

Hugo Susumu Mota Asaga

**Trabalho de Formatura Supervisionado - Jogo
Digital.**

Brasil

2021

Hugo Susumu Mota Asaga

Trabalho de Formatura Supervisionado - Jogo Digital.

Jogo Digital 3D inspirado por jogos de captação de criaturas e RPGs clássicos.

Universidade de São Paulo – USP
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Orientador: Ricardo Nakamura

Brasil
2021

Hugo Susumu Mota Asaga

Trabalho de Formatura Supervisionado - Jogo Digital./ Hugo Susumu Mota Asaga.
– Brasil, 2021-

67p. : il. (algumas color.) ; 30 cm.

Orientador: Ricardo Nakamura

Monografia Final – Universidade de São Paulo – USP

Instituto de Matemática e Estatística

Bacharelado em Ciência da Computação, 2021.

1. Jogo Digital. 2. Videogame. 3. RPG. 4. Pokémon. 5. Dungeon Crawler. 6. Game Design. I. Ricardo Nakamura. II. Universidade de São Paulo. III. Instituto de Matemática e Estatística. IV. Trabalho de Formatura Supervisionado - Jogo Digital.

Este trabalho é dedicado a minha família e amigos por todo suporte que me deram durante os anos de graduação.

Agradecimentos

Este trabalho é resultado de todas as pessoas que me acompanharam e me deram apoio durante toda a minha formação acadêmica.

Agradeço aos meus pais por todo o apoio e carinho que me deram durante minha vida inteira.

Agradeço à minha namorada que sempre me deu forças para continuar seguindo em frente mesmo nos momentos mais difíceis.

Agradeço aos amigos por me darem apoio emocional e me ajudarem a descontrair quando mais precisei.

Agradeço aos professores que tive ao longo da vida por me incentivarem a buscar o conhecimento.

Resumo

O objetivo deste trabalho foi desenvolver o protótipo de um jogo digital, do gênero *Role-Playing Game* (RPG), *Dungeon Crawler*, que busca agradar os jogadores que gostam de jogos de captura e criação de criaturas, e jogos RPG que seguem convenções estabelecidas nas décadas de 1980-1990. A partir da análise de jogos bem avaliados pela crítica se criou o design base do jogo, o qual foi implementado durante a fase de desenvolvimento e refinado por meio de *feedbacks* recebidos durante o primeiro teste com público. Futuramente é possível levar a diante o desenvolvimento do jogo baseado nos sistemas já criados para adicionar mais conteúdo ao jogo.

Palavras-chave: Jogo Digital, Videogame, RPG, Pokémon, Dungeon Crawler, Game Design.

Listas de ilustrações

Figura 1 – Uma scene como apresentada no editor do Unity.	19
Figura 2 – Objects representados no editor do Unity.	19
Figura 3 – Objeto e seu menu de componentes.	20
Figura 4 – Uma prefab.	21
Figura 5 – Evolução da visão do mundo em Pokémon durante os anos.	24
Figura 6 – Evolução das batalhas em Pokémon durante os anos.	25
Figura 7 – Exploração de mundo nos primeiros jogos da série Shin Megami Tensei.	26
Figura 8 – Batalhas nos primeiros jogos da série Shin Megami Tensei.	27
Figura 9 – Evolução da exploração de mundo em Etrian Odyssey.	28
Figura 10 – Evolução das batalhas em Etrian Odyssey.	29
Figura 11 – Etapa de exploração de labirintos.	32
Figura 12 – Main Menu aberto durante a exploração.	33
Figura 13 – SubMenu Bag de ações contendo os itens adquiridos.	33
Figura 14 – Representação de itens espalhados dentro do labirinto.	34
Figura 15 – Interação com atalhos durante a exploração do labirinto.	35
Figura 16 – O minimapa apresenta uma representação do labirinto visto de cima.	35
Figura 17 – Conjunto de Ícones presentes no minimapa.	36
Figura 18 – Os stats de um demônio como representados em jogo.	37
Figura 19 – Base do sistema de batalha.	40
Figura 20 – Sistema de skills em funcionamento.	41
Figura 21 – Submenu para uso dos itens.	41
Figura 22 – Posição dos aliados e inimigos em uma batalha.	42
Figura 23 – A tela de final de batalha mostra a exp e itens ganhos.	42
Figura 24 – Ao aprender mais de 5 skills, o demônio deve esquecer uma.	42
Figura 25 – Linha evolutiva de um dos demônios do jogo.	43
Figura 26 – Template dos menus utilizados durante o jogo.	43
Figura 27 – Template de diversas interfaces criadas para o jogo.	44
Figura 28 – Editor do motor de jogos Unity.	45
Figura 29 – Workspace do GIMP.	45
Figura 30 – O objeto <i>OverworldPlayer</i> visto pelo editor Unity.	46
Figura 31 – Layout de um labirinto criado com Dungeon Map Doodler.	49
Figura 32 – Estrutura básica de uma labirinto de duas perspectivas diferentes.	50
Figura 33 – Visão ortogonal de um labirinto com terreno adicionado.	50
Figura 34 – Visão da cena de Batalha como é vista pelo editor.	51
Figura 35 – Respostas das perguntas de múltiplas escolhas.	61

Lista de tabelas

Tabela 1 – Tabela de resistências	38
Tabela 2 – Stats Modifiers Table	57

Listings

5.1	Trecho do arquivo JSON que contêm os encontros por zoneID	48
5.2	Exemplo da inserção de uma espécie de demônio dentro do jogo	52
5.3	Exemplo da definição de golpes aprendíveis por nível de um demônio	52
5.4	Exemplo de como uma <i>skill</i> é adicionada ao jogo	54
5.5	Exemplo de como um item é adicionado ao jogo	54

Sumário

Listings	11
1 INTRODUÇÃO	15
1.1 Objetivos	15
1.2 Metodologia	16
2 CONCEITOS	17
2.1 Jogos Digitais	17
2.2 Mecânica de jogo	17
2.3 Game Design	18
2.4 Game Engine	18
2.4.1 Scenes	18
2.4.2 Game Objects	19
2.4.3 Prefabs	20
2.4.4 Scripts	21
3 ESTUDO DE JOGOS RELEVANTES	23
3.1 Mecânicas Relevantes	23
3.2 Séries de Jogos Estudas	23
3.2.1 Pokémon	24
3.2.2 Shin Megami Tensei	26
3.2.3 Etrian Odyssey	28
4 DESIGN DO JOGO	31
4.1 Objetivos do Jogador	31
4.2 Obstáculos	31
4.2.1 Labirintos	32
4.2.1.1 Movimentação	32
4.2.1.2 Menus	32
4.2.1.3 Objetos Interativos	33
4.2.1.4 Mini-mapa	35
4.2.2 Batalhas	36
4.2.2.1 Inimigos	36
4.2.2.2 Stats	36
4.2.2.3 Types	38
4.2.2.4 Opções de combate	39

4.2.2.5	Skills e Itens	39
4.2.2.6	Status Effects	39
4.2.2.7	Posição dentro da equipe	40
4.2.2.8	Captura de Demônios	41
4.2.2.9	Progressão de personagens	41
4.3	Interface	43
4.4	Cidades	44
5	DESENVOLVIMENTO	45
5.1	Ferramentas	45
5.2	Assets	46
5.3	Arquitetura das Mecânicas de Exploração	46
5.3.1	Representação do jogador	46
5.3.2	Random Encounters	47
5.3.3	Labirinto	49
5.3.4	Overworld Menu System	51
5.4	Arquitetura das Mecânicas de Batalha	51
5.4.1	Arena de Batalha	51
5.4.2	Demônios	52
5.4.3	Calculo de Stats	53
5.4.4	Skills e Itens	53
5.4.5	Calculos de Dano	55
5.4.6	Precisão e Esquiva	56
5.4.7	Buffs e Debuffs	56
5.4.8	Condições Especiais	57
5.4.9	Captura de demônios	58
5.4.10	Sistema de Batalhas	58
5.5	Game Manager	59
5.6	Teste com jogadores reais	60
5.7	Melhorias feitas com base no <i>feedback</i> dos jogadores	62
6	CONSIDERAÇÕES FINAIS	63
6.1	Escopo	63
6.2	Trabalhos Futuros	63
7	CONCLUSÃO	65
	REFERÊNCIAS	67

1 Introdução

A indústria de jogos digitais é uma das indústrias mais valiosas do mundo contemporâneo, ao todo lucrando mais de 90 bilhões de dólares em 2020 ([DOBRILOVA, 2021](#)). Jogos digitais, assim como filmes, podem ser separados em diversos gêneros, dentre os quais pode-se destacar o RPG (role-playing game).

Um jogo RPG é um gênero de jogos digitais onde o jogador controla as ações de um ou muitos personagens imersos em um mundo bem definido, geralmente envolvendo alguma forma de desenvolvimento de personagem por meio de melhorias em atributos. Vários jogos RPG herdam muitas das suas terminologias, mecânicas e ambientação de jogos de RPG de mesa. ([WIKIPEDIA, 2021c](#))

Pokémon é a maior franquia de jogos de gênero de RPGs e uma das maiores franquias do mundo, mas apesar de ser extremamente bem sucedida, é criticada por muitos fãs, principalmente os mais velhos, por se manter presa a uma fórmula rígida e ser desenvolvida dando prioridade apenas para o público alvo infantil por meio de implementação de mecânicas que diminuem a dificuldade e tornam grande parte do jogo trivial para uma grande parte dos jogadores, tornando a experiência de jogar os jogos da franquia alienadora ao público mais velho que cresceu com a franquia.

1.1 Objetivos

A proposta deste trabalho é criar um protótipo de um jogo que possa atender as necessidades da parcela do público que gostaria de uma experiência semelhante a encontrada em Pokémon e de fãs de jogos RPG ao incorporar mecânicas clássicas de Pokémon e de jogos RPG considerados desafiadores em um mesmo produto.

Vale ressaltar que jogos digitais, por serem um entretenimento interativo, são softwares complexos compostos por diversos componentes, cada um responsável por um elemento diferente, tais como: interpretar inputs do jogador, reprodução audiovisual, simulação de mundos, cálculo de lógica de jogo, apresentação de interfaces, etc. Assim, para implementação do sistema será necessário a aplicação de conhecimentos adquiridos no curso de Ciência da Computação do IME-USP, principalmente nas disciplinas de Princípios de Interação Humano-computador, Algoritmos e Estruturas de Dados II, Design e Programação de Games, Atividade Curricular em Cultura e Extensão, e Técnicas de Programação I. Também serão aplicados os conhecimentos adquiridos durante 1 ano e meio de participação nas atividades extracurriculares do grupo de extensão USPGameDev.

1.2 Metodologia

Antes de começar o desenvolvimento foi feito um estudo de jogos de franquias de jogos digitais do gênero RPG e de suas mecânicas, em seguida se criou do design dos elementos fundamentais do jogo.

A partir do design básico do jogo, se deu início ao processo de desenvolvimento, onde foram implementados os personagens, ambientes, menus, interações entre os objetos, interfaces, sistemas, e reprodução de efeitos sonoros e visuais.

Com os elementos básicos do jogo já funcionais, foi criado um primeiro teste com jogadores reais, os quais forneceram o feedback utilizado para avaliar o estado do projeto.

A partir do feedback fornecido, foram feitas mudanças e melhorias no design do jogo permitindo a melhoria dos sistemas já desenvolvidos, assim como o desenvolvimento de novos sistemas, resultando na versão final do protótipo.

Todas as etapas de produção do jogo digital foram acompanhadas e validadas pelo professor orientador Ricardo Nakamura.

2 Conceitos

Neste capítulo serão apresentados os principais conceitos básicos de desenvolvimento de jogos, que serão importantes para o entendimento e discussão do projeto.

2.1 Jogos Digitais

Segundo Jesse Schell em *The art of game design*, jogos são atividades em que os participantes praticam por vontade própria; apresentam objetivos, conflitos, desafios e regras; possuem ser vencidas e perdidas; são interativas; engajam os participantes; e são sistemas fechados. O autor utiliza essas características para definir jogos em uma única frase: "Um jogo é uma atividade de resolução de problemas, abordada com uma atitude lúdica". ([SCHELL, 2014](#))

De acordo com Raph Koster, autor de *A Theory of Fun for Game Design*, jogos são quebra-cabeças a serem resolvidos, e a diversão de se jogar um jogo provém do domínio e compreensão do jogo, e o ato de resolver problemas é o que torna jogos divertidos. ([KOSTER, 2013](#))

Um jogo digital é um jogo que envolve a interação com uma interface gráfica de vídeo e um dispositivo de entrada, como um joystick, teclado, sensor de movimento ou controle, para gerar um feedback visual. Este feedback é mostrado em um dispositivo de vídeo, geralmente acompanhado por feedbacks de áudio e podendo ter outros tipos de feedback agregado dependendo do jogo. ([WIKIPEDIA, 2021d](#))

2.2 Mecânica de jogo

Mecânicas de jogos são o aspecto do game design, ou regras, as quais o jogador segue dentro do mundo do jogo. Mecânicas de jogo ditam como o jogador age dentro do jogo. Por exemplo, a mecânica de se ter missões e objetivos para a missão estimulam o jogador a seguir uma estrutura desenvolvida pelos designers. ([GAMEDESIGNING, 2021](#))

Algumas das mecânicas mais utilizadas em jogos RPG ,e que foram relevantes para o desenvolvimento do projeto, são o desenvolvimento de personagens jogáveis por meio de melhorias de atributos, batalhas em turnos, encontros aleatórios e exploração de ambientes.

2.3 Game Design

“Game Design é a arte de decidir o que o jogo deve ser”, ou seja, o game design é o ato de se fazer as decisões de como será o jogo. Para se decidir o que o jogo é, se deve fazer centenas, normalmente milhares de decisões. As decisões que se deve fazer durante a produção de um jogo, incluindo as regras, a aparência e sensação, cronologia dos eventos, ritmo, tomadas de risco, recompensas, punições, e qualquer outra coisa que o jogador experiencie é responsabilidade do game designer. Por essa razão o designer geralmente está envolvido com o desenvolvimento do jogo desde o começo até o fim do jogo, fazendo decisões durante o processo de criação do jogo. ([SCHELL, 2014](#))

Vale lembrar que “Designer” é um papel, não uma pessoa em específico. Todos que tomam decisões durante o jogo estão cumprindo o papel de um game designer. Quase todos desenvolvedores da equipe tomam decisões sobre o jogo apenas pelo fato de estarem criando conteúdo para o jogo. Por essa razão, não importando o cargo, todos desenvolvedores da equipe devem ter um entendimento básico dos princípios de design de jogo. ([SCHELL, 2014](#))

2.4 Game Engine

Uma Game Engine, ou Motor de Jogos, é um framework de software projetado para o desenvolvimento de jogos digitais, e geralmente inclui bibliotecas e programas de suporte relevantes. O termo Game Engine também pode se referir a um software de desenvolvimento que utilize este framework, tipicamente oferecendo uma grande variedade de ferramentas e recursos para desenvolvimento de jogos. ([WIKIPEDIA, 2021a](#))

Para este trabalho foi utilizado o motor de jogos Unity, por ser um motor de jogos poderoso para criação de jogos 3D, o motor de jogos mais popular atualmente ([GAMEDEVELOPER, 2021](#)), gratuito, por ser possível exportar jogos para mais de 19 plataformas diferentes, e por conter a maior quantidade de recursos de aprendizagem disponíveis gratuitamente na internet.

