Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Matheus Lima Cunha

Desenvolvimento de um jogo do gênero *Metroidvania* com geração procedural de mapas.

São Paulo
Dezembro de 2020

Desenvolvimento de um jogo do gênero *Metroidvania* com geração procedural de mapas.

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Ricardo Nakamura

São Paulo
Dezembro de 2020

Agradecimentos

Gostaria de agradecer minha família que sempre cuidou de mim e me motivou a seguir meus sonhos. Thalia Laura por me ajudar com a parte visual do jogo e, mais importante, nos momentos difíceis.Por fim, meu orientador, Ricardo Nakamura, por me orientar mesmo observando muitos outros alunos.

Resumo

Neste trabalho foi desenvolvido um jogo do gênero *metroidvania* aplicando-se técnicas de geração procedural de conteúdo na criação do mapa. Essa ideia vêm da suposição que essas técnicas, pouco usadas em jogos do gênero, possam contribuir positivamente para a experiencia: beneficiando sensação de exploração (uma das principais características do gênero *metroidvania*) e estimulariam que o usuário os jogue novamente após ter terminado a historia. O desenvolvimento foi feito em três etapas: criação do jogo base, estudo de algoritmos e implementação do gerador. As primeiras versões serviram para fundamentar as principais mecânicas de jogo, sem o uso de geração procedural. Após um estudo de diferentes algoritmos, as melhores foram implementadas no protótipo final. As avaliações indicam interesse e divertimento da maior parte dos usuários do protótipo final. Próximos passos para este trabalho incluem a melhoria do sistema de combate do jogo como também testes com diferentes algoritmos de geração.

Palavras-chave: *Metroidvania*, *Design* de jogos, Geração Procedural.

Abstract

In this work, a game of the *metroidvania* genre was developed using procedural generation techniques in the map creation. The idea is that these techniques, seldomly used in this genre, will add to the user experience: benefitting the sensation of exploration (one of the main aspects of a *metroidvania* game) and stimulating replayability. There were three stages of development: base game creation, algorithm study and generator implementation. The first prototypes established the main mechanics of the game, without any procedural generation. After the study and analysis of different generators, the desired techniques were implemented in the final prototype. The user's review showed interest and engagement in the final version of the game. Further studies include enhancing the game's combat and testing different algorithms for generation.

Keywords: *Metroidvania*, Game Design, Procedural Generation.

Sumário

1 Introdução **1**

1.1 Motivação . . . 1

1.2 Justificativa . . . 2

1.3 Objetivo . . . 3

2 Fundamentação teórica e tecnológica **4**

2.1 *Design* de jogos . . . 4

2.2 Motores de jogos . . . 5

2.2.1 Arquitetura de Software do Godot Engine . . . 5

2.3 Geração Procedural . . . 5

3 Análise de jogos do estilo Metroidvania **7**

3.1 Design de um jogo do gênero *metroidvania* . . . 7

3.2 Jogos do gênero *metroidvania* com geração procedural . . . 9

3.2.1 *Sundered: Eldritch Edition* . . . 9

3.2.2 *A Robot Named Fight!* . . . 11

4 Desenvolvimento **12**

4.1 Planejamento . . . 12

4.2 Protótipos Iniciais . . . 13

4.2.1 Primeiro Protótipo . . . 14

4.2.2 Segundo Protótipo . . . 15

4.3 Gerador Procedural . . . 17

4.3.1 Estudo Inicial . . . 17

4.3.2 Primeiro Gerador ("Random Walk") . . . 18

4.3.3 Segundo Gerador ("Roomcedural") . . . 19

4.3.4 Terceiro Gerador ("Abstractcedural") . . . 20

4.4 Protótipo final . . . 21

5 Conclusões **28**

5.1 Próximos passos . . . 28

A *Game Design Document* **30**

B *Respostas dos Formulários* **31**

C Repositórios Git **32**

D Versões do Trabalho **33**

Referências Bibliográficas **34**

Capítulo 1

Introdução

Neste trabalho será desenvolvido um jogo do gênero *metroidvania* com a aplicação de algoritmos de geração procedural para a criação do mapa do jogo. É esperado que a implementação desta técnica contribua para a experiência do jogador proporcionando diferentes ambientes a serem explorados. Atualmente ela não é muito utilizada em jogos no gênero, pois muitos consideram que os mapas gerados são "genéricos", sem um *design* e *layout* profundo que uma criação manual teria, sendo só um conjunto de salas ligadas e não um mundo a ser explorado. Esse fator atrapalharia a imersão e prejudicaria a exploração do jogador.

1.1 Motivação

"Metroidvania" é termo criado da mistura dos nomes dos jogos virtuais *Metroid*[1] e *Castlevania*[2], sucessos do final da década de 1980, utilizado para caracterizar outros jogos com um conjunto de regras e interfaces similares.

A característica mais marcante deste gênero consiste na exploração de regiões interligadas. No inicio, barreiras impedem o acesso do jogador a todas as regiões (denominadas "fechaduras"), como um penhasco que o personagem não consegue escalar, por exemplo. Conforme o jogo avança, poderes e habilidades (denominadas "chaves") são encontradas que permitem o acesso a novas partes do mundo. Por exemplo, a habilidade de saltar mais alto permite chegar no topo do penhasco.

Muitos jogadores apreciam o gênero devido a exploração não linear, que os deixa livres para navegar à vontade pelo mapa. Outro ponto importante é a sensação de crescimento provinda da aquisição de novos poderes e aberturas de mais áreas a serem exploradas. Porém, normalmente não há muita motivação para continuar explorando ou jogá-lo novamente uma vez que o fim do jogo é atingido, visto que o jogador já consumiu grande parte do entretenimento provido pela exploração.

Na tentativa de renovar esta experiência, existem programas conhecidos como *"randomizadores"*. Criados pela comunidade de jogadores, seu objetivo é trocar a posição de objetos dentro do jogo, obrigando o jogador a tomar caminhos diferentes do original.

Infelizmente, *randomizadores* nem sempre são efetivos. Como eles dependem da modificação do código original do jogo, o jogador é forçado a utilizar programas externos para

[1] Mais informações: https://www.mobygames.com/game/metroid
[2] Mais informações: https://www.mobygames.com/game/castlevania-ii-simons-quest

Figura 1.1: *Exemplo clássico de uma fechadura: por mais que o jogador consiga ver a saída à esquerda, ele precisa de algum poder de mobilidade (como pulo duplo) para alcança-la.*

utiliza-los: novos erros dentro do jogo podem ser introduzidos, sem falar de possíveis complicações legais. E, considerando a questão da jogabilidade, a maioria dos *randomizadores* se limitam a trocar somente a posição de itens e inimigos, raramente alterando o cenário virtual, o que pode não ser o suficiente para retomar a experiencia de alguns jogadores.

