INSTITUTO SUPERIOR MANUEL TEIXEIRA GOMES

Desenvolvimento de um Jogo Auto-Shooter utilizando a Engine Godot.

Relatório do Trabalho Final de Curso

Cayan Moraes Prola

Trabalho orientado por:
Prof. Cristiano Soares

2024

Desenvolvimento de um Jogo Auto-Shooter utilizando a Engine Godot.

Declaração de autoria

Declaro ser o (a) autor(a) do trabalho apresentado neste relatório, sendo original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

_____

(Assinatura)

O Instituto Superior Manuel Teixeira Gomes tem o direito, perpétuo e sem limites geográficos, de arquivar e publicitar este trabalho através de quaisquer meios, de o divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos não comerciais, desde que seja dado crédito ao autor.

_____

(Assinatura)

Agradecimentos

Gostaria de expressar minha gratidão especialmente a minha família, por terem me apoiado durante todo o meu trajeto.

Gostaria de agradecer também todos os meus amigos, que estiveram sempre comigo durante a duração do curso e do projeto.

Agradeço também aos professores pelo apoio durante o desenvolvimento do projeto e do curso.

Não posso deixar de agradecer aos meus colegas de curso, que estiveram comigo durante todo o trajeto.

Por último, agradeço ao Instituto Superior Manuel Teixeira Gomes e seus associados pelo conhecimento obtido, que foram fundamentais para o meu crescimento pessoal e profissional.

Resumo

Este projeto teve como objetivo a criação de um jogo do subgênero auto-shooter, inspirado nas tendências modernas de jogos que evoluíram a partir do gênero shoot 'em up e, mais especificamente, do bullet hell. Inspirado em títulos como Vampire Survivors e Deep Rock Galactic: Survivor, utilizando a engine open-source Godot. O jogo incorpora características típicas do subgênero, como um sistema de upgrades dinâmico, spawn de hordas de oponentes, um mapa infinito e um sistema de armas com características únicas.

A escolha da engine Godot foi feita levando em conta diversas variáveis, como sua flexibilidade e facilidade de adaptação. A engine oferece suporte a múltiplas linguagens de programação, como C#, C++ e GDScript, sendo esta última uma linguagem muito semelhante ao Python, com a qual já possuo experiência. Essa familiaridade acelerou o processo de desenvolvimento, permitindo a implementação eficiente de funcionalidades complexas. Além disso, o fato de Godot ser open-source foi um fator decisivo, proporcionando maior transparência e confiabilidade no desenvolvimento do projeto.

O desenvolvimento enfrentou desafios técnicos e de design, incluindo a integração dos sistemas do jogo, a criação e otimização das armas, e a implementação de múltiplos upgrades. Embora o projeto esteja concluído, há oportunidades para trabalhos futuros, como a adição de novos inimigos, armas, upgrades e mapas que expandiriam ainda mais a experiência do jogo.

Palavras-chave: Jogo auto-shooter, Godot engine, Desenvolvimento de jogos, Mapa infinito, Aprimoramentos de personagem.

Tabela de Conteúdos

Introdução

A indústria de jogos está passando por um momento peculiar no qual jogos produzidos por gigantes do mercado estão cada vez mais sofrendo com orçamentos gigantes que não geram retorno, falta de inovação nos lançamentos, jogos de serviço que não são atraentes ao público. E naturalmente, o crescimento de jogos indie acabou ocorrendo, projetos que são feitos com carinho e cuidado e que lembram a indústria de que jogos são arte, e não apenas um produto comercial.
Um exemplo notável é o subgênero dos *auto-shooters*, que evoluiu a partir dos jogos de bullet hell, dentro do gênero *shoot 'em up*. Um exemplo de um jogo desse é Vampire Survivors, que foi desenvolvido por apenas um desenvolvedor inicialmente, e revolucionou a indústria com uma jogabilidade simples, porém extremamente viciante e relaxante, disponibilizada em múltiplas plataformas e com um preço acessível, contrariando muito das práticas usuais das grandes empresas da indústria. O sucesso do jogo foi tão grande que inspirou vários outros jogos como Deep Rock Galactic: Survivors, e até sendo incorporado no universo de jogos da Riot Games, uma das maiores empresas da indústria, com o modo de jogo chamado “Swarm”, lançado recentemente no cliente do jogo League of Legends, possivelmente o maior *MOBA* da atualidade.

O objetivo deste projeto é desenvolver um jogo *auto-shooter* inspirado por essas tendências, utilizando a *engine open-source* Godot. O jogo busca ter a mesma jogabilidade que encantou milhões de jogadores pelo mundo e gerar uma sensação descontraída e relaxante de diversão.

A escolha do projeto se deu feita pela minha paixão em jogos, cultivada desde a infância, unida ao desejo de um desafio que nunca havia enfrentado: desenvolver um jogo completo. Juntamente com a oportunidade de expandir meus conhecimentos em programação e design de jogos e ter a experiencia de fazer algo que utilizei durante muito tempo.

Durante o desenvolvimento do jogo, foram empregadas diversas técnicas de otimização e design de gameplay, utilizando as funcionalidades da Godot para garantir um desempenho eficiente e uma experiência de jogo fluida. A escolha da *engine* Godot foi estratégica, considerando sua flexibilidade e suporte ao GDScript, que facilitou o desenvolvimento rápido e eficiente do jogo.

Estado da arte

Introdução ao gênero

O gênero *shoot 'em up* é um dos gêneros mais antigos da história dos videogames, tendo suas raízes na década de 1960. São caracterizados por colocarem o jogador em controle de uma espaçonave ou um personagem enfrentando ondas de inimigos diferentes, tendo como objetivo a sobrevivência e a destruição dos inimigos, necessitando boa coordenação motora para se mover em movimento constante, desviando dos projeteis inimigos e acertando os seus próprios disparos.

No final da década de 1970 e início da década de 1980, o gênero *shoot 'em up* começou a crescer em popularidade com jogos como Space Invaders (1978), Asteroids (1979) e Galaga (1981), jogos esses que estabeleceram conceitos e padrões de design utilizados até os dias de hoje, como sistemas de power-ups e batalhas contra chefes.

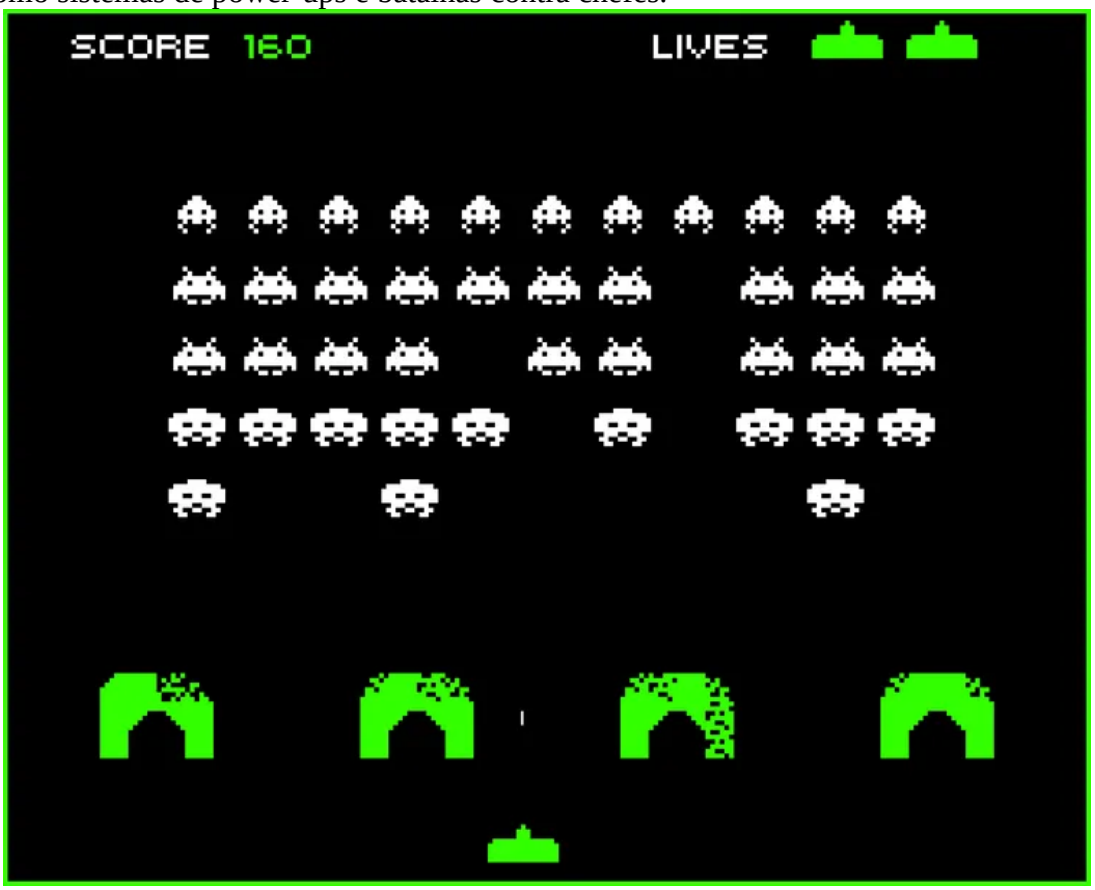

Figura 1: Imagem do jogo Space Invaders (1978).