O Unity, como outros arcabouços de software, estabelece uma arquitetura a partir da qual os jogos devem ser desenvolvidos. Neste caso, existem quatro elementos centrais que precisam ser compreendidos: Scenes, Game Objects, Prefabs e Scripts. Eles serão detalhados nas próximas subseções.

2.4.1 Scenes

Scenes é o local onde se trabalha com o conteúdo do jogo dentro do Unity. Eles são os assets que contém todo ou parte do jogo, sendo que um jogo pode ter quantas scenes

forem necessárias com no mínimo uma. A figura 1 ilustra como uma scene é vista dentro do editor Unity.

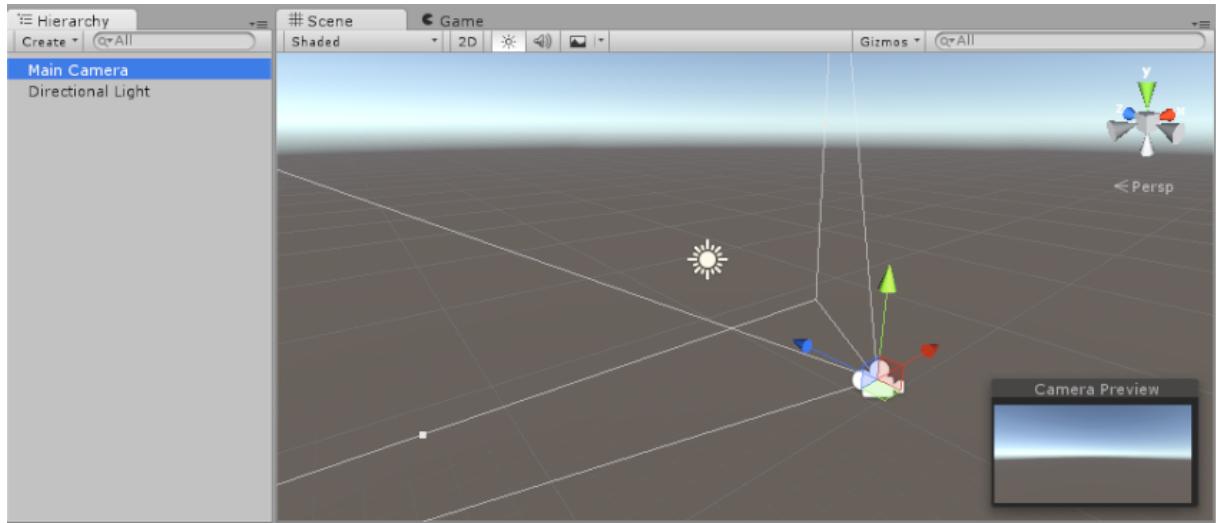


Figura 1 – Uma scene como apresentada no editor do Unity.

2.4.2 Game Objects

Game Objects 2 são os objetos fundamentais no Unity que representam personagens, adereços e cenários, todos objetos de um jogo Unity são Game Objects. Game Objects não apresentam nenhuma funcionalidade própria, mas são containers para Componentes que dão funcionalidade e propriedades a um Game Object.



Figura 2 – Objects representados no editor do Unity.

Dependendo do tipo de Game Object que se deseja criar no Unity, se deve utilizar uma combinação diferente de componentes 3. O Unity apresenta vários tipos de componentes pré-programados, mas é possível se criar novos utilizando a API de script do Unity em C#.

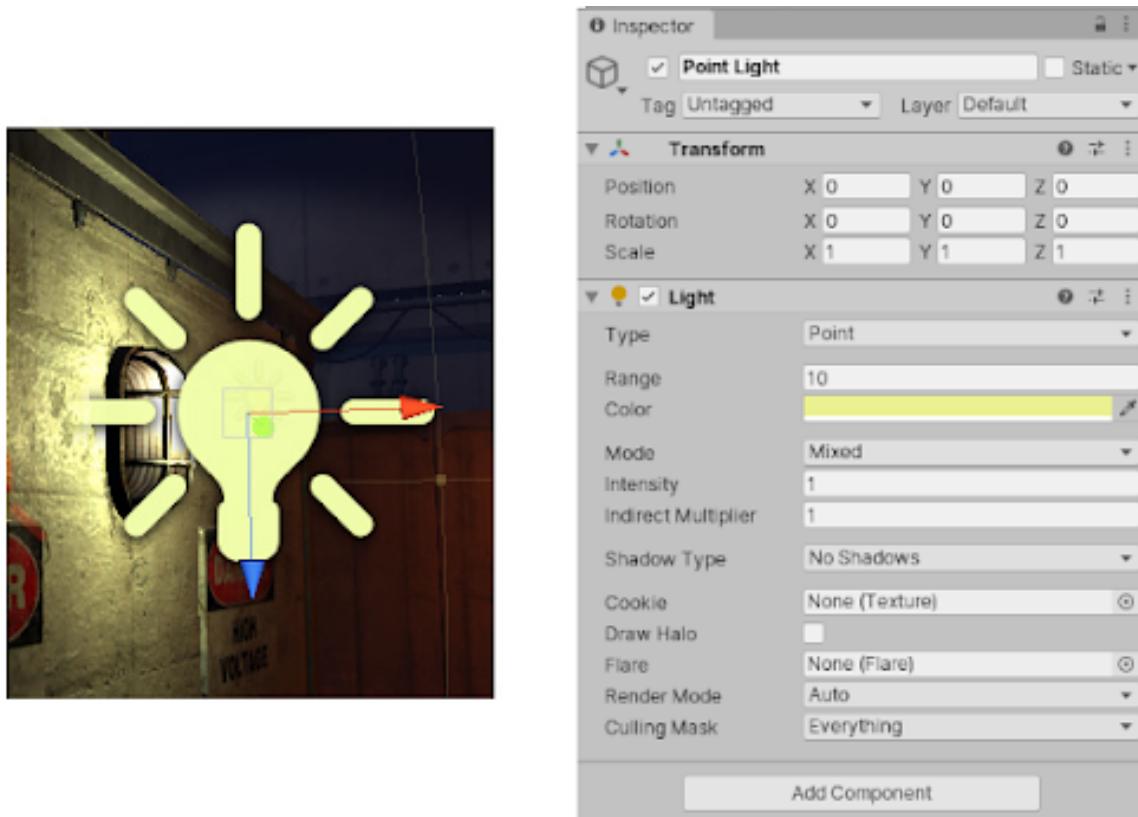


Figura 3 – Objeto e seu menu de componentes.

2.4.3 Prefabs

O sistema de Prefabs permite que o desenvolvedor crie, configure e armazene Game Objects completos com todos seus componentes, propriedades e gameObjects acoplados como um asset reutilizável. O Prefab [4](#) age como um template com o qual o desenvolvedor pode criar novas instâncias dentro de uma Scene.

O sistema de Prefabs é melhor utilizado quando se precisa utilizar o mesmo GameObject múltiplas vezes dentro de uma Scene, ou em múltiplas Scenes dentro de um projeto, pois o uso de Prefabs permite que sejam feitas modificações síncronas em todas suas instâncias a partir de modificações no Prefab original.

Qualquer modificação que se faz em um Prefab é automaticamente refletida em suas instâncias, mas modificações em instâncias do Prefab não afetam as outras instâncias ou o Prefab, possibilitando que as instâncias sejam customizadas de acordo com as necessidades.

Outro utilidade importante de Prefabs é a instanciação de GameObjects em tempo de execução, o que pode ser utilizado para adicionar efeitos especiais, power ups, ou projéteis à Scene durante a execução do jogo.

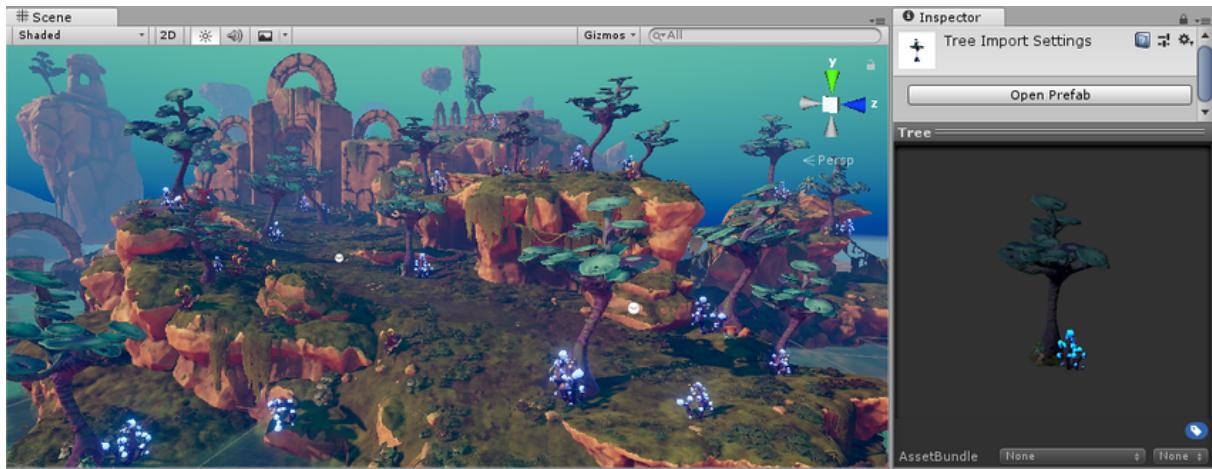


Figura 4 – Uma prefab.

2.4.4 Scripts

O Unity permite que sejam criados novos Componentes por meio de scripts em C#, permitindo assim que o desenvolvedor modifique e adicione propriedades a GameObjects da maneira que desejar.

Um script se conecta ao funcionamento interno do Unity pela implementação de uma classe que deriva da classe interna chamada `MonoBehaviour`, o que permite que scripts sejam acopladas a GameObjects como Componentes.

3 Estudo de Jogos Relevantes

Neste capítulo serão apresentados as principais mecânicas básicas dos gêneros Coleção de Criaturas, *RPG*, e a análise dos jogos que foram estudados como base para o desenvolvimento do projeto.

3.1 Mecânicas Relevantes

As quatro mecânicas mais relevantes consideradas para a produção do projeto foram: Exploração de labirintos, encontros aleatórios, progressão por níveis e a captura de monstros, pois são as mecânicas que formam o pilar base para os jogos da franquia Pokémon e de outros RPGs similares.

A mecânica de Exploração de Labirintos consiste em dividir cada área do jogo em diversos labirintos, onde o jogador deve atravessar para continuar o jogo. O fator diversão desse tipo de mecânica deriva da exploração do ambiente, da batalha contra inimigos encontrados pelo caminho, de descobrir segredos presentes nos labirintos e a sensação de conquista ao superar os obstáculos presentes no labirinto.

As batalhas em encontros aleatórios são obstáculos que os jogadores podem encontrar aleatoriamente ao se movimentar pelos ambientes do jogo. Durante os encontros aleatórios, jogadores precisam derrotar inimigos aleatórios para continuarem a progredir em direção ao objetivo principal. O fator diversão desta mecânica deriva da satisfação de ganhar uma batalha, e das possíveis recompensas que podem ganhar ao superar cada batalha.

A progressão de força se refere a capacidade do jogador de se tornar mais forte durante o jogo, geralmente por meio de aquisição de recursos como equipamentos melhores e itens, por meio de novas habilidades ou por melhoria dos atributos dos personagens jogáveis.

A captura de monstros é a mecânica que permite que o jogador capture criaturas para utilizá-las para progredir no jogo.

3.2 Séries de Jogos Estudas

Durante esta seção será apresentado uma breve análise dos jogos estudados com o funcionamento das principais mecânicas de jogos que foram escolhidos para análises. Uma explicação mais profunda destas mecânicas serão apresentadas durante o capítulo 5, onde as análises foram fundamentais para a implementação do jogo.

3.2.1 Pokémon

Pokémon ([COMPANY, 2021](#)) é a franquia de jogos mais bem sucedida da história, com mais de 90 bilhões de dólares arrecadados em vendas, seus jogos se destacam por serem acessíveis, fáceis de aprender, e terem um forte apelo ao público infantil.

Na maioria dos jogos *mainline* de Pokémon a exploração do mundo é feita por meio de uma perspectiva *top-down* [5](#), uma perspectiva onde o jogador enxerga o mundo de um ponto de vista acima do personagem que controla. No mundo explorável do jogo, o jogador pode descobrir novas áreas, segredos, itens espalhados, além de poder entrar em batalhas contra pokémons selvagens ou treinadores pokémons espalhados.

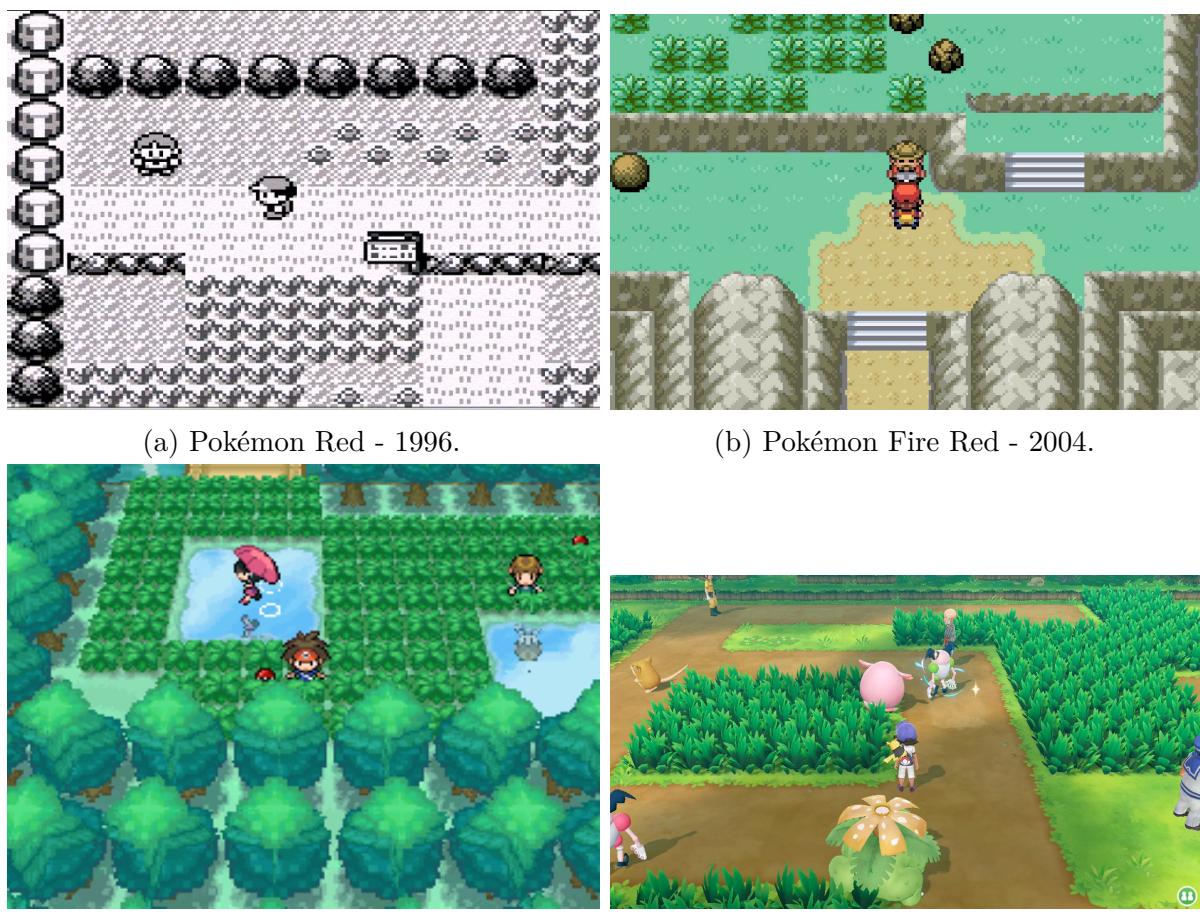


Figura 5 – Evolução da visão do mundo em Pokémon durante os anos.