1.2 Justificativa

Geração procedural é uma forma de geração algorítmica de dados ao invés da criação deles manualmente. Por exemplo, para criar um cenário de jogo representando uma floresta, o programador usa um algoritmo que decide onde e quais as árvores serão colocadas no terreno ao invés de manualmente colocar cada uma.

Ela é utilizada de diversas formas durante o desenvolvimento de um jogo, pois permite a criação automática e variada de dados, economizando tempo e recursos. Casos de aplicações comuns são:

- A geração de uma grande área de terreno (com florestas, lagos, montanhas e afins) que servirão de base para o mundo de jogo. Os mapas de *Far Cry 5* e Ghost Recon Wildlands, por exemplo, utilizam uma ferramenta chamada Hoodini para gerarem os seus ambientes[3].

- Variação de conteúdo. Em *Borderlands*, um jogo de tiro em primeira pessoa, as armas são compostas de diferentes partes, como cano, mira e cabo. Cada parte modifica tanto as propriedades da arma como também seu visual e funcionamento. Sempre que o jogador encontra uma arma nova, as componentes são escolhidas aleatoriamente, dando uma sensação de variedade e unicidade para o jogador.

- Entrega constante de conteúdo. Em jogos do gênero *roguelite*[4], é comum gerar um mapa diferente sempre que o jogador começa uma partida nova, com a finalidade de

[3]Fonte: https://www.sidefx.com/industries/games/

[4]Para mais informações sobre o termo *roguelite*, visite: https://gaming.stackexchange.com/questions/246887/whats-the-difference-between-a-roguelike-and-a-roguelite-game.

desafiar suas habilidades (pois ele não pode depender de decorar informações do mapa) e proporcionar uma experiência diferente.

Por outro lado, a geração procedural deve ser usada com cautela. Além de casos onde conteúdo criado é inviável, como fases que são impossíveis de serem completadas, um gerador ruim pode levar a uma experiência extremamente monótona e repetitiva. Outro fator importante é o tempo gasto desenvolvendo e melhorando o gerador, que pode ser melhor gasto simplesmente produzindo o conteúdo manualmente. Por fim, devemos pensar na performance do jogo. Por exemplo, em *Borderlands* seria extremamente negativo para a experiência do jogador se, sempre que ele encontrasse uma arma nova, demorasse para ela ser gerada.

1.3 Objetivo

O objetivo deste trabalho é o desenvolvimento de um jogo do gênero *metroidvania* com a aplicação técnicas de geração procedural no mapa. Será utilizada a experiência dos jogadores para avaliá-lo.

Considerando que a principal experiência deste gênero vêm da exploração, tem-se como hipótese que a geração procedural possa contribuir para a experiência, pois eliminaria o problema citado anteriormente dos jogadores decorarem o mapa e perderem a motivação de explorar ou jogar novamente.

Enquanto existem jogos bem sucedidos do gênero, como *A Robot Named Fight!* e *Sundered: Eldritch Edition*, eles possuem suas limitações. Enquanto o primeiro consegue se diversificar bem entre cada sessão de jogo, elas têm baixa duração (demorando uma a duas horas até que o jogador consiga vencer o chefe final) e dependem de novo conteúdo liberado após cada sessão para instigar a re-jogabilidade. É provável que essas técnicas não funcionariam para durações maiores ou jogadores que tem seu foco na ambientação e historia. *Sundered*, por outro lado, consegue trabalhar bem a historia e ambientação do jogo, tendo uma duração maior que o anterior (em média, são necessárias 12 horas para o jogador derrotar o chefe final), porém não existe muito motivo para joga-lo novamente após o final dado que não existe variação nos pontos importantes do jogo (como posição das chaves e fechaduras).

Capítulo 2

Fundamentação teórica e tecnológica

2.1 *Design* de jogos

> "*Game Design* é o ato de decidir o que o jogo deveria ser. Só isso. Por cima, parece muito simples." - Schell (2019) (tradução própria)[1]

Desenvolver bem um jogo é complicado. Por um lado, existem todos os aspectos audiovisuais que devem tanto interessar e instigar quanto passar as informações do mundo virtual para o jogador. Por outro, temos toda a questão dele interagir com esse mundo com um conjunto limitado de ações de maneira intuitiva. Também temos que manter um certo nível de dificuldade que o desafie o suficiente para o engajar sem frustra-lo. Além de outros aspectos.

O livro *The Art of Game Design: A Book of Lenses* trabalha com a ideia de lentes exatamente por esse motivo. De acordo com Schell (2019), um bom *design* acontece ao observamos o jogo através de diferentes perspectivas: o jogo é justo? As ações do personagem são interessantes? A interface é intuitiva para o jogador? Com que objetos do mundo físico o jogador vai interagir para jogar?

As respostas para essas perguntas podem vir de muitos lugares: observar jogos que já existem, considerar a opinião de outros desenvolvedores (em sites como *Gamasutra*[2]), ler livros sobre o assunto ou partir de experiencias passadas. Ainda assim, *game design* é um processo de tentativa e erro:

> "*Game design* não é um conjunto de principios, é uma atividade. [...] Você precisa montar o jogo, joga-lo, e deixar outros jogar. Quando ele não o satisfazer (e ele não irá), você deve muda-lo. E muda-lo. E mais uma vez, uma duzia de vezes, até você criar um jogo que as pessoas realmente gostem de jogar. Só quando você tiver passado algumas vezes por isso que você começará a entender o que *game design* é." - Schell (2019) (tradução própria)[3]

[1] Texto original: "Game design is the act of deciding what a game should be. That's it. On the surface, it sounds too simple."

[2] *Gamasutra* é um site que possui diversos conteúdos relacionados a jogos, incluindo publicações de artigos de desenvolvedores de jogos (sem *peer-review).*

[3] Texto original: "Game design is not a set of principles, it is an activity. [...] You must build the game, play it yourself, and let others play it. When it doesn't satisfy (and it won't), you must change it. And change it. And change it again, dozens of times, until you have created a game that people actually enjoy playing. When you have been through this a few times, then you will start to understand what game design is."

Norman (2013) também descreve essa ideia ao explicar o processo de *design* centrado no humano:"*Faça observações no público alvo, gere ideias, produza protótipos e os teste. Repita até que esteja satisfeito.*" (tradução própria)[4]. Afinal, jogos são feitos para os jogadores; não faz sentido exclui-los do processo de desenvolvimento.

De maneira geral, *game design* é sobre fazer escolhas sobre diferentes aspectos do jogo. Para garantir que elas terão o resultado esperado, são necessários testes com usuários reais. Eles vão revelar novos aspectos do jogo que levarão a mais escolhas, que também devem ser testadas. E esse processo se repete até que o jogo atinja as expectativas dos desenvolvedores e jogadores.

Essa ideia de "ciclos interativos"foi seguida durante o desenvolvimento deste trabalho, tendo sido realizados testes com jogadores em toda versão do jogo. Além disso, algumas ideias descritas por Schell (2019) foram utilizadas para melhorar a experiencia do jogador.