Com o passar do tempo, o gênero se diversificou mais ainda, pois os jogos já não eram considerados desafiadores o suficiente, com isso, houve o surgimento do subgênero *bullet hell*, na década de 1990. Este nome vem da expressão japonesa "*danmaku*", que traduzida ao português significa "barragem", referindo-se a imensa quantidade de projeteis que o jogador deve se esquivar, aumentando a dificuldade e tornando a capacidade de completar os jogos uma verdadeira conquista. Este subgênero teve seu nascimento e ascensão no mercado de jogos japonês, e acabou tendo algumas de suas características aplicadas em jogos de outros gêneros, como Nier: Automata e Undertale, ambos jogos imensamente aclamados pela crítica especializada.

Figura 2: Imagem do jogo Touhou 2: The Story of Eastern Wonderland (1997)

Devido à grande dificuldade dos jogos *bullet hell*, o gênero acabou se tornando nichado e de difícil acesso, o que levou a continuação desse processo de evolução. O próximo passo evolucionário foi o surgimento do subgênero *auto-shooter*, que também pode ser chamado de *bullet heaven*, contrastando diretamente com *bullet hell*, pois quem atira todos os projeteis são o jogador, e seu objetivo principal é desviar e sobreviver dos inimigos, focando-se apenas na movimentação. Com os avanços tecnológicos ocorridos durante os anos 2000 e o grande crescimento do mercado de jogos, a possibilidade de jogos estarem disponíveis a qualquer momento através de um dispositivo móvel como um celular ou um vídeo game portátil como o *Steam Deck*, ocasionou com que houvesse uma facilitação a entrada para novos jogadores. Em união com a simplificação dos controles, a automatização dos disparos e a baixa necessidade computacional dos jogos *auto-shooter*, surgiu um cenário perfeito para eles fazerem muito sucesso.

Análise de jogos relevantes

Vampire Survivors

Vampire Survivors é um dos jogos mais influentes dentro do subgênero auto-shooter, e foi a grande inspiração para o desenvolvimento deste projeto. Desenvolvido inicialmente por um único desenvolvedor, este jogo destacou-se pela sua simplicidade e jogabilidade viciante. O jogo coloca o jogador em um ambiente 2D, onde deve sobreviver até um determinado tempo, lutando contra ondas de inimigos contínuas.

A mecânica de *auto-shooting* é a essência do jogo: o jogador apenas controla a movimentação do personagem, enquanto as armas disparam automaticamente. Porém, com um pequeno detalhe que torna o jogo diferente, que são elementos *roguelike*. *Roguelike* é um outro subgênero que cresceu bastante na década de 2010, que consiste em jogos no qual são necessárias várias tentativas para terminar uma missão ou mapa, e a cada tentativa o jogador recebe novas habilidades e até aumentos de estatísticas do personagem. Criando um loop de constante melhora, no qual quanto mais o jogador joga, mais poderoso se torna. Estes

elementos *roguelike* acabaram se tornando algo muito comum nos jogos *auto-shooter*, aumentando a dificuldade inicial dos jogos, mas também criando uma re-jogabilidade muito maior.

No caso de Vampire Survivors, temos melhorias permanentes, que devem ser compradas com o ouro recebido em cada partida, e dentro de cada partida temos 6 espaços para escolhermos diversas opções de armas com atuações diferentes, e mais 6 opções de estatísticas com atuações diferentes, totalizando 12 espaços totais. Isso torna cada partida do jogo diferente, devido à grande possibilidade de escolhas possíveis. Unido a isso, existem combinações únicas para cada arma, que fazem com que as armas evoluam ainda mais, aumentando em muito seu poder e mudando a dinâmica do jogo.
Todas essas características foram inspirações diretas no meu projeto.

Figura 3 Imagem do jogo Vampire Survivors (2022)

Deep Rock Galactic: Survivors

Deep Rock Galactic: Survivors é um spin-off do jogo *Deep Rock Galactic*, adaptando o universo rico e cooperativo do original para o formato auto-shooter. O jogo mantém a ambientação e universo característico do sucessor, porém com uma visão isométrica, elementos *roguelike*, mapas procedurais e com destruição de elementos.
As armas e habilidades disponíveis são as mesmas encontradas no primeiro jogo da série, trazendo um senso de continuidade e familiaridade para os fãs do original.
Além disso, o jogo mantém um foco muito maior em progressão de personagem, não sendo necessário apenas ouro para evoluir uma estatística do jogador, como também a coleta de diferentes ingredientes do universo do jogo para a evolução. O jogo também inclui diferentes missões opcionais dentro de cada partida, aumentando bastante a variedade entre partidas.

O sistema de mapas é uma grande evolução de outros jogos *auto-shooter*, o sistema de destruição do mapa, além de muito bem-feito, faz muito sentido no contexto do jogo, no qual os personagens jogáveis são mineradores. Todos os objetos destrutíveis são minérios, e não servem apenas como itens a serem coletados, mas também como uma maneira de se

movimentar pelo mapa e criar caminhos diferentes.

Figura 4 Imagem do jogo Deep Rock Galactic: Survivor (2024).

Conclusão do estudo da arte

Estudando a evolução dos jogos, principalmente do gênero *shoot 'em up*, produziu muitas informações relevantes na construção do projeto, desde a escolha do tema e conceito do projeto, como também nas diferentes maneiras de executar o desenvolvimento.

Estes jogos não só definem padrões para o gênero, mas também serviram de inspiração direta para o desenvolvimento do projeto, sendo Vampire Survivors a grande motivação por trás da escolha do tema.

Métodos

O processo de desenvolvimento do jogo começou com uma fase de planejamento, para definir os objetivos e as funcionalidades que o jogo deveria ter. O objetivo era criar um jogo completo, com todas as funcionalidades que jogos do gênero auto-shooter possuem.

A fase de desenvolvimento inicial foi utilizada para adquirir conhecimento das funcionalidades da engine e entender a complexidade do projeto e sua viabilidade. Todas as funcionalidades implementadas funcionam como base para o projeto, porém foram implementados inicialmente como teste.

O desenvolvimento do projeto foi feito utilizando a engine open-source Godot na sua versão 4.2.2, escolhida pela sua flexibilidade e confiabilidade. Foi utilizado a linguagem GDScript, que está integrada diretamente na engine. Além disso, foi utilizado Git para controle de

versão, Itch.io, Kenney e OpenGameArt para assets. Todos os assets utilizados são gratuitos e livres para uso.

Desenvolvimento inicial

Escolha de assets

Após o processo de planejamento e a definição de objetos, foi feita a escolha de assets, para visualizar como ficaria a arte do jogo, se as artes se comunicariam bem entre elas e se seria possível atingir a qualidade desejada. Todos os assets escolhidos e utilizados possuem estilos parecidos e são feitos em formato de pixel art. Após a escolha e coleta de todos os possíveis assets, foi iniciado o desenvolvimento inicial do projeto.

Criação do personagem e sua movimentação

A primeira implementação foi a criação do personagem principal e todos seus *nodes* e funcionalidades básicas. Os *nodes* utilizados para o personagem foram um **CharacterBody2D** como *root node*, e **AnimatedSprite2D** e **CollisionShape2D** como seus filhos. O *node* **CharacterBody2D** serve para a criação de personagens 2D com física e colisões que se movem através da funcionalidade implementada em scripts. O personagem é puramente controlado pelo utilizador, possui uma velocidade padrão e se move em qualquer eixo. O *node* **AnimatedSprite2D** é utilizado para utilizar um personagem animado com a utilização dos assets obtidos, o personagem possui animações enquanto anda, quando leva dano e quando morre. Já o *node* **CollisionShape2D** serve para controlar a colisão com os inimigos e possíveis obstáculos. Para a movimentação do personagem, foram adicionados novos inputs na engine, estes inputs foram as teclas W, A, S, D, teclas estas que são o padrão utilizado no mercado de jogos para teclados do padrão QWERTY. Estes inputs são então recebidos pela engine e lidos na função de física padrão da classe **CharacterBody2D**, onde cada um dos inputs representa uma direção específica.