Este tipo de exploração de mundo é simples de se desenvolver, fácil para jogadores se entenderem e se localizarem, porém a imersão dentro do mundo do jogo pode ser mais rasa para alguns jogadores.

As batalhas em pokémon [6](#) podem ocorrer de várias maneiras durante a exploração do mundo, sendo que o tipo mais comum é contra Pokémons selvagens encontrados randomicamente durante a exploração do mundo.

Batalhas normalmente ocorrem no formato 1 contra 1, onde um dos pokémons do jogador troca turnos com um dos pokémons adversários até que os *Hit Points* ou *HP* de um deles cheguem a zero.



Figura 6 – Evolução das batalhas em Pokémons durante os anos.

O sistema de batalhas é simples e fácil de se aprender, possibilitando que até mesmo crianças que não saibam ler consigam se divertir com o jogo, ao mesmo tempo que apresenta mecânicas complexas e profundas, de modo que possa agradar fãs mais avançados. O problema deste modelo é que o design do jogo considera apenas os jogadores mais casuais da série, resultando em um jogo fácil onde não é necessário o uso eficiente das mecânicas de combate mais complexas e profundas que o jogo oferece.

3.2.2 Shin Megami Tensei

Shin Megami Tensei ([WIKIPEDIA, 2021b](#)), também conhecidos pela sigla SMT, é uma franquia de jogos que, em sua maioria, consistem de jogos RPG de exploração de labirintos e negociação com demônios com os quais se podem fazer alianças para utilizá-los em batalhas.

A exploração do mundo dos jogos SMT é feita de diferentes maneiras atualmente, mas esta análise se baseará nos jogos mais antigos, onde a exploração era feita por meio de exploração em primeira pessoa de diversos labirintos que formam cada pedaço do jogo [7](#), durante essa exploração jogadores podem encontrar itens, segredos, e demônios recrutáveis que podem ser utilizados para progredir dentro do jogo.



(a) Digital Devil Story: Megami Tensei
- 1987.

(b) Shin Megami Tensei - 1992.



(c) Shin Megami Tensei II - 1994. (d) Shin Megami Tensei IF PS1 - 2002.

Figura 7 – Exploração de mundo nos primeiros jogos da série Shin Megami Tensei.

A exploração em primeira pessoa aumenta a imersão do jogador dentro do jogo, pode ser utilizada para criar uma sensação de maior claustrofobia, desorientação, e solidão, o que é utilizado para a criação da atmosfera e ambientação do jogo. A principal desvantagem deste tipo de exploração é a dificuldade de se localizar e a grande quantidade de batalhas

que se deve enfrentar durante o percurso dos labirintos, o que pode se tornar uma atividade frustrante ao invés de recompensadora.

As batalhas em SMT ocorrem via encontros aleatórios durante a exploração de labirintos, o jogador deve utilizar seu grupo de aliados, podem ser ou não ser demônios, para derrotar os grupos de inimigos ⁸. Durante o combate, o jogador tem a opção de negociar com demônios para que eles se tornem aliados.



(a) Digital Devil Story - 1987.

(b) Shin Megami Tensei - 1992.



(c) Shin Megami Tensei II - 1994.

(d) Shin Megami Tensei IF PS1 - 2002.

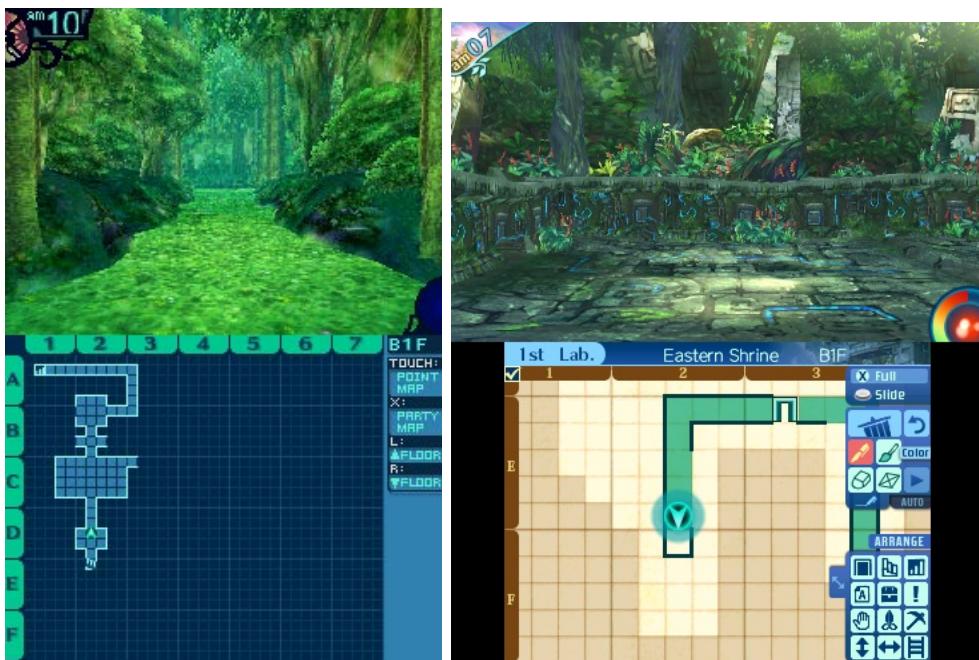
Figura 8 – Batalhas nos primeiros jogos da série Shin Megami Tensei.

O sistema de batalhas dos jogos SMT é bem complexo, com várias opções para o jogador utilizar estrategicamente e derrotar os diversos demônios e inimigos encontrados durante o jogo. A alta dificuldade das batalhas do jogo requer que o jogador utilize de todos os recursos disponíveis de maneira eficiente para progredir, tornando assim muito satisfatório e divertido a aprendizagem e eventual maestria do sistema de combate. A desvantagem deste modelo de batalhas é a necessidade do jogador dedicar tempo para entender todos os sistemas presentes, o que pode ser desencorajador para muitos jogadores mais casuais.

3.2.3 Etrian Odyssey

Etrian Odyssey ([WIKI, 2021](#)) é uma série de jogos de RPGs de exploração de labirintos desenvolvido pela Atlus, a mesma desenvolvedora da série de jogos Shin Megami Tensei, apresenta uma jogabilidade semelhante à série SMT mas com um foco bem maior em exploração de ambientes.

A exploração do mundo em Etrian Odyssey é um dos principais focos do jogos, o jogador se move por labirintos com diversas temas diferentes, devendo criar seus próprios mapas por meio da tela touchscreen do Nintendo DS, e durante a exploração o jogador pode encontrar com diversos monstros, armadilhas, atalhos, itens, chefes e eventos especiais [9](#).



(a) Etrian Odyssey - 2007.

(b) Etrian Odyssey Nexus - 2018.

Figura 9 – Evolução da exploração de mundo em Etrian Odyssey.

As batalhas em Etrian Odyssey podem ocorrer por meio de encontros aleatórios durante a exploração do ambiente, ou por interação direta com monstros presentes nos labirintos [10](#). O jogo usa um sistema de classes semelhante aos presentes em RPGs de mesa tradicionais, onde o jogador deve montar um time de até 5 personagens, cada um com uma especialidade e habilidades diferentes. O sistema do jogo permite que o jogador crie cada personagem de maneira independente de forma que o time tenha as capacidades de ataque, defesa e suporte necessárias para conseguir derrotar todos os inimigos que encontrar em cada labirinto.



(a) Etrian Odyssey - 2007.

(b) Etrian Odyssey Nexus - 2018.

Figura 10 – Evolução das batalhas em Etrian Odyssey.

4 Design do Jogo

O jogo foi dividido em duas partes principais, a exploração de labirintos, onde o jogador percorre o mundo em direção a um objetivo, e as batalhas por encontros aleatórios que podem ocorrer durante a exploração. Por meio do estudo dos jogos selecionados, o jogo foi desenvolvido implementando mecânicas desses diferentes jogos de maneira a criar o protótipo de um produto novo.

Este capítulo é dedicado a uma visão geral de como foi feito o design do jogo e de como cada mecânica presente no jogo funciona, detalhes de implementação e funcionamento das mecânicas serão apresentados no capítulo 5.

4.1 Objetivos do Jogador

Objetivos em jogos servem para focar o jogador em uma determinada tarefa, dão propósito ao jogador, e servem como uma forma de guiar a experiência do jogador, no caso do protótipo feito, o objetivo apresentado ao jogador no começo do jogo é simples, entregar um pacote dado ao jogador por um NPC (*Non-Player Character*) na cidade inicial a um NPC em uma outra cidade, e para chegar nesta cidade o jogador deve atravessar uma floresta. Apesar de parecer um objetivo muito simples e genérico, a real motivação para o jogador continuar jogando é a curiosidade de descobrir novos ambientes, novos monstros para interagir, e novas habilidades.

O objetivo inicial de entregar o pacote de uma cidade a outra se traduz em vários pequenos objetivos intermediários durante o jogo, tais como: ficar mais forte para progredir; conseguir novos aliados; encontrar o caminho pelo labirinto; encontrar itens úteis; e desenvolver estratégias para otimizar batalhas.

4.2 Obstáculos

Obstáculos em jogos são os elementos que desafiam o jogador, que testam sua capacidade de resolver problemas, e que testam a habilidade do jogador de usar os recursos presentes no jogo.

Neste protótipo os obstáculos são os labirintos que representam o ambiente explorável do jogo, e as batalhas aleatórias contra inimigos encontrados durante a exploração do ambiente.

4.2.1 Labirintos

A escolha de se ter a exploração do mundo em forma de labirintos em primeira pessoa partiu da limitação de assets disponíveis para se criar um vasto mundo que fosse interessante para o jogador. O uso de labirintos semelhantes aos encontrados em Shin Megami Tensei e Etrian Odyssey permitiu que se fosse usado poucos assets visuais para se criar um mapa grande e interessante para os jogadores explorarem [11](#).



Figura 11 – Etapa de exploração de labirintos.

4.2.1.1 Movimentação

O jogador tem três opções de movimento: dar passos na direção da câmera, rotacionar em incrementos de 90° ao redor do eixo vertical, e passos para direita e esquerda. A cada passo o que o jogador executa é incrementado um contador que decide se o jogador encontrou inimigos para iniciar uma batalha ou não.

4.2.1.2 Menus

Durante a exploração do labirinto o jogador pode abrir um Menu dividido em várias opções: *Party* (Equipe), *Bag* (Bolsa), *Map* (Mapa), e Save [12](#). Apenas as funcionalidades de *Party* e *Bag* foram implementadas por serem as relevantes ao funcionamento do protótipo.

A opção *Party* permite que o jogador veja informações dos demônios em seu grupo atual e os selecione para executar ações por meio de um novo menu que permite executar ações com o demônio selecionado. Essas ações são: Dar apelido ao demônio; Trocá-lo de posição no grupo; e utilizar habilidades [13](#).

A opção *Bag* permite que o jogador use e veja informações sobre os itens que adquirir durante o jogo, uma caixa de mensagem é aberta e apresenta as informações do item atualmente selecionado.

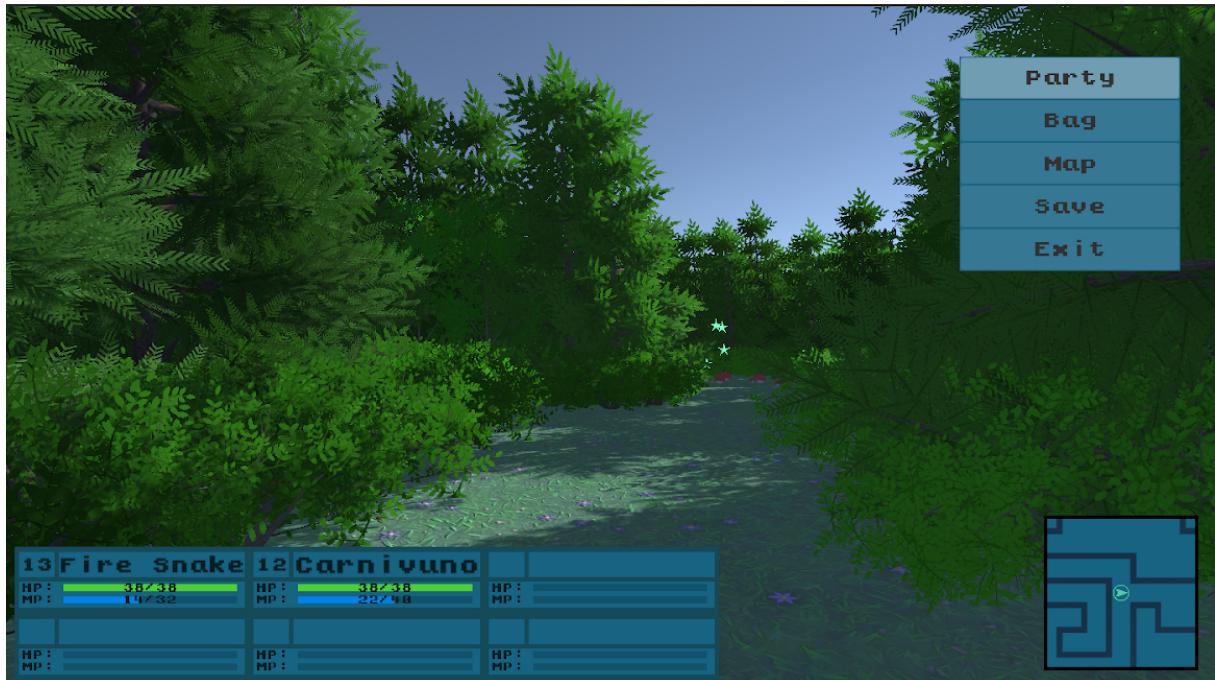
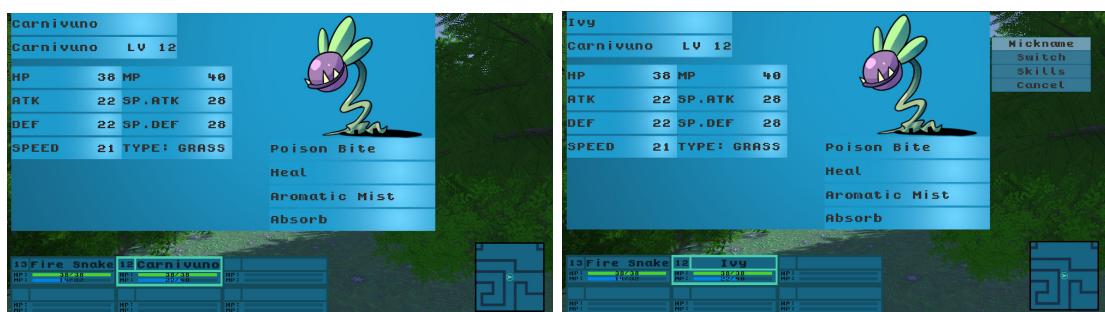


Figura 12 – Main Menu aberto durante a exploração.