2.2 Motores de jogos

Motores de jogos, conhecidos popularmente como *Game Engines*, são programas desenvolvidos com a finalidade de auxiliar a criação de jogos eletrônicos, permitindo a reutilização de código e estruturas de dados comuns entre jogos além de abstrair a implementação de código de baixo nível. Exemplos de funcionalidades comuns entre *engines* são a exibição de imagens na tela, gerenciamento de entrada do usuário e a implementação do *Game Loop*[5].

Atualmente, diferentes *engines* tem diferentes áreas que se sobressaem. *Twine*[6], por exemplo, é focado na criação de historias interativas, mas não seria eficaz na criação de jogos do gênero *platform* ou tiro em primeira pessoa.

Para este trabalho, será utilizada a *Godot Engine*[7]. Essa escolha foi feita baseada na eficacia deste motor na criação de jogos 2D com implementação de simulação física, além do conhecimento prévio do autor em sua utilização.

2.2.1 Arquitetura de Software do Godot Engine

Muitos dos importantes padrões de programação mencionados por Nystrom (2014) já são implementadas internamente pela *Godot Engine* (como o *Game Loop* e o *Object Pool*). Dentre os outros padrões, a maioria dos personagens utilizam o padrão *State* para organizar os seus diferentes comportamentos em uma maquina de estados, além da aplicação do *Singleton* para a uniformizar o áudio da música de fundo do jogo.

2.3 Geração Procedural

Como discutido na Introdução, a geração procedural em jogos é uma técnica de geração algorítmica de conteúdo. Como mostra o livro *Procedural Generation in Game Design*, de

[4]Texto original: "Make observations on the intended target population, generate ideas, produce prototypes and test them. Repeat until satisfied."

[5]O *Game Loop* é uma importante padrão de programação que define o laço interativo da maioria dos jogos atuais. Para mais informações, acesse: http://gameprogrammingpatterns.com/game-loop.html

[6]https://twinery.org/

[7]https://godotengine.org/

Short e Adams (2017), muitos aspectos em um jogo (como mapas, *puzzles*, sons e historias) podem ser gerados proceduralmente. Mesmo considerando só a geração de mapas, objetivo deste trabalho, ainda existe grande variedade. Por exemplo, considere a ideia geral de como os seguintes jogos geram os seus mapas:

- Em *The Binding of Isaac*, o mapa é uma matriz quadrada. Como todas as salas tem tamanho fixo e saídas para todos os lados, o gerador consegue definir o caminho e escolhe aleatoriamente salas feitas manualmente de uma lista. No fim de cada caminho é escolhida uma sala especial, como tesouro ou chefe. Além disso, os inimigos parecem serem definidos pela sala.

- Em *Spelunky*, o mapa também é uma matriz quadrada. Como descrito por Yu (2016), primeiro é definido um caminho. Então as salas (também feitas manualmente) são escolhidas de maneira que sempre é possível completar o mapa sem precisar utilizar quaisquer itens. As posições do mapa que não fazem parte do caminho são povoados aleatoriamente com outras salas. Por fim, o gerador analisa o mapa criado e posiciona inimigos e itens por ele.

- Em *Enter the Gungeon*, enquanto as salas também são feitas manualmente, ele não utiliza uma matriz para o seu mapa: quando necessário, são gerados corredores conectando as salas. O posicionamento de inimigos também parece definido pelas salas em si.

Todos os três exemplos usam salas feitas manualmente e a geração procedural se concentra no posicionamento delas pelo espaço disponível. Mesmo assim, cada um implementa essa ideia de maneira diferente, seguindo as suas necessidades e interesses.

Para estudos iniciais, foi lido o livro *Procedural Content Generation in Games* de Shaker *et al.* (2016). Ele é um dos primeiros livros sobre geração procedural em jogos e dá uma visão geral diferentes metodologias e algoritmos dessa técnica. Ele foi feito com base em um curso de geração procedural em jogos na Universidade de TI de Copenhaga, portanto sua estrutura é bem didática.

Então, foram lidos os capítulos sobre geração procedural de mapas do livro *Procedural Generation in Game Design*. Mesmo sendo explicados em alto nível, raramente mostrando o *pseudocódigo*, os algoritmos descritos serviram de fonte de inspiração do que poderia ser feito neste trabalho. Porém, eles ainda teriam que ser adaptados para implementar a principal característica de um jogo do gênero *metroidvania*: as chaves e fechaduras que determinam a exploração do mundo.

Capítulo 3

Análise de jogos do estilo Metroidvania

3.1 Design de um jogo do gênero *metroidvania*

Inicialmente, foram estudados jogos bem sucedidos do gênero *metroidvania* a fim de entender como a estrutura do mapa influencia a experiência do jogador: *Super Metroid*, *Metroid: Fusion*, *Hollow Knight* e *Guacamelee*. É interessante citar:

- Quando o jogador adquire um poder, é comum obrigá-lo a utilizar sua nova habilidade para sair da sala. Isso força o jogador a entender melhor o funcionamento de seu poder e relacioná-lo a locais previamente visitados.
- É importante deixar claro onde o jogador pode usar os poderes. Isso o ajuda a lembrar de áreas que ele poderá acessar ao adquirir a nova habilidade.
- Poderes que servem outros propósitos além da exploração adicionam bastante na experiência de jogo. Em *Guacamelee* e *Hollow Knight*, por exemplo, boa parte dos poderes também auxiliam no combate.

Figura 3.1: *Em Super Metroid, o jogador entra na sala caindo pela direita, mas não consegue pular de volta.*

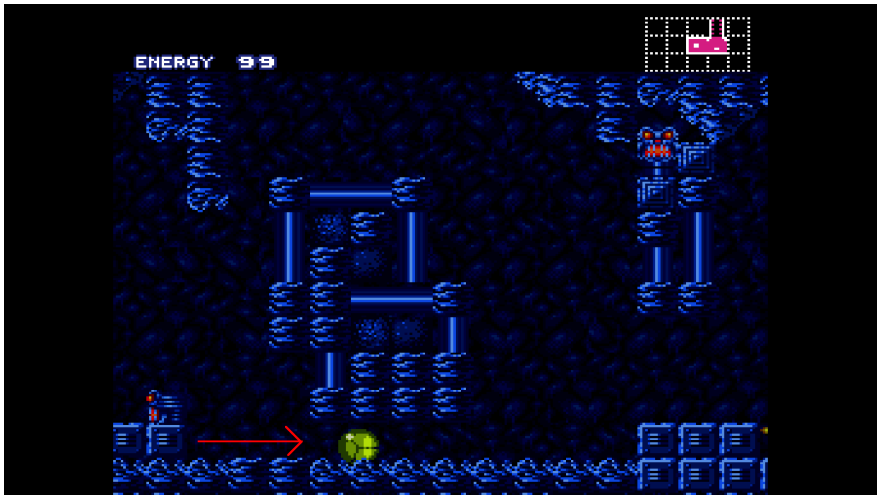

Figura 3.2: *Após pegar o poder morthball, o jogador consegue se transformar em uma pequena esfera e sair da sala.*

Figura 3.3: *Guacamelee utiliza blocos coloridos para indicar que poderes o jogador precisa para continuar.*

3.2 Jogos do gênero *metroidvania* com geração procedural

Por fim, foram pesquisados jogos do gênero *metroidvania* que utilizassem geração procedural. Mesmo não encontrando muitos casos, foi possível notar um certo desprezo da comunidade de jogadores em relação a aplicação destas técnicas. A principal crítica é o *level design*: os mapas gerados deixam a exploração monótona e repetitiva, enquanto a ambientação se torna muito superficial.