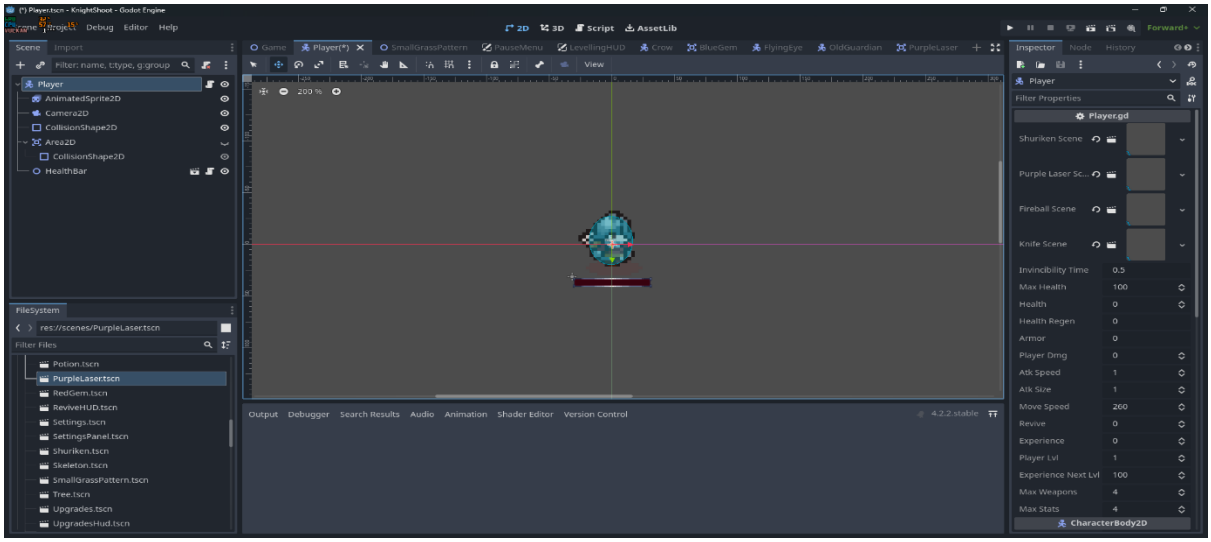

Figura 5 Imagem do jogador dos seus filhos.

Implementação da câmera

Para garantir que o personagem fosse sempre o foco principal, foi adicionado um *node* **Camera2D** como filho do personagem. A câmera se movimenta juntamente com o personagem pelo mapa, atingindo o resultado desejado.

Sistema de armas inicial

Inicialmente, foi feito o desenvolvimento de apenas uma arma, o laser roxo. Esta arma, inicialmente, atira sempre para a direita do personagem, e possui atributos como dano e velocidade de movimento. Foi utilizada inicialmente para testar as funcionalidades da engine e seu modelo de física A arma foi feita com um *node* de **Area2D**, que serve para criar objetos 2D com colisões, e utiliza também um **CollisionShape2D** para o controle de colisões.

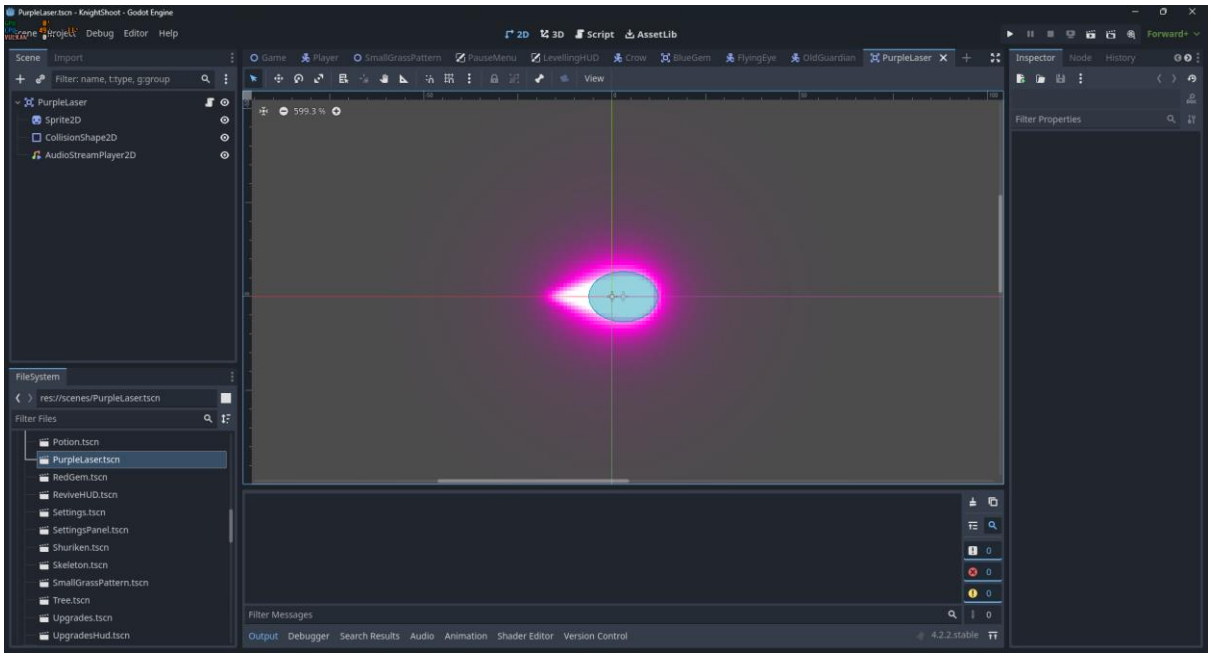

Figura 6 Imagem da cena do Laser roxo e seus nodes.

Com a primeira arma adicionada, foi feita a implementação dos primeiros inimigos, os *Orcs*, para testar se a sua funcionalidade estava funcionando corretamente.

Criaçao dos Orcs

Os primeiros inimigos criados foram os *Orcs*. Inicialmente, eram adicionados manualmente na cena principal do jogo. Possuem a mesma estrutura de *nodes* do jogador, com exceção da

câmera.

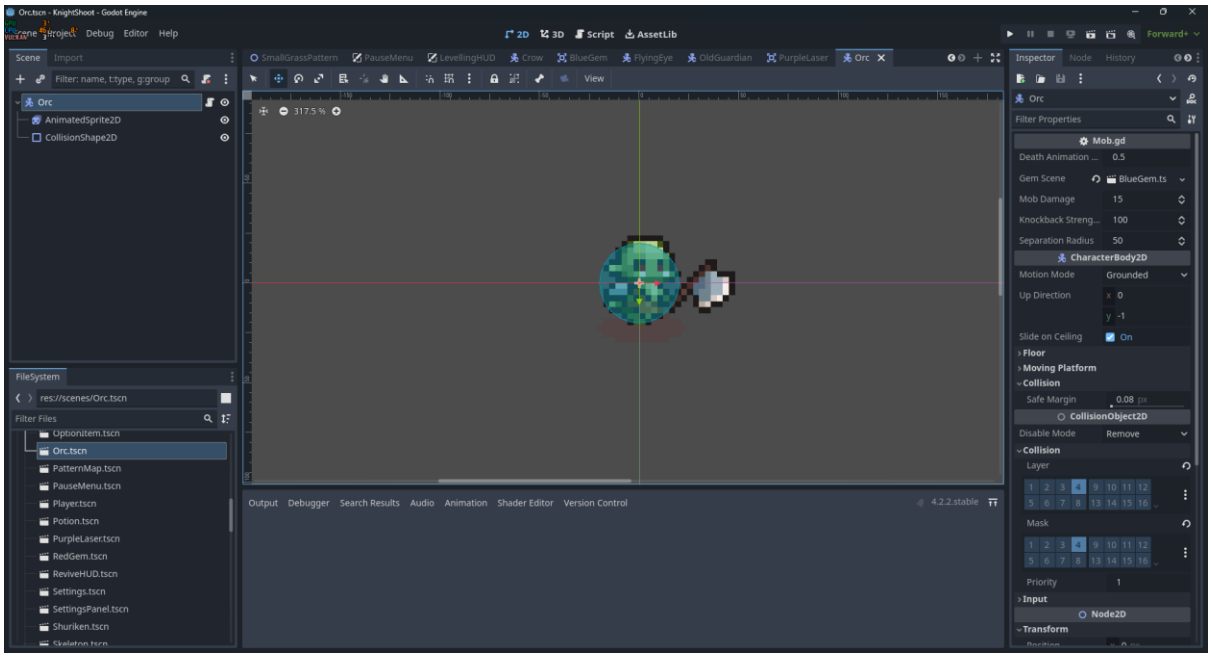

Figura 7 Imagem do inimigo Orc e seus nodes.

Para a funcionalidade dos *Orcs*, e necessário que recebam a posição do jogador e se movimentem em sua direção constantemente. Caso entrem em contato com ele, devem infringir dano. Para atingir essa funcionalidade, é recebida a posição global do jogador e utiliza-se uma função base da engine para direcionar a posição do *Orc* até a do jogador.

Na criação do script, foi feito um código modular e flexível que fosse capaz de ser utilizado para todos os possíveis tipos diferentes de inimigos que poderiam ser adicionados.