(a) O jogador escolhe entre um dos membros da equipe após selecionar "Party".

(b) Um submenu de ações do personagem escolhido.

(c) SubMenu de *skills* do personagem.

(d) Um submenu de ações é aberto após a escolha de "Bag".

Figura 13 – SubMenu Bag de ações contendo os itens adquiridos.

4.2.1.3 Objetos Interativos

Para deixar a exploração dos labirintos mais interessante e recompensadora para o jogador, foram criados objetos com os quais o jogador pode encontrar e interagir dentro dos labirintos.

Itens podem ser encontrados espalhados pelo labirinto ??, indicados por partículas que saem do chão e por um ícone presente no minimapa no canto inferior direito. Partículas foram escolhidas para indicar a localização de itens devido a limitação de *assets* disponíveis.



(a) Estrelas representam a localização de itens coletáveis.
(b) Uma caixa de mensagem é aberta ao se pisar sobre o item.

Figura 14 – Representação de itens espalhados dentro do labirinto.

Em jogos que apresentam ambientes em forma de labirintos e requerem que o jogador volte ao começo do labirinto como ocorre em jogos das franquias *Etrian Odyssey* e *Dark Souls*, o jogo garante que o jogador possa liberar atalhos para alcançar as áreas previamente alcançadas de maneira mais rápida e fácil, garantindo assim que o jogador não perca o sentimento de progresso mesmo que precise voltar ao começo do labirinto. Por essa razão foram criados atalhos semelhantes aos encontrados em *Etrian Odyssey*, que permitem que o jogador atravesse paredes do labirinto.

Os atalhos começam permitindo a passagem apenas por um dos lados da parede, mas após serem utilizados pelo menos 1 vez, eles passam a permitir a travessia por ambos os lados da parede. Os atalhos devem ser algo que o jogador deve procurar ativamente, assim por design os locais com atalho são indicados apenas por pequenos indicadores visuais (como colônias de cogumelo crescendo juntos) 15, ou pelo próprio design do labirinto em si.

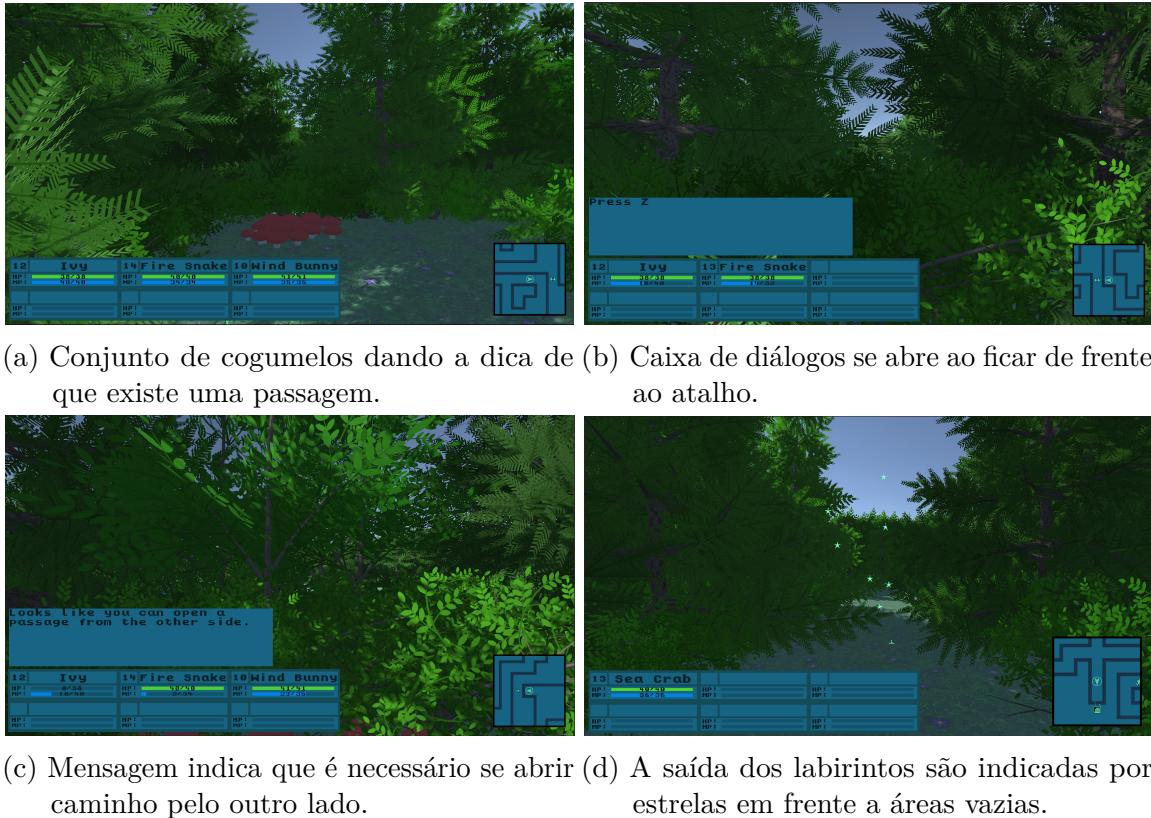


Figura 15 – Interação com atalhos durante a exploração do labirinto.

4.2.1.4 Mini-mapa

A visão em primeira pessoa torna muito desorientador trafegar pelo labirinto portanto para melhorar a capacidade de localização durante a exploração dos labirintos, foi criado um mini mapa 16 que apresenta a visão superior do labirinto com ícones 17 que indicam objetos interativos e que não rotaciona junto com o jogador, permitindo que o jogador mais facilmente se localize e não fique andando em círculos, o que poderia se tornar um fator muito frustrante dentro do jogo.



Figura 16 – O minimapa apresenta uma representação do labirinto visto de cima.

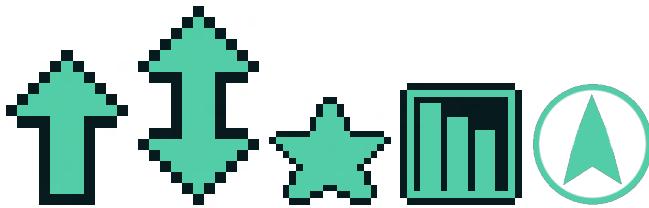


Figura 17 – Conjunto de Ícones presentes no minimapa.

4.2.2 Batalhas

Encontros aleatórios são uma mecânica que os videogames adotaram diretamente dos RPGs de mesa, onde ocasionalmente o DM (do inglês *Dungeon Master*) jogava dados para determinar quando e quais inimigos o grupo de jogadores iria ter que enfrentar durante o jogo.

Encontros aleatórios adiciona ao jogo um elemento de surpresa e mistério, onde o jogador nunca sabe quando e o que terá que enfrentar diante, e assim é capaz de invocar sentimentos de suspense, curiosidade, alívio e até medo. Esta mecânica também apresenta suas desvantagens, encontros aleatórios interrompem o fluxo de exploração do mundo e progresso dos jogadores podendo se tornar muito frustrantes, o que é um problema recorrente nos jogos mais antigos da série *Shin Megami Tensei*.

O sistema de encontros aleatórios desenvolvido é semelhante ao encontrado em *Etrian Odyssey*, que por apresentar um sistema onde o jogador deve dar um certo número de passos antes de encontrar uma batalha contra inimigos, possibilitando assim um fluxo mais natural e menos frustrante da exploração do labirinto.

As batalhas são em turnos divididos em duas partes: A escolha de ações, onde o jogador deve escolher estrategicamente as ações de cada um de seus personagens; e a etapa de batalha, onde as ações de todos os personagens em batalha são executadas por ordem de prioridade e velocidade. O sistema de batalha foi desenvolvido aplicando conceitos aprendidos ao estudar os sistemas de batalha encontrados em *Shin Megami Tensei*, *Etrian Odyssey*, e *Pokémon*.

4.2.2.1 Inimigos

Os inimigos são encontrados aleatoriamente em grupos de 1 a 6 demônios, cada grupo de demônios é predefinido por uma tabela de encontros e são balanceados de acordo com as zonas em que são encontrados.

4.2.2.2 Stats

Stats (do inglês *Statistics*) são as características dos demônios que determinam sua força dentro do jogo.

- Level: Determina a força geral de um demônio, quanto maior o level de um demônio maior seus outros *stats*.
- HP: Sigla para *Hit Point*, determina quanto dano um demônio pode receber antes de ficar incapacitado.
- MP: Sigla para *Mana Point*, são pontos que podem ser gastos para utilizar habilidades especiais em batalha.
- ATK: O status de ataque, determina parcialmente o quanto de dano um demônio causa quando utiliza golpes de corpo a corpo.
- DEF: O status de defesa, determina parcialmente o quanto de dano um demônio recebe de um golpe corpo a corpo.
- SP.ATK: O status de ataque especial, determina parcialmente o quanto de dano um demônio causa quando utiliza golpes a distância.
- SP.DEF: O status de defesa especial, determina parcialmente o quanto de dano um demônio recebe de um golpe a distância.
- SPEED: O status de velocidade, determina o ordem em que os demônios executam uma ação em batalha
- TYPE: O tipo elemental do demônio, determina que habilidades pode aprender, e suas resistências e fraquezas contra habilidades de outros tipos elementais.

Um exemplo dos *stats* de um demônio como são representados dentro do jogo pode ser visto na figura 18

Ivy			
Carnivuno		LV	12
HP	36	MP	40
ATK	22	SP.ATK	28
DEF	22	SP.DEF	28
SPEED	21	TYPE: GRASS	

Figura 18 – Os stats de um demônio como representados em jogo.

4.2.2.3 Types

Uma das mecânicas mais comuns em jogos RPGs é a mecânica de existir diferentes tipos de ataques e habilidades como atributos diferentes e cada personagem dentro do jogo ter fraquezas e resistências a diferentes tipos de ataques. Em *Dark Souls* o jogador pode utilizar uma grande variedade de armas e magias para atacar as fraquezas dos inimigos e causar dano adicional, e podem utilizar diferentes tipos de armadura para resistir a diferentes tipos de dano que inimigos podem causar a ele. Em *Fire Emblem* cada tipo de unidade que o jogador controla durante as batalhas possuem suas respectivas resistências e fraquezas, dando ao jogador uma camada a mais de estratégia a ser considerada para posicionar suas unidades a fim de derrotar inimigos sem que perca suas próprias unidades. Em *SMT* e *Etrian Odyssey* cada monstro encontrado tem diferentes resistências e a diferentes tipos de armas e diferentes tipos de magia, forçando o jogador a criar uma equipe equilibrada a fim de conseguir enfrentar todos os desafios enfrentados durante o jogo. E por fim em Pokémon as resistências, fraquezas e ataques que cada pokémon pode aprender são todas definidas pelo seu tipo, que pode ser um ou dois de um total de 18 tipos diferentes.

Para este jogo foi utilizado um sistema de tipos semelhante ao encontrado em pokémon, pois torna a implementação da mecânica mais simples e fácil de ser desenvolvida, dado que ao invés de precisar determinar as resistências e fraquezas de cada demônio, é apenas necessário determiná-las para cada tipo elemental.

Para o jogo temos 13 tipos diferentes com cada resistência e fraqueza definidas de acordo com a tabela 1, onde as linhas representam o tipo do ataque, e as colunas o tipo do demônio defensor, e os valores representam o multiplicador aplicado ao dano.

Tabela 1 – Tabela de resistências.

ATK DEF	NORMAL	FIRE	GRASS	WATER	AIR	DARK	LIGHT	ICE	ROCK	POISON	DRAGON	BUG
NORMAL	1	1	1	1	1	1	1	1	0.5	1	1	1
FIRE	1	0.5	2	0.5	1	1	1	2	0.5	1	0.5	2
GRASS	1	0.5	0.5	2	1	1	1	0.5	2	0.5	1	0.5
WATER	1	2	0.5	0.5	0.5	1	1	0.5	2	1	0.5	1
AIR	1	1	2		1	1	1	0.5	0.5	1	0.5	2
DARK	2		1	1	1	1	0.5	2	1	1	1	2
LIGHT	1	1	1	1	1	1	2	0.5	1	1	2	1
ICE	1	0.5	2	0.5	2	1	1	0.5	0.5	1	2	1
ROCK	1	1	1	0.5	2	1	1	2	0.5	2	1	2
POISON	2		1	2	2	1	1	1	0.5	1	0.5	0.5
DRAGON	1	1	1	1	1	1	1	1	1	1	2	1
BUG	1	0.5	2		1	1	2	1	1	0.5	0.5	1

4.2.2.4 Opções de combate

Durante a etapa de seleção de ações o jogador cinco diferentes opções de ação para selecionar cada um de seus aliados vivos, sendo elas *Attack*, *Guard*, *Skill*, *Bag*, e *Escape*. O menu de ações e exemplos de demônios utilizando *Attack* e *Guard* está ilustrado na figura 19.

- *Attack*: O aliado executa um fraco ataque corpo sem custo.
- *Guard*: O aliado assume uma postura defensiva recebendo menos dano durante o turno, apresenta prioridade alta para que a ação seja executada antes da maioria dos ataques inimigos.
- *Skill*: Abre um novo menu onde o jogador pode selecionar uma habilidade do aliado para ser utilizada.
- *Bag*: Abre o inventário de itens do jogador, dos quais pode se selecionar uma para utilizar ao invés de executar uma ação com o membro da equipe.
- *Escape*: Tenta fugir da batalha durante o turno do aliado.

4.2.2.5 Skills e Itens

As skills são habilidades especiais que demônios podem utilizar durante as batalhas para causar mais dano, curar aliados, e aplicar efeitos especiais sobre aliados e inimigos 20, e assim podem ser divididas em diferentes categorias: Ataque, Cura, Suporte, Efeitos Especiais, Alvos únicos, Alvos Múltiplos, Corpo a Corpo e A distância.

