de cima para baixo, rodopiando e com velocidades aleatórias dentro de uma faixa de valores predefinidos. Outro aspeto interessante, que também é atribuído de forma arbitrária, é o tamanho dos asteroides quando criados.

Os atributos deste elemento são: a durabilidade, o dano que causa, o valor mínimo e máximo de velocidade que o asteroide pode ter, e por fim a velocidade de rotação que pode atingir.

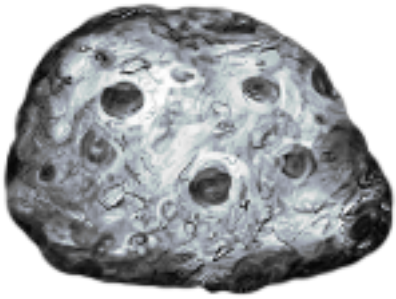


Figura 39 - Asteroide 1

Na **Figura 39**, a representação do asteroide 1, de formato circular. A sua durabilidade é de 10, o dano causado quando embate com a nave do jogador é de 5. A sua velocidade de queda mínima será de 5 e a máxima de 10. Já a velocidade de rotação será de 80.

Figura 40 - Asteroide 2

Ilustrado na **Figura 40**, o asteroide 2, o qual possui um formato cilíndrico. Por esse motivo, os seus atributos diferem do anterior. A durabilidade ao dano causado pela nave do jogador será de 20, o dano que causa ao colidir com a nave será de 10. Quanto à sua velocidade de queda, ela pode ter valores entre 5 e 7, e o valor da velocidade de rotação será de 60.

Estas representações de asteroides, da Figura 39 e Figura 40, fazem parte do mesmo *package* de *assets* do subcapítulo “Capítulos” [8].

O seguinte inimigo, é o soldado alienígena vindo da galáxia “Horn”. Estarão presentes em jogo quatro tipos de representações, cada uma com um nome e história diferentes.

São eles os da família dos “Insetroid_02” e “Insetroid_04”, inspirados na aparência de um inseto com características robóticas, são alienígenas biónicos que têm a capacidade de expelir um raio verde de forma a causar dano à nave do jogador.

Dispõem-se de mais uma família de soldados, que inclui os “Oculon” e os “Omnivision”. Estes possuem olhos enormes e características que lembram uma mistura de louva-a-deus e um inseto alienígena, conta a história que são excelentes espiões. Estes têm a mesma capacidade de expelir raios, contudo mais poderosa que a dos “Insetroid”.

Os atributos de todos eles serão os seguintes: nível de vida, dano causado na nave quando existe colisões, velocidade de voo, número de pontos de disparo, tipo de disparo e tempo de intervalo de disparo, um pouco à imagem da nave do jogador.

Figura 41 - Insetroid_02

A **Figura 41**, mostra a representação do “Insetroid_02”. A sua vida terá o valor de 20, o dano que causa quando embate com a nave é de 10, a velocidade de voo é de 4, pois é um ser biomecânico e as asas são parte do inseto, logo será mais lento a deslocar-se. Consegue expelir o seu ataque por um único orifício, este será o raio verde, que será descrito mais adiante neste documento. O tempo de intervalo em que expele o raio será de 2,5 segundos.

Figura 42 - Insetroid_04

A **Figura 42**, representa a versão seguinte da família “Insetroid”, o “Insetroid_04”. Tem uma aparência mais obscura que a sua versão anterior, o “Insetroid_02”. Disponibiliza como vida o valor de 30, sendo mais difícil de eliminar que a sua antiga versão. Causa um pouco mais de dano quando embate com a nave do jogador, tendo um valor de 15. Será mais rápido no seu voo, com um valor de 5 no atributo da velocidade. Terá a possibilidade de expelir por dois orifícios, o raio será o mesmo da versão anterior, contudo será mais rápida a sua expulsão, com o valor de 2 segundos de intervalo.

Figura 43 - Oculon

A **Figura 43**, exibe o alienígena “Oculon”. Tem como valor de vida 35, o dano causado à nave será de 20. A velocidade de deslocamento é de 5, bem mais rápida que a dos soldados anteriores. Consegue expelir o seu ataque por um orifício, o seu ataque é um raio amarelo, que será detalhado neste mesmo capítulo do documento. Este alienígena consegue expelir o seu raio em intervalos de 1,5 segundos.

Figura 44 - Omnvision

A ilustração da **Figura 44**, representa o alienígena “Omnivision”, uma versão mais sombria, devido aos seus tons negros. A sua vida tem o valor de 40, o dano causado à nave, caso interajam diretamente é de 25. Ele, consegue expelir o seu ataque por dois orifícios, os raios de luz amarela. O intervalo de tempo em que esses raios serão expelidos é de 1 segundo, apenas. Este é o soldado mais mortífero e rápido disponível no jogo.

Passando agora para outro tipo de inimigo, os inimigos finais de nível, mais conhecidos como os líderes dos exércitos que tomaram posse dos planetas que invadiram.

Cada um tem a sua identidade e características próprias. No total, são cinco: “Krakthorn”, “Glacithorn”, “Ruckhorn”, “Redhorn” e “Toxicaris”.

Têm como atributos os mesmos dos seus soldados, diferenciando-se por terem um ataque especial. (No momento da escrita deste documento, o tipo de disparo e os ataques especiais ainda não estão definidos.)

Figura 45 – Krakthorn

A **Figura 45**, ilustra o líder “Krakthorn”, que tem como alcunha o “mestre da guerra”, líder que está em posse do planeta “Rosara”. Possui uma vida de 60 valores. Como são criaturas maiores, a sua velocidade diminui, logo terá apenas 2 valores. Contudo, são astutos no disparo e desta forma, têm a capacidade de disparar em intervalos de apenas 1 segundo. Estes disparos são feitos por quatro orifícios nos seus tentáculos. Caso a nave colida diretamente com ele, causa um dano de 30 valores, a mesma.

Figura 46 – Glacithorn

O líder “Glacithorn”, conhecido pela sua frieza, está representado na **Figura 46**. Este é o principal invasor do planeta “Celestia”, do qual tomou posse. A sua vida tem o valor de 70, e a sua velocidade de deslocamento será de 2,5. Como é da mesma família de “Krakthorn”, uma vez que os dois têm genes de polvo, consegue também disparar por 4 dos seus tentáculos, com um intervalo de tempo de 1 segundo. Se no combate final, este colidir com a nave do jogador, causará um dano à mesma no valor de 30.

Figura 47 – Ruckhorn

Como exibido na **Figura 47**, vemos o primeiro dos dois irmãos gémeos, o “Ruckhorn”, este tem uma aparência feroz e intimidante. É o líder que ocupou o planeta “Aurelion” e o tem sobe ameaça. Tem a durabilidade de 80 de vida, robustez dada pelo seu casco forte e espinhoso. A velocidade a que se desloca é de 3, sendo assim mais difícil ao jogador de o eliminar. Consegue expelir ataques por 6 orifícios do seu corpo, dos quais a nave do jogador se tem de esquivar. Estes ataques são feitos em intervalos de 0.8 segundos. Caso colida diretamente com a nave do jogador, este sofre 40 de dano.

Figura 48 - Redhorn

A **Figura 48**, remete para o segundo gêmeo, o líder “Redhorn”, invasor do planeta “Violurn”. Caracterizado pela sua cor vermelha, blindado de espinhos e com olhos azuis brilhantes, este possui uma durabilidade de 90 de vida. A sua velocidade também melhora em relação ao seu antecedente, passando a deslocar-se a 3,5 valores de velocidade. Tendo o mesmo aspeto que o “Ruckhorn”, consegue expelir os seus ataques pelo mesmo número de orifícios, o intervalo de tempo com que os reproduz é também exatamente o mesmo, por fim o dano causado na nave permanece no valor de 40 caso exista uma colisão entre ambos.

Figura 49 - Toxicaris

Por último na **Figura 49**, temos o líder dos líderes, “Toxicaris”, o último inimigo deste subcapítulo.

O seu nome surge da combinação da palavra “tóxico” e “caris”, que é um sufixo usado em nomes científicos para crustáceos ou insetos. É o maior de todos os inimigos, caracterizado pelo seu aspeto robusto e pelos seus olhos penetrantes. Foi aquele que tomou de assalto o planeta “Bronzara”, o planeta mais importante da galáxia “Supernova”.

É o que demonstra mais capacidade de aversão ao dano causado pela nave do jogador, com o poderoso valor de 100 de vida. Mesmo sendo uma criatura gigantesca, consegue mover-se rapidamente, apresentando 4 valores de velocidade. Porém, o que mais surpreende é a sua capacidade de projetar 8 disparos vindos do seu corpo, com um tempo de intervalo de apenas 0,5 segundos. Quando, por algum acaso, atinge a nave do jogador, esta sofre um dano de 40 valores. Todos estes fatores, tornam “Toxicaris”, o elemento mais perigoso do jogo, por esse motivo, é o guardião do planeta mais valioso que queremos reconquistar.

As imagens mostradas desde a **Figura 41** à **Figura 49**, fazem parte dos mesmos *assets* usados no subcapítulo “Naves” [9].

4.2.4.3 Projéteis

Existem dois tipos de projéteis no jogo: os que são disparados pela nave do jogador e os que são disparados pelos inimigos.

Os disparos feitos pela nave do jogador, podem ter quatro variações que combinam com as naves que estão presentes no jogo. Têm como características diferenciadoras, a cor, o dano que causam ao inimigo e a velocidade com que atravessam o ecrã de jogo.

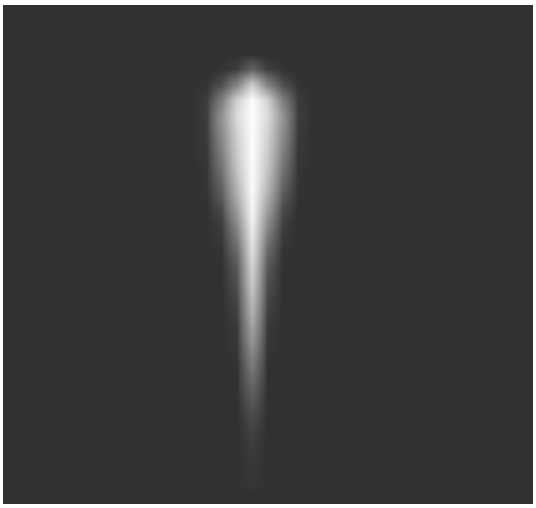


Figura 50 - Projétil da nave do jogador

Como ilustrado na **Figura. 50**, verificamos a representação gráfica que o projétil disparado pela nave do jogador terá. Nesta ilustração, é representado o disparo branco, que será realizado pelas naves, "Azure Horizon" e "Celestial Blue", terá um dano de 10 valores e uma velocidade de 8.

A **Tabela 3**, ilustra a correspondência entre projéteis e naves, assim como os valores de cada atributo dos projéteis.

Tabela 3 - Projéteis da nave do jogador

Naves	Cor	Dano	Velocidade
"Azure Horizon"; "Celestial Blue"	Branco	10	8
"Inferno Falcon"; "Phoenix's Talon"	Vermelho	15	9
"Verdant Flame"; "Ember Leaf"	Laranja	20	10
"Midnight Voyager"; "Shadow Star"	Azul	25	11

O outro tipo de projétil presente no jogo é o que é disparado pelos exércitos inimigos. Existem dois tipos de projéteis, que se diferenciam pela cor, dano e velocidade.

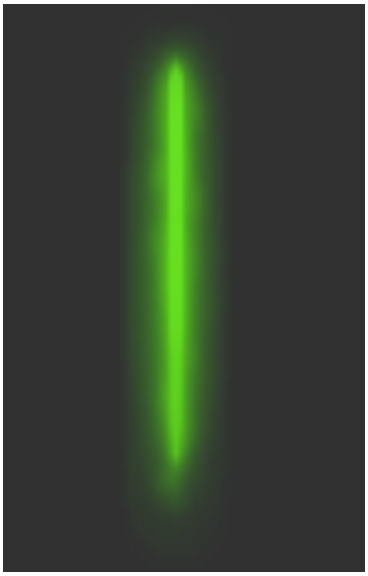


Figura 51 - Projétil dos inimigos

Na **Figura 51**, a representação de um projétil disparado por um dos inimigos, caracterizando-se por ser mais longo que os tiros das naves do jogador.

A **Tabela 4**, descreve detalhadamente qual é a correspondência entre esses disparos e os inimigos do exército da galáxia “Horn”, além disso, pode-se encontrar os valores dos atributos de cada um deles.

Tabela 4 - Projéteis dos inimigos

Inimigos	Cor	Dano	Velocidade
“Insetroid_02”; “Insetroid_04”	Verde	10	10
“Oculon”; “Omnivision”	Amarelo	20	10

Estas representações de projéteis fazem parte do pack “2D Space Kit” [8], já anteriormente referenciado, contudo fiz-lhes pequenas alterações, de tamanho e cor.

4.2.4.4 Power Up's

No decorrer do jogo, quando o jogador elimina inimigos, aparecem de forma aleatória *power-ups,* estes trazem uma melhoria momentânea ao jogador. Os três tipos de *power-ups* presentes no jogo são: "*Shield Protection*", "*Power Boost*" e "*Ultra Shot*".

Figura 52 - Shield Protection

A **Figura 52**, define a representação visual do *power-up* "*Shield Protection*". É representado por um hexágono preto com umas abas cinza, com uma seta azul ao fundo. Outro aspeto que o diferencia, é a letra 'S' no centro do hexágono. Este *power-up,* proporciona ao jogador uma proteção feita ao redor da nave. Essa proteção tem a forma circular, conforme representado na **Figura 55**, que será destacada mais à frente neste relatório. Na interface de jogo, no canto superior do ecrã, em baixo da barra de "*power*" da nave, aparecerá uma barra de cor azul. Essa barra representa o nível de proteção que a nave ainda possui, caso essa proteção receba qualquer tipo de dano, essa barra vai decrementando conforme o dano causado. Se o jogador já estiver com essa proteção ativa e conseguir coletar um *power-up* do mesmo tipo, a barra será restaurada na percentagem máxima.

Figura 53 - Power Boost

A **Figura 53**, representa o *power-up* "*Power Boost*", que possui a mesma representação gráfica do *power-up* "*Shield Protection*", mas por sua vez com tons laranjas e no centro, a letra 'P'. Este *power-up,* tem a capacidade de restituir totalmente a percentagem de "*power*" da nave. Caso o nível atual de "*power*" da nave esteja no seu nível máximo, quando é coletado este tipo de *power-up*, este mantém-se no valor máximo, não sendo incrementado em qualquer tipo de valor.

Figura 54 - Ultra Shot

Representado na **Figura 54**, o *power-up* "*Ultra Shoot*", com a mesma representação gráfica dos anteriores, porém de tons verdes e a letra 'U' ao centro. Este *power-up,* tem a capacidade de incrementar o número de canhões com os quais a nave do jogador vai conseguir disparar projéteis.

Tendo como exemplo a nave "Azure Horizon", esta dispõe apenas de 1 canhão inicial, caso o jogador colete este *power-up*, esse número passa a ser de 3, se coletar uma segunda vez, o número aumenta para 5, este será o valor máximo de canhões que a nave pode ter. Se a nave for atingida por um inimigo, o número de canhões é decrementado tendo em conta a mesma ordem de incremento, até ao número de canhões original da nave.

As representações gráficas dos power-ups, fazem parte do package "Free Stylized 2D", já referenciado anteriormente [9].

4.2.4.5 Escudo

Outro elemento presente no jogo, é o escudo da nave. Este é ativado através do *power-up* "*Shield Protection*", e proporciona uma proteção extra à nave, permitindo que o jogador consiga sobreviver por mais tempo aos ataques inimigos. Adiciona, uma capacidade de suporte de dano de 50 valores.

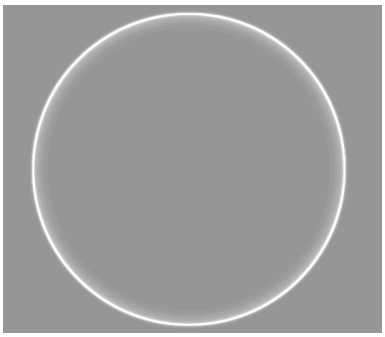


Figura 55 – Escudo

Como ilustrado na **Figura 55**, o escudo de proteção da nave. Este tem formato circular e tenta imitar uma bolha, que abriga a nave do jogador de qualquer tipo de ataque inimigo.

4.2.4.6 Moedas

O último elemento presente no jogo serão as moedas. Estas, são obtidas a partir da eliminação de inimigos, pela conclusão de capítulos ou pelas respostas certas aos conteúdos dados pela *framework*. Esta moeda, pode ser trocada por naves na loja de naves do jogo. Até ao momento, ainda não encontrei a representação gráfica desejada deste item para o jogo.

Como nota final deste subcapítulo "4.2.4 Elementos", acrescenta-se que as ilustrações e os valores dos atributos dos elementos, podem sofrer alterações no decorrer da implementação do jogo, se os testes de jogabilidade assim o justificarem. Por essa razão, estes valores não têm um vínculo definitivo.

4.3 Jogabilidade

Neste subcapítulo, será descrita a experiência de jogo que o jogador irá encontrar, como: a sequência de ecrãs; quais as mecânicas de jogo; o comportamento dos inimigos; as recompensas obtidas; a perspetiva de jogo e por último o tempo médio de duração do jogo.

O jogo, quando aberto no dispositivo móvel, direciona o jogador para o ecrã inicial, onde encontram-se três botões: um para ativar ou desativar o som do jogo; outro botão para entrar na loja, onde é possível escolher a nave que o jogador quer utilizar; e o botão de "*play*". Este botão, conduz o jogador ao ecrã de seleção de capítulos, onde é apresentada uma lista de planetas, que podem estar bloqueados ou desbloqueados conforme o progresso feito pelo jogador.

Selecionando um dos capítulos, o jogador é finalmente enviado para o ecrã de jogo. No capítulo de interface do utilizador, os ecrãs serão explicados de forma mais detalhada com o auxílio visual de alguns esboços.

No ecrã do jogo, o jogador tem a possibilidade de pilotar a nave através do toque no ecrã do dispositivo. É possível mover a nave em todas as direções que o jogador assim o desejar, apenas existe uma limitação, que é não sair com a nave do ecrã de jogo. Ela só pode circular dentro de um retângulo, que é criado de forma automática no ecrã do dispositivo móvel.

A nave comandada pelo jogador, dispara de forma automática os projéteis que eliminam todos os inimigos que se apresentam no ecrã de jogo.

Estes inimigos, são criados de forma aleatória, em espaços de tempo previamente definidos. As posições que surgem no ecrã também são arbitrárias,

dentro dos limites do ecrã de jogo. A sua trajetória no ecrã de jogo é simples, seguem uma linha na vertical desde o seu ponto de origem, descendo ao longo do eixo y, enquanto o ponto x mantém-se fixo.

Desta forma, o jogador deve ter a astúcia de mover a sua nave de forma a não ser atingido pelos inimigos, assim como pelos projéteis disparados pelos mesmos.

Por cada inimigo eliminado, serão libertadas moedas e *power-ups* de forma aleatória. As moedas dão ao jogador a possibilidade de comprar novas naves disponíveis na loja. Estas modificarão a experiência de jogo, uma vez que todas têm atributos diferentes de dano e disparos. Os *power-ups*, oferecem ao jogador uma melhoria provisória de jogo, para que possa eliminar os inimigos de forma mais rápida e eficaz.

O fim de cada capítulo, é definido pela eliminação do líder que tomou de assalto o planeta. Este líder, apenas aparece no ecrã de jogo quando um limite de tempo é atingido, e todos os inimigos "normais" foram destruídos, limpando o ecrã de qualquer distúrbio visual. Assim, os únicos elementos presentes no ecrã nesse momento, serão a nave do jogador e o líder alienígena.

O líder, tem quatro estados de comportamento: a entrada, o ataque, o ataque especial e a morte. O estado de entrada, é quando o elemento inimigo final é gerado e desce na vertical até aparecer no ecrã do jogador, esta descida é limitada à parte superior do ecrã, onde este deve ficar ancorado numa posição pré-programada. O segundo estado, é o de ataque, onde o inimigo, neste momento, encontra-se parado um pouco acima do centro do ecrã e começa a movimentar-se em ziguezague. Esta movimentação, é acompanhada do início do ataque normal do líder, disparando assim projéteis na direção da nave do jogador. Este estado dura um intervalo de tempo previamente definido. O terceiro estado, é o estado de ataque especial, o inimigo fica imóvel um pouco acima do centro do ecrã, prepara-se para realizar um tipo de ataque personalizado e muito mais danoso para a nave do jogador. Este estado, dura também um espaço de tempo anteriormente definido. Ao encerrar este estado, o inimigo volta ao segundo estado, o de ataque normal, e assim permanece neste ciclo até que o último estado seja ativado. O último estado, o da morte, ocorre quando o jogador consegue eliminar o líder que invadiu o planeta que tentamos salvar. Nesse momento, o inimigo desaparece do ecrã e o jogador recebe uma recompensa em moedas, sendo guiado para o ecrã de vitória do jogo.