A detecção de colisões é feita nos scripts do jogador e da arma. Caso o *node* de colisão do *Orc* entre em contato com o *node* de colisão do jogador, o jogador leva dano e fica invulnerável por alguns milissegundos. Caso entre em contato com o disparo de uma arma, o *Orc* leva dano, se a vida estiver igual ou menor que zero, suas colisões são removidas e é utilizada uma animação para sua morte, seguindo da remoção do *Orc* do jogo. Já se o *Orc* for atingido e sua vida esteja maior que zero, é utilizada uma animação para indicar que foi atingido e sua funcionalidade continua igual, até ter sido eliminado ou ter eliminado o jogador.

Continuação do desenvolvimento

Após a implementação das funcionalidades descritas acima, e com um melhor entendimento da complexidade do projeto e suas funcionalidades. Foi feita a continuação do desenvolvimento, com a criação de todas as funcionalidades e interfaces necessárias para o jogo, e o sistema de áudio e configurações.

Sistema de ondas

Na continuação do desenvolvimento, foi feita a criação do sistema de ondas de inimigos que surgem automaticamente. Para isso, foram adicionados mais 2 tipos diferentes de inimigos,

os *Skeleton*s e os *Flying Eyes*. Funcionam exatamente como os *Orcs* e tem atribuído o mesmo script, para manter a funcionalidade.

O sistema possui um tempo total que vai sendo reduzido com o passar do tempo até a conclusão da partida ou a eliminação do jogador. Enquanto o tempo é maior que zero, inimigos surgem em um círculo a volta da posição do jogador. Foi decidido o uso de 10 minutos para a duração da partida, e para cada minuto que passa, surge uma nova onda de inimigos diferente da anterior, com valores diferentes para vida, tempo para o surgimento do próximo inimigo, quantidade máxima de inimigos para uma onda, quantidade máxima de inimigos na tela ao mesmo tempo e duração em segundos.

Implementação das interfaces

Para a criação das interfaces, foram utilizadas cenas diferentes para cada uma das interfaces do jogo. Em todas estas interfaces são utilizados estilos e layouts diferentes utilizando os *nodes* específicos para interfaces na engine.
Classes essas como **Control**, que possibilita o controle irrestrito dentro daquele espaço, como mudanças de tamanho, posição. É a classe base para todos os *nodes* relacionados à interface de usuário. Foram implementados diversos **VBoxContainer** e **HBoxContainer**, ambos containers usados para criar layouts verticais e horizontais, respectivamente. Foi utilizado também o *node* **PanelContainer**, para criar painéis com texturas, **TextureButton,** nodes de botões com texturas, entre outros nodes de interface da engine.

Menu Principal

Este menu funciona como base para o jogo. Nele é possível começar uma partida, acessar a página de aprimoramentos permanentes, acessar as configurações ou sair do jogo.

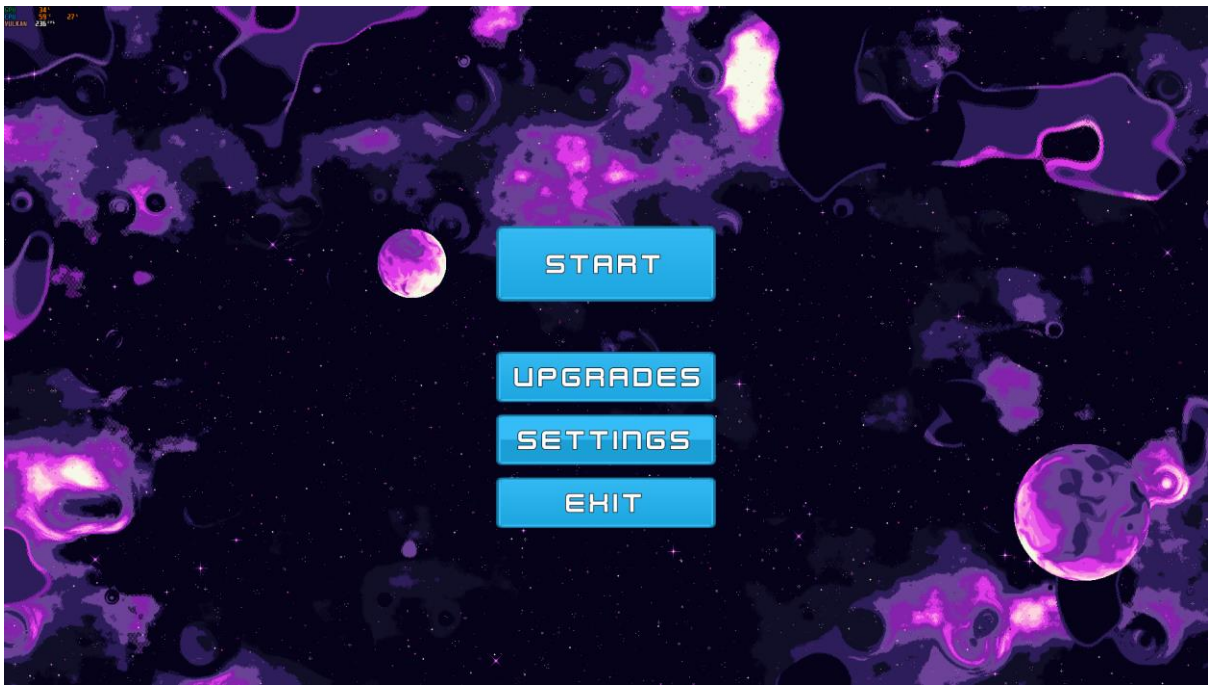

Figura 8 Imagem do menu principal.

Menu de aprimoramentos permanentes

Neste menu, podem ser feito os aprimoramentos permanentes com o ouro obtido durante as partidas. Ao clicar em um aprimoramento, sua descrição e preço surgem na parte inferior, sendo possível

também reembolsar os aprimoramentos feitos, no desejo de um desafio maior. Possui também um botão para retornar ao menu principal.

Figura 9 Imagem do menu de aprimoramentos permanentes sem nenhuma opção selecionada.

Figura 10 Imagem do menu de aprimoramentos permanentes com uma opção selecionada.

Figura 11 Imagem do menu de aprimoramentos permanentes com uma opção selecionada e no seu nível máximo.

Menu de configurações

Para o menu de configurações, foi utilizado um sistema um pouco diferente dos outros menus, devido a necessidade da utilização do mesmo painel dentro de uma partida. Para isso, são utilizadas três cenas diferentes. Uma para o painel e todas as suas características como a troca de resolução e a capacidade de alterar o volume do áudio. E as outras duas cenas, instanciam o painel na situação desejada, seja no menu, seja dentro da partida.

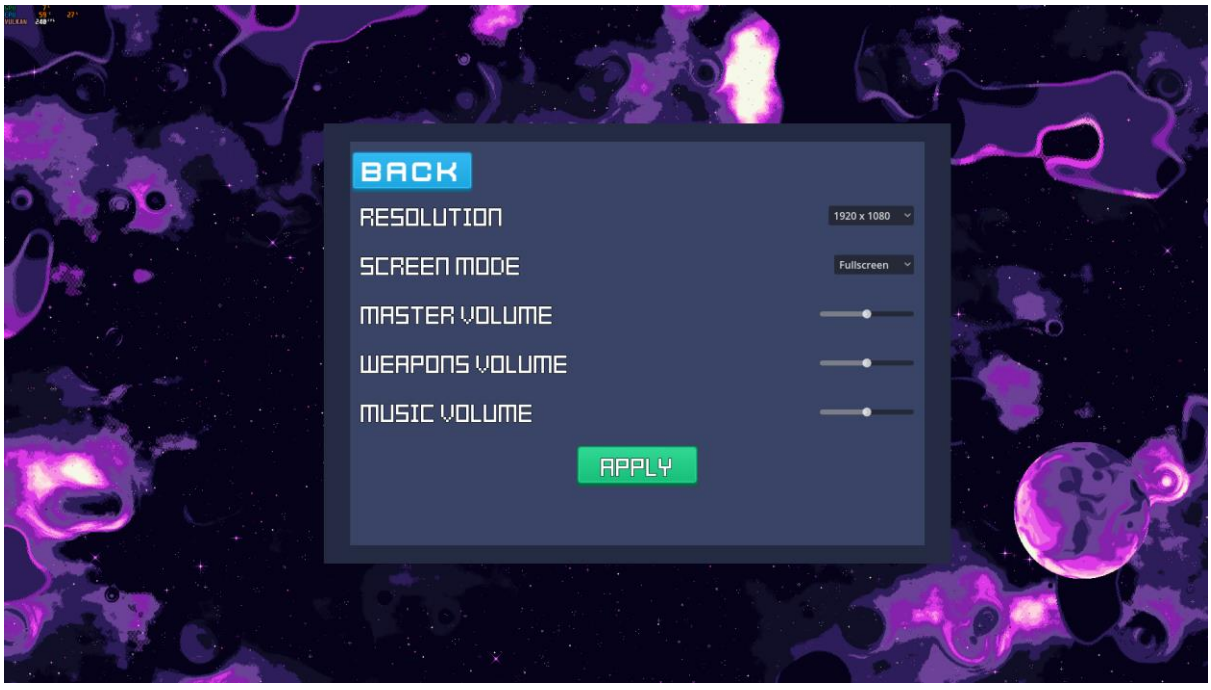

Figura 12 Imagem do menu de configurações.