Skills por serem ações de batalha mais impactantes requerem que o usuário “pague” um custo dependendo do impacto da skill para utilizá-las. Skills corpo a corpo custam HP e a distância custam MP, as habilidades e personagens são balanceados de forma que os personagens que podem aprender skills a distância poderosas possuem mais MP, mas são mais frágeis, e os personagens que possuem skills corpo a corpo fortes possuem mais HP e são mais robustas.

Skills que causam dano também podem ter atributos *Tipo* diferentes, sendo mais ou menos efetivas dependendo da resistência do alvo contra o *Tipo* da skill.

Itens 21 funcionam de maneira semelhante à skills, podendo ser utilizados por qualquer demônio aliado e a quantidade do item ao ser utilizado é reduzido por 1.

4.2.2.6 Status Effects

Status Effects ou "efeitos de status" são efeitos que afetam a capacidade do demônio de batalhar, podendo ser condições especiais como paralisia, ou *buffs* e *debuffs*.



(a) Menu de opções de ação para o demônio ativamente selecionado.



(b) Demônio aliado executa um ataque ao demônio inimigo.



(c) Demônio aliado se defendendo.

Figura 19 – Base do sistema de batalha.

Esses efeitos podem ser aplicados a demônios aliados e inimigos por meio de *skills* e itens dando assim maior profundidade e complexidade estratégica ao construir uma equipe de demônios e ao utilizar cada demônio eficientemente em batalha.

4.2.2.7 Posição dentro da equipe

Cada grupo de demônios, sendo aliados ou inimigos, possuem uma linha frontal e uma linha posterior [22](#). Demônios na linha frontal são mais prováveis de serem atacados por demônios inimigos, e demônios na linha posterior causam e recebem menos dano por golpes corpo a corpo.

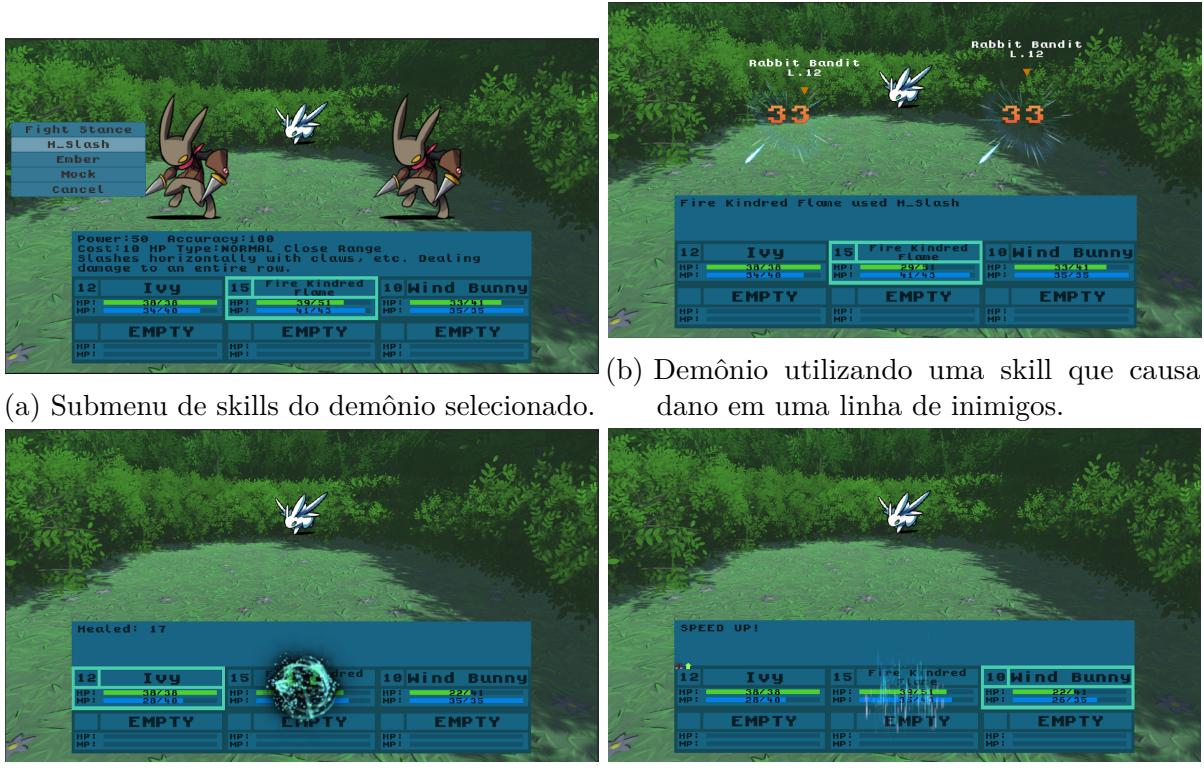


Figura 20 – Sistema de skills em funcionamento.



Figura 21 – Submenu para uso dos itens.

4.2.2.8 Captura de Demônios

Uma das principais maneiras do jogador aumentar sua força para progredir no jogo é capturar novos demônios para utilizar em sua jornada. Todos os demônios inimigos que o jogador podem ser capturados terminar de fazer isso aqui

4.2.2.9 Progressão de personagens

A progressão de cada demônio individual do jogo é medida por meio de seu nível, que é utilizado para calcular cada um de seus *stats*. Após cada batalha os demônios do jogador ganham pontos de experiência que são responsáveis por aumentar o nível de um



(a) Formação de batalha dos inimigos.

Ivy HP: 38/38 MP: 34/40	Fire Kindred Flame HP: 39/39 MP: 41/43	Wind Bunny HP: 33/34 MP: 35/35
EMPTY HP: MP:	EMPTY HP: MP:	EMPTY HP: MP:

(b) Formação de batalha do equipe aliada.

Figura 22 – Posição dos aliados e inimigos em uma batalha.

demônio. A quantidade de experiencia ganha é mostrada na tela de resultados ao final de cada batalha como é visto na figura 23.

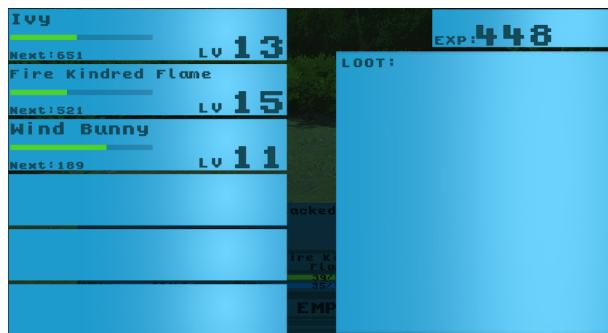


Figura 23 – A tela de final de batalha mostra a exp e itens ganhos.

Cada demônio pode aprender *skills* novas ao chegar em níveis específicos assim assim aumentando o arsenal de opções que o jogador tem ao seu dispor durante as batalhas. Caso um ou mais demônios possam aprender uma *skill* nova ao alcançar um nível novo, uma tela é aberta um demônio por vez para que o jogador possa escolher qual das *skills* deseja deletar 24.



Figura 24 – Ao aprender mais de 5 skills, o demônio deve esquecer uma.

Alguns demônios ao chegarem em certos limiares de nível podem evoluir para

espécies mais poderosas de demônios, a figura 25 apresenta a linha evolutiva de um dos demônios que o jogador pode receber no início do jogo.

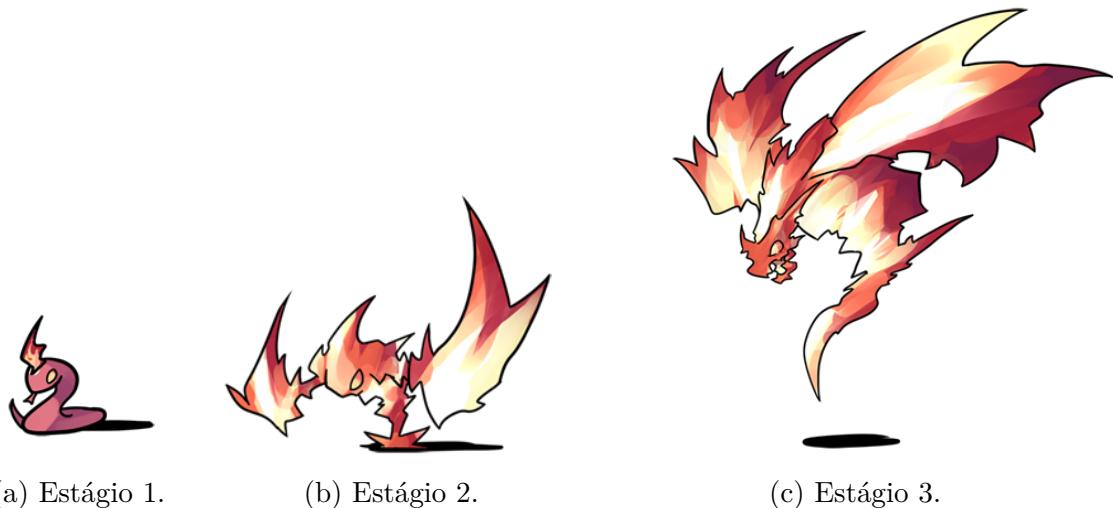


Figura 25 – Linha evolutiva de um dos demônios do jogo.

4.3 Interface

A interação com o jogo é feita em grande parte por menus pelos quais o jogador têm várias opções de ação apresentados a ele, como exemplo na figura 26 abaixo podemos ver o menu de ações de batalha, o menu de skills e o menu de itens, o qual é instanciado sempre vazio. Os menus estáticos têm sempre o mesmo número de opções, como é o caso dos menus de ação e o de skills, mas o menu que representa os itens por ter tamanho variável é feito como um menu que apresenta barra de rolagem.

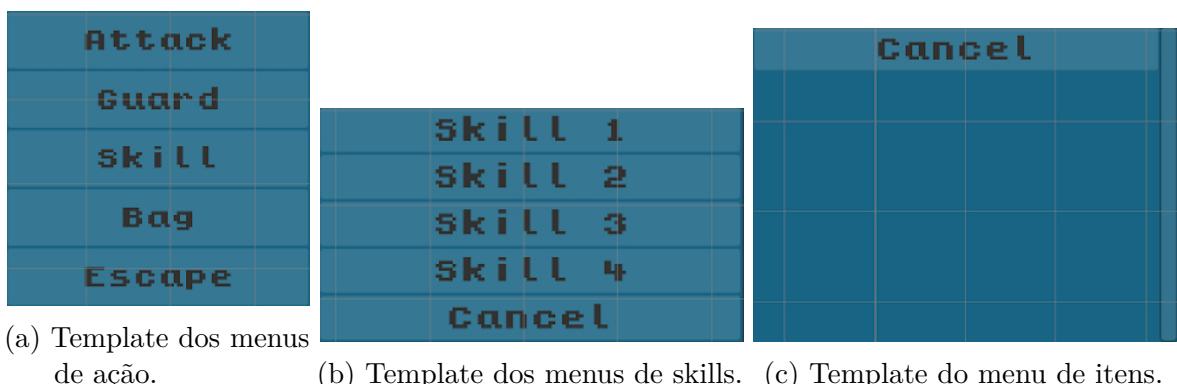


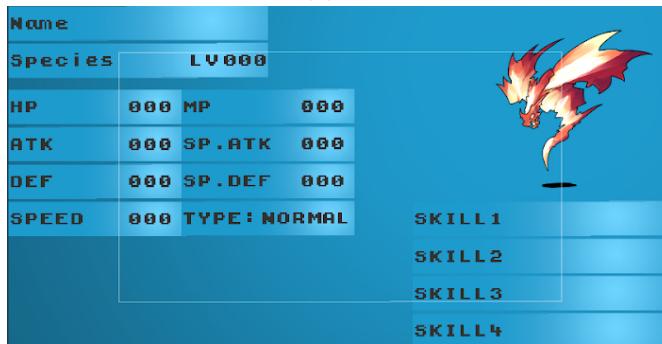
Figura 26 – Template dos menus utilizados durante o jogo.

Para apresentar as informações essenciais para o jogador durante o jogo, foram criados diversas telas com informações de fácil acesso e entendimento 27. Durante a batalha os status de cada demônio são apresentados por meio de caixas que mostram a condição do demônio aliado, ao final de uma batalha a tela de resultados mostra ao

jogador o que ele ganhou ao final da batalha, e durante o jogo o jogador tem acesso a telas de informações sobre cada demônio obtido durante o jogo.



(a) Template da tela de resultados.



(b) Template da tela de stats.



(c) Template da HUD.

Figura 27 – Template de diversas interfaces criadas para o jogo.

4.4 Cidades

As cidades dentro do jogo nasceram da necessidade de se existir um local considerado seguro onde o jogador poderia fazer ações como curar seus demônios, trocar demônios do grupo com demônios salvos no demonomicon, e para se ter um elemento narrativo que sirva de objetivo e propósito para o jogador.

Um dos NPCs criados para habitar a cidade é o Mestre da Guilda, o jogador o encontra ao entrar na Guilda. Ao encontrar com o Mestre da Guilda o jogador recebe seu primeiro demônio, alguns itens para iniciar a aventura e uma missão em troca do demônio inicial.

O segundo NPC implementado é a Demonóloga, que dá ao jogador um meio de curar seus demônios, invocar demônios do demonomicon e selar demônios do grupo de volta para o demonomicon.

5 Desenvolvimento

Neste capítulo será detalhado o processo de desenvolvimento do jogo, que consiste na implementação das decisões de design feitas durante a etapa de design inicial e durante a revisão de *feedback* recebidos durante a etapa de teste.

5.1 Ferramentas

As ferramentas mais utilizadas durante o projeto foi o motor de jogos Unity [28](#), e o editor de imagens GIMP (GNU Image Manipulation Program) [29](#) o qual foi utilizado para a criação e edição de *assets* gráficos por ser uma poderosa ferramenta para edição de imagens, open source, gratuita e fácil de utilizar.

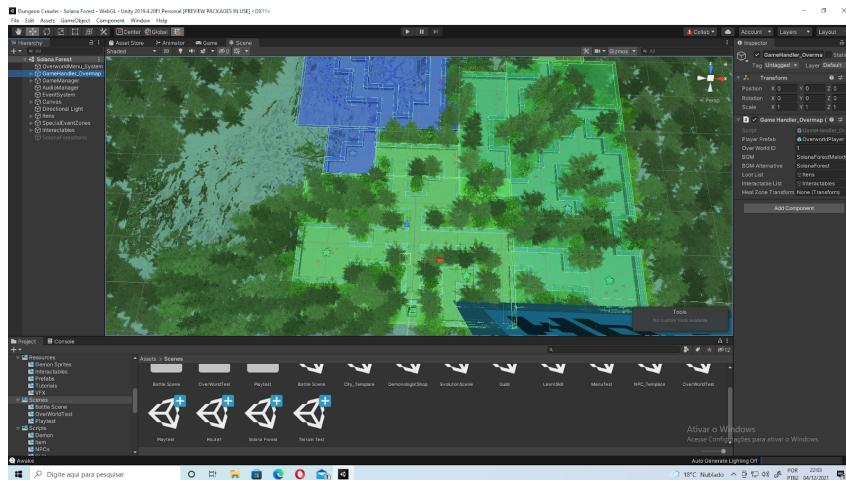


Figura 28 – Editor do motor de jogos Unity.