3.2.1 *Sundered: Eldritch Edition*

Em *Sundered*, o jogador é uma exploradora que está presa em outra dimensão. Para escapar, ela faz um pacto com um entidade misteriosa e deve vencer três guardiões para abrir o caminho de volta.

No mapa, a posição das salas especiais (que dão chaves, são fechaduras, contam parte da história ou similares) são fixas. Conectando-as, existem grandes salas "vazias" que são preenchidas proceduralmente por salas menores para cada vida do personagem. Isso garante uma consistência "externa", fixando o caminho geral que os jogadores têm de seguir (como ordem dos poderes e partes acessíveis do mapa por vez), enquanto o caminho especifico é variado.

Além disso, não existe uma separação clara onde uma sala começa ou termina enquanto se joga, ajudando com a fluidez do jogo. Porém esse técnica pode acabar deixando os jogadores perdidos devido a falta de um ponto de referência.

Figura 3.4: *Exemplo de uma possível geração do mapa em Sundered.*

Figura 3.5: *Outro exemplo de uma possível geração do mapa em Sundered. Note que as salas "especiais"não mudam de posição.*

3.2.2 *A Robot Named Fight!*

Em *A Robot Named Fight!*, o jogador é um robô que luta contra invasores alienígenas. Para conseguir ficar forte o suficiente para parar a invasão, ele deve descer nas ruínas abaixo da terra em busca de poderosos artefatos perdidos.

Todo o mapa do jogo é gerado proceduralmente: salas pré-criadas são posicionadas aleatoriamente pelo mapa de maneira consistente. Outro fator importante é que a ordem dos poderes também varia, mudando toda nova partida. O gerador parece garantir que sempre é possível completar o mapa criado (não coloca a "chave" após a "fechadura"), além de gerar caminhos opcionais para serem explorados.

A jogabilidade se assemelha bastante a série *Metroid*. Para trocar de salas, o jogador deve passar por uma porta e ver uma curta animação, o que acaba quebrando a fluidez do jogo. Por outro lado, a distinção mais clara entre as salas ajuda a planejar e percorrer caminhos já explorados com mais facilidade.

Figura 3.6: *Exemplo de uma possível geração do mapa em A Robot Named Fight!.*

Capítulo 4

Desenvolvimento

Neste capítulo será detalhado o processo de criação e desenvolvimento do jogo proposto neste projeto.

4.1 Planejamento

Antes de começar o desenvolvimento, foi feito um *Game Design Document*. Ele reúne as ideias e decisões iniciais do projeto, como a ambientação das áreas, poderes do jogador e itens, servindo de base durante o processo de desenvolvimento e ajudando a manter o *design* consistente.

Além das regras e objetos do jogo, também foi desenvolvida uma narrativa que contextualiza as ações do jogador. A história começa com o protagonista acordando no mundo dos mortos. Possuindo um desejo de vingança e nenhuma memória de quem era ou como morreu, ele procurando uma saída. Uma voz misteriosa o contacta, prometendo ajudá-lo. Porém, ao chegar no mundo dos vivos, a voz revela que foi aprisionada e obriga o personagem a libertá-la. Para isso, ele tem de viajar por um mundo tomado pelo caos, em busca de poderosas almas para ficar mais forte e quebrar sua maldição.

Dentre os problemas pensados durante o planejamento do jogo, o principal foi como amenizar a questão do mapa gerado ficar genérico e monótono: ele foi dividido em diferentes áreas, cada uma com sua própria ambientação e estilo. Além de manter o jogador entretido, essa divisão tem o intuito de garantir uma consistência no mundo, pois evita casos como um sistema de cavernas posicionado acima de uma cidade que deveria estar na superfície.

Outra questão que exigiu planejamento durante o *design* do jogo foi sobre manter os poderes únicos entre si, garantindo que cada "fechadura" do mapa só poderá ser ultrapassada com um poder específico. Isso evita que o jogador acabe quebrando a ordem de progressão e fique preso em áreas que não poderia ter acessado. Além disso, todos os poderes também foram pensados com utilidade também em combate: como proposto no capítulo 10 de (Schell (2019)), aumentar o número de usos de uma mesma ação e contribui a experiencia do jogador e jogabilidade emergente do jogo.

A progressão do jogador pelo mapa foi planejada utilizando um esquema de "miniciclos"baseados nos jogos *A Robot Named Fight!* e *Metroid: Fusion*. Cada área do jogo terá o seu próprio ciclo de progressão (como coletar um poder, derrotar o chefe e chegar na próxima área) isolado das demais. Isso evita que o jogador tenha que lembrar e re-explorar áreas

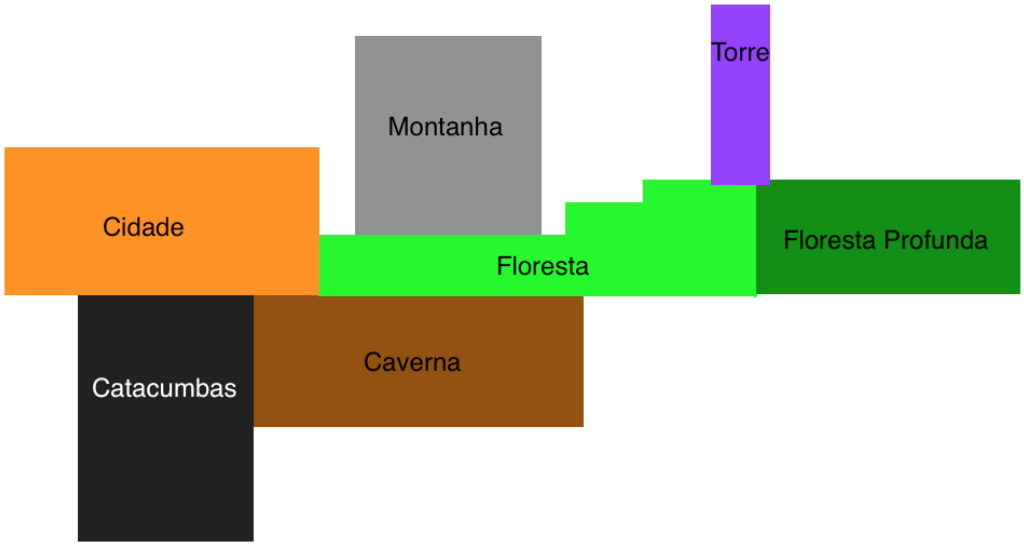

Figura 4.1: *Rascunho inicial do mapa com a divisão de áreas.*

passadas para progredir. Isso não impede que existam locais que só passam ser acessados com poderes de outras áreas, mas eles devem ser opcionais.