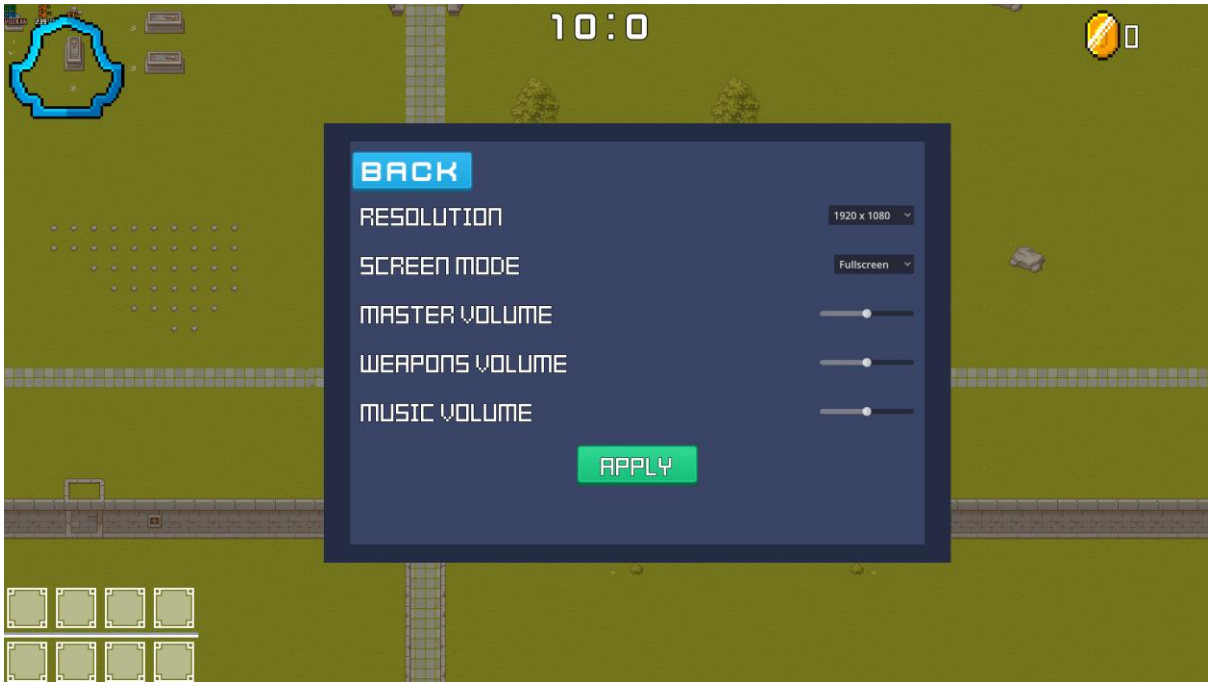

Figura 13 Imagem do painel de configurações sendo acessado dentro de uma partida.

Menu de pause

Para o menu de pause, todas as funcionalidades do jogo são pausadas, exceto o input pelo mouse. A seguir o painel de pause aparece com três opções, uma para continuar a partida, uma para acessar o painel de configurações e outra para terminar a partida e voltar ao menu.

Figura 14 Imagem do menu de pause durante uma partida.

Interface para subir de nível

Para esta interface, foi utilizado um sistema parecido com o sistema da interface de

configurações. São utilizadas duas cenas, uma para criar o estilo do painel e a outra é utilizada para popular as diferentes opções de aprimoramento. O jogador pode pular até três níveis, para tentar escolher as opções que deseja.

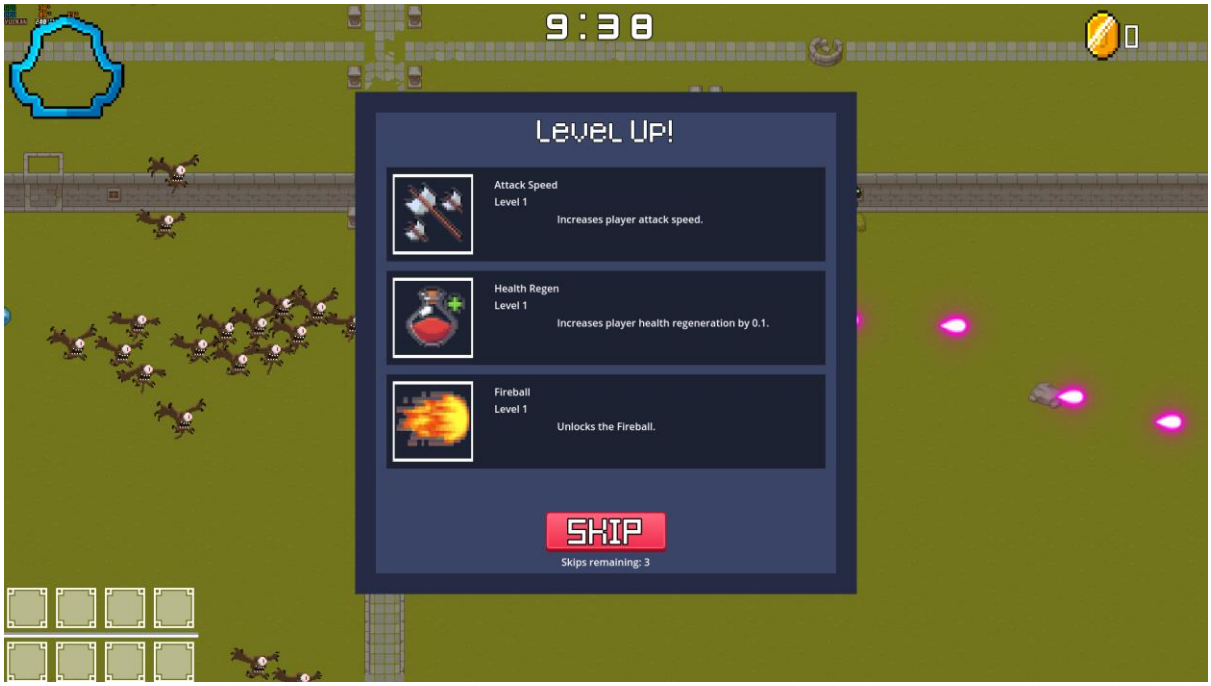

Figura 15 Imagem da interface de aprimoramento em uma partida.

Interface de fim de jogo

Esta é uma interface utilizada caso o jogador seja eliminado antes do tempo e não possua a opção de reviver ou sobreviva a duração completa da partida.

Figura 16 Imagem da interface de fim de jogo.

Interface de ressureição

Esta interface é utilizada caso o jogador possa o aprimoramento de ressurreição, possuindo

uma opção para o jogador reviver e outra caso o jogador desejar sair da partida.

Figura 17 Imagem da interface de ressureição.

Interface do jogo

Para a interface do jogo, possuímos um indicador do nível do personagem que vai sendo alterado dinamicamente com a experiencia coletada, a quantidade de ouro obtida durante a partida, um contador de 10 minutos até o fim da partida e uma pequena interface com as oito opções selecionadas pelo jogador, sendo as quatro opções de cima para as armas e as quatro opções de baixo para as estatísticas.

Figura 18 Imagem da interface durante uma partida.

Implementação das funcionalidades

Em conjunto com a implementação de cada uma das interfaces, foram implementadas as

funcionalidades necessárias para cada uma.

Menu Principal

O menu principal funciona como a base do projeto, pois é a primeira tela que aparece ao abrir o jogo e é o local de onde as partidas começam. Para a sua funcionalidade, é apenas necessário a conexão entre os botões e suas cenas respectivas.

Sistema de funcionalidades das partidas

Grande parte das funcionalidades necessárias estão sendo chamadas diretamente na cena “Game”. Entretanto, outras funcionalidades foram implementadas diretamente no script com o mesmo nome.