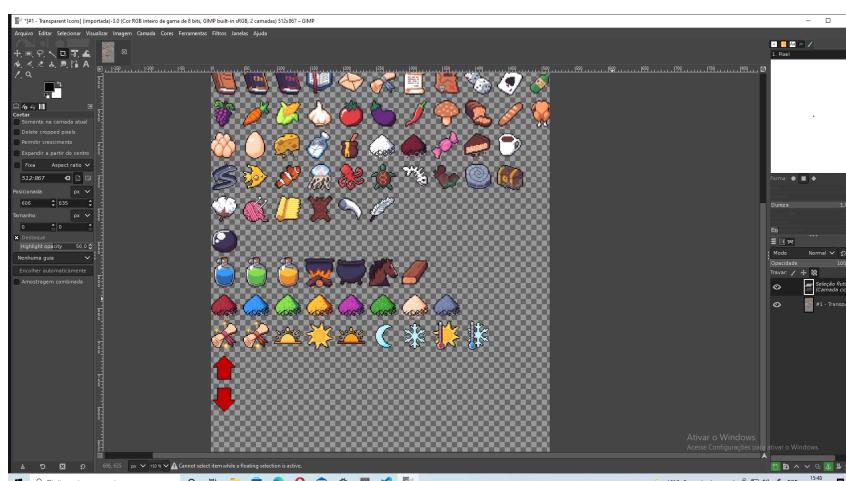


Figura 29 – Workspace do GIMP.

Outras ferramentas utilizadas para a criação do jogo foram: Visual Studio Code para escrita de código; Github Desktop para o versionamento do projeto; Google Docs, Google Meet, e e-mail para a comunicação com o orientador e manter o controle de horas trabalhadas e a plataforma Itch.io em conjunto com o Google Forms e Discord para os testes com jogadores.

5.2 Assets

Para concentrar o esforço de criação do jogo na implementação de sistemas e lógica de jogo a maior parte dos assets gráficos e todos os assets sonoros são *Royalty-free* obtidos de diversas fontes encontradas online, sendo a maioria encontrada no Itch.io.

5.3 Arquitetura das Mecânicas de Exploração

5.3.1 Representação do jogador

O jogador, como é representado durante a exploração do labirinto, é um objeto 3D com duas câmeras acopladas, a primeira é a câmera principal que dá o jogo a visão em primeira pessoa, já a segunda se encontra bem acima do jogador, a qual gera a imagem reproduzida pelo mini mapa do jogo.

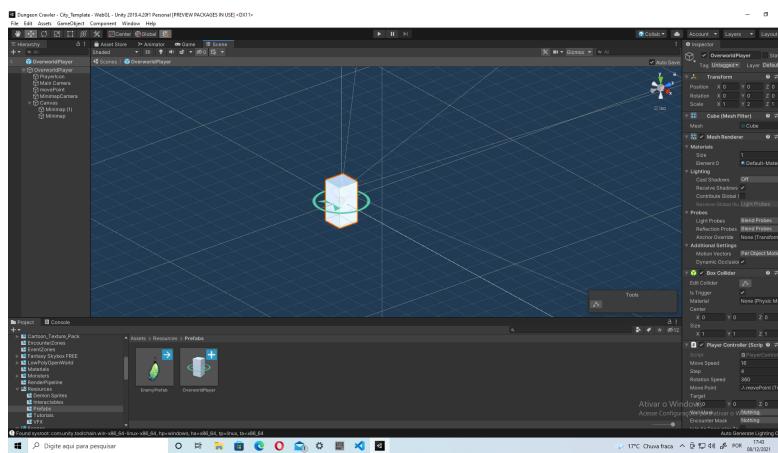


Figura 30 – O objeto *OverworldPlayer* visto pelo editor Unity.

Atrelado à caixa de colisão do jogador há rotinas que definem como o jogador consegue interagir com o ambiente do labirinto, como interação com os itens e objetos interativos, e zonas especiais que indicam quando e como encontros aleatórios e eventos especiais, tais como tutoriais, ocorrem durante a exploração do labirinto.

5.3.2 Random Encounters

Os diferentes métodos de realizar encontros aleatórios estudados foram os encontrados nos jogos da primeira geração de *Pokémon* e em *Etrian Odyssey V*.

Em *Etrian Odyssey V*, ao entrar no labirinto ou após cada batalha, um número aleatório de 64 a 256 é setado como limite de um contador, e cada espaço do labirinto em que o jogador pode dar um passo têm a ele associado um valor de 0 a 20, o qual é adicionado ao contador cada vez que o jogador entra em um desses espaços. Quando o contador ultrapassa o limite definido uma encontro aleatório é iniciado de acordo com uma tabela que indica quais são os possíveis encontros que o jogador pode batalhar em cada espaço. Este método garante que o jogador possa dar um número mínimo de passos antes que encontre outra batalha, e permite que os designers façam ajustes finos a taxa de encontros para que a experiência do jogador não se torne frustrante durante o jogo. A desvantagem deste método é grande quantidade de recursos que se deve ser alocada a implementação da mecânica, dado que cada passo do labirinto deve ser ajustado e reajustado a cada modificação do design do labirinto.

Em *Pokémon*, a cada passo que o jogador executa dentro de áreas onde pode encontrar pokémons selvagens é gerado um valor aleatório entre 0 e 255, este valor então é comparado com a taxa de encontro da área em que se encontra, caso o valor gerado seja maior que a taxa de encontro o jogo inicia uma batalha. A espécie e nível do pokémon selvagem gerado para esse encontro é decidido a partir de uma tabela de encontros. A vantagem deste método é de ser muito simples e fácil de implementar no jogo, a desvantagem é que prejudica a exploração em áreas onde não se pode desviar de zonas em que encontros aleatórios ocorrem ou em áreas com taxa de encontro muito altas.

Para o jogo desenvolvido para este trabalho, se optou por uma estratégia semelhante à utilizada em *Etrian Odyssey V* porém simplificada para ficar coerente com o escopo do trabalho. O labirinto foi dividido em diversas zonas com diferentes valores X_i , e tabelas de encontros associados a eles, e a cada passo que o jogador executa dentro dessas zonas se adiciona o valor X_i a um contador. Uma batalha é iniciada quando o contador X_i ultrapassa um certo valor Y definido como limite, e o grupo de demônios que é encontrado é decidido aleatoriamente a partir da tabela de encontros associada a área em que o jogador se encontra.

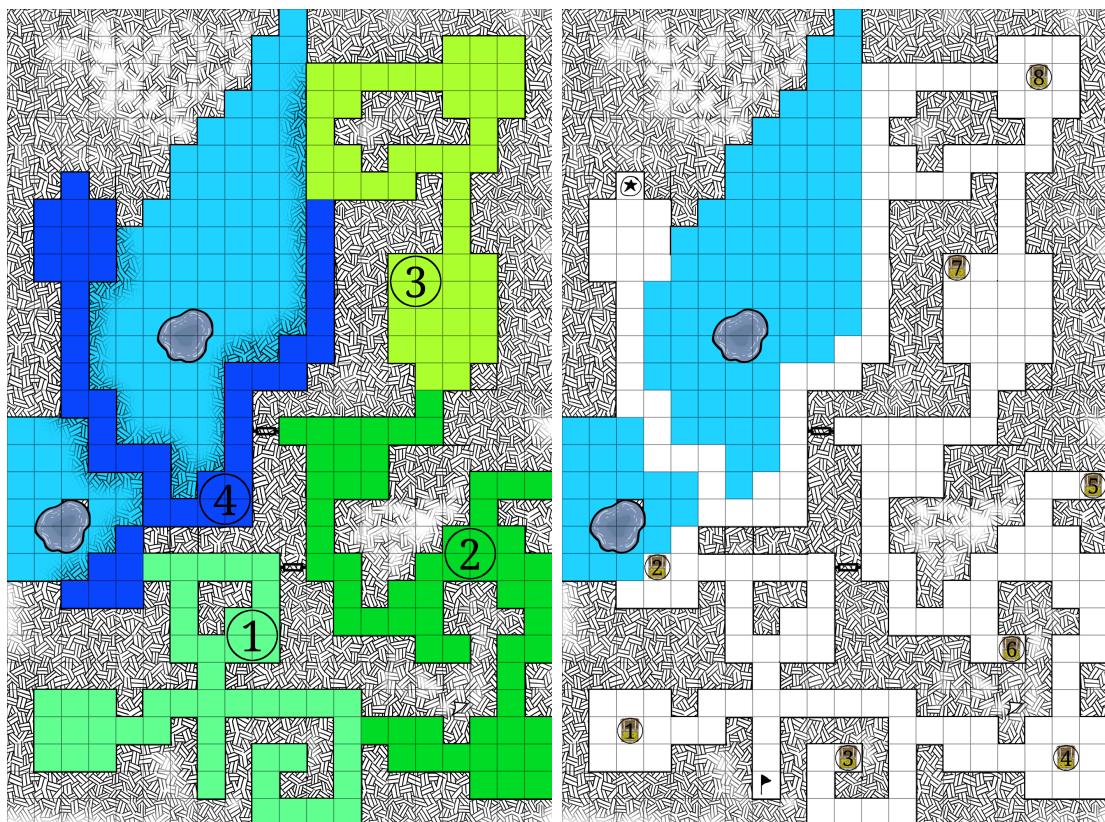
Listing 5.1 – Trecho do arquivo JSON que contém os encontros por zoneID

```
1 {
2     "table" : [
3         {
4             "zoneID": "Default",
5             "encounterList": [
6                 {
7                     "min_level": 1,
8                     "max_level": 1,
9                     "enemy_formation": [ "", "", "", "",
10                               "", "", "" ]
11                 },
12                 {
13                     "min_level": 6,
14                     "max_level": 7,
15                     "enemy_formation": [ "", "", "", "",
16                               "", "", "" ]
17                 }
18             ]
19         },
20         {
21             "zoneID": "OverWorldTest_0",
22             "encounterList": [
23                 {
24                     "min_level": 3,
25                     "max_level": 4,
26                     "enemy_formation": [ "", "Fire Snake", "", "",
27                                           "Fire Snake", "", ""
28                                           "Fire Snake" ]
29                 },
30                 {
31                     "min_level": 3,
32                     "max_level": 4,
33                     "enemy_formation": [ "", "Fire Kindred
34                                         Flame", "", "",
35                                         "", "Fire Snake", "" ]
36                 }
37             ]
38         },
39     ]
```

No trecho de código apresentado em 5.1 podemos ver em detalhes como o jogo gera o grupo de inimigos que o jogador encontra em cada encontro aleatório, de acordo com a *zoneID* de cada área explorável do mapa, o algoritmo escolhe aleatoriamente dentro da *encounterList* um grupo de inimigos de nível entre *min_level* e *max_level* e com a formação *enemy_formation* contra qual o jogador deve batalhar.

5.3.3 Labirinto

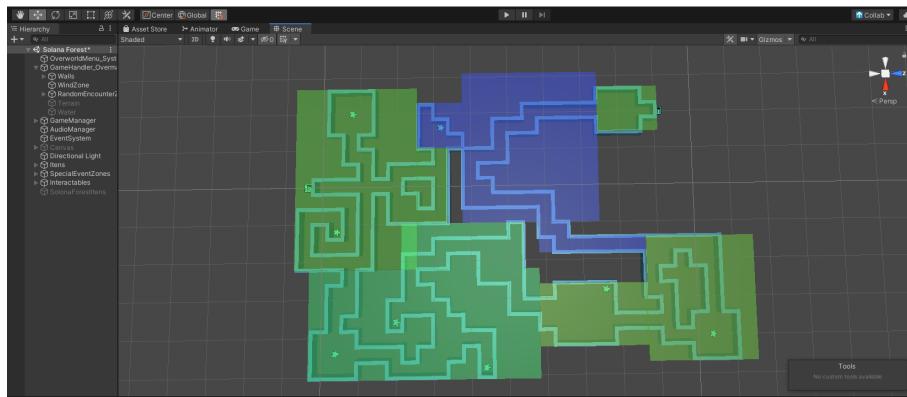
Todo labirinto antes de ser implementado dentro do jogo é criado utilizando um software gratuito chamado Dungeon Map Doodler, que permite a criação de mapas de RPGs de tabuleiro de maneira semelhante a lápis e papel. O uso deste método tornou mais rápido e eficaz o processo de criação de novos labirintos para o jogo. Na figura 31 temos um exemplo de um dos *layout* criados para um labirinto do jogo.



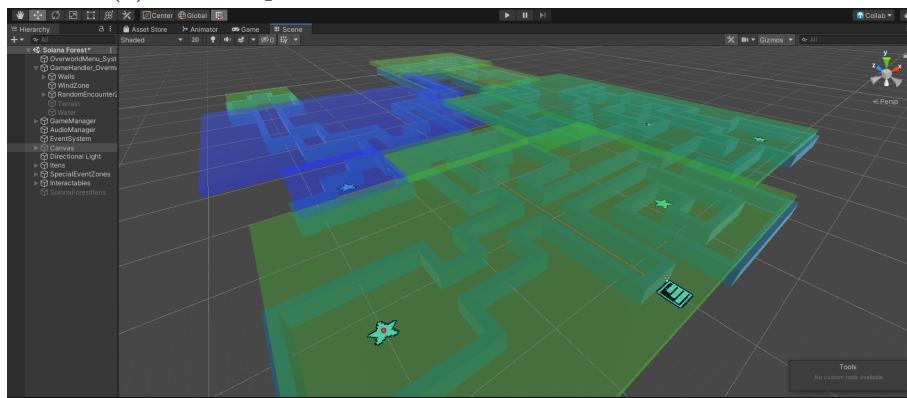
(a) Layout do labirinto com suas diferentes zonas.
(b) Layout do labirinto com a posição de todos itens.

Figura 31 – Layout de um labirinto criado com Dungeon Map Doodler.

A estrutura básica dos labirintos se resume a paredes que são visíveis apenas pela câmera do minimapa, caixas de colisão que definem as diferentes zonas do labirinto e a objetos interativos espalhados com os quais o jogador pode interagir, como está ilustrado na figura 32.



(a) Visão superior da estrutura básica de um labirinto.



(b) Visão ortogonal da estrutura básica de um labirinto.

Figura 32 – Estrutura básica de uma labirinto de duas perspectivas diferentes.