4.2 Protótipos Iniciais

No começo do projeto, foi priorizado o desenvolvimento das principais ações do personagem, que foram avaliadas em um cenário fixo, sem geração procedural. A interação do jogador com o jogo se dá através das ações do personagem, portanto é importante fundamenta-las para garantir a imersão e uma boa experiencia de jogo (Schell (2019)). Além disso, um cenário fixo fornece um ambiente controlado para testarmos e avaliarmos melhor as ações propostas.

Cada protótipo foi avaliado com usuários reais, a maioria através de um formulário online implementado na *Google Forms*, que continha links de download para o protótipo e instruções de uso. Os questionário são compostos de quatro seções:

- Introdução, onde é explicado o objetivo, a estrutura, utilização dos dados e o tempo esperado.
- Perfil, composto de 2 a 5 perguntas sobre a experiência prévia do entrevistado em relação a jogos digitais e jogos do gênero de *platform*, *fighter* e *metroidvanias*.
- Jogo, contendo as informações para *download* do projeto. Também há uma seção de ajuda que serve tanto para guiar o jogador quanto para enumerar os problemas mais comuns.
- Avaliação, que pergunta sobre diversos aspectos do jogo, indo de questões mais abertas (como, “o que você gostou no jogo?”) para mais específicas (sobre movimentação, combate, clareza).

Todos os questionários ficaram disponíveis por um período de uma semana. Após esse tempo, os dados coletados eram analisados e discutidos. Para melhor visualização do *feedback*, foram montados painéis para agrupar os comentários por tópico.

4.2.1 Primeiro Protótipo

A primeira versão do jogo foi desenvolvida entre 20 de março até 15 de abril. Ela consiste de duas áreas que servem como tutorial para as principais mecânicas de jogo:

- O "Mundo dos Mortos", que ensina a movimentação básica como andar e pular. Ela também começa a introduzir ideias do gênero *metroidvania*: o jogador deve se movimentar para a esquerda para conseguir progredir, o que vai contra o padrão de jogos do gênero *plataform* (como *Super Mario Bros.*), onde o objetivo é ir para a direita.

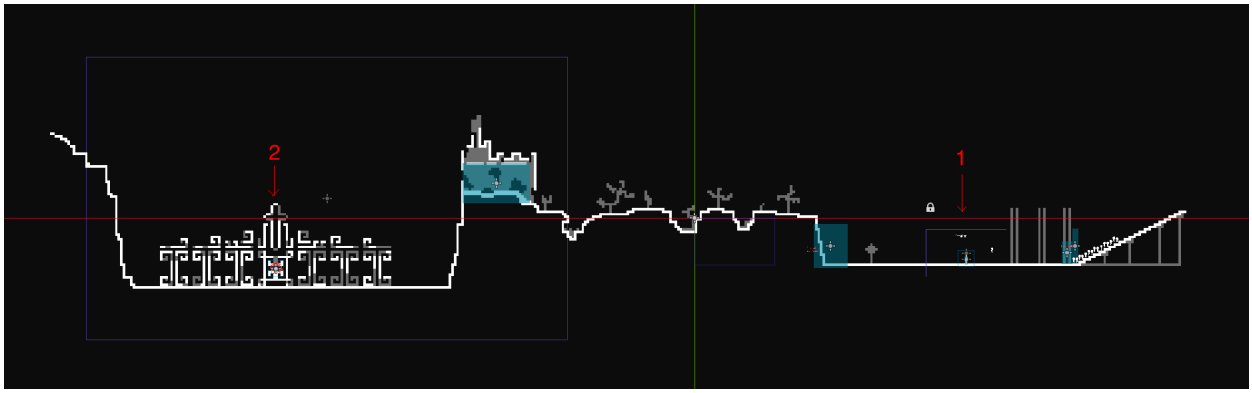

Figura 4.2: *Imagem do "Mundo dos Mortos"dentro do editor do jogo. O jogador começa em (1) e seu objetivo é chegar em (2).*

- O "Mundo dos Vivos", que ensina o combate e a utilização de poderes. Ele também utiliza o principal conceito de um *metroidvania*: perto do começo do mapa, o jogador encontra uma área venenosa que leva para o final. Porém, para conseguir cruza-lá sem morrer, ele deve dar a volta no mapa e adquirir um poder de cura.

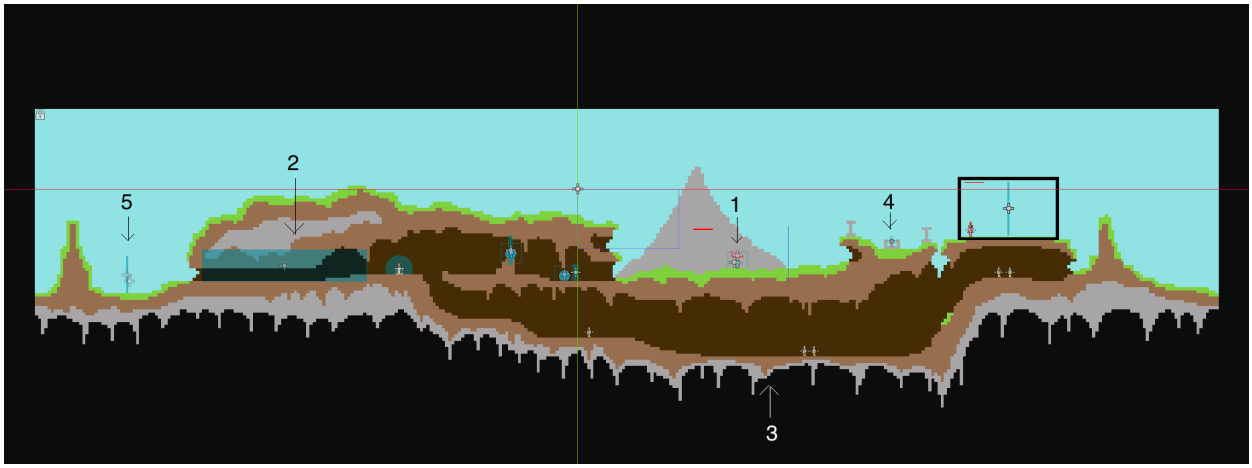

Figura 4.3: *Imagem do "Mundo dos Vivos"dentro do editor do jogo. O jogador começa em (1) e seu objetivo é chegar em (5), porém (2) é uma área venenosa que matará o jogador. Para conseguir atravessá-la, ele precisa dar a volta pode um poder localizado em (4), o obrigando a dar a volta pela parte de baixo do mapa (3).*