Neste momento, é feita uma chamada à *framework* educacional, onde devemos responder corretamente ao conteúdo didático que aparecer no ecrã, caso a resposta seja correta, recebemos uma recompensa em moedas. Daqui, partimos novamente para o ecrã de escolha de capítulos, onde se pode selecionar o capítulo seguinte, até chegar ao último e assim concluir o jogo.

Caso, por algum percalço, o jogador fique com o "*power*" da nave vazio, será direcionado para o ecrã de derrota do jogo. Aqui, tem duas opções: a de retornar ao ecrã de capítulos ou restaurar o "*power*" da nave. Para este último, é feita uma

chamada à *framework* e caso responda de forma acertada, é-lhe dada a hipótese de continuar o jogo do ponto em que ficou quando ficou sem "*power*".

A perspetiva do jogo, é a de vista de cima, onde é possível observar todos os elementos do jogo em um plano superior.

Por último, tendo em conta a estrutura de dificuldade demostrada na **Tabela 2**, e dependendo muito da habilidade do jogador, este jogo poderá ter em média, uma duração de 40 minutos.

4.4 Storyboard

Neste capítulo, serão apresentados alguns *storyboards* que vão ajudar o leitor a visualizar como será o aspeto visual do jogo "*Supernova QuEST*".

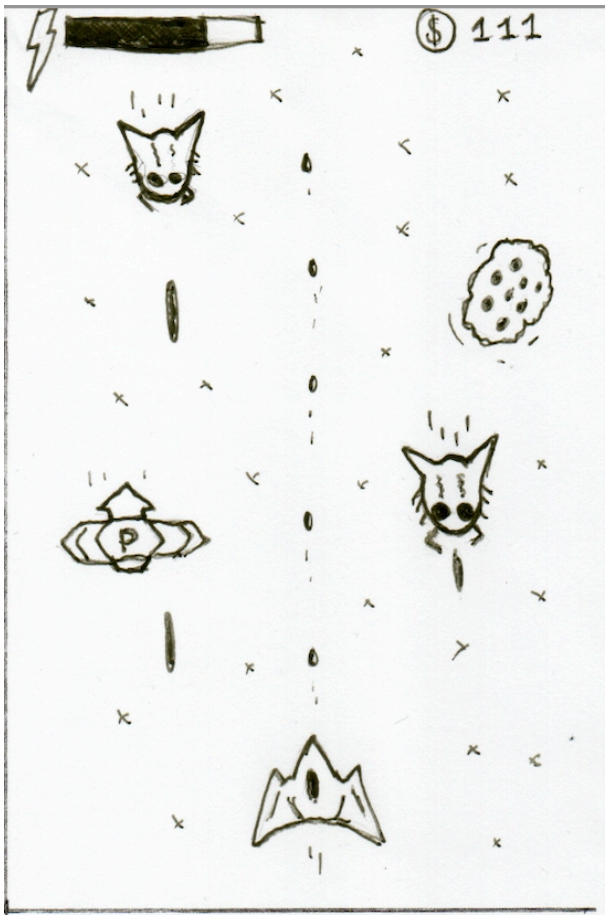


Figura 56 - Storyboard da ação do jogo

Como ilustrado na **Figura 56**, pode-se visualizar o contexto da ação do jogo ao longo do capítulo.

Posicionada na parte inferior do ecrã, é possível identificar a nave do jogador a efetuar disparos. Nesta figura, é também identificável um *power-up* de "*Power Boost*" no lado esquerdo. No centro, do lado direito, pode-se identificar um inimigo do tipo "Insetroid_02", a efetuar disparos na direção do jogador. Em cima desse elemento, inclui-se a representação do asteroide, que gira e é mais um objeto que traz desafio ao jogo.

Por último, é identificável os elementos pertencentes ao *HUD,* à esquerda o medidor do *power* atual da nave, à direita o contador de moedas ganhas pelo jogador.

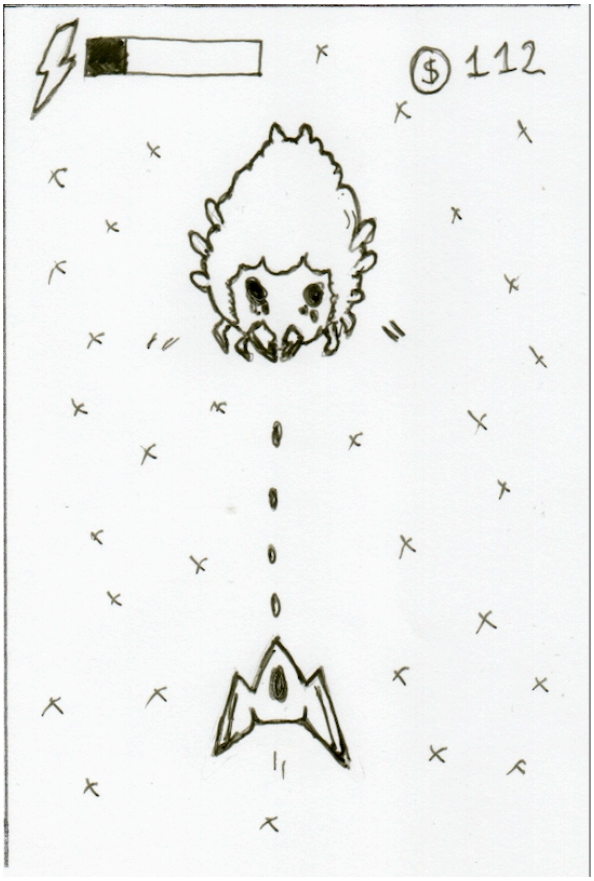


Figura 57 - Storyboard do combate com o líder

Na **Figura 57**, o *storyboard* ilustrativo do combate entre a nave do jogador e o líder final do capítulo.

Neste caso, dispõem-se da luta final do jogo, em que é enfrentado o alienígena "Toxicaris", facilmente identificado na parte superior do ecrã, dadas as suas características. Nesta fase do jogo, que é o combate final do capítulo, apenas estes dois elementos estarão presentes no ecrã. Na parte superior, é identificável o *HUD*, que acompanha o jogador durante todo o jogo.

4.5 Interface do utilizador

Este capítulo, serve para mostrar os ecrãs que o jogo irá ter. Desde o ecrã de início de jogo, passando pelo ecrã de seleção de capítulos, à loja do jogo, entre outros.

Figura 58 - Ecrã inicial

Nesta **Figura 58** está ilustrado o ecrã inicial do jogo. Esta interface, é constituída por uma representação fictícia de um planeta com um anel à sua volta, de seguida ao centro do ecrã é identificável o título do jogo “Supernova QuEST”. Este ecrã, terá três botões: um para tirar o som do jogo, colocado no canto superior esquerdo; um botão para aceder a loja, disponível no canto superior direito; e por fim, o botão de “Play”, bem destacado ao centro abaixo do nome do jogo. Este botão permite ao jogador aceder à interface seguinte, que é a de seleção de capítulos do jogo. No fundo deste ecrã, verifica-se uma representação do espaço celeste que estará de acordo com a temática do jogo.

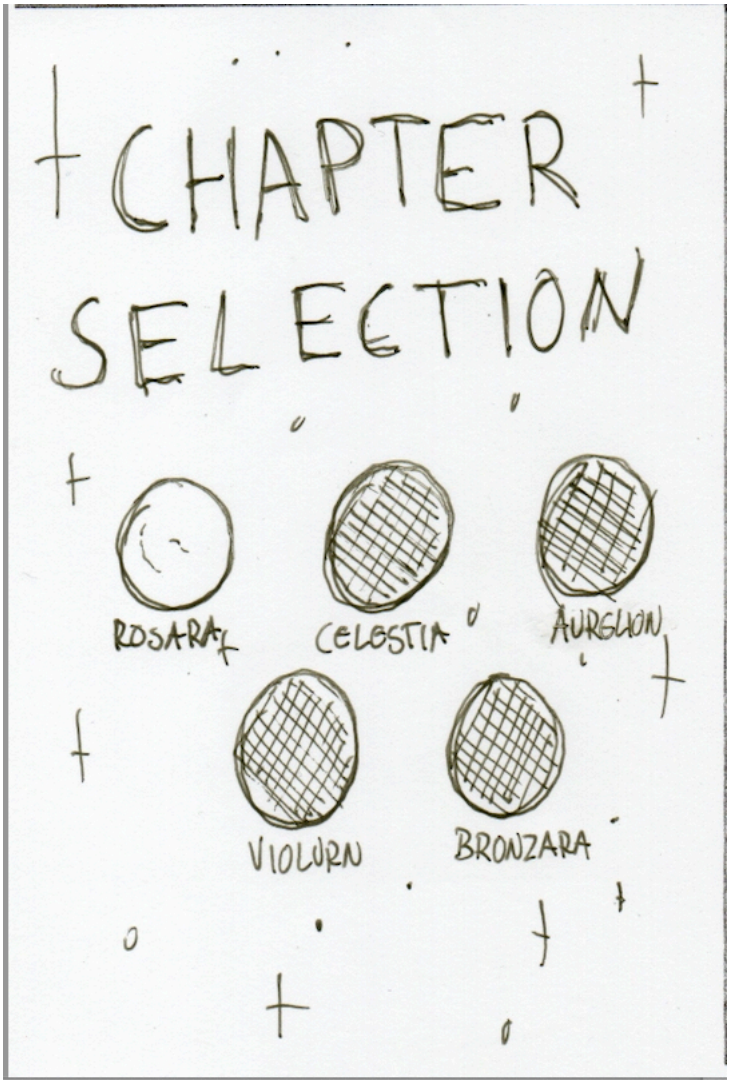


Figura 59 - Ecrã de seleção de capítulos

A **Figura 59** representa a segunda interface presente no jogo, que é a seleção de capítulos. Esta interface, permite ao jogador selecionar qual o capítulo que deseja jogar, desde que este se encontre desbloqueado. Neste caso, apenas o capítulo do planeta “Rosara” está desbloqueado, e a sua representação terá cores, os outros planetas presentes na grelha central terão um tom mais escuro e não estarão selecionáveis para o jogador.

Em cima desta grelha de seleção de capítulos, observa-se o título do ecrã que o identifica, com o mesmo tipo de letra do ecrã inicial.

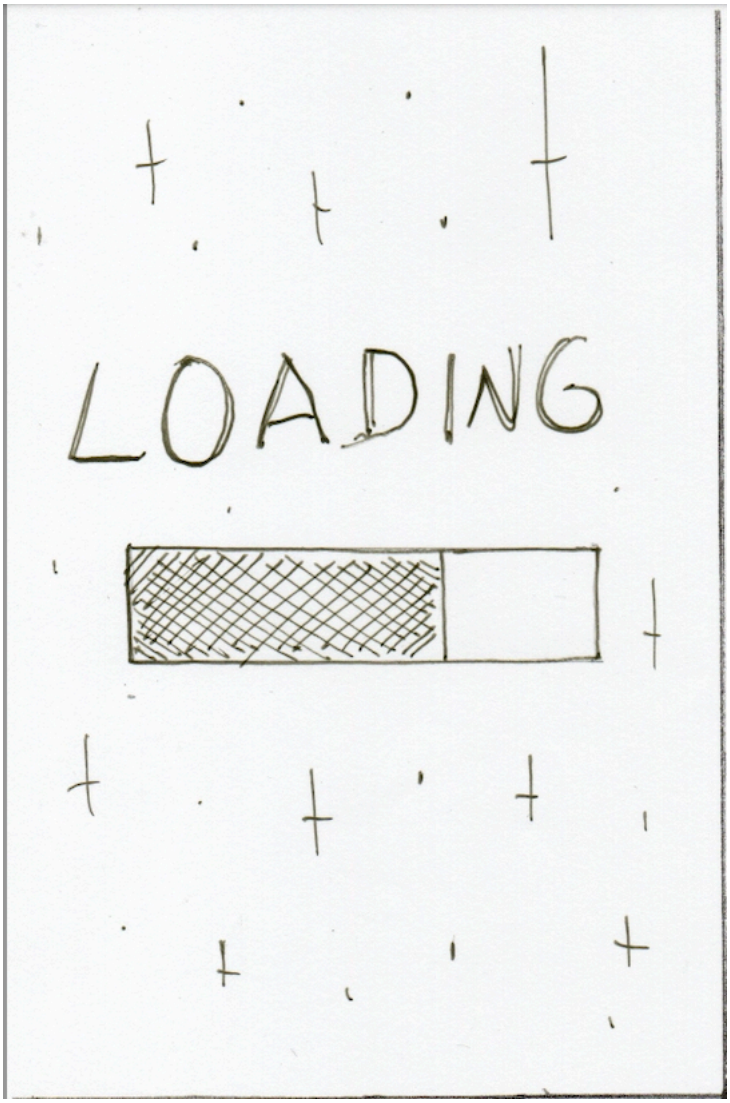


Figura 60 - Ecrã de processamento

Quando o jogador seleciona um dos capítulos, é apresentada ao jogador uma interface de processamento entre ecrãs, comumente chamado de ecrã de *loading*, representado na **Figura 60**. O aparecimento desta interface dependerá sempre da capacidade de processamento que o dispositivo móvel do jogador possui, existe a possibilidade de alguns jogadores nem chegarem a vê-la, uma vez que alguns níveis não necessitam de tanto processamento.

A interface será constituída por um texto “Loading”, com o mesmo tipo de letra dos ecrãs anteriores, e logo abaixo desse texto, haverá uma barra que mostra a quantidade de processamento que já foi efetuado e o que falta para prosseguir para o ecrã seguinte do jogo.

Figura 61 - Ecrã de fim de capítulo

Na **Figura 61**, é possível ver o ecrã de fim de capítulo. Este será um pop-up que aparece no centro do capítulo que foi concluído com sucesso, contendo o texto "Chapter Completed". Por baixo deste texto, existe um botão, com uma representação gráfica de uma seta virada à direita. Este botão conduz o jogador ao ecrã de seleção de capítulos, onde se pode observar que o capítulo que jogamos foi finalizado com sucesso e estará desbloqueado e disponível o capítulo seguinte do jogo.

Ainda na interface do fim de capítulo, sendo este um pop-up, identifica-se na parte superior o *HUD,* que contém as informações com as quais se terminou o capítulo. O fundo desta interface, será o fundo do capítulo que o jogador acabou de completar.

Dois elementos que farão parte deste pop-up, e que não estão presentes na **Figura 61**, serão a indicação de quantas moedas foram ganhas durante o decorrer do capítulo e um botão que dará a possibilidade de duplicar as moedas ganhas, através de uma chamada a *framework* educacional.

Figura 62 - Ecrã de fim de jogo

A **Figura 62**, mostra a interface de fim de jogo. Esta, será a interface que aparecerá quando a nave do jogador fica sem *power*. Assim, como na **Figura 61**, é um pop-up que surge no ecrã do jogo, com o texto "Game Over", e dois botões circulares: um com uma representação gráfica de uma seta apontada para a direita, este botão, direciona o jogador para o ecrã de seleção de capítulos; o outro botão, que terá a representação gráfica de uma seta que dá meia-volta sobre si mesma, fará uma chamada à *framework* educacional. Em caso de sucesso na interação com a *framework*, o *power* da nave do jogador é reconstituído e poderá assim continuar o nível no ponto em que ficou sem *power*.

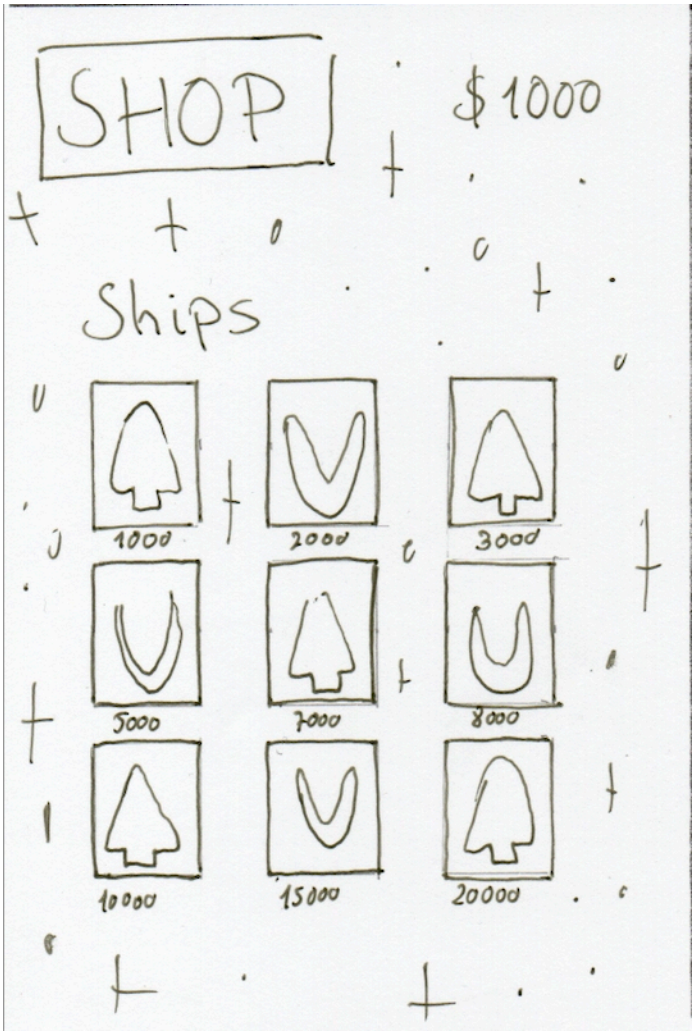


Figura 63 - Ecrã da loja

A Figura 63, é a interface da loja do jogo. Aqui, o jogador pode selecionar a nave que deseja usar em jogo, porém só pode ter essa nave à sua disposição se tiver em sua posse a quantidade de moedas que é necessária para adquiri-la. Na parte central do ecrã, existe uma grelha que mostra todas as naves disponíveis no jogo, representadas por uma mini versão gráfica delas, dentro de botões retangulares. Por baixo de cada um destes botões, está o respetivo preço da nave.

Na parte superior desta interface, verifica-se o texto identificativo do ecrã onde o jogador se encontra, com o seguinte texto, “Shop”, que segue o mesmo tipo de letra de todos os outros ecrãs.

No canto superior direito, está identificado o valor de moedas que o jogador tem em sua posse, para que possa adquirir novas naves. A cada nave adquirida, este valor deve ser retirado ao valor total representado neste ecrã.

Aqui, será dada a opção ao jogador de fazer uma chamada a *framework*, de forma a ganhar moedas para adquirir novas naves.

4.6 Música e efeitos sonoros

Neste capítulo, serão apresentados a música e os efeitos sonoros presentes neste videojogo.

Cada capítulo terá associado um tema musical diferente, que estará de acordo com o tema principal do videojogo, o espaço. Estes, serão temas que trarão um sentimento de coragem e dinamismo ao jogador. Nos ecrãs fora da ação jogo, terão uma música de fundo, leve e com um tom mais melancólico, também ligada ao tema do espaço.

O jogo, terá efeitos sonoros em várias condições, tais como: disparo por parte da nave do jogador; disparo feito pelo inimigo; na eliminação de inimigos, com sons distintos caso este seja um alienígena ou um asteroide; na "morte" do jogador; na conclusão de um capítulo; entre outros.

4.7 Momentos de uso da framework

Existem quatro momentos de interação do jogo, com a *framework* educacional.

O primeiro momento, será feito logo que o jogo é iniciado, aqui o jogador terá a primeira interação com a *framework*. Caso a resposta fornecida pela *framework* seja positiva, o jogador será recompensado com um montante predefinido de moedas de jogo.

Outros dois momentos de interação que estarão presentes, serão em casos de sucesso ou insucesso no jogo. Caso, o jogador complete um capítulo eliminando o inimigo final, vai visualizar o ecrã de fim de capítulo, no qual terá a hipótese de duplicar o montante de moedas que ganhou durante o decorrer do capítulo, através da interação com a *framework* educacional. Para isso, deve responder positivamente a um conteúdo didático, que será apresentado pela *framework*, caso isso aconteça, o valor das moedas ganhas será duplicado e adicionado ao total de moedas do jogador. Se a resposta for negativa, o valor de moedas deve-se manter o inalterável.