Para dar vida e tornar os labirintos visualmente interessantes foi utilizado o sistema de terrenos disponível pelo motor de jogos Unity em conjunto a *assets* gratuitos para representação de árvores, arbustos, chão e água encontrados no Itch.io e na loja de *assets* do Unity. O terreno foi desenhado estratégicamente para dar a ilusão ao jogador de que ele está colidindo com o terreno, quando na realidade todas colisões são com as paredes invisíveis. O resultado da adição do terreno pode ser observado na figura 33

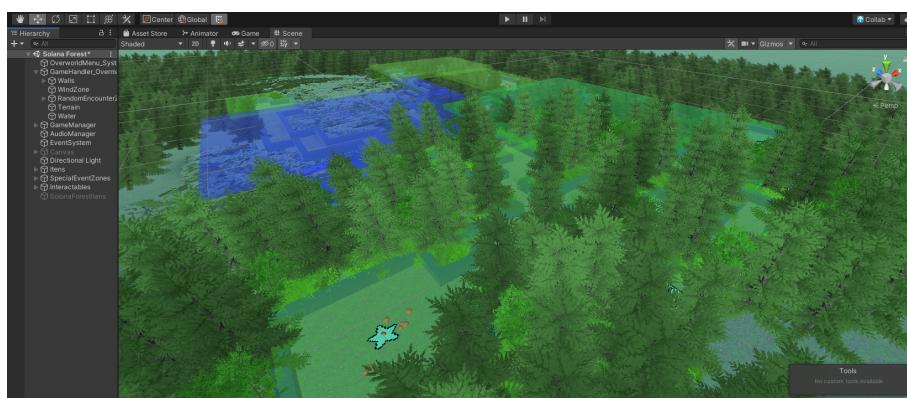


Figura 33 – Visão ortogonal de um labirinto com terreno adicionado.

5.3.4 Overworld Menu System

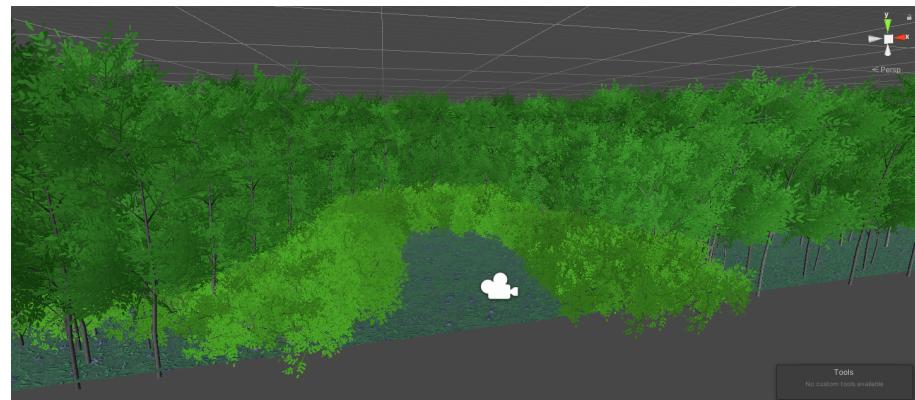
Durante a exploração dos labirintos, as ações que o jogador pode fazer utilizando o menu [4.2.1.2](#) estão implementadas no arquivo OW_MenuSystem.cs, que por meio de uma máquina de estados foram criadas ações bem definidas como abrir menus, utilizar itens e abrir caixas de dialogo úteis para o progresso do jogador.

5.4 Arquitetura das Mecânicas de Batalha

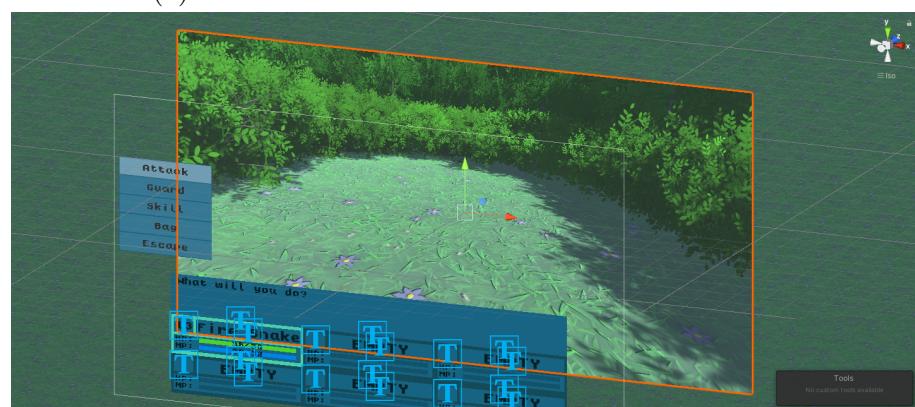
5.4.1 Arena de Batalha

A cena de batalha é composta basicamente de partes diferentes, o *background* tridimensional que representa a arena de batalha, o grupo dos inimigos, e o grupo do jogador.

O *Background* é uma arena construída com os mesmos *assets* tridimensionais utilizadas na construção do labirinto e uma câmera auxiliar renderiza a imagem da arena sobre um plano bidimensional atrás dos objetos da cena que são capturados pela câmera principal, como ilustra a figura [34](#).



(a) Arena construída com *assets* tridimensionais.



(b) Background projetado sobre um plano bidimensional.

Figura 34 – Visão da cena de Batalha como é vista pelo editor.

5.4.2 Demônios

Demônios são os objetos dentro do jogo com os quais o jogador deve utilizar para derrotar outros demônios encontrados dentro dos labirintos, cada demônio encontrado dentro do jogo é membro de uma espécie, que define seu *sprite* (imagem bidimensional) que o representa, seus *stats* e quais *skills* pode aprender.

As definições de *stats* base de cada demônio estão escritas no arquivo BaseStats.cs 5.2, e os golpes que cada demônio pode aprender estão definidos no arquivo MovePool.cs 5.3.

```
new DemonStats /*name*/ "Carnidos",
    /*HP*/ 60,
    /*MP*/ 70,
    /*ATK*/ 62,
    /*DEF*/ 63,
    /*SPATK*/ 80,
    /*SPDEF*/ 80,
    /*SPEED*/ 60,
    /*TYPE*/ TYPE.GRASS,
    /*CATCH*/ 45,
    /*EXP*/ 141,
    /*GROWTH*/ GROWTH_RATE.MEDIUM_SLOW,
    /*EVOID*/ "Carnivores",
    /*EVOLVL*/ 30)
```

Listing 5.2 – Exemplo da inserção de uma espécie de demônio dentro do jogo

```
new Demon_MovePool("Carnidos",
    new List<LevelUp_Move>() {
        new LevelUp_Move(/*Level*/1, /*skillID*/5),
        new LevelUp_Move(/*Level*/3, /*skillID*/1),
        new LevelUp_Move(/*Level*/5, /*skillID*/18),
        new LevelUp_Move(/*Level*/6, /*skillID*/6),
        new LevelUp_Move(/*Level*/9, /*skillID*/39),
        new LevelUp_Move(/*Level*/13, /*skillID*/4),
        new LevelUp_Move(/*Level*/16, /*skillID*/14),
        new LevelUp_Move(/*Level*/18, /*skillID*/26)
    })
}
```

Listing 5.3 – Exemplo da definição de golpes aprendíveis por nível de um demônio

5.4.3 Calculo de Stats

O método de calculo dos *stats* de cada demônio foi baseado no método utilizado nos jogos da franquia Pokémon, onde os *stats* são calculados a partir de fórmulas que levam em conta os *stats* base da espécie do pokémon, do nível, e de alguns atributos individuais. Este método foi escolhido pois outros métodos estudados, que envolviam a definição manual de como os *stats* de cada personagem do jogo crescem por nível, ou o uso de "lançamento de dados" para definir o crescimento de cada personagem ao ganhar um nível, foram descartados por serem ineficientes para a grande quantidade de demônios presentes no jogo ou por serem mais difíceis de balancear por dependerem de números gerados aleatoriamente.

As fórmulas 5.1, 5.2 e 5.3 foram utilizadas para o cálculo dos *stats* de cada demônio a partir dos *stats* base da espécie, nível Level do demônio, nível máximo MaxLevel e duas constantes X e Y definidas por meio de teste.

$$HP = X * baseHP * Level / MaxLevel + Level + 2 * Y \quad (5.1)$$

$$MP = X * baseMP * Level / MaxLevel + Level + 2 * Y \quad (5.2)$$

$$OutrosStats = X * baseSTAT * Level / MaxLevel + Y \quad (5.3)$$

A fórmula 5.4 define quantos de pontos de experiencia totais um demônio deve ter para alcançar um nível Level dado um fator de crescimento G dependente da taxa de crescimento da espécie e um valor constante E de experiencia base.

$$TotalEXP = E * Level^G \quad (5.4)$$

A fórmula 5.5 define quantos pontos de experiencia cada demônio inimigo fornece ao ser derrotado com base na EXP base da espécie, do seu nível Level e constante Z determinada por meio de testes.

$$YieldEXP = EXP * Level / Z \quad (5.5)$$

5.4.4 Skills e Itens

As *Skills* foram implementadas como um classe no script SkillData.cs, e todas as *Skills* devem estãos salvas em uma lista *SkillList* com informações necessária para o uso de cada *skill*: nome, tipo elemental, tipo de alvo, prioridade, poder, precisão, custo,

efeito especial, se é especial, se é de longa distância, ID (número de identificação), efeito visual, cor do efeito visual, descrição e se é utilizável durante a exploração do labirinto. Um exemplo de como uma *skill* é adicionada ao jogo pode ser vista no trecho de código retirado de SkillData.cs 5.4.

```
new SkillData /*name*/          "Absorb",
               /*type*/           BaseStats.TYPE.GRASS,
               /*target_type*/    TARGET_TYPE.SINGLE,
               /*priority*/       PRIORITY.NORMAL,
               /*power*/          30,
               /*accuracy*/      100,
               /*cost*/           6,
               /*effect*/         new EFFECT[] {EFFECT.LIFESTEAL},
               /*isSpecial*/      true,
               /*isRanged*/       true,
               /*ID*/              6,
               /*VFX*/             "Absorb",
               /*VFX_COLOR*/      new Color(0, 1, 0.93f, 1),
               /*DESC*/            "An attack that absorbs half the
                                 damage inflicted.",
               /*OWuse*/          false),
```

Listing 5.4 – Exemplo de como uma *skill* é adicionada ao jogo

Os itens foram implementados a partir da implementação das *skills* em uma classe *Item* no arquivo ItemData.cs, e por isso são adicionadas ao jogo praticamente da mesma maneira que as *skills*, porém em uma lista *ItemList*. Um exemplo de como um item é adicionado ao jogo pode ser vista no trecho de código retirado de ItemData.cs 5.5.

```
new ItemData /*name*/          "H_UltraPotion",
               /*type*/           BaseStats.TYPE.NORMAL,
               /*target_type*/    Skill.TARGET_TYPE.ALLY_ROW,
               /*priority*/       PRIORITY.NORMAL,
               /*power*/          -40,
               /*accuracy*/      100,
               /*cost*/           0,
               /*status_effect*/ new Skill.EFFECT[]
                                 {Skill.EFFECT.NULL},
               /*isSpecial*/      true,
               /*isRanged*/       true,
               /*ID*/              5,
               /*VFX*/             "Heal",
               /*VFX_COLOR*/      new Color(0.6f, 1.0f, 0.92f,
                                 1.0f),
```

```

/*DESC*/           "Medicine that recovers a great
                  amount HP of an entire row.",
/*OWuse*/          true)

```

Listing 5.5 – Exemplo de como um item é adicionado ao jogo

As animações que tocam quando se utiliza um item ou uma skill são determinadas pelo campo VFX que define qual animação deve ser mostrada ao jogador, e devido a quantidade limitada de animações disponíveis gratuitamente, o campo VFX_COLOR define a cor da animação, criando a possibilidade de se utilizar a mesma animação para diferentes efeitos.

O parâmetro OWuse indica quais *skills* ou itens podem ser utilizados durante a exploração do labirinto, já que alguns itens e *skills* possuem utilidade apenas em batalha.

5.4.5 Calculos de Dano

Em jogos RPG há 3 maneiras diferentes de se calcular dano, primeiro de forma que o dano é resultado da força do ataque subtraido pela defesa do alvo 5.6 como é visto em *Etrian Odyssey*; a segunda forma é o dano mitigado pela defesa ser uma porcentagem do valor total do ataque 5.7 como é feito em jogos MOBAs (*Multiplayer Online Battle Arena*), como por exemplo em *League Of Legends*; e por ultimo o dano é resultado da razão entre o poder de ataque do atacante e o poder de defesa do defensor 5.8 como é feito por exemplo em *Pokémon*.

$$DMG = ATK - DEF \quad (5.6)$$

$$DMG = ATK * (1 - DEF/MAXDEF) \quad (5.7)$$

$$DMG = ATK/DEF \quad (5.8)$$

Como o ataque, defesa e quantidade de vida dos demônios são calculados a partir de valores básicos são proporcionais ao nível do demônio, como visto no capítulo 5.4.3, foi decidido que a implementação do calculo de dano seria baseado na equação 5.8 resultando na formula 5.9.

$$(X * \frac{ATKLVL}{Y} + Z) * \frac{SPOWER}{BPOWER} * \frac{ATK * ATKMOD}{DEF * DEFMOD} * DEFLANE * ATKLANE * RES * COND \quad (5.9)$$

Na equação 5.9 temos que o valores X, Y e Z são valores constantes decididos por experimentação, SPOWER é o poder do ataque, BPOWER é o valor base da ação *ATTACK* 4.2.2.4, ATK é o *ATK* ou *SP.ATK* 4.2.2.2 do demônio atacante dependendo se o ataque é de curta ou longa distancia respectivamente, DEF é o *DEF* ou *SP.DEF* 4.2.2.2 do demônio defensor dependendo se o ataque é de curta ou longa distancia respectivamente, ATKMOD e DEFMOD são dependente dos modificadores de ataque e defesa dos demônios atacante e defensor respectivamente, DEFLANE e ATKLANE são modificadores referentes a posição das unidades defensoras e atacantes para casos em que o ataque é corpo a corpo de acordo com a decisão de design vista em 4.2.2.7, RES é o modificador baseado na resistência do defensor contra o tipo do ataque de acordo com a tabela 1, e por fim COND é um valor referente a condições especiais que estejam afetando o demônio atacante.

A função para o calculo de cura é semelhante, mas mais simples, à função de dano, pois não leva em consideração a defesa do alvo e pode ser vista na equação 5.10.

$$(X * \frac{ATKLVL}{Y} + Z) * \frac{SPOWER}{BPOWER} \quad (5.10)$$

5.4.6 Precisão e Esquiva

A precisão de golpes é calculada pela formula 5.11 onde ACC é a precisão da habilidade ACCMOD é o modificador de precisão resultante de efeitos especiais aplicados sobre o demônio que executa a ação, e EVA é o modificador de evasão resultante de efeitos especiais aplicados sobre o demônio alvo.

$$TOTALACC = ACC * ACCMOD/EVA \quad (5.11)$$

A precisão resultante então é comparada com um número gerado aleatoriamente de 1 a 100, caso a precisão resultante seja o maior, o golpe acerta o alvo, caso negativo, o golpe não acerta o alvo e nada acontece.