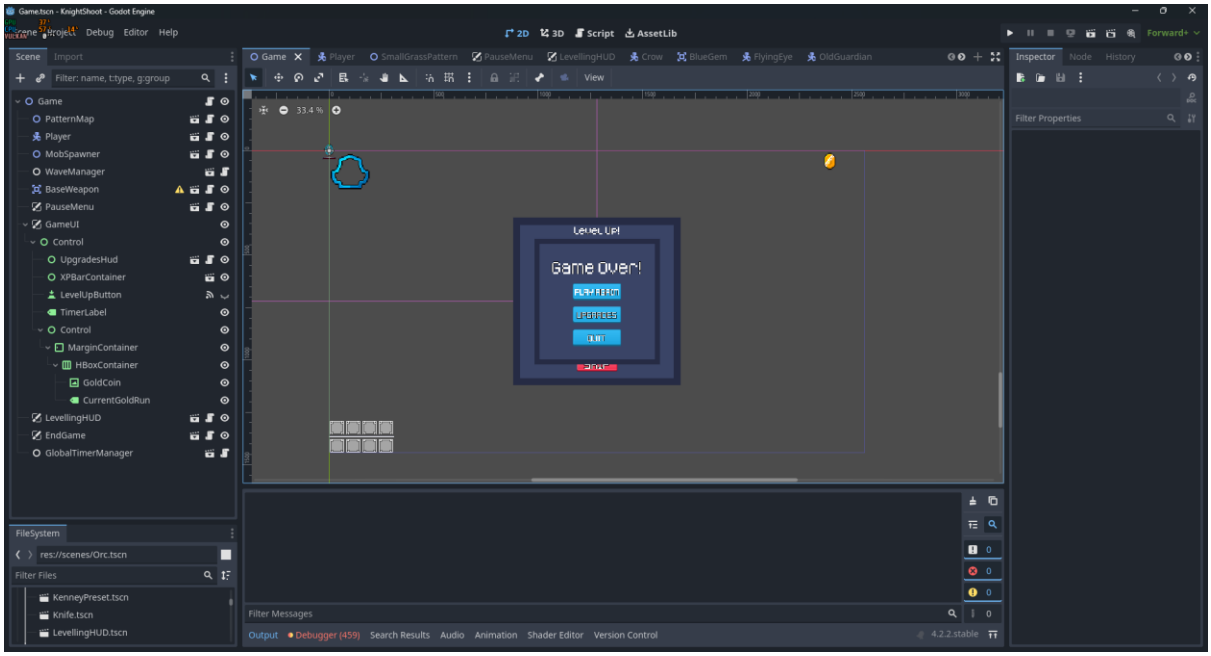

Figura 19 Imagem da cena "Game" e dos nodes instanciados.

Inicialmente foi utilizado um *node* **Timer** para controlar a duração do jogo, quando o tempo acaba, é chamada a função de fim de jogo, pausando todos as funcionalidades e mostrando a respectiva interface. Foi também implementada uma função para atualizar o *viewport* a cada frame, que serve para atualizar a sua posição em relação a posição da câmera que segue o jogador, mantendo os dois iguais.

Dentro do script que controla as funcionalidades da partida, são feitas as chamadas da interface de pause, caso o jogador pressione a tecla “ESC”, e a chamada para o jogador resumir a partida. O sistema de ouro é atualizado sempre que o jogador entrar em contato com uma das moedas de ouro durante a partida, no mesmo momento, a interface que mostra o ouro é atualizada com o valor coletado. Quando o jogo termina, o ouro coletado é adicionado ao ouro permanente e os dados do jogo são guardados. Foi implementado também um sistema para esconder o ícone do mouse durante a partida caso fique sem se movimentar por alguns segundos, o ícone só volta a aparecer caso se mova novamente, ou alguma das interfaces apareça.

Sistema de aprimoramentos permanentes, configurações e salvamento de dados

Neste sistema, foi utilizado juntamente com a interface, uma funcionalidade da engine

chamada “Autoload”, que serve para guardar informações que serão utilizadas em mais de uma cena. Neste *autoload*, são salvos os valores de cada um dos aprimoramentos, e seus valores máximos possíveis, juntamente com o preço em ouro necessário para cada um. Estes valores são passados tanto para a interface, que os atualiza sempre que o jogador faz uma alteração, quanto para o script do jogador, em que esses atributos são adicionados ao mesmo, dado os valores salvos no *autoload*.

Para o sistema de configurações, foi utilizado um script com funções já existentes na engine para mexer com os valores necessários, como a resolução e o áudio. O áudio foi separado em 3 canais diferentes. Um canal para manipular o áudio geral, um canal apenas para a música e um apenas para o som dos disparos. Foram adicionadas múltiplas resoluções, comportando valores entre 1440pX900p até 2560pX1440p.

Todos os dados mencionados acima, como valores de configurações, atributos permanentes e ouro são salvos em um arquivo durante múltiplas fases do jogo, em um arquivo “*JSON*”, estes dados são lidos quando o jogador inicia o jogo, atribuindo os valores corretos e mantendo as escolhas e progressão ao jogador.

Sistema de aprimoramentos durante a partida

Durante a partida, quando um dos inimigos morre, ele deixa uma pequena gema no local da morte. Quando o jogador passa pela gema, sua experiencia aumenta em um determinado valor. Foram implementadas três gemas diferentes que são deixadas por inimigos diferentes, e possuem valores diferentes. Caso o jogador chegue no valor necessário para passar de nível, a interface de seleção de opções é aberta, na qual o jogador pode selecionar uma entre três opções disponíveis, ou utilizar um botão para pular aquele nível em busca de opções diferentes, porém só pode utilizar até três por partida. O jogador possui quatro espaços para selecionar armas, e quatro espaços para atributos. Enquanto os oito espaços não foram preenchidos, novas opções serão mostradas em conjunto com opções já selecionadas. Quando o jogador estiver com os oito espaços ocupados, só é possível o aparecimento de alguma das opções já selecionadas. Já quando o jogador está com todas as escolhas no nível máximo, a interface de aprimoramento não aparece e o jogador recebe um pouco de ouro.

Sistema de armas

O jogo possui quatro armas: Laser Roxo, Shuriken, Faca e Bola de Fogo. Cada uma das armas possui funcionalidades, atributos e aparências diferentes.

Para o Laser Roxo, ele é a arma primaria do jogador e começa com ele em todas as partidas, enquanto para as outras armas o jogador deve evolui-las para nível 1 para que disparem. Inicialmente, atira apenas para a direita, mas nos níveis 1, 3 e 5, uma direção nova é adicionada, sendo elas esquerda, cima e baixo respectivamente. Já nos níveis 2 e 4, apenas os atributos de dano e velocidade da arma são aprimorados.
Para a Shuriken, quando desbloqueado, um temporizador é acionado com 5 segundos de duração. No fim da contagem, uma Shuriken surge a volta do jogador, orbitando-o em seu movimento e dando dano nos inimigos que entram em contato, sendo repetido o processo a cada 5 segundos. Para cada nível, uma Shuriken nova é adicionada a orbita. Já nos níveis 4 e 5, a velocidade de orbita é aumentada.

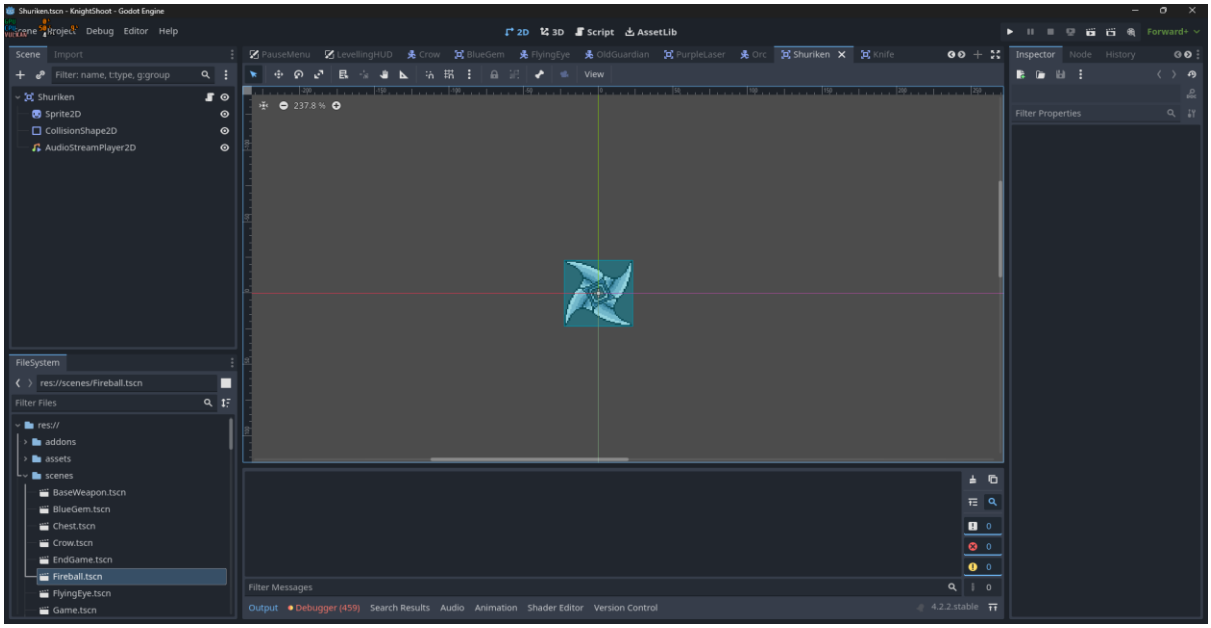

Figura 20 Imagem da arma Shuriken e seus filhos.