Ao final do período de testes, as respostas foram analisadas da seguinte maneira: elas foram divididas em frases menores e categorizadas em “positivas”, “negativas”, “neutras” ou “sugestão”, cada uma com uma cor (azul, vermelho, amarelo e verde, respectivamente). Então, essas frases foram agrupadas em categorias, cada categoria consecutivamente sendo agrupada em macro categorias. Por fim, foi montado um painel com todas essas informações:

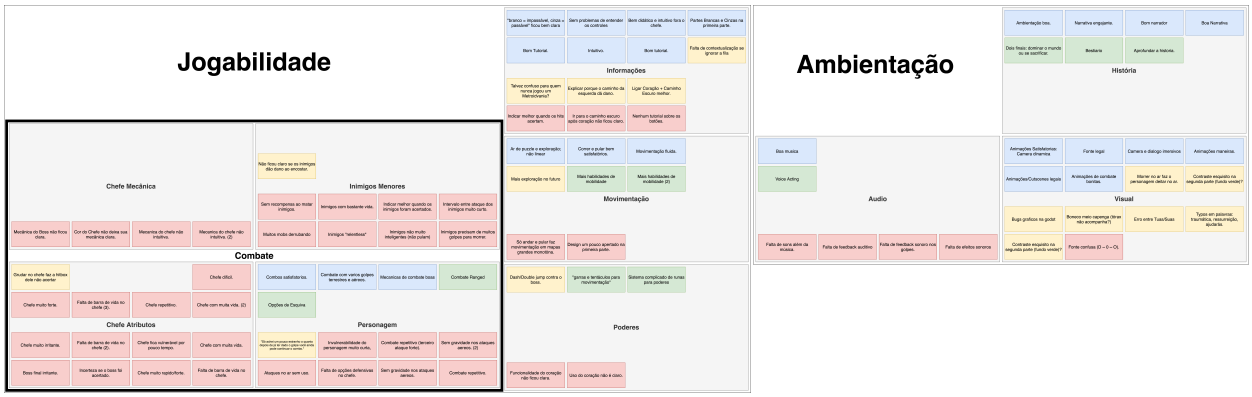

Figura 4.4: *O painel montado, mostrando a divisão de categorias e as frases em cada uma delas.*

Este método de análise (baseado no Kanban[1] e *clustering* manual) ajuda a organizar os comentários em categorias, permitindo uma consulta rápida e a representação visual da avaliação. Por exemplo, na imagem acima é fácil de perceber que existe uma grande quantia de comentários negativos em relação ao combate do protótipo.

De todas as respostas recebidas, o combate foi o mais criticado:

- Inimigos muito fortes, especialmente o chefe.
- Combate repetitivo.
- Escassez de resposta nos golpes, tanto auditivo quanto visual.
- Falta de recompensas por vencer os inimigos.

A combinação destes elementos foi prejudicial para a experiência do jogador, muitos preferindo evitar o combate quando possível. Outros pontos importantes citados incluem a falta de efeitos sonoros de maneira geral e o fato da maioria não entender como passar pela área venenosa. Mesmo com esses problemas, muitas pessoas apoiaram a ideia devido a ambientação e história do jogo. Também foi elogiaram a estrutura do tutorial, que ensina uma ação por vez e proporciona um espaço de "teste"para o jogador.

4.2.2 Segundo Protótipo

O segundo protótipo foi desenvolvido de 20 de abril até 30 de junho.

Devido às críticas anteriores, esta versão é mais linear na tentativa de fundamentar as principais mecânicas do jogo. A área "Mundo dos Vivos" foi substituída pela "Entre mundos", que o representa o limbo, local entre a vida e a morte. A intenção nesta mudança

[1]Para mais informações: https://en.wikipedia.org/wiki/Kanban_%28development%29

foi criar uma área para ensinar o básico do combate sem a adição de elementos do gênero metroidvania, evitando confusões passadas dos jogadores.

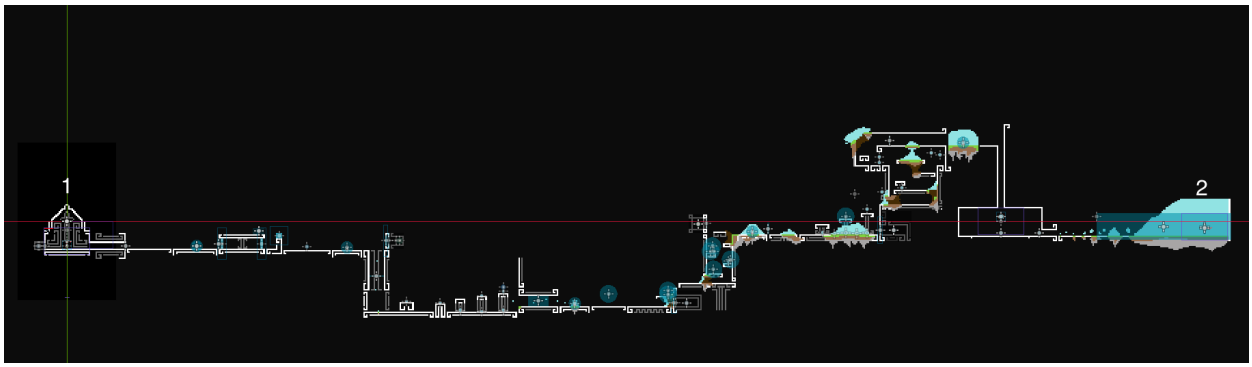

Figura 4.5: *Imagem do "Entre Mundos"dentro do editor do jogo. O jogador começa em (1) e deve chegar em (2). Em relação ao "Mundo dos Vivos", este mapa é linear e maior.*

O "Entre mundos" foi projetado para ser um local onde o jogador possa experimentar as mudanças deste protótipo sem muitos riscos. Todas elas foram projetadas tendo em mente as criticas anteriores e a experiência de jogo. Dentre elas, vale a pena citar:

- Combate modificado, substituindo a sequência fixa de três golpes da versão anterior para golpes únicos e direcionais. Além disso, o jogador agora pode desferir golpes enquanto se movimenta.
- Novos inimigos foram criados, sendo mais fracos e com uma IA mais simples.
- A adição de "almas"para o jogo, que são derrubadas por inimigos mortos ou escondidas pelo mapa. Enquanto sem utilidade no protótipo além de pontuação, foi planejado utilizá-las como moeda no futuro.
- Áreas secretas espalhadas pelo mapa, que só se tornam visíveis quando o personagem as acessa. Elas foram pensadas para estimular a exploração do jogador e contribuir com a experiencia de jogo.

Os testes foram realizados da mesma maneira do primeiro protótipo. Também foi adicionada uma seção especial que pedia para o participante, caso ele tivesse jogado a primeira versão, para compara-la com a segunda.