5.4.7 Buffs e Debuffs

Quando um *stat* de um demônio é utilizado para algum calculo em batalha, um número de modificadores podem ser aplicados durante os cálculos. Durante a batalha, os *stats* efetivos de um demônio pode ser aumentado ou diminuido por algumas *skills*, que são chamadas de *Buffs* e *Debuffs* respectivamente.

Os modificadores operam em uma escala deslizante de estágios. quando um dado *stat* é elevado ou diminuído, o seu estágio atual é aumentado ou diminuído pela quantidade ditada pela *skill*, até um máximo de +6 e um mínimo de -6. Um dado estágio corresponde

a um dado multiplicador que irá modificar o *stat* quando ele é utilizado para calculos de batalha. Os multiplicadores exatos para cada estagio estão detalhados na tabela 2.

Tabela 2 – Stats Modifiers Table

Stage	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
Modifier	2/8	2/7	2/6	2/5	2/4	2/3	2/2	3/2	4/2	5/2	6/2	7/2	8/2

5.4.8 Condições Especiais

Assim como *Buffs* e *Debuffs* condições especiais podem ser infligidos a demônios por meio de *skills*, essas condições especiais afetam a capacidade do demônio afetado de batalhar eficientemente. Atualmente no protótipo estão implementados 5 diferentes condições especiais sendo elas:

- Poison
 - A cada turno o demônio envenenado perde 1/8 da vida.
 - No labirinto a cada 4 passos dado faz o demônio envenenado perder 1/16 de vida arredondado para baixo (mínimo de 1 por passo).
 - Pode ser curada por item ou *skill*.
 - Demônios do tipo *POISON* não podem ser envenenadas.
- Burn
 - A cada turno o demônio queimado perde 1/8 da vida.
 - O demônio queimado causa 50% a menos de dano com golpes corpo a corpo.
 - Pode ser curada por item ou *skill*.
 - Unidades do tipo *FIRE* não podem ser queimadas.
- Freeze
 - O demônio congelado não consegue agir.
 - Todo turno o demônio congelado têm 20% de chance de descongelar e pode atacar no mesmo turno que descongelar
 - Podem ser curada por item ou *skill*.
 - O demônio é descongelado ao final da batalha.
 - Demônios do tipo *ICE* são imunes a congelamento.
- Paralysis

- O demônio paralisado tem 25% de chance de perder a vez durante seu turno.
- A velocidade do demônio paralisado é diminuída pela metade.
- Rage
 - O demônio fica incapacitado de fazer qualquer ação além de *ATTACK*.
 - O demônio causa 25% a mais de dano corpo a corpo.
 - Podem ser curada por item ou *skill*.
 - O demônio é curado ao final da batalha.

Adicionalmente, existem diferentes chances de se aplicar condições especiais podendo ser uma chance baixa, uma chance alta ou em alguns poucos casos, sempre aplicam.

5.4.9 Captura de demônios

A ação de capturar demônios foi implementada como um efeito especial de alguns itens criados especificamente para possibilitar que jogadores obtenham novos demônios para seu grupo. Para verificar se a ação de captura foi concluída com sucesso, um número X é calculado a partir da equação 5.12 onde maxHP é a vida máxima do demônio alvo, currentHP é a vida atual do demônio alvo, catch_rate é um valor associado a especie do demônio alvo que define o quanto fácil ele é capturado, sealBonus é um bônus associado ao item utilizado, e A é um valor definido de acordo com o game design, sendo que a chance de captura de um demônio com vida atual igual a máxima é 1/A da chance de quando o demônio está com apenas 1 HP restante.

$$X = \frac{A * \text{maxHP} - (A - 1) * \text{currentHP}}{A * \text{maxHP}} * \text{catch_rate} * \text{sealBonus} \quad (5.12)$$

Caso o jogador esteja com o grupo cheio, as informações do demônio capturado são salvas dentro de um objeto chamado *Demonomicon* e o jogador pode trocar os demônios que possui dentro do grupo com os salvos no *Demonomicon* por meio da NPC Demonóloga 4.4.

5.4.10 Sistema de Batalhas

As batalhas são gerenciadas pelo sistema de batalha implementado no arquivo Battle_System.cs que por meio de uma máquina de estados divide cada turno de uma batalha em 6 estados diferentes: START; MOVESELECTIONTURN; TARGETSELECTIONTURN; BATTLETURN; WON; LOST;

Durante o estado START são inicializados todos os parâmetros utilizados durante a batalha e a função SetupBattle() é chamada onde os demônios aliados e inimigos são instanciados na *scene*.

Ao iniciar a fase de escolha de ações, as ações dos inimigos são escolhidas por meio de uma algoritmo e então o sistema alterna entre os estados MOVESELECTIONTURN e TARGETSELECTIONTURN para o jogador conseguir registrar as ações de todos os demônios em seu grupo.

Após o jogador selecionar todas as ações de seus demônios, o sistema de batalha entra no estado BATTLETURN, onde todas as ações de batalha são executadas.

A ordem em que as ações são executadas dependem primeiro da prioridade da ação e em segundo da velocidade do demônio, por exemplo, se um demônio utiliza *GUARD*, uma ação de prioridade alta, contra um demônio que utiliza *ATTACK*, a ação *GUARD* vai ser sempre executada primeiro independente da velocidade de ambos os demônios. Em outro exemplo um demônio com *SPEED* 10 e outro demônio com *SPEED* 11 executam ações de mesma prioridade, o demônio com *SPEED* 11 sempre irá agir primeiro.

Depois de cada ação que pode encerrar uma batalha, como por exemplo a captura ou morte de um demônio, o sistema de batalha checa condições para o fim de uma batalha, caso todos os demônios aliados sejam derrotados, o estado muda para LOST indicando que o jogador foi derrotado e retornado o jogador para a cidade, caso todos os demônios inimigos sejam derrotados, o estado muda para WIN e o sistema calcula as recompensas recebidas por vencer a batalha e inicia o processo de retornar o jogador ao labirinto.

5.5 Game Manager

O Game Manager é um dos dois únicos objetos que não são destruídos ao se trocar de *scene* dentro do jogo, sendo o outro o AudioManager, que é instanciado na primeira *scene* aberta no jogo e só é destruído ao se fechar o jogo. Este comportamento é necessário pois dentro do Game Manager são salvas informações necessárias para o funcionamento do jogo como um todo, e serve como uma ponte entre as diversas *scenes* que compõem o jogo.

Dentro do Game Manager é salvo informações de progresso do jogador como a posição do jogador dentro do labirinto, em qual labirinto o jogador se encontra, os demônios presentes no grupo do jogador, os demônios salvos no *demonomicon* e os itens que possui.

Além das informações do jogador listas mantém registro dos eventos que devem acontecer dentro do jogo e quais já aconteceram e não devem acontecer novamente, como tutoriais já vistos, quais atalhos foram desbloqueados, quais itens já foram recolhidos do labirinto, e quais missões já foram concluídas.

O Game Manager também é responsável por manter gerenciar o estado atual do

jogo, e quais *scenes* devem ser abertas em cada etapa do jogo. A implementação completa do Game Manager pode ser encontrado no script *GameManager.cs*.

5.6 Teste com jogadores reais

Depois de implementado as mecânicas básicas de exploração de labirintos, foi executado um teste com jogadores reais para avaliar o protótipo e com base no *feedback* recebido implementar mudanças e novas mecânicas para melhorar a experiência do jogador e o jogo como um todo.

O participantes escolhidos para o testar a primeira iteração do protótipo foram pessoas próximas que pudessem testar o jogo pessoalmente ou com comunicação pelo Discord, pois neste estágio não estavam implementados nenhum tutorial ou ferramentas que ensinassem ao jogador o funcionamento básico de combate e exploração do jogo, necessitando assim da assistência do desenvolvedor para um teste bem sucedido.

Para auxiliar na coleta de *feedback* foi utilizado um Google Forms que todos os jogadores responderam após testarem o jogo. O resultado das perguntas de múltiplas escolhas pode ser vista na figura [35](#).



Figura 35 – Respostas das perguntas de múltiplas escolhas.

Outras perguntas feitas no Forms foram: Qual foi o momento ou aspecto mais frustrante durante o jogo?; Qual foi momento ou aspecto que mais te agradou durante o jogo?; Teve alguma coisa que você queria fazer mas não pôde?; Se pudesse, o que você gostaria de mudar, adicionar, ou remover da experiência?; Como você descreveria o jogo para uma familiar ou amigo?.

A análise dos *feedbacks* levou as seguintes melhorias que deveriam ser implementadas em uma segunda iteração:

- Precisa aumentar a quantidade de XP que se obtém por batalha, ou melhorar o balanceamento de inimigos.
- Habilidade de buff e debuff precisam ser mais fortes para incentivar uso pelos jogadores.
- Aumentar a variedade de habilidades e efeitos.
- Ensinar melhor o jogador a jogar.
- Não rotacionar o mapa para causar menos desorientação ao jogador.
- Melhor a claridade de quando golpes são super efetivos, são pouco efetivos e quando golpes erram o alvo.
- Quando o alvo for um monstro que já está morto, focar o próximo alvo.
- Criar uma tela de resultado de batalha para maior visibilidade.

5.7 Melhorias feitas com base no *feedback* dos jogadores

Com base nos testes com jogadores, uma segunda iteração do protótipo foi implementada, nesta segunda iteração foram feitos ajustes pequenos como retirar a rotação do minimapa, ajustes na quantidade de experiência que os demônios aliados ganham em batalha, mudanças visuais para indicar quando golpes causam dano super-efetivo ou pouco efetivos, adição de elementos interativos no labirinto [4.2.1.3](#), criação de uma tela de resultados exibida ao final de batalhas [4.2.2.9](#), redirecionamento de ações que têm como alvo demônios mortos, e melhorias feitas aos tutoriais. Também foi necessário ajustes que exigiram uma implementação mais robusta como foi o caso da reimplementação e rebalanceamento do sistema de buffs e debuffs, a introdução de condições especiais ao jogo [5.4.8](#), e a introdução de cidades e NPCs ao jogo [4.4](#).

6 Considerações Finais

Neste capítulo estarão descritos os principais problemas encontrados durante o desenvolvimento do projeto e possíveis implementações futuras que podem ser feitas para transformar o protótipo em um produto final.

6.1 Escopo

O principal problema enfrentado durante o projeto foi a dificuldade de se prever com precisão todos os sistemas e mecânicas que precisavam ser implementadas para se ter um protótipo jogável, e de se calcular o tempo necessário para cada implementação.

Durante o planejamento inicial do projeto se esperava que a implementação dos sistemas básicos de exploração de labirintos e batalhas seriam mais rápida e fácil do que se mostraram na realidade, além de que ao decorrer do desenvolvimento e após o primeiro teste com jogadores novos sistemas se mostraram essenciais para o funcionamento do jogo como foi o caso, por exemplo, das cidade e NPCs [4.4](#), das Condições Especiais [5.4.8](#) e de tutoriais.

Como resultado, apenas de todos os componentes do jogo terem sido implementados, não foi possível fazer tantos testes com jogadores quanto planejado inicialmente, se esperava fazer pelo menos três testes com usuários e três iterações de design decorrente do ?? desses testes, e só foi possível a realização de apenas um teste com jogadores e um iteração do protótipo decorrente deste teste.

O teste com jogadores, como resultado do tutorial ainda não ter sido implementado, necessitou ser feito com auxílio e supervisão aos jogadores para que conseguissem jogar e entender todas a mecânicas presentes no jogo. Como resultado não foi possível testar com muitas pessoas e todas tinham um perfil semelhante, possivelmente gerando um viés no resultado do teste.

6.2 Trabalhos Futuros

O jogo em seu estado atual possui todos os sistemas e mecânicas necessários para um jogo completo, faltando apenas a criação de mais conteúdos como mais itens, demônios, labirintos, cidades, NPCs, *skills* e missões, assim a prioridade para versões futuras deve ser concentrar o esforço no design do jogo e em novas iterações do jogo com base em testes com jogadores reais. Além da criação de mais conteúdo para o jogo, possíveis planos para o futuro podem incluir mudanças que tornem o jogo mais interessante visualmente e

conserto de alguns *bugs* conhecidos que permanecem no jogo.

7 Conclusão

O desenvolvimento do protótipo de um jogo com um escopo bem maior em relação a projetos passados feitos dentro e fora do grupo USPGAMEDEV revelou ser uma forma eficiente de se aplicar os conhecimentos obtidos tanto na graduação no IME quanto na graduação não finalizada na POLI-USP e, além disso, também representou uma grande oportunidade de se aprender sobre todo o processo de criação de jogos, por ser um projeto que necessitou tanto o desenvolvimento de áreas 3D quanto 2D e componentes audiovisuais.

Apesar de não ter sido possível executar todos os planos iniciais como vários testes com usuários e de se ter uma quantidade maior de conteúdo jogável, o objetivo principal de se criar um protótipo jogável com todas partes funcionais de um jogo completo foi atingido com sucesso, sendo que além do que se foi proposto inicialmente, foram implementados sistemas e mecânicas adicionais para o funcionamento total do jogo.

Adicionalmente, o processo de expansão de conteúdo do jogo é facilitado pelo fato de se ter toda a base do jogo implementada, possibilitando assim a continuação do projeto e até mesmo um possível lançamento comercial do jogo no futuro.

Referências

- COMPANY, P. *Pokémon*. 2021. Disponível em: <<https://www.pokemon.com/br/>>. Citado na página 24.
- DOBRILOVA, T. *How Much Is the Gaming Industry Worth in 2021? [+25 Powerful Stats]*. 2021. Disponível em: <<https://techjury.net/blog/gaming-industry-worth/>>. Citado na página 15.
- GAMEDESIGNING. *basic game mechanics*. 2021. Disponível em: <<https://www.gamedesigning.org/learn/basic-game-mechanics/>>. Citado na página 17.
- GAMEDEVELOPER. *game engines on steam the definitive breakdown*. 2021. Disponível em: <<https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown>>. Citado na página 18.
- KOSTER, R. *A Theory of Fun for Game Design*. [S.l.]: OReillyMedia, 2013. Citado na página 17.
- SCHELL, J. *The Art of Game Design a Book of Lenses*. [S.l.]: AK Peters, 2014. Citado 2 vezes nas páginas 17 e 18.
- WIKI, E. O. *Etrian Odyssey Wiki*. 2021. Disponível em: <https://etrian.fandom.com/wiki/Etrian_Odyssey_Wiki>. Citado na página 28.
- WIKIPEDIA. *Game Engine*. 2021. Disponível em: <https://en.wikipedia.org/wiki/Game_engine>. Citado na página 18.
- WIKIPEDIA. *Megami Tensei*. 2021. Disponível em: <https://en.wikipedia.org/wiki/Megami_Tensei>. Citado na página 26.
- WIKIPEDIA. *Role-playing video game*. 2021. Disponível em: <https://en.wikipedia.org/wiki/Role-playing_video_game>. Citado na página 15.
- WIKIPEDIA. *Video game*. 2021. Disponível em: <https://en.wikipedia.org/wiki/Video_game>. Citado na página 17.