Em caso de insucesso durante o capítulo, ou seja, se a nave do jogador ficar sem "*power*", aparece o ecrã de fim de jogo, no qual o jogador tem a possibilidade de chamar a *framework* educacional. Esta interação, dá ao jogador a chance de restaurar o "*power*" total da nave, podendo continuar o jogo a partir do ponto em que o ecrã de fim de jogo foi exibido.

O último momento de interação entre jogo e *framework* ocorrerá na loja, onde, caso o jogador queira aumentar o valor de moedas que possui, pode fazer uma chamada à *framework.* Dependendo da resposta recebida, se for positiva, o jogador irá receber um montante de moedas que lhe permitirá adquirir as naves que tanto deseja.

5. Protótipo

Neste capítulo, será apresentado o protótipo que foi desenvolvido para mostrar uma ideia geral do que será o jogo final.

Como já foi abordado anteriormente, o motor de criação de jogos *Unity*, será utilizado para o desenvolvimento do projeto. Criado e mantido pela *Unity Technologies*, este motor foi escolhido por ser multiplataforma, permitindo a criação de videojogos para *Android*, *iOs*, *macOS*, *Windows*, entre outros.

Umas das suas principais características, é a facilidade de uso, com uma interface intuitiva e personalizável.

Outro ponto interessante é que tem uma loja de *assets* própria, que disponibiliza de vários pacotes de sprites, animações e áudios gratuitos.

Por último, mas não menos importante, o *Unity* oferece uma versão gratuita para desenvolvedores independentes ou pequenas empresas, sendo uma excelente escolha para quem está a iniciar uma carreira como criador de videojogos.

De forma, a apresentar detalhadamente todos os aspetos mais relevantes do protótipo, foram criados cinco subcapítulos: configuração do projeto; fundo dinâmico; movimentação da nave; inimigos; e por último a nave do jogador.

5.1 Configuração do projeto

Sendo este projeto um jogo para dispositivos móveis com o sistema operativo Android, é necessário ter em conta alguns cuidados especiais ao iniciar a sua criação no *Unity*.

5.1.1 Criação do projeto

Ao abrir o *Unity,* é apresentado o ecrã de projetos, onde o utilizador tem a opção de criar um projeto através do botão “New Project”.

Quando selecionado, são apresentados vários *templates* disponíveis para a criação do projeto. Entre eles, está o “2D Mobile”, que, apesar de parecer a escolha mais adequada para este projeto, não foi a escolhida. A razão para essa decisão é que essa opção instala vários pacotes pré-configurados, que podem não ser necessários, tornando o projeto mais pesado.

Por esse motivo, optou-se pelo *template* básico “2D” e, posteriormente foram feitos os ajustes necessárias para a configurar o projeto para plataformas móveis, no caso, Android.

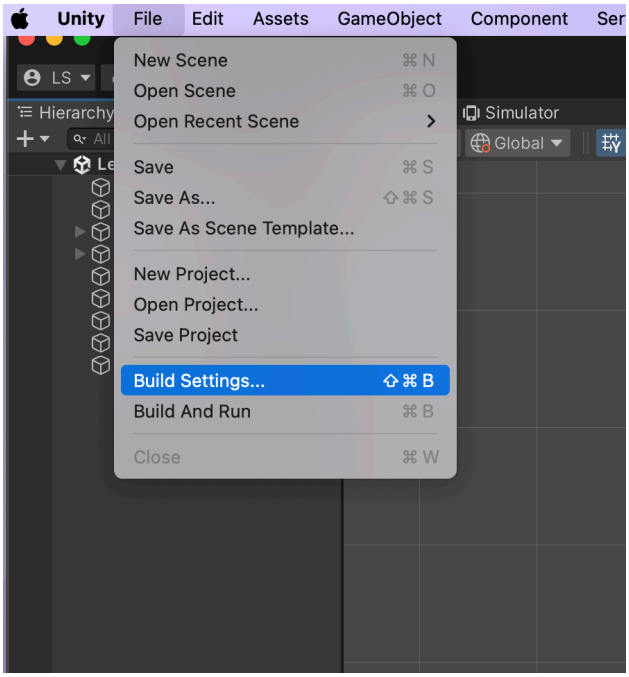


Figura 64 - Acesso a configuração de construção do projeto

A **Figura 64**, retrata o caminho necessário para aceder ao ecrã de configuração da *build* do projeto. No caso de utilizar o sistema operativo *macOS,* o utilizador deve seguir o seguinte caminho: “File > Build Settings”.

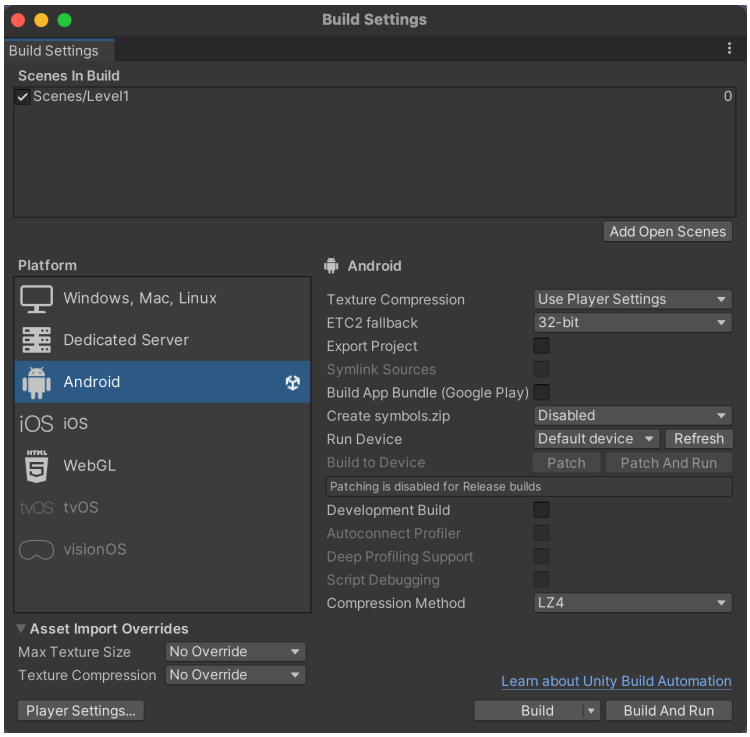


Figura 65 - Ecrã Build Settings

Na **Figura 65**, apresenta o ecrã onde é possível alterar a plataforma para a qual o projeto será desenvolvido, neste caso, a plataforma de Android. Para isso, basta selecionar “Android” na lista de plataformas e clicar no botão “Switch Platform”. O *Unity,* de forma automática, instalará todos os ficheiros e pastas necessários para que o projeto seja configurado para a plataforma selecionada. [10]

Outro menu essencial durante a criação do projeto é o menu de definições do projeto. Nele, podem ser ajustadas várias opções associadas ao jogo, como, por exemplo, forçar o modo de retrato do ecrã de jogo, este é o formato desejado para o jogo em desenvolvimento. Essa configuração é importante para garantir que o jogador utilize o dispositivo móvel na vertical, eliminando a possibilidade de alternar para o modo paisagem.

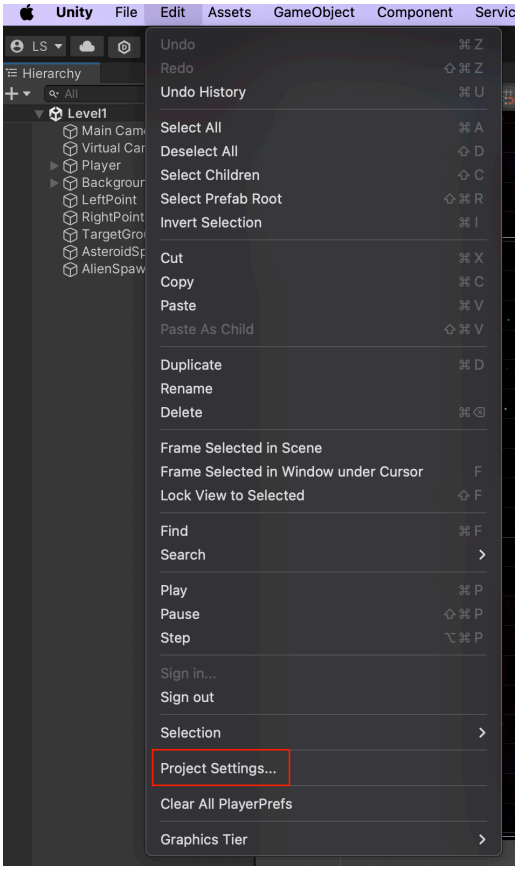


Figura 66 - Acesso as definições do projeto

A **Figura 66**, mostra como aceder ao menu de definições do projeto. Este menu será muito útil para o utilizador caso deseje fazer alterações ao projeto inicial.

O próximo passo consiste na definição de como o jogo deve ser apresentado no ecrã do dispositivo móvel. Para isso, o utilizador deve abrir o submenu "Player" e, em seguida, clicar na aba "Resolution and Presentation".

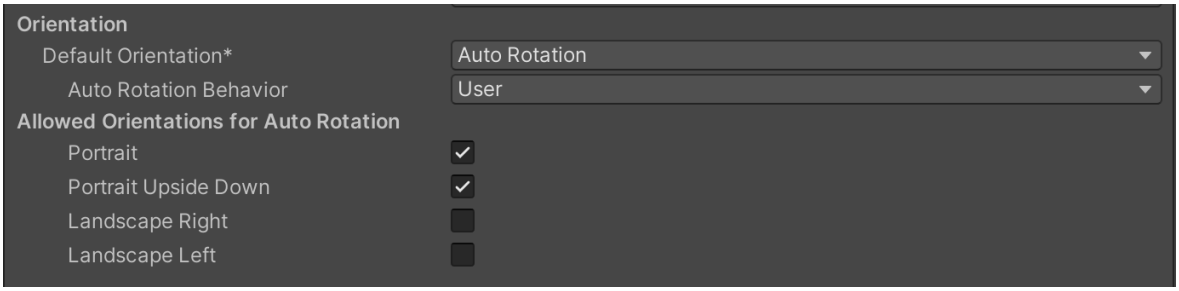


Figura 67 - Definição da orientação do ecrã de jogo

Nesse menu, o utilizador encontrará as opções representadas na ***Figura 67***. No caso do projeto "Supernova QuEST", é possível permitir a rotação automática do dispositivo móvel, porém, apenas entre duas orientações: *portrait* e *portrait upside down* [10].

5.1.2 Simulador

Por último, um fator importante nesta configuração para um projeto de videojogos para plataformas móveis é a inclusão de simuladores, permitindo testar o jogo diretamente na plataforma *Unity*.

No ecrã principal do *Unity,* encontram-se ao centro duas abas, que por predefinição, são a aba "Scene" e a aba "Game".

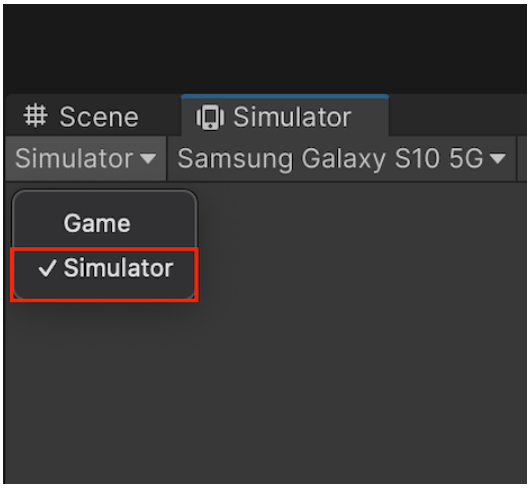


Figura 68 - Opção de simulador

A **Figura 68**, exemplifica como alterar a aba identificada como “Game” para “Simulator”. Para isso, basta selecionar a aba “Game” e, em seguida, clicar na seta à direita, em baixo da aba. Isso abrirá um submenu onde será possível selecionar a opção “Simulator”.

Neste momento, o utilizador terá acesso a vários simuladores de diferentes marcas de dispositivos móveis, permitindo testar o comportamento do jogo de forma fácil e direta.

Outra opção oferecida pelo *Unity* para testar jogos em dispositivos móveis é através da aplicação *Unity Remote* [11]. Esta estabelece uma conexão direta entre o dispositivo móvel e o jogo que está a ser desenvolvido, sem a necessidade de criar um executável.

5.2 Fundo Dinâmico

Neste subcapítulo, será descrito o processo de criação de um fundo dinâmico para o jogo, utilizando várias técnicas, como o “*parallax effect*”, amplamente utilizado nos jogos retro.

5.2.1 Fundo infinito

O primeiro passo consistiu em criar um fundo de *scroll* infinito, que transmite a impressão de que a nave está a movimentar-se no ecrã, na realidade, é o fundo que se move, proporcionando um efeito de movimento interessante ao jogo [12].

A técnica utilizada começa por copiar a imagem usada como fundo e colocá-la acima da original, estendendo assim o fundo na direção vertical.

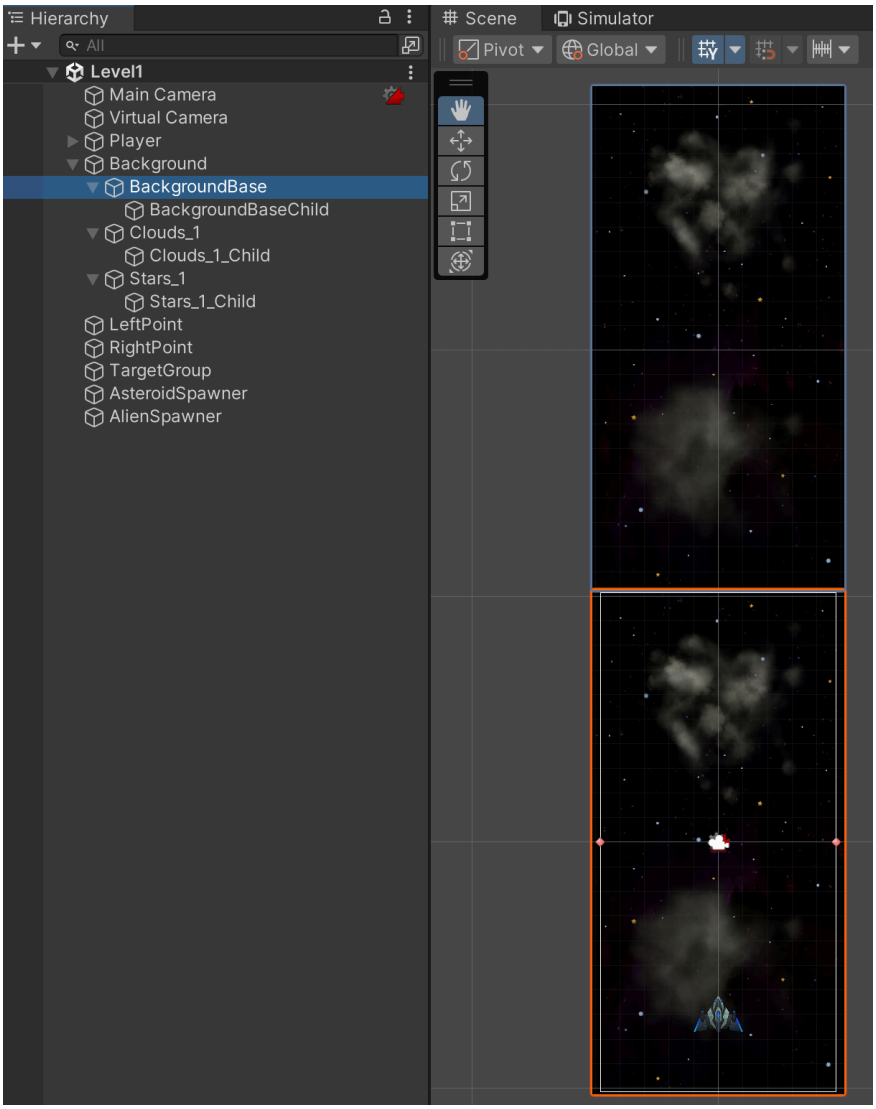


Figura 69 - Fundo dinâmico

Na **Figura 69**, é possível identificar que o fundo original demarcado pela cor laranja, foi estendido na vertical e adicionado como “filho” do fundo principal.

Após duplicar o fundo e atribuí-lo ao objeto “pai”, foi criado o script “*ParallaxEffect*”, que está disponível no Anexo I presente neste documento, intitulado de “*Script* ParallaxEffect”.

É importante ter especial cuidado na escolha das imagens a serem usadas como fundo, pois o objetivo é garantir uma transição suave e invisível aos olhos do jogador, criando uma experiência imersiva sem interrupções visuais.

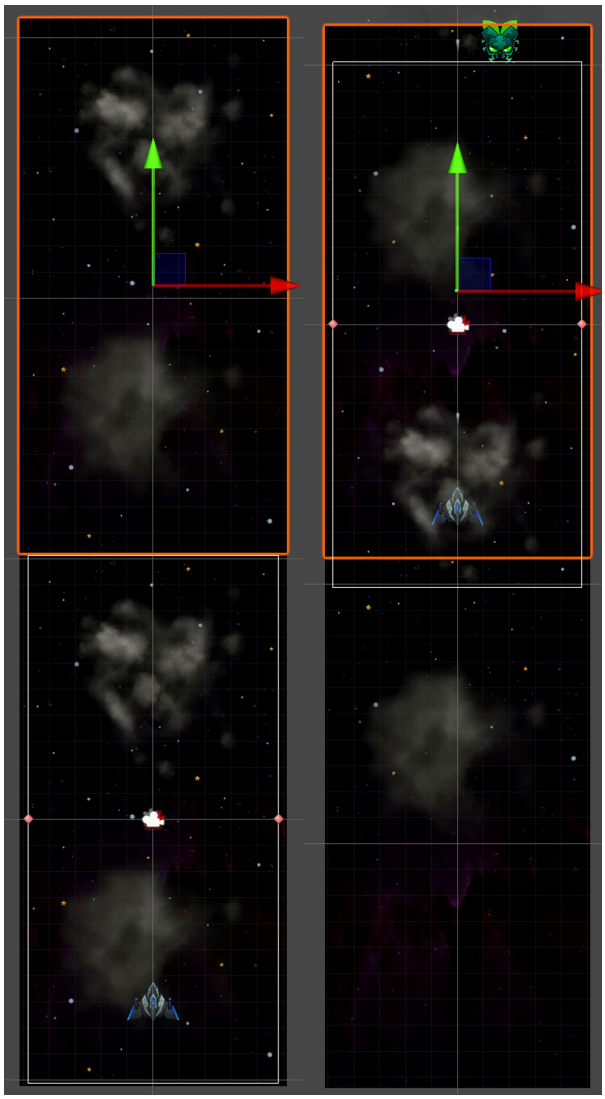


Figura 70 - Movimentação do fundo

Como se pode verificar na **Figura 70**, no lado esquerdo encontram-se as imagens de fundo na sua posição inicial. Já no lado direito da imagem, é possível observar que as imagens estão prestes a atingir o ponto limite, momento em que é feito o *reset* e o fundo retorna à posição inicial, como mostrado à esquerda.

5.2.2 Efeito Parallax

Para tornar o fundo ainda mais dinâmico, foi utilizada uma técnica chamada "Parallax Effect". Muito comum em videojogos dos anos 80, esta técnica cria a ilusão de profundidade em um ambiente 2D. Para isso, são criadas várias camadas com fundos diferentes, que se movem a velocidades distintas [13].

Figura 71 - Fundo de nuvens

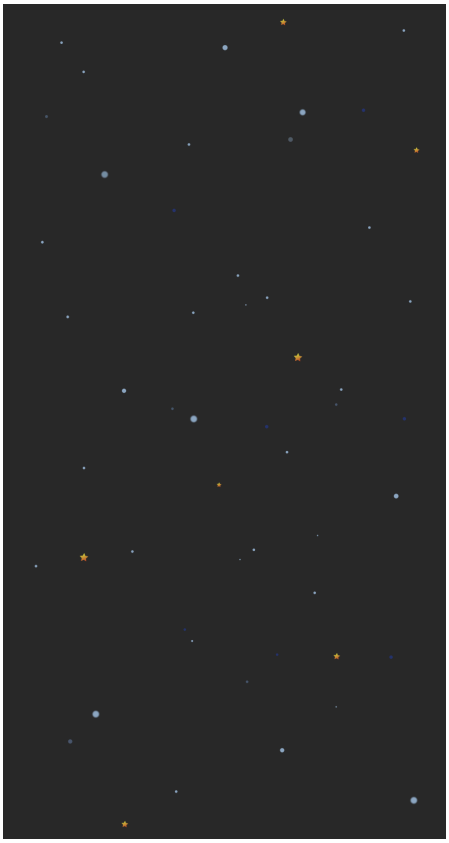


Figura 72 - Fundo de estrelas

A **Figura 71** e **72** mostram os outros fundos utilizados para criar o efeito "*Parallax*". A **Figura 71** apresenta um fundo com representações de nuvens, que irão passar por cima dos outros elementos presentes no ecrã de jogo. Já a **Figura 72** apresenta um fundo com estrelas, que será mais uma camada a contribuir para criação do efeito "*Parallax*".