As críticas mostraram melhora em relação à versão anterior: tanto o combate quanto a movimentação se tornaram mais fluidos, e as áreas secretas foram efetivas em entreter os jogadores. O ponto mais levantado foi a dificuldade, especialmente do chefe no final da fase. Alguns jogadores não conseguiram avançar e desistiram devido a frustração de morrer repetidamente, enquanto outros se sentiram desafiados e gostaram da experiência, por mais que também tenham morrido várias vezes. Em relação aos outros inimigos, houveram críticas em relação ao comportamento "irritante" deles.

Levando em consideração estas críticas, foram feitos alguns leves ajustes no jogo, compondo uma nova versão (nomeada 2.1).

As principais mudanças foram a redução da vida e do dano de diversos inimigos e a adição de uma habilidade de bloqueio para o personagem, fornecendo uma opção defensiva no combate para o jogador. Também foram feitos ajustes no "Entre Mundos" para adequar essas alterações.

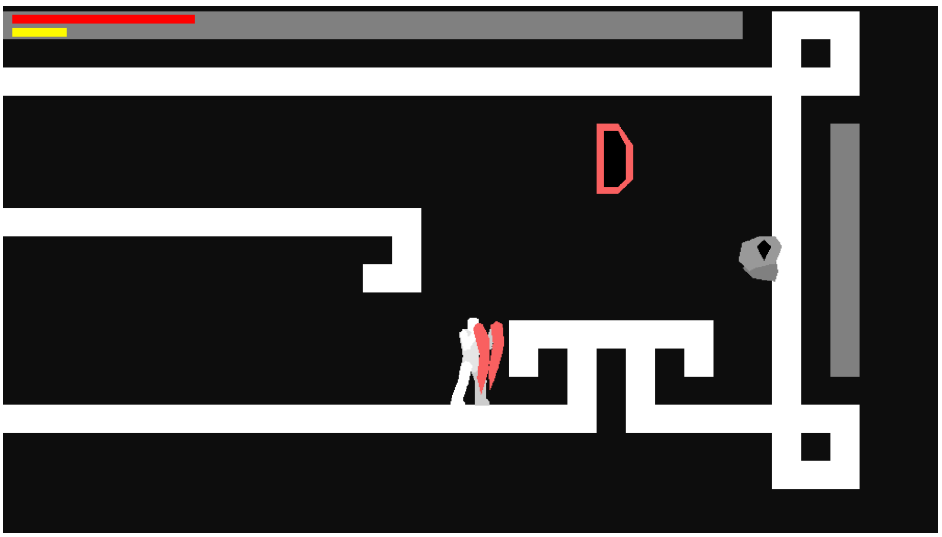

Figura 4.6: *Jogador em uma seção do "Entre Mundos", utilizando a nova habilidade de bloqueio.*

Após outra sessão de testes, as respostas foram positivas de maneira geral. Mesmo assim, houveram comentários sobre a defesa ser muito forte e sobre a redução da dificuldade dos inimigos, que teria deixado o jogo demasiadamente fácil.

4.3 Gerador Procedural

Através das avaliações dos jogadores, o protótipos mostraram ter uma base sólida de ações. Portanto, após a versão 2.1, foi iniciada um estudo sobre o gerador procedural. Ele foi desenvolvido a parte do jogo, em um ambiente onde o foco seria somente a criação do algoritmo. Ao final do estudo, os algoritmos e as estruturas de dados seriam aplicadas ao jogo base.

4.3.1 Estudo Inicial

Antes do inicio do desenvolvimento, foram estudados os livros *Procedural Generation in Game Design* e *Procedural Content Generation in Games*, além de análise de jogos do gênero *platformer* que utilizam geração procedural, como *A Robot Named Fight!*, *20XX* e *Spelunky*.

A principio, o algoritmo de geração cíclica, descrito por Dormans (2017), pareceu interessante. Ele é representado por um grafo: os nós seriam salas e as arestas, uma conexão entre elas. No começo, só existem dois nós, um representando a sala inicial e outro o fim, então o gerador utiliza gramaticas de transformações de grafos para expandi-lo e criar a sala[2].

[2]Video explicativo: https://www.youtube.com/watch?v=_wvkTT-6P3Q&feature=emb_title.

Essa ideia chamou a atenção pois soava como uma boa aplicação para o conceito de "chave fechadura"do gênero *metroidvania*. Porém, sua implementação pareceu bastante complexa e ela foi abandonada, dando-se preferencia a outros algoritmos mais simples.

4.3.2 Primeiro Gerador ("Random Walk")

O primeiro gerador foi baseado no algoritmo *Agent-based dungeon growing* descrita por Shaker *et al.* (2016), como também parte do algoritmo usado por Yu (2016) no jogo *Spelunky*. A geração é dividida em duas etapas:

1. O gerador faz um *random walk* pelo espaço do mapa, garantindo que ele só passará por cada local uma vez e não sairá dos limites definidos. Sempre que ele passa por uma sala, é anotado a direção de entrada e saída dela. Após um número de passos (ou caso ele fique “preso”), é colocado um “power-up” e essa caminhada termina. Esse processo se repete um número definido de vezes, sempre começando de um espaço já visitado.

2. Após ter criado todos os caminhos, o gerador usa as informações dos espaços visitados e das entradas e saídas de cada um para escolher as salas adequadas e posicioná-las pelo mapa. Como todas as entradas e saídas já estão definidas, é garantido que toda sala é acessível e nenhuma delas leva para “fora do mapa”.

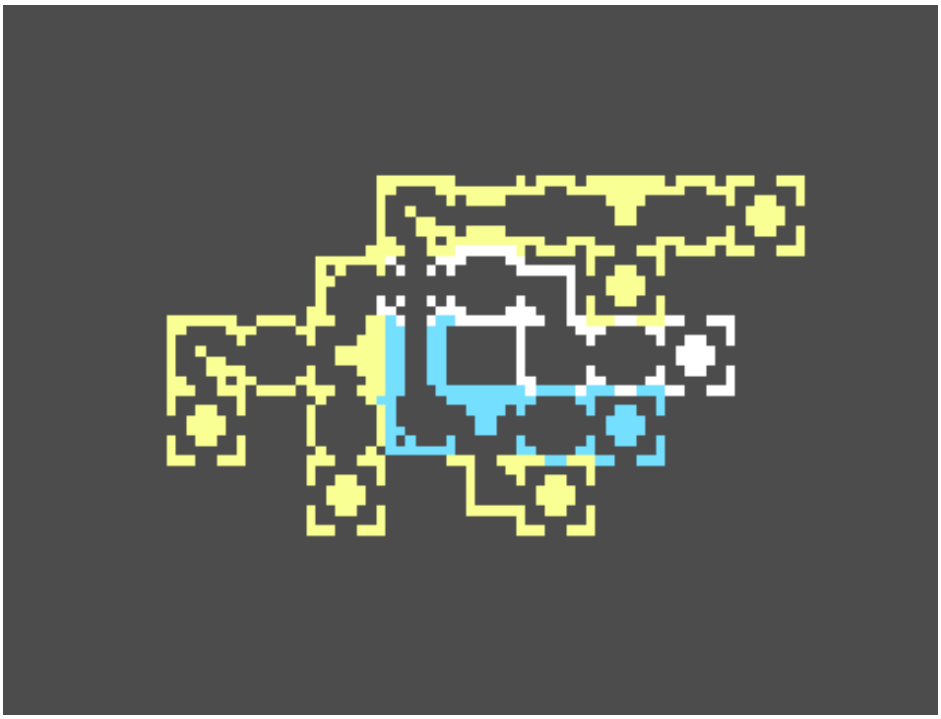

Figura 4.7: *Exemplo de mapa gerado utilizando o gerador "Random Walker".*

O comportamento por trás deste gerador é simular a navegação do jogador dentro do jogo: temos um primeiro caminho que leva a um poder, depois outro caminho encontrado anteriormente que precisa deste poder e assim em diante. Além disso, podemos adicionar caminhos menores que levam a melhorias opcionais sem dificuldades.