A Faca, quando desbloqueada, é disparada em uma direção aleatória, no momento que colide com o fim do viewport, ricocheteia no mesmo. Entretanto, é utilizado um temporizador para a duração de cada disparo, após o fim do temporizador, as facas disparadas são removidas, para não ficarem batendo infinitamente e manter o balanceamento. A cada nível desbloqueado, aumenta em um a quantidade de facas disparadas ao mesmo tempo, unido a isso, seu dano e velocidade são aumentados em todos os níveis.

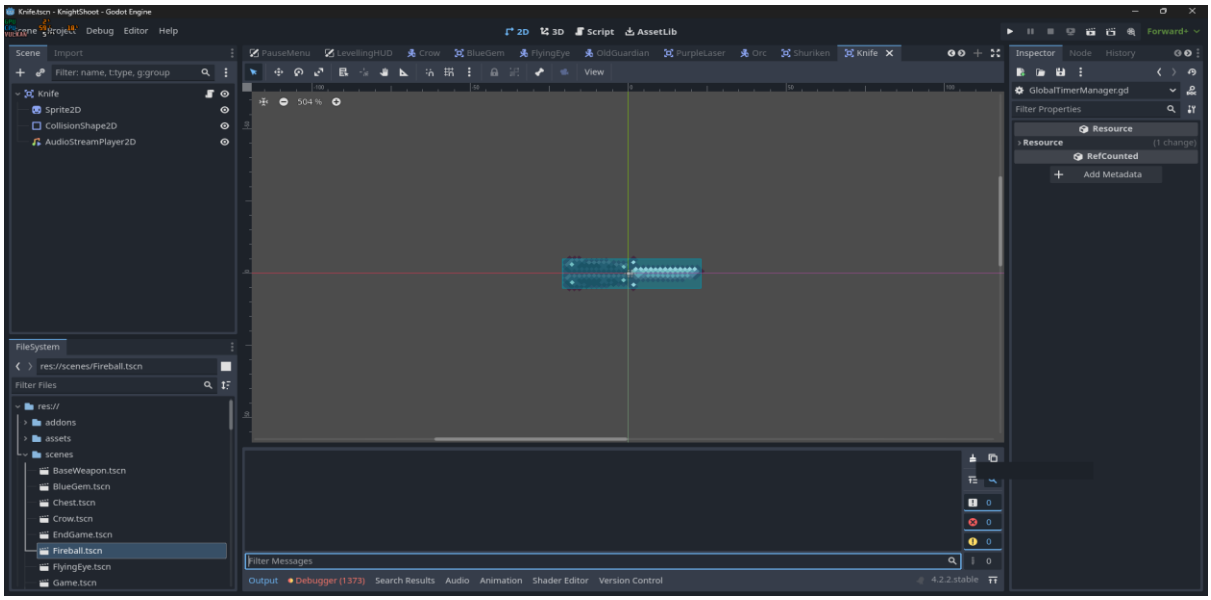

Figura 21 Imagem da arma Faca e seus filhos.

Por último, a Bola de Fogo é a única arma que possui uma animação, portanto foi implementado o *node* **AnimatedSprite2D**, ao invés de apenas um **Sprite2D**.
Quando desbloqueada, utiliza a localização global de um dos inimigos para disparar diretamente contra ele, portanto possui um sistema de mira automático.
Inicialmente, apenas uma Bola de Fogo é disparada, mas nos níveis 3 e 5, são adicionados mais um projetil por disparo, tendo três Bolas de Fogo disparadas no último nível.
Já nos níveis 2 e 4, recebe aumento no dano e na velocidade.

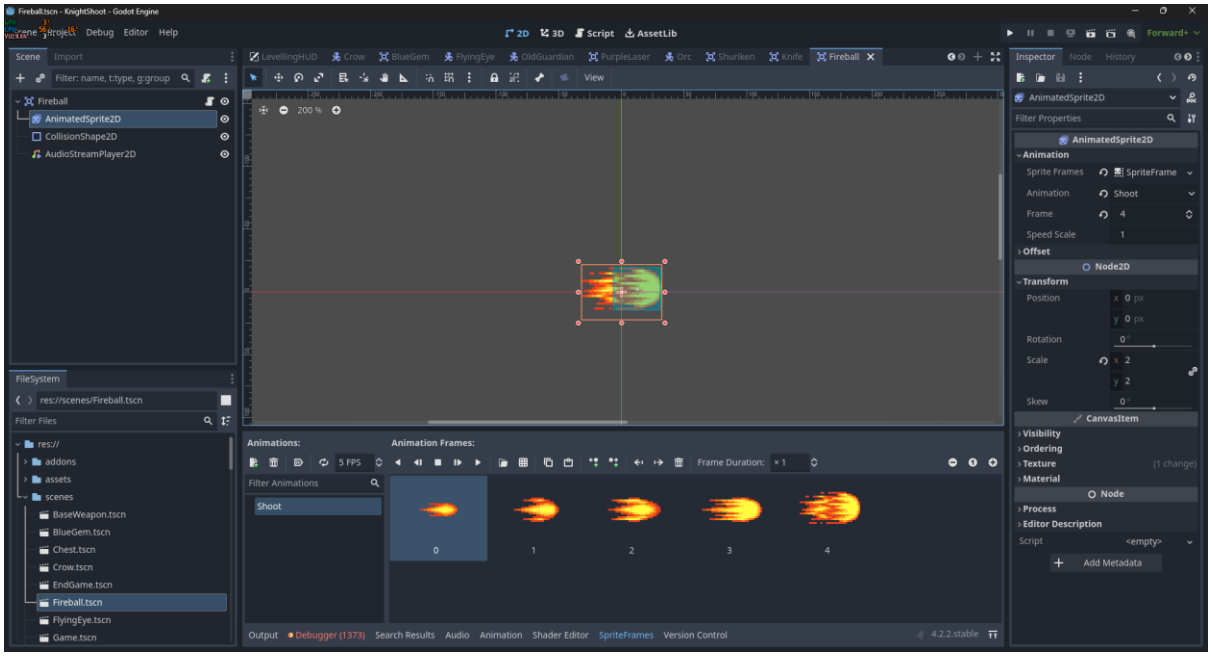

Figura 22 Imagem da arma Bola de Fogo e seus filhos.

Sistema de itens

Durante a partida, dois tipos diferentes de itens surgem aleatoriamente em um raio ao redor do jogador, os itens são: uma poção e uma moeda de ouro. Quando o jogador coleta o item, recebe um dos dois bônus, a poção restaura 20 de vida para o jogador, e a moeda de ouro dá um valor de ouro aleatório entre 20 e 60. Para balancear a quantidade de itens, são utilizados um valor de tempo mínimo e um valor de tempo máximo para o surgimento de um dos dois itens. Os itens vão surgindo aleatoriamente entre o tempo especificado durante toda a duração da partida.

Implementação do mapa

Para atingir o objetivo desejado de criar um mapa infinito e que fosse eficiente computacionalmente, foram utilizadas duas cenas diferentes. Em uma cena foi criado um mapa com um padrão específico, na segunda cena, o padrão vai sendo repetido enquanto o jogador se mexe, mantendo uma renderização otimizada. Foi utilizado um sistema de “chunks”, uma técnica para renderizar “pedaços” de um certo tamanho em uma certa quantidade, quanto mais pedaços renderizados, mais pesado fica computacionalmente. Portanto o algoritmo também descarrega os pedaços caso o jogador esteja a uma distância grande deles, mantendo a otimização do programa.

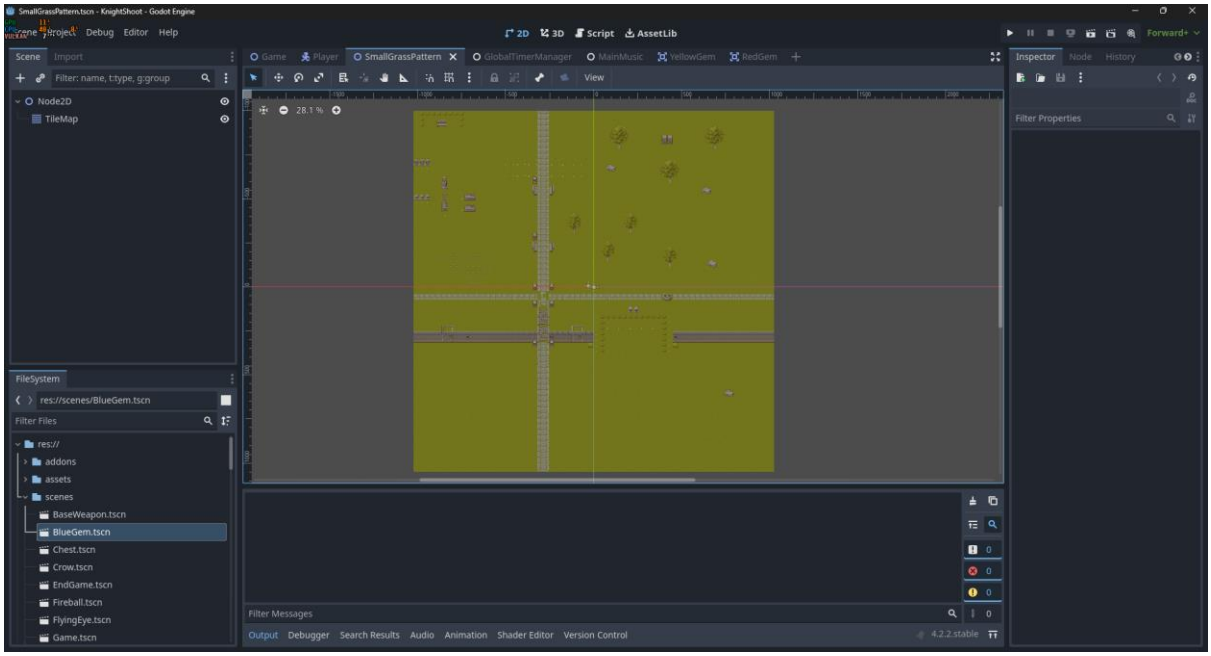

Figura 23 Imagem do padrão desenhado para o mapa.