Estas três camadas são organizadas em três objetos diferentes, dos quais fazem parte o objeto "pai" e objeto "filho". O script é aplicado apenas ao primeiro objeto "pai", não sendo necessário aplicá-lo ao objeto "filho", pois ele já faz parte do primeiro.

Em anexo, está a configuração de um desses objetos, que pode ser consultado no Anexo I, com o título "Configurações de um objeto aplicado no efeito Parallax*",* onde são explicadas todas as suas propriedades e detalhes.

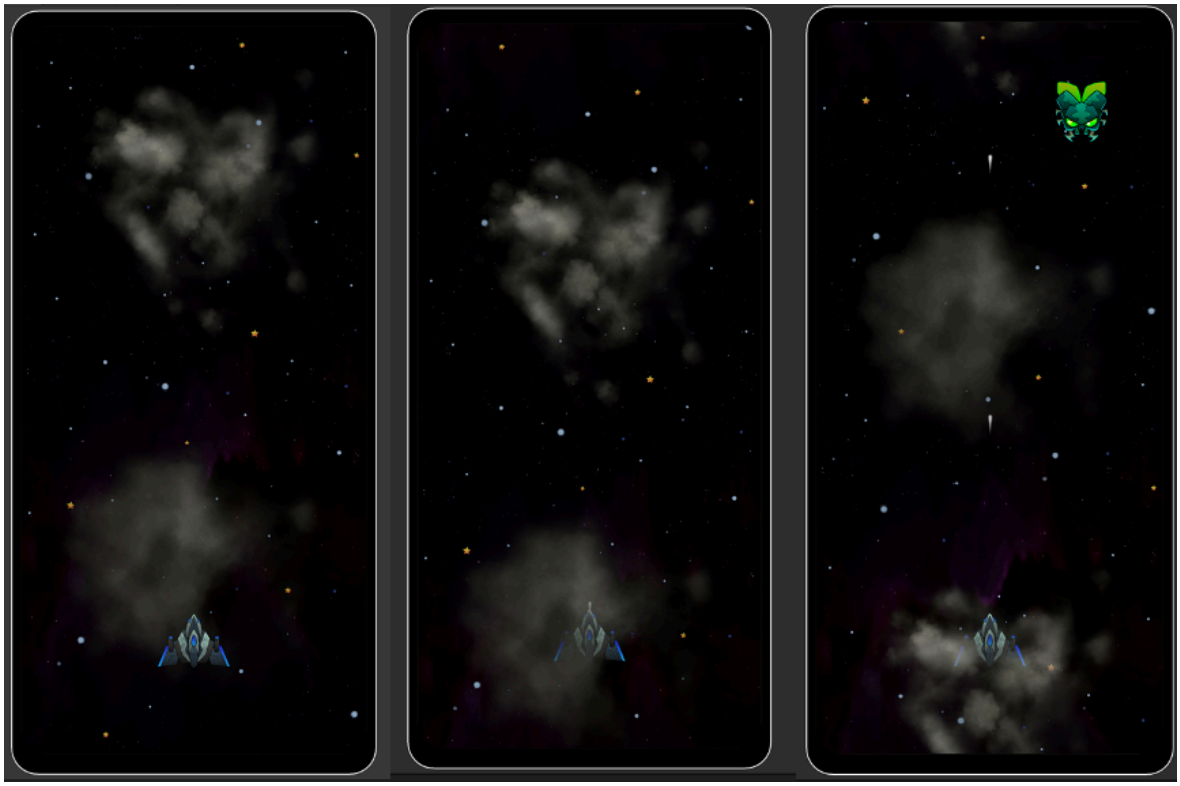


Figura 73 - Fundo dinâmico

O resultado obtido é um fundo dinâmico e divertido. Na **Figura 73**, são apresentadas três capturas de ecrã do simulador. Nessas capturas, é possível verificar que os três fundos se movimentam como desejado. As nuvens, presentes em todas as capturas, estão em movimento, e com um pouco mais de atenção, é possível observar que as estrelas se movem a uma velocidade mais lenta, criando o efeito de profundidade característico do "*Parallax*".

5.3 Movimentação da nave

Este subcapítulo tem como objetivo explicar todo o processo de desenvolvimento da movimentação da nave no jogo.

5.3.1 Input System

Após algumas pesquisas, decidiu-se utilizar o novo "*Input System"* [14], que, como o nome indica, é um sistema do *Unity* responsável por gerir *inputs* provenientes de teclados, ratos, toques e gestos. Este sistema veio substituir o antigo "*Input Manager".*

O "*Input System*" deve ser instalado através do "*Package Manager*", uma funcionalidade do *Unity* que permite a adição de novos *packages* ao projeto. No Anexo I presente neste relatório, no separador "Instalação Input System" é possível consultar o procedimento necessário para realizar essa instalação.

Assim que a instalação for concluída, o novo "*Input System"* estará disponível no simulador.

Agora, pode-se utilizar o "*EnhancedTouch"* [15], uma funcionalidade introduzida para melhorar o suporte a interações de toques nos ecrãs dos dispositivos móveis, proporcionando maior flexibilidade e precisão. Esta funcionalidade oferece informações detalhadas, como a posição exata do toque, o estado em que se encontra ("began", "moved", "ended"), a mudança de posição entre *frames*, entre outras.

Seguindo este processo, foi criada da classe responsável por mover a nave através do toque do jogador no ecrã, denominada "PlayerController". A implementação completa dessa classe pode ser encontrada no Anexo I, intitulado "Implementação da classe PlayerController".

Ao testar o código desenvolvido, observou-se que a nave é controlada conforme o esperado. No entanto, não há limitações na sua navegação, o que permite que ela saia do ecrã de jogo, causando assim um problema que tem de ser resolvido.

5.3.2 Limitações de navegação

Para implementar essa restrição, optou-se por criar quatro variáveis na classe "*PlayerController*", que irão armazenar os valores máximos nos quais a nave pode se movimentar. Estas variáveis serão usadas no método de *Update* da classe. A implementação desta atualização pode ser encontrada no Anexo I, no separador "Limitações da nave" onde todos os detalhes são explicados.

Com esta implementação, conseguiu-se limitar a movimentação da nave dentro do ecrã de jogo, garantindo que ela não saia do campo de visão do jogador.

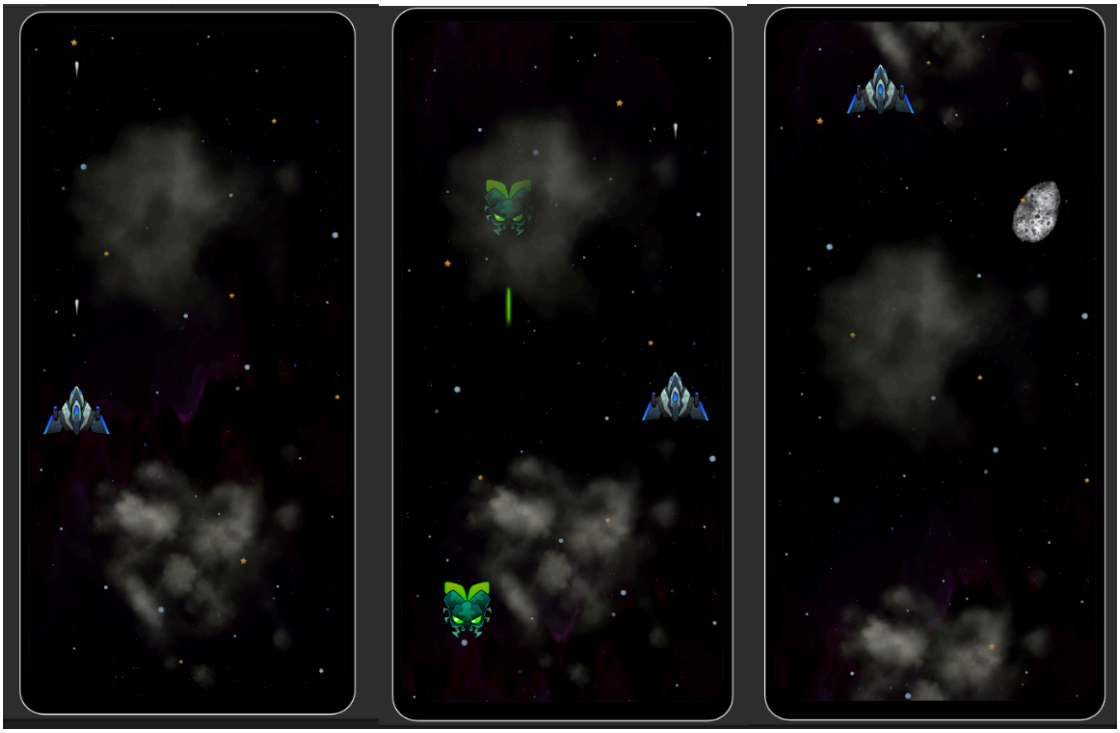


Figura 74 - Limitações de movimentação

A **Figura 74** mostra três capturas de ecrã do simulador, ilustrando três dos limites pelos quais a nave pode se movimentar. Na primeira captura, observa-se que a nave está posicionada no seu limite mais à esquerda possível. A captura do meio mostra a limitação de navegação da nave para o lado direito, alcançando a borda do dispositivo móvel. Por fim, a última captura de ecrã apresenta a limitação da movimentação da nave na parte superior do ecrã de jogo.

5.3.3 Cinemachine

A abordagem anterior foi suficiente para limitar a movimentação da nave no ecrã do jogo. No entanto, quando essa movimentação foi testada em simuladores com dimensões de ecrã diferentes, percebeu-se que, por vezes, a nave ficava restrita a um pequeno plano de movimentação.

Após se realizar uma pesquisa sobre como se adaptar a câmera do jogo a diferentes tipos de ecrãs, foi descoberto o "*Cinemachine*" [16], um sistema avançado de controlo de câmeras utilizado no *Unity.* Uma das suas principais funções é o enquadramento dinâmico da câmera, que se mostrou fundamental para resolver o problema relacionado aos diferentes tamanho e resoluções dos dispositivos móveis. O "Cinemachine" é uma extensão gratuita disponível no "*Package Manager*" da plataforma.

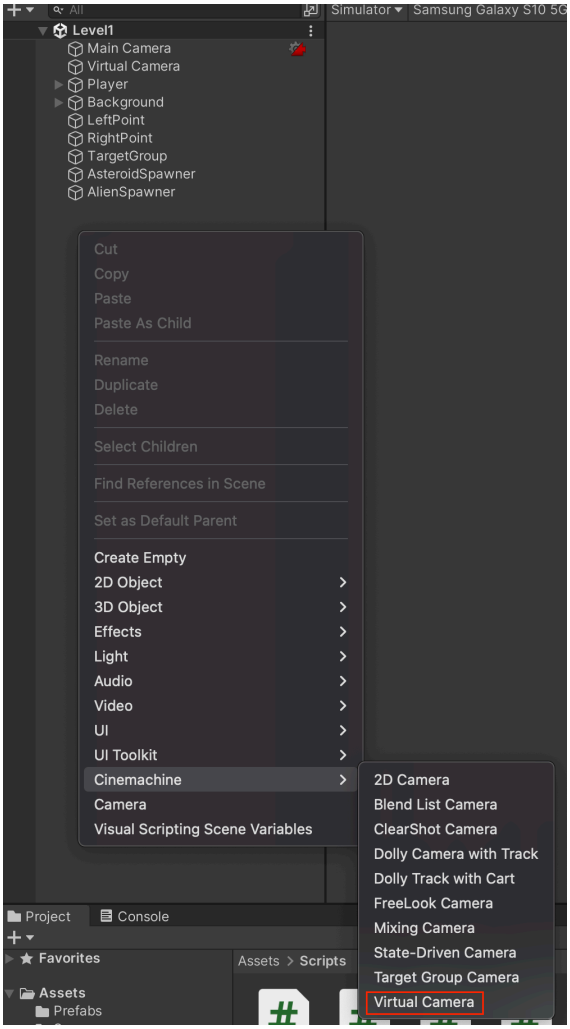


Figura 75 - Criação da câmera virtual

A **Figura 75** ensina como se pode criar uma “*Virtual Camera*” através da extensão “*Cinemachine*”. Na janela da hierarquia de objetos do *Unity*, ao clicar com o botão direito do rato, aparece o menu de criação de objetos, onde existe a opção de selecionar o “*Cinemachine*”. Após essa seleção, um submenu será exibido, permitindo clicar em “*Virtual Camera*” para criar o objeto.

Com o objeto da “*Virtual Camera*” selecionado, serão feitos alguns ajustes através da interface do “*Inspector*” no *Unity*. O primeiro passo é redefinir os valores do “*Transform” para zero e ajustar o valor de “Position*”, no eixo z para um valor negativo alto, garantindo que todos os objetos fiquem posicionados à frente da câmera virtual criada.

Figura 76 - Pontos de referência da câmera virtual

Através da criação de dois objetos, chamados "*LeftPoint*" e "*RightPoint*", aos quais é atribuído um ícone vermelho, os limites da câmera virtual foram definidos pelo posicionamento destes dois objetos, que serviram como pontos de referência. Estes objetos são posicionados nos limites mínimo e máximo do eixo do x do ecrã do jogo. Pode-se verificar esses dois pontos na **Figura 76**.

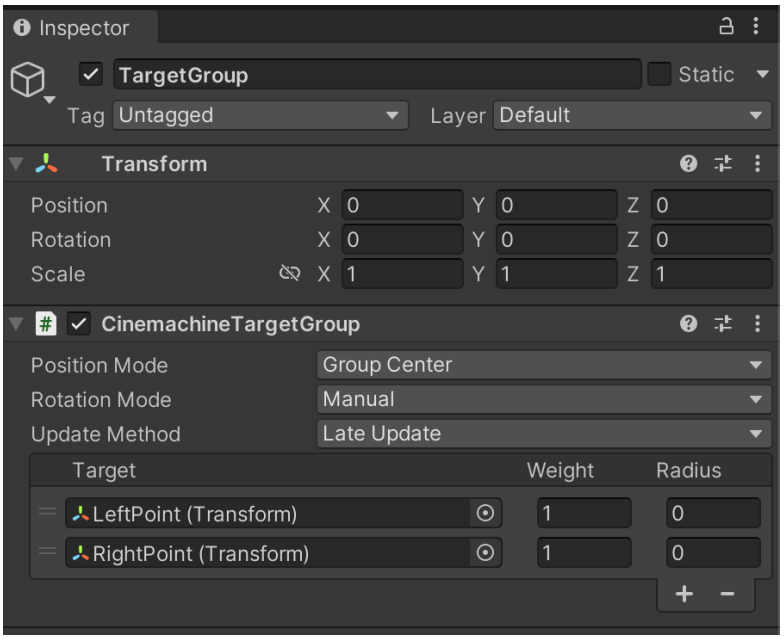


Figura 77 - Objeto TargetGroup

Na **Figura 77**, são implementadas as configurações feitas em um novo objeto chamado "*TargetGroup*", criado para atribuir o script "*CinemachineTargetGroup*". Foram atribuídos como "*Target"* os dois objetos criados anteriormente, "LeftPoint" e "RightPoint".

O "*CinemachineTargetGroup*" [17] é um componente que faz parte do "Cinemachine", permitindo agrupar vários objetos como alvos de uma câmera. As suas principais funcionalidades incluem: seguir múltiplos objetos ao mesmo tempo, ajustar dinamicamente a posição, a orientação e o zoom da câmera para manter todos os alvos visíveis.

Esta última funcionalidade foi a utilizada para resolver o problema identificado no projeto. Agora que foi criado o objeto "*TargetGroup*" [18], este está pronto para ser adicionado à câmera virtual.

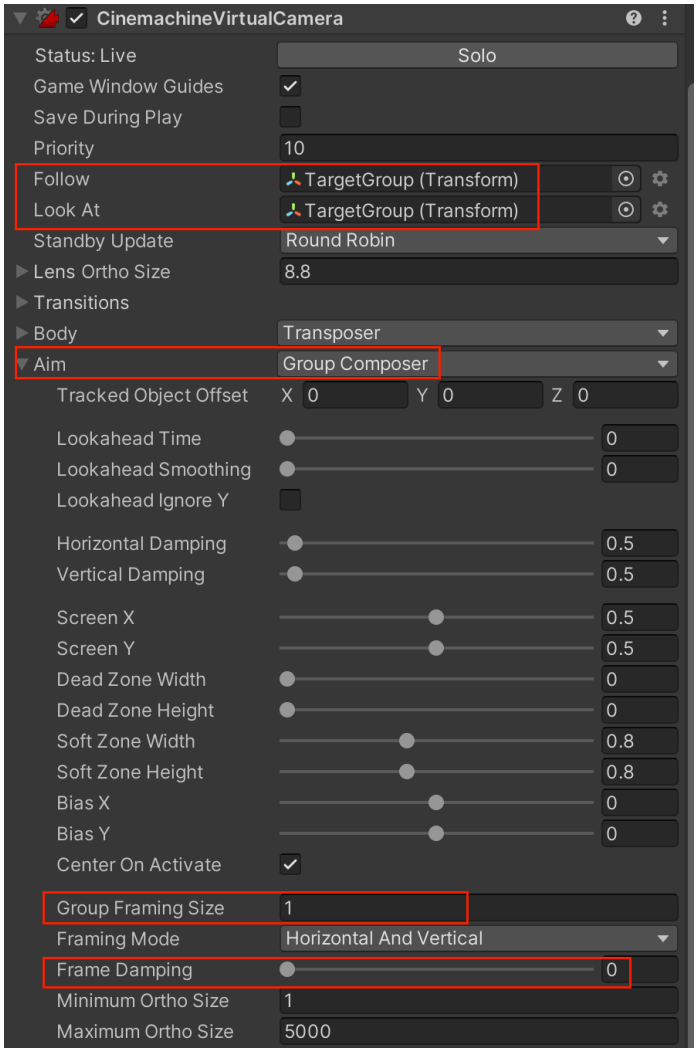


Figura 78 - Configuração da câmera virtual

A **Figura 86** apresenta a configuração final da câmera virtual. Como se pode verificar, o objeto "*TargetGroup*", foi adicionado aos campos de "*Follow*" e "*Look at*", fazendo com que este objeto seja permanentemente seguido e focado pela câmera virtual.

Em seguida, é necessário alterar o "*Aim*", de "*Composer*" para "*Group Composer*", de forma a exibir o submenu com os outros atributos relevantes a serem ajustados, como o "*Group Framing Size*" e o "*Frame Damping*".

O "*Group Framing Size"* [19] é uma funcionalidade que permite à câmera enquadrar automaticamente todos os membros de um "*Target Group"* dentro da cena. Para garantir que todos os elementos do grupo estejam visíveis na cena, o seu valor deve ser de 1. Já o "*Frame Damping*" ajusta a rapidez com que o "*Target Group*" deve ser focado. Quanto mais baixo for o valor, mais rápida será a resposta, logo o valor foi alterado para 0.

Após realizar um novo teste ao jogo para verificar se os limites da nave se ajustavam dinamicamente aos diferentes tipos de ecrãs, foi identificado um novo problema.

Na classe “*PlayerController*”, os valores máximos de movimentação da nave estavam a ser definidos na função “*Start*”, que é executada no primeiro frame do jogo. Isso não dava tempo suficiente para que a câmera virtual se adaptasse.

Este problema foi resolvido utilizando uma “*Coroutine*”, que é uma função especial no *Unity* que permite executar código em múltiplos frames, criando pausas ou atrasos sem bloquear a execução normal do jogo. As “*Coroutines*” são ideais para tarefas assíncronas, como a execução de temporizadores, ao contrário das funções normais que são executadas em um único frame.

A implementação desta “Courotine”, está presente no Anexo I, intitulado “Coroutine SetBoundaries”.

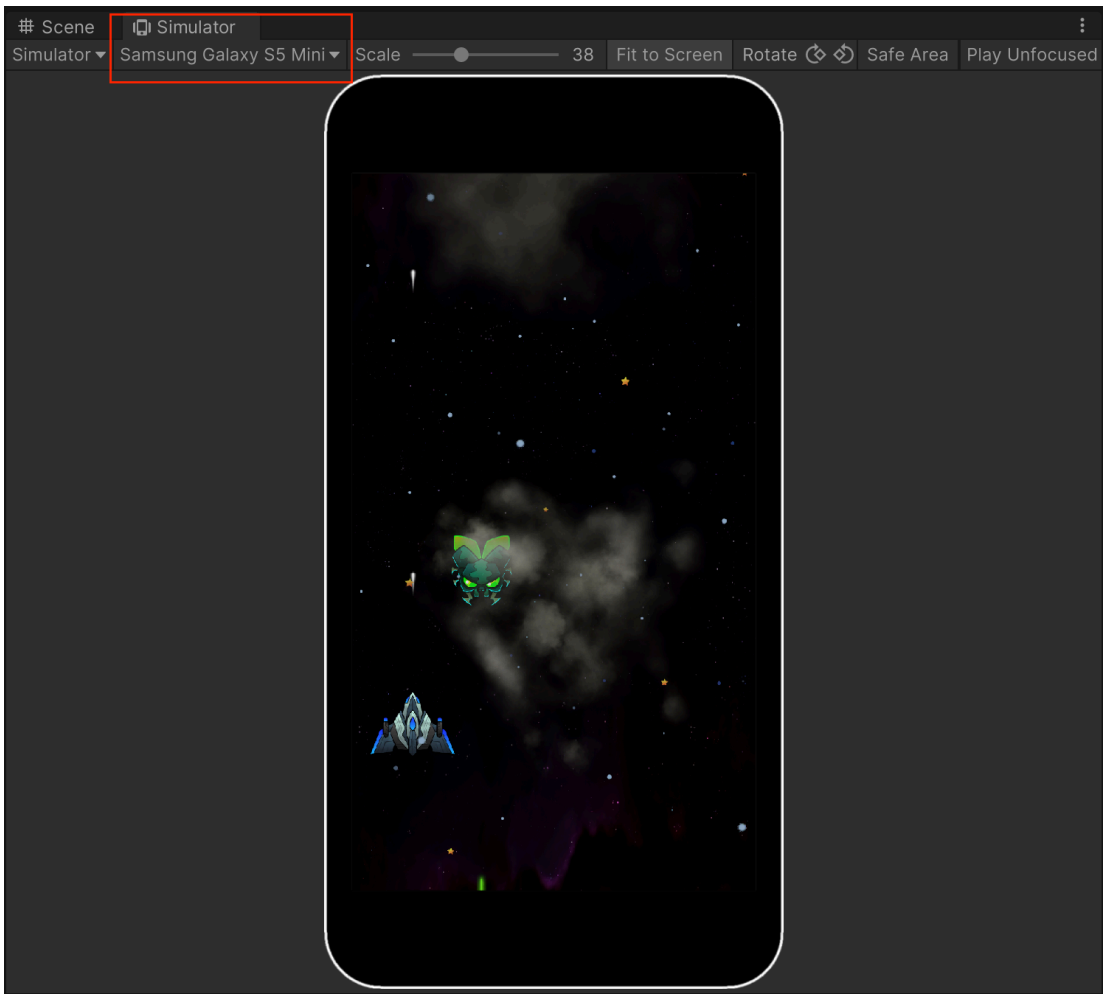


Figura 79 - Simulador Samsung Galaxy S5 Mini

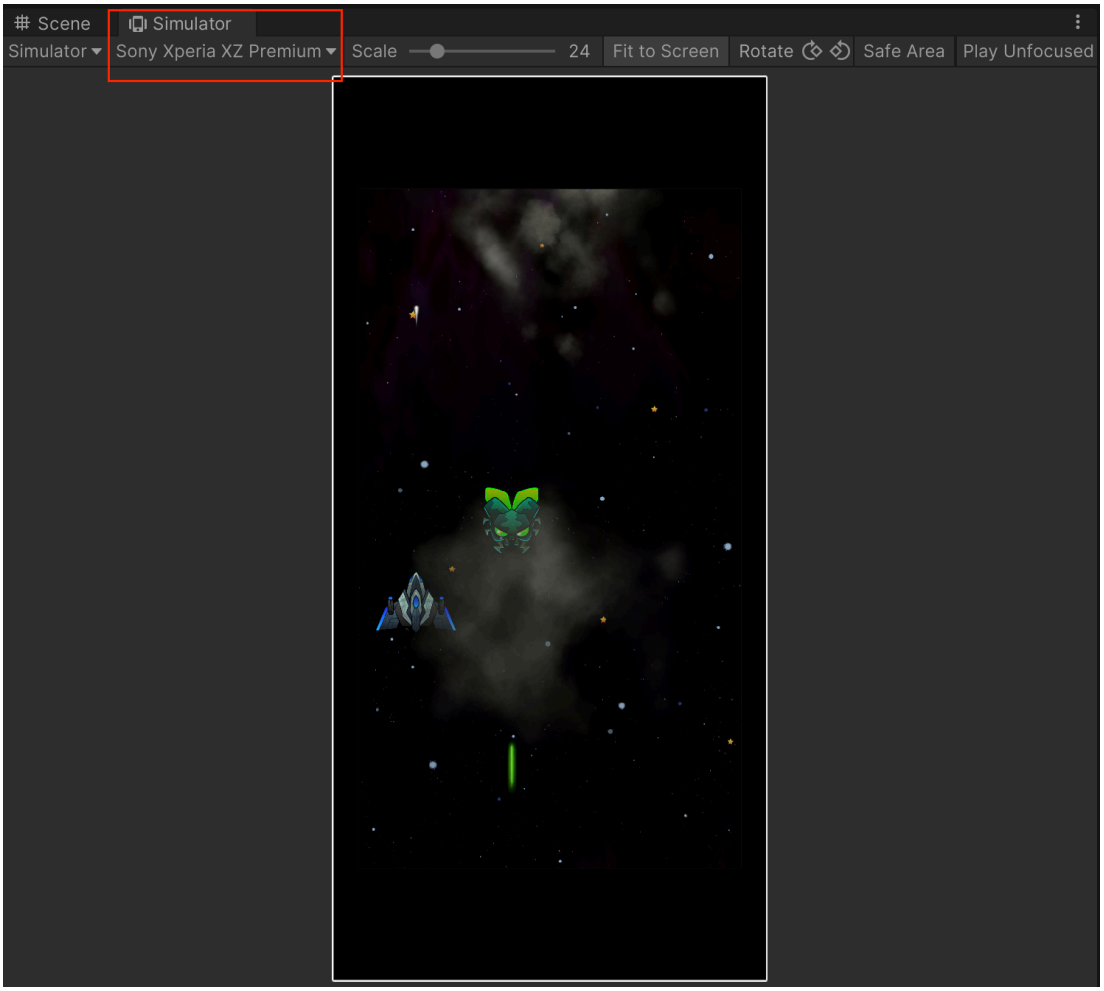


Figura 80 - Simulador Sony Xperia XZ Premium

O resultado obtido está presente na **Figura 79** e **Figura 80**, onde se pode verificar a utilização de dois simuladores com dimensões de ecrã diferentes: o simluador “Samsung Galaxy S5 Mini”, com 4,5 polegadas, e o “Sony Xperia XZ Premium”, com 5,46 polegadas.

É possível observar que a nave pode ser movida até o seu limite no lado esquerdo, nos dois simuladores com dimensões diferentes. Isso demonstra que o jogo agora se adapta automaticamente aos limites de navegação definidos, independentemente do dispositivo móvel utilizado.

Para concluir este capítulo sobre a movimentação da nave, o processo foi moroso, mas o resultado final correspondeu às expectativas. Agora, é possível jogar o jogo em qualquer tipo de dispositivo, mantendo a nave sempre dentro dos limites do ecrã.

5.4 Inimigo

Este subcapítulo tem como objetivo descrever o processo de construção dos inimigos presentes no jogo. No protótipo, serão descritos apenas dois: o “*Insetroid*” e o “*Asteroid*”.

5.4.1 Superclasse Enemy

Para a construção dos inimigos do jogo, foi utilizado um dos pilares fundamentais da programação orientada a objetos: a herança. Ela permite que a superclasse compartilhe atributos e métodos com as suas subclasses, sendo utilizada para reaproveitar código, melhorar a organização e facilitar a extensibilidade. A implementação dessa superclasse está detalhada no Anexo I, intitulado “Superclasse Enemy”.

5.4.2 Asteroid

Com a criação da superclasse “Enemy”, o próximo passo foi a criação do objeto 2D que represente o asteroide.

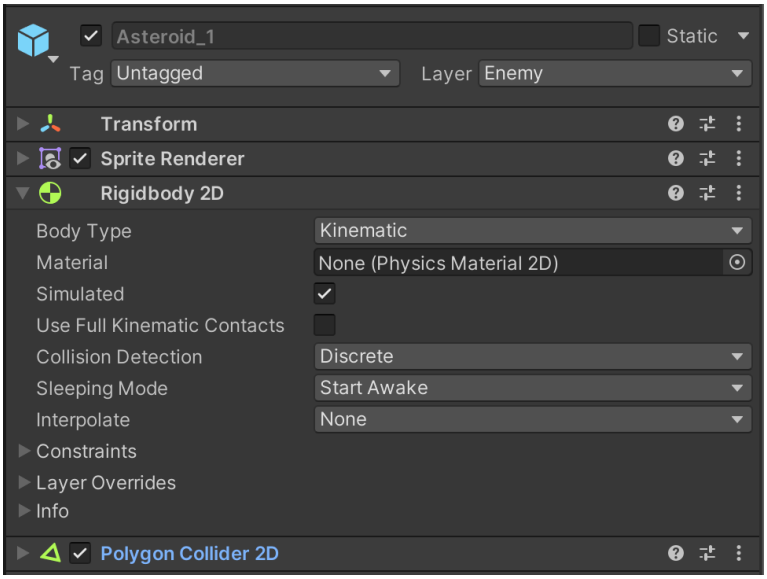


Figura 81 - Objeto 2D do Asteroide

Na **Figura 81**, é apresentada a construção do objeto 2D referente ao asteroide.

O primeiro componente adicionado a este objeto, foi o “*Sprite Renderer*”, responsável por associar uma imagem ao objeto. O segundo componente adicionado foi o “*Rigidbody 2D*” [20], que confere propriedades físicas ao objeto em um espaço bidimensional, permitindo que ele seja afetado por forças como a gravidade, colisões e atrito. Após adicionar este componente, a primeira

modificação realizada foi alterar o valor do "*Body Type*" para "*Kinematic*", garantindo que todas as movimentações do asteroide sejam controladas apenas por meio de script.

Em seguida, foi adicionado o componente "*Collider 2D*", que define a área de colisão do objeto e permite que ele interaja fisicamente com outros objetos. Para o asteroide, foi escolhido o tipo "*Polygon Collider 2D*", o qual é ideal para objetos com formas irregulares, pois adapta-se automaticamente à forma da imagem, permitindo ajustes, os quais foram necessários no caso em específico.

Após a construção do objeto, iniciou-se a implementação da subclasse "*Asteroid*", que é derivada da classe "*Enemy*". O código dessa classe, juntamente com a explicação detalhada, pode ser consultado no Anexo I, intitulado "Subclasse Asteroid."

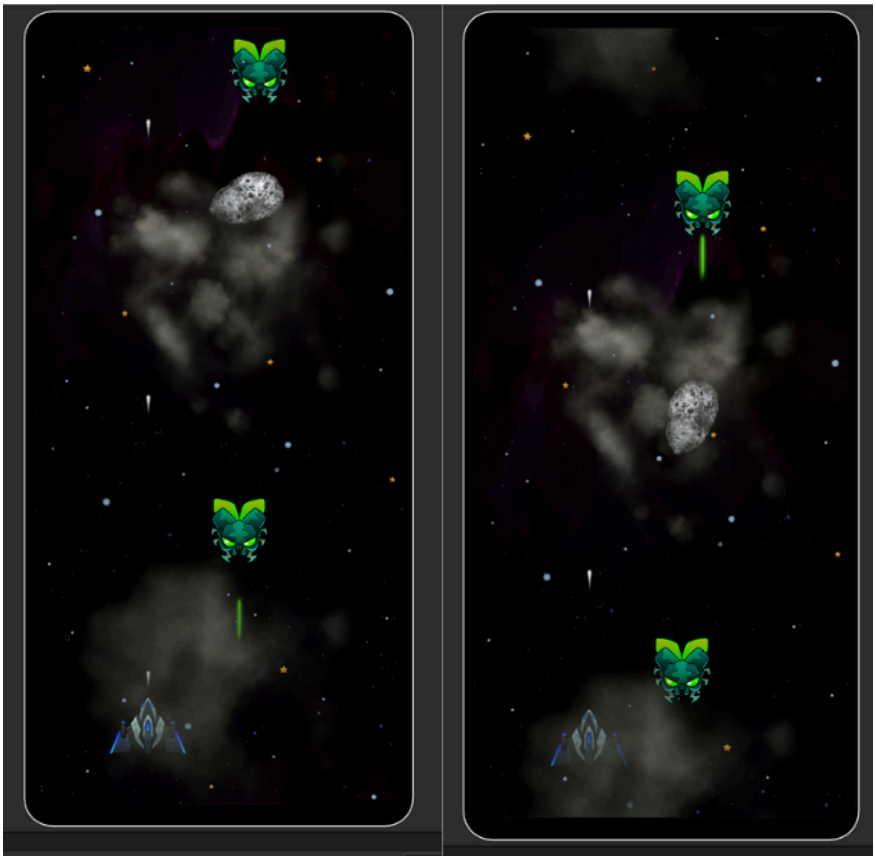


Figura 82 - Asteroides

Na **Figura 82** são demonstradas duas capturas do simulador, onde é possível identificar o objeto de jogo "Asteroid" e um dos seus atributos que é a rotação. Verifica-se que, na imagem à direita, o asteroide está em uma posição vertical, enquanto na imagem à esquerda já esta encontra numa posição horizontal, comprovando que ocorreu uma rotação.

Considera-se que o resultado final ficou bastante interessante, devido ao dinamismo atribuído a este objeto.

5.4.3 Asteroid Spawner

Após a implementação da classe "*Asteroid*", procedeu-se à criação de uma classe geradora de asteroides, chamada "*AsteroidSpawner*". A implementação desta classe pode ser encontrada no Anexo I, com o título "Classe AsteroidSpawner".

Concluída a implementação, esta classe foi atribuída ao objeto de jogo através do componente "*Script*" do *Unity*.

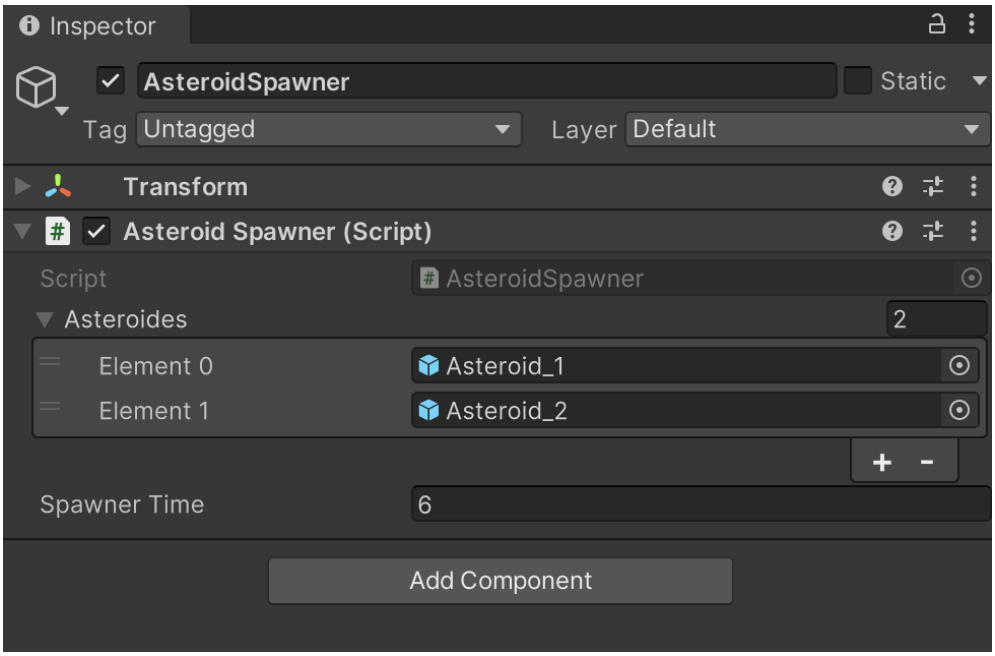


Figura 83 - GameObject AsteroidSpawner

Na **Figura 83** é representado o "*Inspector*" do objeto de jogo "*AsteroidSpawner*". São observáveis os atributos públicos que recebem os valores necessários ao funcionamento do script, como o "*Spawner Time*", definido com o valor de 6 segundos, e o *array "Asteroids",* no qual se podem adicionar os objetos desejados, neste caso, dois "*prefabs"* de asteroides que o jogo utilizará.

A palavra "*prefab*" [21], mencionada anteriormente, carece de uma explicação a respeito do que é e o do que faz. O "*prefab*" é um tipo especial de objeto do *Unity*, que serve como modelo reutilizável para a criação de objetos durante a execução do jogo. Normalmente, é criada uma pasta chamada "Prefabs", onde se guardam este tipo de objetos.

A forma de como os "Insetroids" são criados no jogo é muito semelhante à dos asteroides. Existe a classe "*AlienSpawner*", que neste momento recebe apenas um objeto, ao contrário do *array* de objetos presente na "*AsteroidSpawner*". Além disso, não é dada qualquer rotação ao objeto quando criado, pois queremos manter os "Insetroids" com a sua disposição inicial.

No futuro, devido a estas semelhanças entre as classes, poderá ser implementado um *design pattern* chamado "*Factory*", com o objetivo de centralizar a criação dos inimigos em uma única classe.

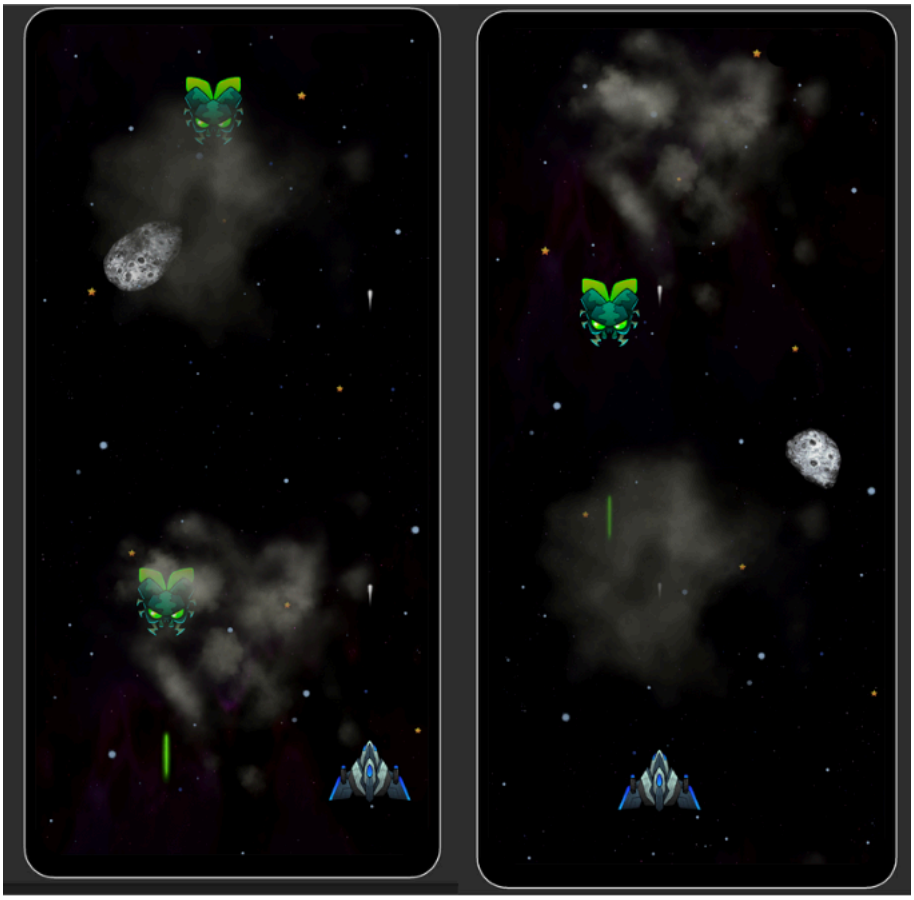


Figura 84 - Geração de Asteroides

A **Figura 84** mostra o processo da geração dos asteroides no jogo. Como é possivel ver, estes podem surgir em posições diferentes e ser de vários tipos e tamanhos. Na figura à esquerda, é possível observar um asteroide com dimensoes avantajadas e com um tipo de *sprite* gerado numa posição mais à esquerda. Já na imagem à direta, verifica-se que o asteroide gerado tem um sprite diferente do anterior e as suas dimensões são mais reduzidas. Quanto a sua posição, é claramente diferente, pois surge numa posição à direita do ecrã.