Final do desenvolvimento

No processo de finalização do desenvolvimento do projeto, foram adicionados mais quatro inimigos, que são: Crow, Necromancer, Mushroom e OldGuardian, ao sistema de ondas dos inimigos, que só havia três tipos de inimigos diferentes até então. Todos foram implementados da mesma maneira que os inimigos anteriores e foram adicionados ao sistema de ondas.

Discussão

Com o desenvolvimento deste projeto, muitos dos objetivos desejados foram atingidos, criando uma jogabilidade dinâmica, satisfatória e leve, muito parecido com Vampire Survivors. Entretanto, algumas mecânicas poderiam ser expandidas e/ou melhoradas, como a funcionalidade da arma Shuriken. Foi notado durante testes que devido ao sistema de física da engine, a orbita da Shuriken no seu primeiro nível não parece estar exatamente como desejado, o que parece ser devido ao movimento do personagem. Durante testes, a Shuriken estava surgindo de um ponto fixo, no qual o movimento da sua orbita no primeiro nível estava perfeito, mas logo ao adicionar a física da Shuriken com a movimentação do jogador, no intuito de orbitar o mesmo, a sua orbita aparenta estar um pouco estranha. Nos níveis seguintes, com a adição de mais uma Shuriken, a sua orbita parece funcionar como desejado, mesmo com a movimentação do jogador. Uma dificuldade que não foi ultrapassada com o fim do desenvolvimento e que pode afetar diretamente a experiencia dos jogadores.

Outra mecânica que não atingiu a funcionalidade foi o sistema de música. Devido a maneira que a engine se comporta quando o jogo é pausado, ou seja, quando alguma das interfaces é chamada durante a partida, a música para juntamente com as outras mecânicas. Outro desafio que não foi ultrapassado e que afeta diretamente a experiencia dos jogadores, devido à quebra de ritmo durante a partida.

E por último, o maior desafio encontrado foi o balanceamento e conexão entre todos os

sistemas do jogo. Devido a complexidade do projeto, foi necessário um grande período de testes para atingir um resultado satisfatório. O balanceamento foi a parte mais difícil, pois foi necessário balancear todos os atributos e armas, junto com os atributos dos inimigos, quantidade de inimigos em cada onda e seu tempo de surgimento, quantidade de experiencia obtida em uma partida. Este foi um processo que necessitou bastante tempo e testes, mas que acabou tendo um resultado positivo e ocasionando uma experiencia melhor aos jogadores.

Entretanto, ainda há espaço para a expansão de alguns sistemas, como a adição de armas, personagens, mapas, inimigos, áudios, configurações e melhorias de qualidade de vida.

Conclusão

Em conclusão, o projeto atingiu os objetivos definidos com sucesso, demonstrando as capacidades da engine Godot na criação de um jogo auto-shooter. O seu desenvolvimento proporcionou muitos aprendizados e conhecimentos tanto em desenvolvimento e design de jogos como programação.

O projeto possui todas as funcionalidades de um jogo auto-shooter como Vampire Survivors e ainda abre espaço para novas expansões. Comprovando a capacidade do desenvolvimento de jogos independentes.

Glossário

Auto-Shooter: Um subgênero de jogos originado de outro subgênero chamado *Bullet Hell*, no qual o jogador apenas movimenta seu personagem pelo mapa, enquanto as armas do personagem disparam automaticamente durante as partidas, tendo como objetivo a sobrevivência de ondas de inimigos. Pode ser chamado também por *Bullet Heaven*.

Shoot 'em up: É um gênero de jogos em que o jogador deve disparar contra vários inimigos e evitando seus ataques.

Bullet Hell: Um subgênero de jogos no qual o jogador deve derrotar seus inimigos desviando de imensos projeteis ao mesmo tempo.

Indie: Abreviação de "independente", usada no mercado de jogos para se referir a jogos desenvolvidos por pequenos estúdios ou indivíduos, sem o apoio financeiro de grandes publicadoras.

Engine: Conjunto de ferramentas e bibliotecas que simplificam o desenvolvimento de jogos ou outros programas gráficos em tempo real, facilitando a criação de elementos como física, gráficos e som.

Open-Source: Aplicação com código fonte público, permitindo que desenvolvedores ao redor do mundo façam contribuições.

MOBA: Arena de Batalha Online Multijogador, é um gênero de jogos de estratégia onde jogadores controlam um personagem em um time, onde o principal objetivo é derrotar a base do time inimigo.

Spawn: Termo em inglês muito utilizado em jogos para se referir a um local de surgimento de objetos, sejam jogadores, objetos do mapa ou inimigos.

HUD: Interface gráfica que exibe informações importantes ao jogador durante o jogo, como vida, pontuação, mapa, entre outros.

Riot Games: Empresa norte-americana conhecida por desenvolver, publicar e organizar campeonatos de jogos extremamente populares, como *League of Legends*.

GDScript: Linguagem de programação de alto nível, orientada a objetos e imperativa, semelhante a Python. Otimizada e integrada com a engine Godot.

GIT: É um sistema de controle de versões, muito utilizado para desenvolvimento de aplicações.

Itch.io*:* Website que disponibiliza a compra ou download de jogos e assets para jogos ou outros projetos de renderização em tempo real.

Kenney: Website que disponibiliza assets gratuitos de domínio público para jogos ou outros projetos de renderização em tempo real.

Assets*:* Conjunto de artes, texturas e materiais utilizados na criação de jogos ou projetos de renderização em tempo real.

Autoload: Ferramenta utilizada na engine Godot para manter cenas carregadas durante o projeto todo, assim garantindo acesso a dados necessários em múltiplas partes de um projeto.

Node: Um node é um bloco de construção, são como ingredientes em uma receita. Podem ser atribuídos como filhos de outros nodes, criando uma arvore de nodes. Esta arvore é conhecida como uma Cena.

Cena*:* É o um conjunto de nodes, após salvo, pode ser utilizado como um filho de outro node.

Viewport*:* Termo utilizado para definir a região de uma tela utilizada para mostrar um conteúdo.

Bibliografia

- arithmetic1938. (2021). *Shooting a Bullet - Godot Tutorial* [Video]. Fonte: https://www.youtube.com/watch?v=2G41KECXXn4&t=1s

- Game Desgning. (7 de Outubro de 2023). *All About Shoot-Em-Up Games: History, Asteroids, Space Invaders*. Fonte: https://www.gamedesigning.org/gaming/shoot-em-up/

- Gdquest. (2023). *Your First 2D GAME From Zero with GODOT 4! **Vampire Survivor Style*** [Video]. Fonte: https://www.youtube.com/watch?v=GwCiGixlqiU

- Godot Engine Documentation. (n.d.). *Godot Docs – 4.3 branch — Godot Engine (stable) documentation in English*. Fonte: https://docs.godotengine.org/en/stable/

- GoTut. (6 de Junho de 2024). *Save and Load System for Godot 4 - Tutorial*. Fonte: https://www.gotut.net/save-and-load-system-for-godot-4/

- itch.io. (n.d.). *Download the latest indie games - itch.io*. Fonte: https://itch.io/

- JackieCodes. (n.d.). *Godot 4 Stardew Valley Tutorial* [Video Playlist]. Fonte: https://www.youtube.com/playlist?list=PLflAYKtRJ7dwtqA0FsZadrQGal8lWp-MM

- Kenney. (n.d.). *Assets · Kenney*. Fonte: https://kenney.nl/assets

- OpenGameArt.org. (n.d.). *OpenGameArt.org*. Fonte: https://opengameart.org/

- PlayWithFurcifer. (2022). *We made Vampire Survivors BUT in 10 Lines of Code* [Video]. Fonte: https://www.youtube.com/watch?v=LgvLbahJOTA

- Zenva GameDev Academy. (18 de Junho de 2024). *Welcome To Zenva - GameDev Academy*. Fonte: https://gamedevacademy.org/json-in-godot-complete-guide/