Este era o resultado procurado para a implementação da geração de asteroides no jogo. E desta forma pode-se concluir que a implementação foi realizada com sucesso.

5.4.4 Disparo do Insetroid

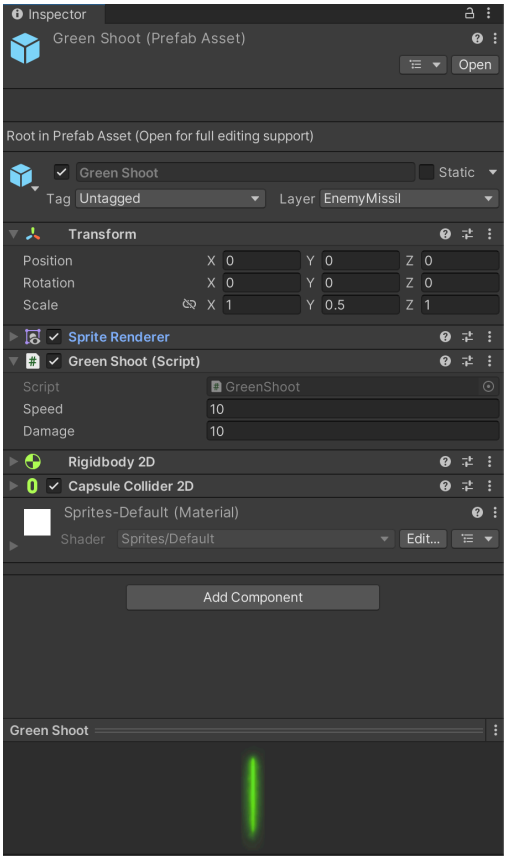


Figura 85 - Prefab Green Shoot

A **Figura 85** mostra o "*Prefab*" do tiro feito pelo alienígena "*Insetroid*". Foram introduzidos quatro componentes: o "Sprite Render", para associar a imagem ao objeto; o script "AlienShoot", que será explicado mais à frente; o "*RigidBody 2D*"; e por fim, o "*Capsule Collider 2D*". Este foi o tipo de "*collider*" escolhido, por ser o que mais se ajusta à forma do objeto.

A partir deste "*prefab*", foram criados outros para os diferentes tipos de tiro que os inimigos irão utilizar. Apenas foi necessário alterar a cor do objeto, e no script, ajustar os valores de "*speed*" e "*damage*" de acordo com o que foi definido anteriormente.

De seguida, serão apresentados alguns pontos comuns entre scripts, tendo como o exemplo a implementação do script "AlienShoot", presente no Anexo I, com o título "Script AlienShoot".

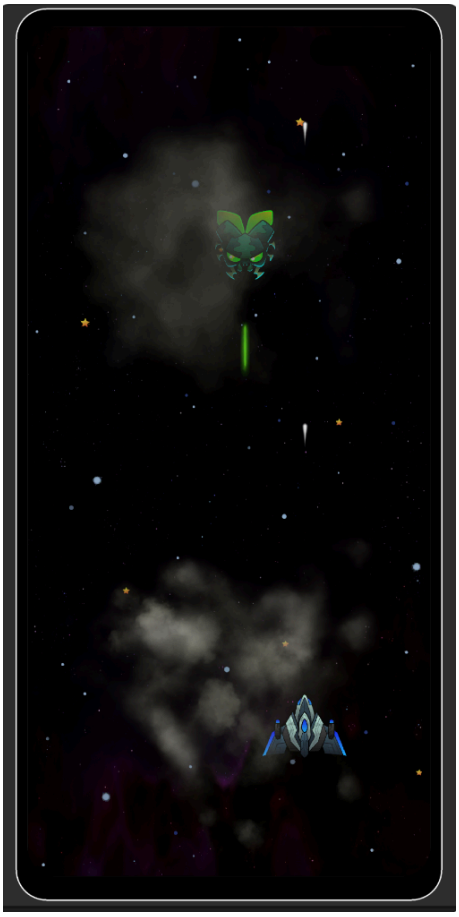


Figura 86 - Disparo do Insetroid

A **Figura 86** demostra a criação do disparo do alienígena. É possível verificar que este é produzido a partir do objeto “Insetroid” e segue uma trajetória descendente em direção à nave do jogador.

A escolha da dimensão e da cor revelou-se acertada, encaixando-se harmoniosamente no cenário do jogo. Assim, pode-se concluir que este objeto foi implementado com sucesso, sem quaisquer problemas significativos.

5.4.1 Insetroid

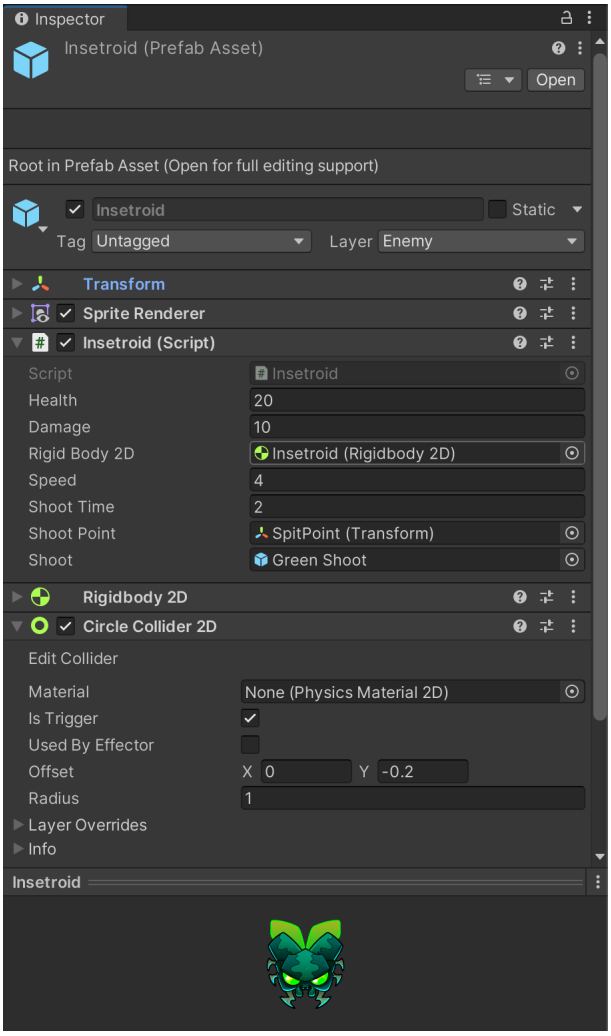


Figura 87 - Prefab Insetroid

Na **Figura 87**, observa-se o aspeto final da criação do “prefab” do "Insetroid”.

Este contém o componente script “Insetroid”, que disponibiliza ao programador vários atributos configuráveis, como, por exemplo, a vida do alienígena, definida no atributo “*Health*”; o nível de dano causado ao embater com o jogador, no atributo “*Damage*”, a velocidade de deslocação, definida em “Speed”; o intervalo de tempo entre disparos, atribuído em “*Shoot Time*”; o ponto de origem dos disparos, determinado por “*Shoot Point*”; e, por fim, o tipo de tiro utilizado pelo alienígena, especificada no atributo “*Shoot*”.

Este script será descrito pormenorizadamente nos Anexo I deste documento, intitulado “Script Insetroid”.

5.5 Nave

Neste subcapítulo, serão explicadas duas características do funcionamento da nave do jogador. A primeira diz respeito ao sistema de dano e às estatísticas da nave, enquanto a segunda aborda o modo como a nave efetuará disparos para eliminar os inimigos.

5.5.1 Sistema de dano e estatísticas

Para a gestão da vida e das estatísticas do jogador, foi criada uma classe específica, posteriormente associada ao objeto de jogo "*Player*". A implementação desta classe está presente no Anexo I, com o título de "Classe PlayerStats".

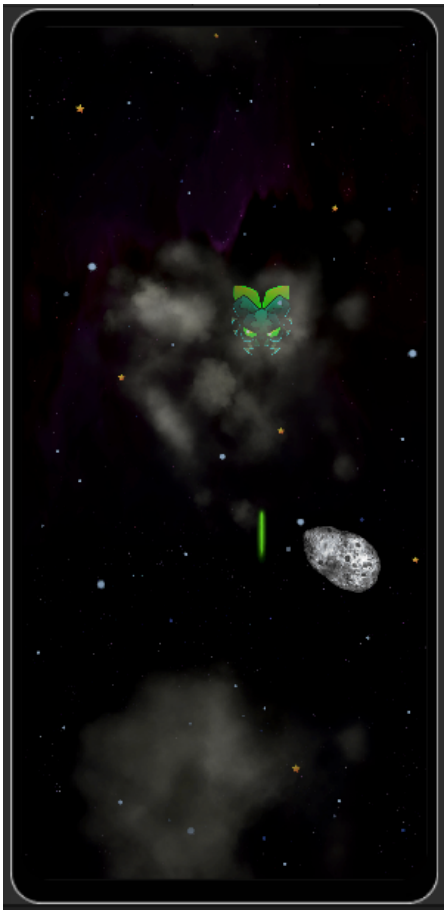


Figura 88 - Nave sem power

A **Figura 88** mostra uma captura onde se verifica que a nave foi atingida várias vezes, ficando sem "power" e desaparecendo assim do ambiente de jogo.

A implementação desta classe ainda não se encontra completa, talvez por esse motivo não tenha gerado grandes problemas na sua concretização.

5.5.2 Disparo da nave

A nave do jogador, conforme descrito anteriormente, realiza disparos de forma automática em intervalos de tempo ajustáveis, os quais variam de acordo com o tipo de nave utilizada.

A implementação detalhada dessa funcionalidade pode ser encontrada no Anexo I, no capítulo intitulado “Classe PlayerShooting”.

Figura 89 - Disparos da nave

O resultado dessa implementação está apresentado na **Figura 89**.

É possível observar três disparos efetuados pela nave no ecrã de jogo, destacados dentro de três círculos vermelhos A nave em questão, denominada “Azure Horizon”, possui um projétil com a cor branca associado ao seu disparo.

Não foram identificados problemas na implementação desse sistema de disparo para a nave, obtendo-se o resultado final conforme o idealizado.

5.6 Matriz de Colisão

Uma configuração descoberta durante as pesquisas para a implementação do jogo, quase que por acaso, foi a “Collision Matrix”, do Unity. [22]

Essa configuração é utilizada para controlar quais “*layers*” podem colidir entre si. Por exemplo, ela permite que os inimigos e o jogador interajam fisicamente, ao mesmo tempo em que ignora colisões entre inimigos, evitando o chamado “*friendly fire*”.

A grande vantagem dessa configuração é redução o custo de processamento, pois evita a deteção de colisões desnecessárias.

No entanto, ela possui limitações, pois não identifica objetos individuais, apenas interações entre “*layers*”.

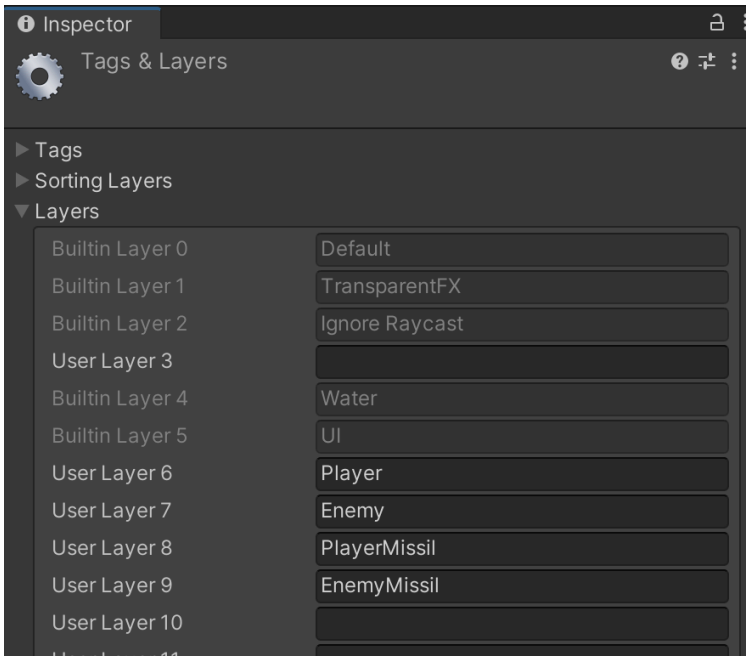


Figura 90 - Criação das Layers

O primeiro passo para a implementação desta configuração foi a criação das *layers no menu de “Tags & Layers”* [23], como se pode verificar na **Figura 90**. A primeira layer a ser criada foi a “*Player*”, associada à nave do jogador. A segunda layer, “*Enemy*”, foi associada a todas as classes que representam os inimigos. Por fim, foram criadas as *layers* associadas aos disparos feitos pelo jogador e pelos inimigos: “*PlayerMissil*” e “*EnemyMissil*”, respetivamente.

O segundo passo consiste em aceder à interface da “*Layer Collision Matrix*”. Para tal, deve seguir o caminho “*Edit* > *Project Settings* > *Physics 2D*”.

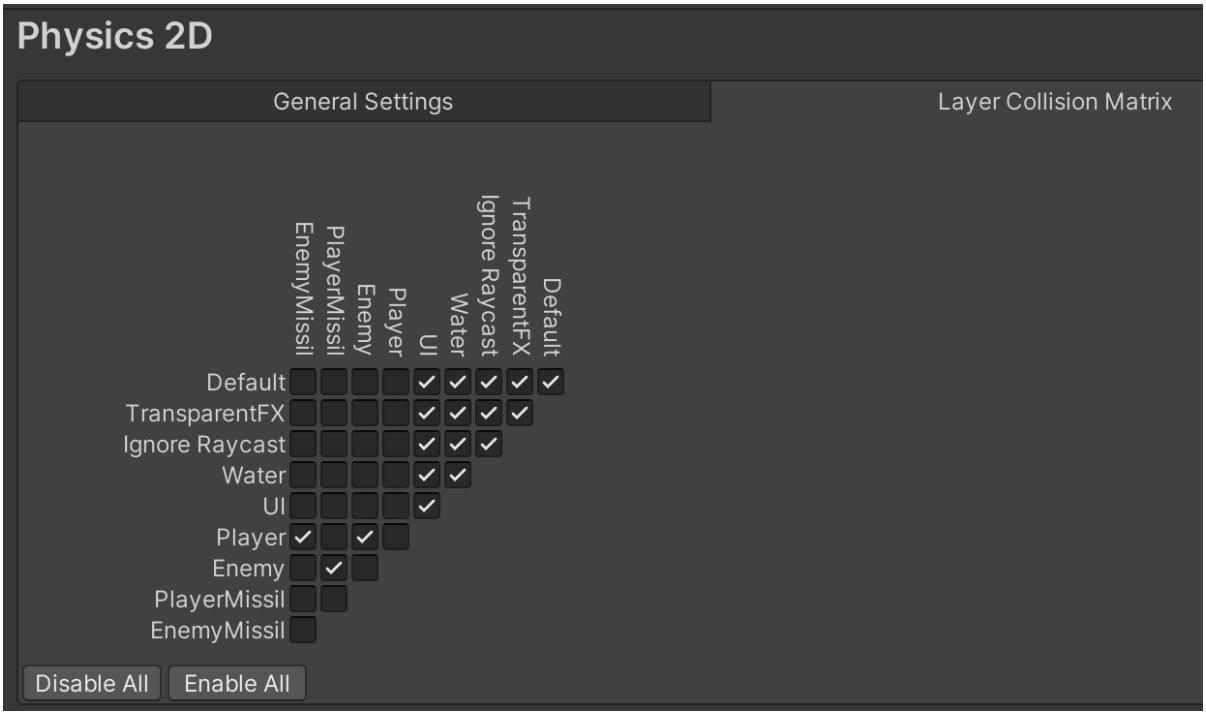


Figura 91 - Interface Physics 2D

Aberta a interface "*Physics 2D*", encontra-se a aba "*Layer Collison Matrix*", conforme ilustrada na **Figura 91**, onde é possível definir quais as *layers* que irão interagir entre si. Por defeito, todas as interações estão selecionadas, sendo necessário desmarcar as interações que não são necessárias.

Por exemplo, na **Figura 91**, pode-se observar que a layer "Player" terá interações apenas com a *layer*" EnemyMissil" e "*Enemy*". Já a *layer* "*Enemy*" detetará colisões apenas com a *layer* "*PlayerMissil*".

Este recurso é extremamente útil, especialmente no contexto do escalonamento das colisões no jogo.

6. Conclusão e trabalho futuro

Este capítulo, dedica-se a apresentar os principais contributos documentados neste relatório e os objetivos alcançados.

É possível através deste documento, ter uma perspetiva geral de como os jogos de tiro espacial são desenvolvidos, tanto por grandes empresas, como por programadores independentes, através de uma análise ao estado da arte de quatro jogos do mesmo gênero do jogo a ser desenvolvido. Foi interessante perceber, que muitos deles utilizam as mesmas técnicas de desenvolvimento, a jogabilidade é bastante parecida, a utilização de anúncios para rentabilizar o jogo é idêntica, entre outros. As maiores diferenças encontradas são os *assets* utilizados, as animações diferem bastante, e um facto curioso, é que os jogos criados por grandes companhias são os que têm mais poluição visual nas suas interfaces, ao invés do jogo criado pelo desenvolvedor independente.

Foi verificado, que a documentação do jogo através do *GDD,* é fundamental. Todos os pontos abordados nesse documento são essenciais para perceber o contexto do jogo e tornar clara e objetiva a implementação do que era apenas uma ideia, é um documento altamente técnico, no qual não devem surgir obstáculos nem dúvidas na hora da implementação do videojogo.

Na construção do protótipo do jogo, foi possível enfrentar os primeiros desafios técnicos. Sendo este projeto direcionado a construção de um jogo para dispositivos móveis, estes requerem especial atenção. A primeira dificuldade enfrentada foi a navegação da nave, tendo em conta que o jogo pode ser jogado em diferentes dispositivos com resoluções de ecrã diversas, o que requereu alguma pesquisa e vários testes. De seguida, foram aplicados conhecimentos de programação orientada a objetos, tendo em conta o contexto do *Unity*. Foi desafiante desenvolver estes conceitos de novo e mostrar como podem ser aplicados ao mundo dos videojogos, aplicando as boas práticas de desenvolvimento de software. Este desenvolvimento, como já foi referido anteriormente, necessitou de várias pesquisas e foi aplicado o método de tentativa e erro ao que diz respeito à implementação. Este método comprovou ter resultados satisfatórios no que diz respeito a absorção de conhecimentos.

Tendo em conta todo o trabalho desenvolvido neste documento, o projeto encontra-se no caminho certo, contando já com bases sólidas para a implementação e conclusão do mesmo. Porém existe trabalho a ser feito, desde a inclusão de animações e sons, a criação de todos os elementos ativos do jogo e as interfaces. Resumidamente, o objetivo é ter o videojogo concluído e o executável criado, para que os testes finais sejam realizados num dispositivo móvel Android. Outro ponto que será um fator importante, é a integração com a framework “GamEducation”, a qual se espera que seja bem-sucedida.

7. Referências

[1] The TreeTop, "*Average Human Attention Span By Age: 31 Statistics*", July 17, 2024. Accessed: Jan, 2025. [Online]. Available: https://www.thetreetop.com/statistics/average-human-attention-span

[2] J. Dias and L. Mateus, "*GamEducation Projeto 1*", ESTCB, Castelo Branco, Portugal, 2024.

[3] App Store Preview, "*Galaxiga - Classic 80s Arcade*", Accessed: Dec, 2024. [Online]. Available: https://apps.apple.com/us/app/galaxiga-classic-80s-arcade/id1519367184

[4] App Store Preview, "*Galaxy Attack: Space Shooter*", Accessed: Dec, 2024. [Online]. Available: https://apps.apple.com/us/app/galaxy-attack-space-shooter/id1225548580

[5] App Store Preview, "*RETRO: Space*", Accessed: Dec, 2024. [Online]. Available: https://apps.apple.com/us/app/retro-space/id1376056227

[6] App Store Preview, "*Retro Space 3D*", Accessed: Dec, 2024. [Online]. Available: https://apps.apple.com/us/app/retro-space-3d/id6502924583

[7] P. Neves, "*GDD – abordagem de Tom Sloper"*, ESTCB, Castelo Branco, Portugal, 2016"

[8] Asset Store, "*2D Space Kit*", Accessed: Nov, 2024. [Online]. Available: https://assetstore.unity.com/packages/2d/environments/2d-space-kit-27662#content

[9] Asset Store, "*Free Stylized 2D Space Shooter Pack*", Accessed: Nov, 2024. [Online]. Available: https://assetstore.unity.com/packages/2d/free-stylized-2d-space-shooter-pack-245185#content

[10] Blackthornprod, "Making an IOS/Android game in Unity – Beginner Tutorial - #1", Accessed: Out, 2024. [Online]. Available: https://www.youtube.com/watch?v=CGleQZVgdN4&t=370s

[11] Unity Documentation, "Unity Remote", 2025. Accessed: Jan, 2025. [Online]. Available: https://docs.unity3d.com/6000.1/Documentation/Manual/UnityRemote5.html

[12] Adamant Algorithm, "Unity Beginners - How to Loop the Background Image", Accessed: Out, 2024. [Online]. Available: https://www.youtube.com/watch?v=U72trwZ7AT8&t=78s

[13] RenderHub, "Depth in Motion: The Parallax Effect in Games", Nov 28, 2023. Accessed: Oct, 2024. [Online]. Available: https://www.renderhub.com/blog/depth-in-motion-the-parallax-effect-in-games

[14] Unity Manual, “Input System”, 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.12/manual/index.html

[15] Unity Manual, “Class EnhancedTouchSupport”, 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/api/UnityEngine.InputSystem.EnhancedTouch.EnhancedTouchSupport.html

[16] Unity Manual, “Cinemachine package”, 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/index.html

[17] Medium, Kenny McLachlan, “*Cinemachine Target Group*”, July 12, 2023. Accessed: Out, 2024. [Online]. Available: https://medium.com/@kennethmclachlan11/cinemachine-target-group-1c66a42a1a3f

[18] Unity Manual, “Cinemachine Target Group”, 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.cinemachine@2.3/manual/CinemachineTargetGroup.html

[19] Unity Manual, “Group Framing”, 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.cinemachine@3.1/manual/CinemachineGroupFraming.html

[20] Unity Manual, “Rigidbody 2D”, 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/Manual/2d-physics/rigidbody/rigidbody-2d-landing.html

[21] Unity Documentation, “Introduction to instantiating prefabs”, 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/Manual/instantiating-prefabs-intro.html

[22] Electronics Arduino Esp, “Physics Layer Collision Matrix in Unity”, Accessed: Nov, 2024. [Online]. Available: https://www.youtube.com/watch?v=KrnfWezKh2k

[23] Unity Documentation, “Layers”, 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/Manual/Layers.html

Anexo I – Detalhes de implementação

Script ParallaxEffect

```
using UnityEngine;

0 references
public class ParallaxEffect : MonoBehaviour
{
    1 reference
    [SerializeField] float speed;
    2 references
    private float backgroundHeight;
    3 references
    private Vector3 startPosition;

    0 references
    void Start()
    {
        startPosition = transform.position;
        backgroundHeight = GetComponent<SpriteRenderer>().bounds.size.y;
    }

    0 references
    void Update()
    {
        // Vector3.down -> Vector3(0, -1, 0)
        // speed -> 5
        // Time.deltaTime -> calculated based on time, not frames
        transform.Translate(Vector3.down * speed * Time.deltaTime);

        // transform.position = value will decreasing with the upddate() call
        // startPosition.y = 0
        // backgroundHeight = 20.48
        // startPosition.y - backgroundHeight = -20.48
        if (transform.position.y < startPosition.y - backgroundHeight)
        {
            transform.position = startPosition;
        }
    }
}
```

A figura mostra a implementação do script "*ParallaxEffect*". Este contém três variáveis: a "speed", que controla a velocidade a que o fundo desce no ecrã do jogador. Esta variável é definida diretamente pelo programador no editor do *Unity*. Para que tal seja possível, foi atribuído o atributo "*SerializeField*" a essa variável.

A segunda variável é a "backgroundHeight", que armazena a altura do fundo, a qual será utilizada para o efeito desejado. Por fim, a variável "startPosition", que é do tipo "Vector3"[1], uma estrutura que armazena posições de x, y e z em um vetor. A utilização do "Vector2" poderia parecer a opção mais óbvia, por se tratar de um jogo 2D. Porém, a escolha recaiu sobre "Vector3", o que se justifica por uma razão

[1] Unity Documentation, "Vector3", 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Vector3.html

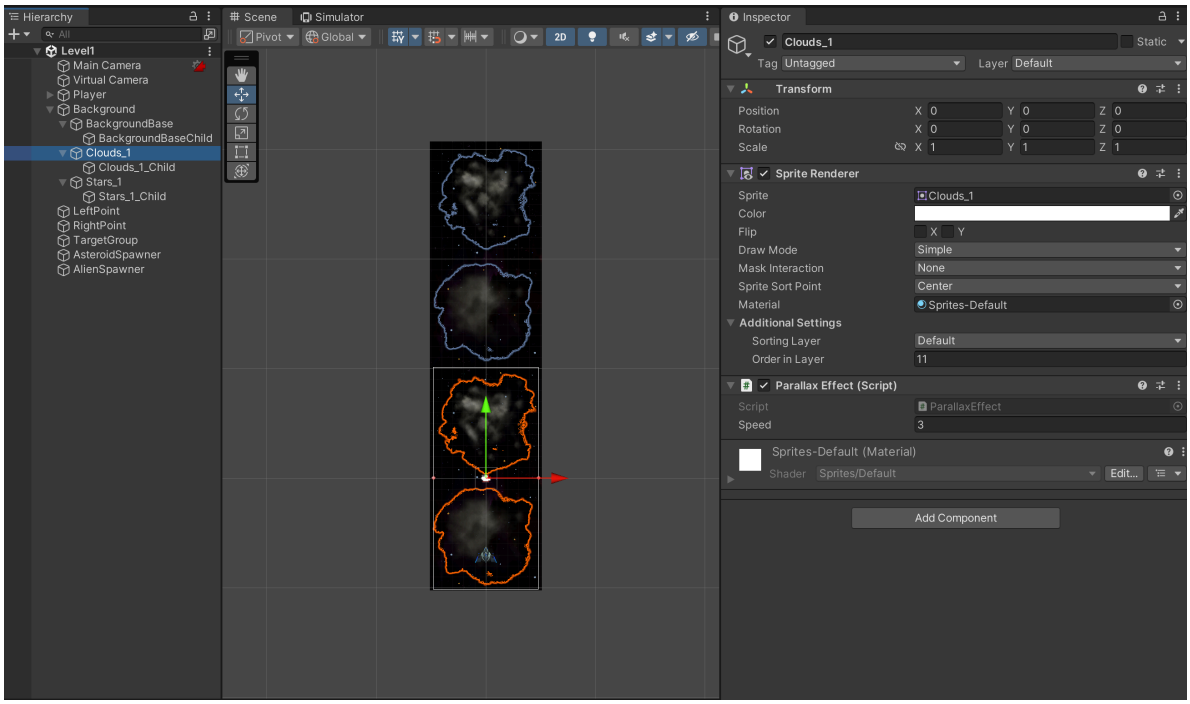
simples: um dos métodos utilizados mais adiante é o "*transform.Translate*"[2], que aceita apenas um "Vector3" como argumento e não um "Vector2".

No método *Start,* é feita a recolha da posição inicial do fundo. Além disso, é armazenado o valor que representa a altura do fundo. O método *Start* faz parte de todos os scripts que herdam a classe *MonoBehaviour* e, normalmente, é utilizado para inicializar variáveis. Este método é chamado apenas uma vez, quando o script é executado.

Já o método *Update* é invocado a cada *frame* do jogo. É nesse método que ocorre a movimentação do fundo na direção vertical, utilizando o método *Translate*, ao qual é passado como argumento o produto de três valores responsáveis por mover o objeto na direção e velocidade desejadas.

Ainda dentro do método *Update*, realiza-se a verificação para determinar quando a posição do fundo em movimento atinge o limite estipulado para o seu deslocamento. Ao atingir esse limite, ocorre o *reset* da posição do *transform*, retornando à posição inicial. O processo repete-se, criando assim o efeito de fundo infinito.

Configurações de um objeto aplicado no efeito Parallax



[2] Unity Documentation, "Transform.Translate", 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/ScriptReference/Transform.Translate.html

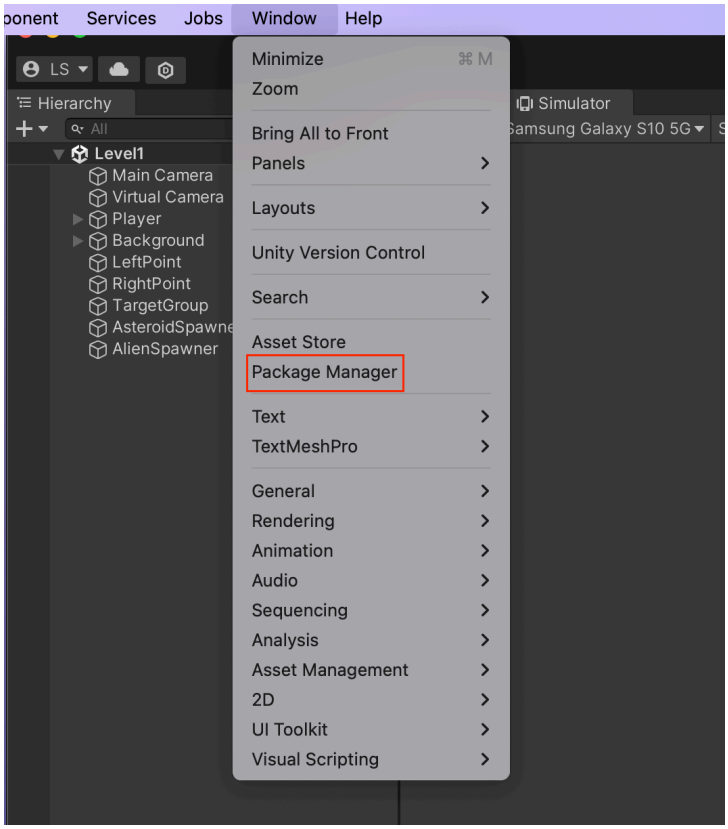
Na figura, está presente um exemplo da configuração do objeto de jogo “*Clouds_01”*, utilizado para obter o efeito “Parallax”.

Este objeto está selecionado e aparece na figura ao centro, delineado pela cor laranja. À direita deste, encontra-se a aba “*Inspector”*, onde é possível visualizar todos os componentes associadas ao objeto selecionado.

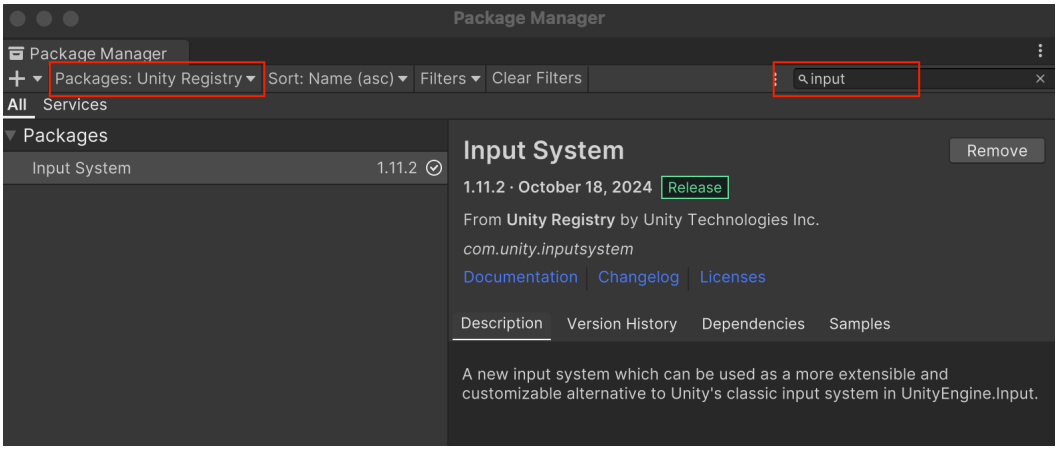
Verifica-se que um desses componentes é a inclusão do script “ParallaxEffect”. É neste script que se faz o acesso à variável “*speed”,* à qual é atribuído um valor numérico correspondente à velocidade com que o fundo se moverá.

Cada um dos objetos “pai” possui esta associação ao componente script “ParallaxEffect”, no qual foram definidos outros valores para a variável “speed”, permitindo assim obter com sucesso o efeito desejado, que cria a sensação de profundidade e velocidade na cena do jogo.

Instalação Input System



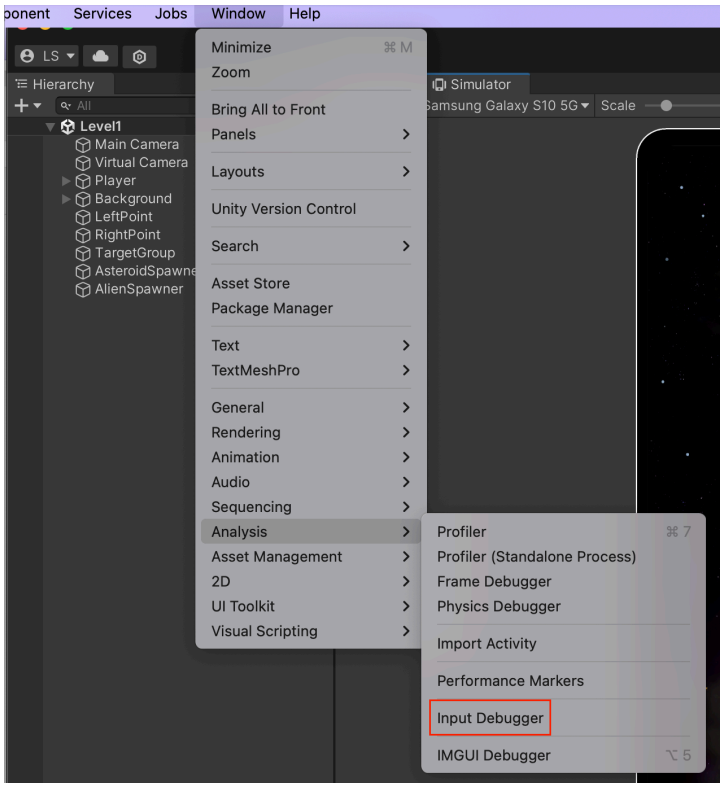
A figura mostra como aceder à interface “*Package Manager”*, para em seguida ser feita a instalação do package “*Input System*” Neste caso, o caminho retratado, “Window > Package Manager”, é relativo a um utilizador do sistema operativo macOS.

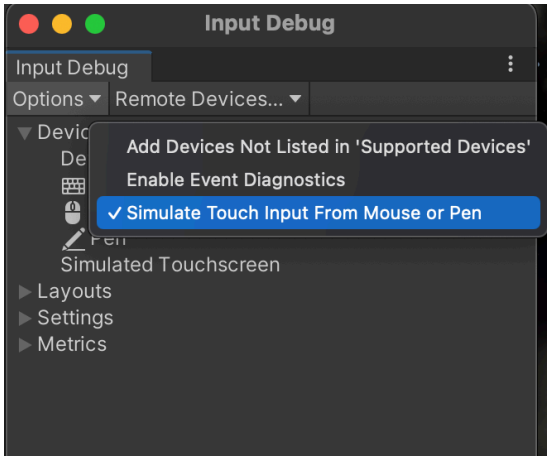


Nesta figura, constata-se a representação da interface do "*Package Manager*". O primeiro passo é selecionar o "*Packages: Unity Registry"*, no canto superior esquerdo, o que permite procurar *packages* no *Unity*.

De seguida, deve escrever "*input*" na barra de pesquisa situada no canto superior direito da interface. Ao centro, surge o package "*Input System*", onde apenas é necessário clicar no botão "*Install*" e aguardar que a instalação seja concluída com sucesso.

Assim que a instalação termine, poderá então utilizar este novo "*Input System"* no simulador, permitindo que os cliques do rato sejam traduzidos como toques no ecrã do dispositivo móvel. Para isso, é necessário ativar uma opção no "*Input Debug".* Na figura, pode-se observar como aceder à interface: "*Window > Analysis > Input Debugger*".





Na figura acima, apresenta-se a interface do "*Input Debug",* na qual, ao selecionar "*Options*", surgem três opções disponíveis. Deve-se selecionar a opção "*Simulate Touch Input From Mouse or Pen".*

Implementação da classe PlayerController

Para que seja possível utilizar as funcionalidades do "EnhancedTouch"[3], é necessário importá-lo na classe "Player Controller", adicionando a seguinte linha no topo da classe: "using UnityEngine.InputSystem.EnhancedTouch;"

De seguida, e conforme indicado nos manuais do *Unity,* devem ser implementados os seguintes métodos: "*OnEnable*", necessário para ativar as funcionalidades fornecidas pelas "*API's*": "*Touch"* e "*Finger"*. Estas adicionam processamento extra ao jogo, razão pela qual se encontram desativas por padrão; "*OnDisable*", que faz o inverso do anterior, desativando novamente as "*API's"*" que foram ativas. Esta implementação pode ser observada na figura abaixo.

```
0 references
private void OnEnable()
{
    EnhancedTouchSupport.Enable();
}

0 references
private void OnDisable()
{
    EnhancedTouchSupport.Disable();
}
```

[3] Unity Manual, "Struct Touch", 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.1/api/UnityEngine.InputSystem.EnhancedTouch.Touch.html

```
0 references
void Update()
{

    if (Touch.activeTouches.Count > 0) // verify if is touching the screen
    {
        if (Touch.activeTouches[0].finger.index == 0) // verify if only one finger is touching
        {
            Touch myTouch = Touch.activeTouches[0]; // store the active touch
            Vector3 touchPosition = myTouch.screenPosition; // get the touch position
            touchPosition = mainCamera.ScreenToWorldPoint(touchPosition); // transform to world points

            // to be possible to control the ship without clicking over the ship
            if (Touch.activeTouches[0].phase == TouchPhase.Began) // verify if the phase of the first touch is Began
            {
                offset = touchPosition - transform.position; // distance between two positions
            }

            if (Touch.activeTouches[0].phase == TouchPhase.Moved) // verify if the phase of the first touch is Moved
            {
                transform.position = new Vector3(touchPosition.x - offset.x,
                 touchPosition.y - offset.y, 0); // made the transform.position with the offset value subtracted
            }
```

Na imagem acima, é demonstrada a implementação realizada no método "*Update*", da classe "*PlayerController*".

Primeiro, é feita a verificação de se o ecrã está a ser tocado de momento, através do "*Touch.activeTouches.Count*". Este método retorna o número de toques ativos no ecrã e, caso o valor seja superior a 0, significa que o ecrã está a ser tocado.

A seguir, a validação do número de dedos que estão em contato com o ecrã. Esta verificação é essencial para evitar movimentos inesperados da nave, caso mais de um dedo esteja a tocar no ecrã do dispositivo móvel.

Após essas duas validações, o toque ativo no momento é armazenado na variável "*myTouch*", do tipo "*Touch*". A partir desse valor, pode-se extrair a posição exata do toque no ecrã, guardando essas coordenadas num "*Vector3*" na variável "*touchPosition*".

O "*touchPosition*" contém coordenadas do ecrã, sendo necessário converter esses valores para coordenadas no mundo do jogo. Para isso, foi utilizado o método "*ScreenToWorldPoint*"[4], pertencente à classe "*Camera*".

Numa primeira fase, para que seja possível movimentar a nave sem que o dedo esteja diretamente sobre a mesma, o que poderia dificultar a visualização da sua posição no jogo, foi criada a variável "offset", do tipo "Vector3".

Esta variável recebe o valor da subtração entre "*touchPosition*" e "*transform.position*", que corresponde à distância entre o toque feito no ecrã e a

[4] Unity Documentation, "Camera.ScreenToWorldPoint", 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Camera.ScreenToWorldPoint.html

posição da nave. No entanto, essa atribuição só é realizada após a validação de que o toque ativo está na fase inicial, comparando com o "*TouchPhase.Began*"[5]. Isto significa que o dedo apenas tocou no ecrã pela primeira vez.

A segunda fase a ser analisada ocorre quando o dedo se movimenta no ecrã. Para efetuar essa validação, compara-se o toque ativo com o "*TouchPhase.Moved*", o que indica qua a posição do toque foi alterada. Caso essa condição seja verdadeira, a movimentação da nave é realizada através do "*transform.position*", atribuindo-lhe um novo "*Vector3*", no qual se aplica a subtração do offset nos eixos x e y.

Dessa forma, os movimentos da nave seguem os movimentos do dedo, independentemente da posição do ecrã, eliminando a necessidade de um toque direto sobre a nave.

Limitações de navegação

```
maxLeft = mainCamera.ViewportToWorldPoint(new Vector3(0.15f, 0, 0)).x;
//By appending .x, we extract the x-coordinate of the returned Vector3
maxRight = mainCamera.ViewportToWorldPoint(new Vector3(0.85f, 0, 0)).x;
maxUp = mainCamera.ViewportToWorldPoint(new Vector3(0, 0.90f, 0)).y;
//By appending .y, we extract the y-coordinate of the returned Vector3
maxDown = mainCamera.ViewportToWorldPoint(new Vector3(0, 0.10f, 0)).y;
```

A figura apresenta o excerto de código que define os valores máximos dentro dos quais a nave do jogador pode movimentar-se.

Para isso, é utilizado o método *"ViewportToWorldPoint"*[6], pertencente à classe "*Camera*", que converte coordenadas de "*viewport"* em coordenadas do mundo do jogo.

A "*viewport*", é um espaço normalizado utilizado para representar o ecrã do dispositivo. As suas coordenadas variam de (0,0), no canto inferior esquerdo até (1,1) no canto superior direito. O ponto (1,0) identifica o canto inferior do lado direito, enquanto o centro da viewport corresponde a (0.5,0.5). Já no canto superior equerdo possui a coordenada (0,1).

---

[5] Unity Manual, "Enum TouchPhase", 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/api/UnityEngine.InputSystem.TouchPhase.html

[6] Unity Documentation, "Camera.ViewportToWorldPoint", 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Camera.ViewportToWorldPoint.html

```
0 references
void Update()
{

    if (Touch.activeTouches.Count > 0) // verify if is touching the screen
    {
        if (Touch.activeTouches[0].finger.index == 0) // verify if only one finger is touching
        {
            Touch myTouch = Touch.activeTouches[0]; // store the active touch
            Vector3 touchPosition = myTouch.screenPosition; // get the touch position
            touchPosition = mainCamera.ScreenToWorldPoint(touchPosition); // transform to world points

            // to be possible to control the ship without clicking over the ship
            if (Touch.activeTouches[0].phase == TouchPhase.Began) // verify if the phase of the first touch is Began
            {
                offset = touchPosition - transform.position; // distance between two positions
            }

            if (Touch.activeTouches[0].phase == TouchPhase.Moved) // verify if the phase of the first touch is Moved
            {
                transform.position = new Vector3(touchPosition.x - offset.x,
                 touchPosition.y - offset.y, 0); // made the transform.position with the offset value subtracted
            }

            // Clamp is used to constrain a value within a specified range (value, min, max);
            transform.position = new Vector3(
                Mathf.Clamp(transform.position.x, maxLeft, maxRight),
                Mathf.Clamp(transform.position.y, maxDown, maxUp), 0);
        }
    }
}
```

Esta figura apresenta o método "*Update"* completo da classe "*PlayerController".* A última alteração realizada encontra-se na parte inferior da figura, onde é atribuída a nova posição ao "*transform.position".* Neste ponto, são utilizadas as variáveis que armanezam os valores máximos dentro dos quais a nave se pode movimentar-se. Para isso, recorre-se ao método "*Clamp*", pertencente à classe "*Mathf*", que restringe o valor do "*transform.position*" dentro do intervalo definido para cada um dos eixos.

No caso do eixo do x, o intervalo é delimitado pela variável "*maxLeft"* como valor mínimo e pela variável "*maxRight"* como valor máximo. Já no eixo do y, o valor mínimo corresponde à variável "*maxDown",* enquanto o valor máximo é determinado pela variável "*maxUp*".

Coroutine SetBoundaries

```
1 reference
private IEnumerator SetBoundaries()
{
    yield return new WaitForSeconds(0.4f); // wait to adjust the camera to the mobile device
    maxLeft = mainCamera.ViewportToWorldPoint(new Vector3(0.15f, 0, 0)).x;
    //By appending .x, we extract the x-coordinate of the returned Vector3
    maxRight = mainCamera.ViewportToWorldPoint(new Vector3(0.85f, 0, 0)).x;
    maxUp = mainCamera.ViewportToWorldPoint(new Vector3(0, 0.90f, 0)).y;
    //By appending .y, we extract the y-coordinate of the returned Vector3
    maxDown = mainCamera.ViewportToWorldPoint(new Vector3(0, 0.10f, 0)).y;
}
```

Na figura acima , é possível ver a implementação da "*Coroutine*"[7] chamada "*SetBoundaries*".

O código que anteriormente era executado diretamente quando o método "*Start*" era chamado foi movido para dentro deste novo método. A palavra-chave "yield" é utilizada para pausar a execução do método, seguido do "*WaitForSeconds(0.4f)",* que, como o nome indica, faz com que a execução desta "Coroutine" seja pausada por 0,4 segundos.

Após alguns testes, verificou-se que este é o tempo necessário para garantir que a câmera se ajuste adequadamente às dimensões do ecrã do dispositivo móvel antes de calcular os limites de movimento da nave.

[7] Unity Documentation, "Splitting tasks across frames", 2024. Accessed: Out, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/Manual/coroutines.html

Superclasse Enemy

```
using UnityEngine;

4 references
public class Enemy : MonoBehaviour
{
    2 references | 2 references
    [SerializeField] protected float health, damage;
    2 references
    [SerializeField] protected Rigidbody2D rigidBody2D;

    1 reference
    public void Damaged(float damage)
    {
        health -= damage;
        Hitted();
        if (health <= 0)
        {
            Destroyed();
        }
    }

    3 references
    public virtual void Hitted(){}

    3 references
    public virtual void Destroyed(){}
}
```

A figura, demostra a implementação da superclasse “*Enemy*”. Esta classe tem três atributos: “health”, “damage” e “rigidBody2D”, definidos com o modificador de acesso “*protected*”. Este modificador permite que apenas a própria classe ou as suas classes derivadas tenham acesso a esses atributos.

É importante prestar atenção ao atributo “*rigidBody2D*”, pois todos os objetos criados a partir desta classe terão este atributo associado. Algumas linhas abaixo, encontra-se a função “*Damaged*”, responsável por decrementar a vida do inimigo até que este valor seja igual ou inferior a zero, caso isso aconteça, o objeto será destruído.

Por fim, são declaradas duas funções, “*Hitted*” e “*Destroyed*”, com a particularidade de serem declaradas com a palavra “virtual”. Isso significa que essas funções podem ser sobrescritas (*override*) na subclasse, permitindo personalizar a sua implementação conforme necessário. Este conceito está diretamente relacionado com um dos quatro pilares da programação orientada a objetos: o polimorfismo.

Subclasse Asteroid

```
public class Asteroid : Enemy
{
    1 reference | 1 reference | 1 reference
    [SerializeField] private float minSpeed, maxSpeed, rotateSpeed;
    2 references
    private float speed;

    0 references
    void Start()
    {
        speed = Random.Range(minSpeed, maxSpeed);
        rigidBody2D.velocity = Vector2.down * speed;
    }

    0 references
    void Update()
    {
        transform.Rotate(0, 0, rotateSpeed * Time.deltaTime);
    }

    0 references
    private void OnTriggerEnter2D(Collider2D collider2D)
    {
        if (collider2D.CompareTag("Player"))
        {
            // decrease the power of the player
            collider2D.GetComponent<PlayerStats>().takeDamage(damage);
            // asteroid destroyed
            Destroy(gameObject);
        }
    }

    2 references
    public override void Hitted()
    {
        // Animation
    }

    2 references
    public override void Destroyed()
    {
        // Animation
        Destroy(gameObject);
    }

    0 references
    private void OnBecameInvisible()
    {
        Destroy(gameObject);
    }
}
```

A figura refere-se à implementação da classe "*Asteroid*", subclasse de "*Enemy*".

Esta classe possui três atributos do tipo "*float*": "*minSpeed*", "*maxSpeed*" e "*rotateSpeed*", todos associados à rotação e à queda dos asteroides no ecrã do jogo.

No método "*Start*", é estabelecida uma velocidade aleatória dentro dos limites definidos pelo programador, através das variáveis "*minSpeed*" e "*maxSpeed*". Na linha seguinte, essa velocidade é aplicada no sentido descendente.

No método "*Update*", é utilizado "*transform.Rotate*"[8], um método responsável por realizar a rotação de um objeto em torno de um ou mais eixos. Para obter uma rotação no eixo do z, é aplicada a variável "*rotateSpeed*", que representa a velocidade de rotação em graus por segundo. Esta é multiplicada pelo "*Time.deltaTime*", garantindo que a rotação seja ajustada de forma proporcional ao tempo, independentemente da taxa de "*frames"*.

Posteriormente, foi implementado o método "*OnTriggerEnter2D*", que é um evento fornecido pelo *Unity* para identificar a interação entre dois objetos com o componente "*Collider 2D*"[9], desde que um desses objetos tenha a propriedade "*Is Trigger*" ativa. Na classe "*Asteroid*", esta função identifica quando a nave do jogador colide com o asteroide.

Para isso, é usado o método "*CompareTag*", que verifica a "*tag*" do objeto em colisão com o objeto criado pela classe. No caso, de o objeto 2D "*Asteroid*" colidir com o objeto 2D "*Player*", será retirado um valor correspondente a "*damage*" do "*power*" da nave do jogador. Isto é realizado através do método "*GetComponent<T>"*, que permite aceder a componentes anexados ao objeto, neste caso, "*PlayerStats*", do qual queremos utilizar o método "*takeDamage(damage)".* O método "*GetComponent<T>*"[10] é uma das formas mais comuns de interagir entre scripts e componentes de um objeto.

Nas linhas seguintes da classe, encontram-se os dois métodos herdados da superclasse, "*Hitted*" e "*Destroyed*", nos quais serão implementadas as animações correspondentes ao asteroide. No método "Destroyed", já é feita a chamada ao método "*Destroy*", uma função do *Unity* que elimina objetos, componentes ou recursos durante o jogo.

O último método presente na classe "*Asteroid*" é o "*OnBecameInvisible*", um método do *Unity* chamado quando a renderização de um objeto do jogo se torna invisível para qualquer câmera. Na classe "Asteroid", este método é utilizado para remover automaticamente o próprio objeto quando deixa de ser visível na cena, contribuindo para otimizar o desempenho do jogo.

---

[8] Unity Documentation, "Transform.Rotate", 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Transform.Rotate.html

[9] Unity Manual, "Polygon Collider 2D component reference", 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/Manual/2d-physics/collider/polygon-collider-2d-reference.html

[10] Unity Documentation, "GameObject.GetComponent", 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/GameObject.GetComponent.html

Classe AsteroidSpawner

```
0 references
public class AsteroidSpawner : MonoBehaviour
{
    2 references
    [SerializeField] private GameObject[] asteroids;
    1 reference
    [SerializeField] private float spawnerTime;
    3 references
    private float timer = 0f;
    2 references
    private int asteroidType;
    4 references
    private Camera mainCamera;
    2 references | 2 references | 2 references
    private float maxLeft, maxRight, spawnerPositionY;

    0 references
    void Start()
    {
        mainCamera = Camera.main;
        StartCoroutine(SetBoundaries());
    }

    0 references
    void Update()
    {
        timer += Time.deltaTime;
        if (timer > spawnerTime)
        {
            CreateAsteroid();
            timer = 0f;
        }
    }

    1 reference
    private void CreateAsteroid()
    {

        asteroidType = Random.Range(0, asteroids.Length);

        GameObject myObject = Instantiate(
            asteroids[asteroidType], new Vector3(Random.Range(maxLeft, maxRight),
             spawnerPositionY, 0), Quaternion.Euler(0, 0, Random.Range(0, 360)));

        float asteroidSize = Random.Range(0.9f, 1.1f);
        myObject.transform.localScale = new Vector3(asteroidSize, asteroidSize, 1);

    }

    1 reference
    private IEnumerator SetBoundaries()
    {
        yield return new WaitForSeconds(0.4f);
        maxLeft = mainCamera.ViewportToWorldPoint(new Vector2(0.15f, 0)).x;
        maxRight = mainCamera.ViewportToWorldPoint(new Vector2(0.85f, 0)).x;
        spawnerPositionY = mainCamera.ViewportToWorldPoint(new Vector2(0, 1.2f)).y;
    }
}
```

A figura revela a implementação da classe “AsteroidSpawner”, cuja principal função é gerar asteroides no ecrã do jogo da forma mais natural possível.

O primeiro atributo desta classe chama-se “asteroids”, sendo um *array* de “*GameObject”,* que armazena os diversos tipos de asteroides presentes no jogo. O segundo atributo, denominado “spawnerTime”, define o intervalo de tempo entre a geração dos asteroides. Estes dois atributos são definidos na interface do *Unity* pelo programador, permitindo testar várias possibilidades de forma mais rápida.

O atributo “asteroidType”, é o índice utilizado para selecionar de forma aleatória um elemento dentro do *array* “asteroids”.

Os últimos atributos desta classe são: “mainCamera”, “maxLeft”, “maxRight” e “spawnerPositionY”, todos com o objetivo de manter a criação dos asteroides dentro das posições de jogo.

No método “*Update*”, existe um temporizador que invoca o método “*CreateAsteroid*” em intervalos de tempo previamente definidos. Quando esse tempo é atingido, o método “*CreateAsteroid*” é chamado, sendo responsável pela criação do objeto de jogo asteroide.

Primeiramente, é feita uma escolha aleatória do tipo de asteroide a ser gerado. Em seguida, cria-se o “*GameObject*”, utilizando o método “*Instantiate*”, que serve para criar ou duplicar objetos durante a execução do jogo. A sintaxe deste método é a *“Instantiate (original, position, rotation)”*, onde o parâmetro “original”, refere-se ao objeto ou “*prefab*” que desejamos instanciar. O segundo parâmetro, “position”, define a posição onde o objeto será instanciado, para isso, usa-se o “*Vector3”.* Por fim, o parâmetro “rotation” usa o método “*Quaternion.Euler”*[11], do *Unity,* que permite criar uma rotação a partir de ângulos de “Euler” [12]em três dimensões (x, y, z). Neste caso, a rotação desejada é na dimensão z, sendo gerado um valor aleatório entre 0 e 360. Isso garante que os asteroides apareçam com uma rotação inicial diferente, proporcionando uma dinâmica distinta ao jogo.

Por tornar os asteroides ainda mais variados, é feito um ajuste aleatório no seu tamanho, utilizando “*transform.localScale*”[13], no qual é atribuído um fator de escala entre os valores de 0.9 e 1.1f. Esta modificação é sempre realizada em relação ao objeto pai. Na prática, os asteroides terão variações de tamanho entre 90% a 110% do seu tamanho original, sendo que essas modificações ocorrem apenas nos eixos x e y, sem afetar o eixo z.

[11] Unity Documentation, “Quaternion.Euler”, 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Quaternion.Euler.html

[12] Unity Documentation, “Quaternion.Euler”, 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Quaternion.Euler.html

[13] Unity Documentation, “Transform.localScale”, 2024. Accessed: Nov, 2024. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Transform-localScale.html

Script AlienShoot

```
public class AlienShoot : MonoBehaviour
{

    1 reference | 1 reference
    [SerializeField] private float speed, damage;
    2 references
    private Rigidbody2D rigidBody2D;
    0 references
    void Start()
    {
        rigidBody2D = GetComponent<Rigidbody2D>();
        rigidBody2D.velocity = Vector2.down * speed;
    }

    0 references
    private void OnTriggerEnter2D(Collider2D collider2D)
    {
        if (collider2D.CompareTag("Player"))
        {
            collider2D.GetComponent<PlayerStats>().takeDamage(damage);
            Destroy(gameObject);
        }
    }

    0 references
    private void OnBecameInvisible()
    {
        Destroy(gameObject);
    }
}
```

A figura destaca a implementação do script "AlienShoot".

Esta imagem demonstra que existem vários pontos em comum entre classes de videojogos. O atributo "*rigidbody2D*" está presente em qualquer uma das classes em que são necessárias capacidades de movimento no objeto.

Quanto aos métodos, verifica-se que o método "*OnTriggerEnter2D*" está presente, pois este é essencial para detetar colisões entre objetos de jogo, no caso, entre o tiro feito pelo alienígena e a nave do jogador.

Outro método muito utilizados é o "*OnBecameInvisible*", um *callback* usado para detetar quando um objeto deixa de ser visível, eliminando-o e, com isso, libertando recursos que melhoram o desempenho do jogo.

Resumindo, neste tipo de desenvolvimento, existem sempre alguns padrões de implementação que podem ser repetitivos, mas são necessários.

Script Insetroid

```
public class Insetroid : Enemy
{

    1 reference | 1 reference
    [SerializeField] private float speed, shootTime;
    1 reference
    [SerializeField] private Transform shootPoint;
    1 reference
    [SerializeField] private GameObject shoot;
    3 references
    private float shootTimer;

    0 references
    void Start()
    {
        rigidBody2D.velocity = Vector2.down * speed;
    }

    0 references
    void Update()
    {
        shootTimer += Time.deltaTime;
        if (shootTimer >= shootTime)
        {
            Instantiate(shoot, shootPoint.position, Quaternion.identity);
            shootTimer = 0;
        }
    }
```

A figura apresenta parte da implementação da subclasse “Insetroid”.

Destaca-se, nesta figura, o uso do “*Quaternion.identity*”, que representa uma rotação neutra, ou seja, sem qualquer tipo de rotação aplicada, equivalente a 0 graus em todos os eixos. Neste caso, pretende-se manter a posição inicial em que o alienígena é gerado, ao contrário do que acontece com os asteroides.

Outro ponto a destacar é a utilização da palavra-chave “[*SerializeField*]” nos atributos. Esta permite ao programador aceder a esses atributos na interface do *Unity* e modificá-los conforme o necessário.

Classe PlayerStats

```
3 references
public class PlayerStats : MonoBehaviour
{
    1 reference
    [SerializeField] private float powerMax;
    3 references
    private float power;

    0 references
    void Start()
    {
        power = powerMax;
    }

    3 references
    public void takeDamage(float damage)
    {
        power -= damage;
        if (power <= 0)
        {
            Destroy(gameObject);
        }
    }
}
```

A figura ilustra a classe "*PlayerStats*". Como pode ser verificado, esta ainda se encontra numa fase inicial de implementação. No entanto, já realiza a gestão do "*power*" da nave, associando o atributo "*powerMax*" ao atributo "*power*" dentro do método "*Start*" da classe.

É nesta classe que se encontra o método "*takeDamage*". Este método reduz o "*power*" da nave até que atinja um valor igual ou inferior a 0, momento em que o jogo chega ao fim para o jogador.

Classe PlayerShooting

```
0 references
public class PlayerShooting : MonoBehaviour
{
    1 reference
    [SerializeField] private GameObject missil;
    1 reference
    [SerializeField] private Transform shootingPoint;
    4 references
    [SerializeField] private float shootingTimer;
    2 references
    private float timer;

    0 references
    void Start()
    {
        timer = shootingTimer;
    }

    0 references
    void Update()
    {
        shootingTimer -= Time.deltaTime;
        if (shootingTimer <= 0)
        {
            Shoot();
            shootingTimer = timer;
        }
    }

    1 reference
    private void Shoot()
    {
        Instantiate(missil, shootingPoint.position, Quaternion.identity);
    }
}
```

A figura representa a implementação da classe "*PlayerShooting*".

Esta classe é composta por quatro variáveis, sendo que três delas possuem o atributo "*SerializeField*" associado. São elas: o "*missil*" do tipo "*GameObject*", que define o tipo de tiro disparado pela nave, permitindo a personalização do projétil para cada modelo de nave; o "*shootingPoint*" do tipo "*Transform*", que determina a posição de onde o disparo será efetuado; e o "*shootingTimer*", responsável por definir o intervalo de tempo entre os disparos.

No que diz respeito aos métodos, destaca-se o "*Shoot*", cuja função é instanciar o míssil que será disparado pela nave. Este método é chamado dentro do "*Update*", onde ocorre a temporização necessária para a execução dos disparos.