Por mais que este algoritmo tenha mostrou bem satisfatório, ele tem uma grave limitação: suas salas tem um tamanho fixo (um “quadrado do mapa”). O motivo disso é que, para manter

a segunda etapa do gerador eficiente, as salas são colocadas em uma árvore binária que as separa pelas suas saídas (cima, baixo, esquerda e direita). Salas de diferentes tamanhos teriam um número variado de saídas, impossibilitando a ideia de usar a árvore binária. Além disso, ele assume que existe pelo menos uma sala para cada combinação de saídas, o que seria inviável para ambientes maiores.

4.3.3 Segundo Gerador ("Roomcedural")

O segundo gerador, apelidado de "Roomcedural", traz uma abordagem diferente para contornar as limitações do primeiro.

A estrutura de dados de uma sala, que no primeiro gerador era composta somente de paredes e de um conjunto de valores para representar as saídas disponíveis, foi expandida:

- O tamanho ("quadrados ocupados no mapa") é definido pela sala, permitindo que elas tenham tamanhos arbitrários.
- As saídas (suas posições e direções) também são definidas por sala.
- Adição de funções dentro de cada sala para auxiliar a geração.

O algoritmo deste gerador foi baseado nos jogos *A Robot Named Fight!* e *20XX* que, por mais que o código fonte não seja acessível, foram estudados durante o desenvolvimento deste trabalho. O processo de geração pode ser descrito como:

1. O gerador escolhe e posiciona uma sala valida no mapa. Essa escolha depende de um conjunto de fatores:
    - Caso ela venha de uma outra sala, ela possui uma entrada que corresponda a saída da sala anterior.
    - Ela pode ser posicionada corretamente dentro do espaço do mapa, encaixando com a saída anterior caso aplicável.
    - Após posicionada, existe pelo menos uma saída desta sala que leva a um espaço vazio (ou seja, que permite outra sala ser posicionada).
2. Com a sala posicionada, o gerador escolhe aleatoriamente uma de suas saídas validas e repete o processo.
3. Caso não seja possível colocar mais uma sala, ele pode abortar o processo ou posicionar o fim do caminho. Considerando ele só posiciona uma sala caso exista uma saída valida, o fim do caminho sempre pode ser colocado.
4. Caso mais caminhos precisem ser criados, o algoritmo de criar caminhos é chamado novamente, começando de uma sala aleatória no mapa.

Esse algoritmo funciona bem. Comparando com o anterior, ele consegue lidar com uma variedade de salas diferentes, porém introduz o problema das salas levarem para “fora do mapa” (saídas que não levavam para outra sala).

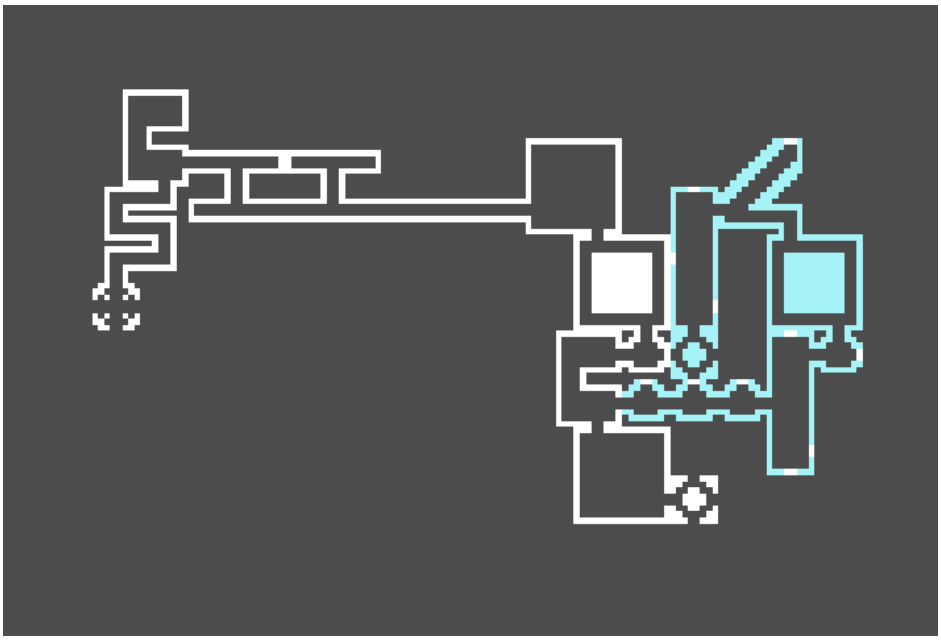

Figura 4.8: *Exemplo de mapa gerado utilizando o gerador "Roomcedural".*

4.3.4 Terceiro Gerador ("Abstractcedural")

O terceiro e último gerador (apelidado de "Abstractcedural") é uma aplicação do segundo gerador com uma implementação mais próxima da real.

Sempre que uma sala é escolhida pelo gerador anterior, ela é posicionada e instanciada no mundo. Isso faz com que todas as salas estejam ativas, possivelmente gastando tempo de processamento com locais que estão longe do jogador. Para resolver este problema, o terceiro gerador cria o mapa "abstratamente", fazendo um grafo onde cada nó é uma sala e cada aresta uma a conexão entre salas. Conforme o jogador sai de um quarto para outro, o jogo dinamicamente troca a sala atual.

Para se adequar a esse novo sistema, as salas receberam algumas modificações:

- Áreas que detectam quando o jogador sai da sala. Eles são usados pelo jogo para definir quando um quarto deve ser trocado.
- Funções para "desbloquear"as saídas. Em vista de outro problema do gerador anterior de salas que levam "para fora do mapa", todas as salas do atual começam fisicamente bloqueadas e são liberadas no final da geração, caso elas tenham alguma conexão valida.

Além disso, foram feitas algumas adições ao algoritmo em si:

- Categorização de salas por ranque, que indicam quantos poderes são necessários para acessa-la. O jogador começa no ranque 0 e, conforme adquire poderes, aumenta seu ranque e os locais exploráveis do mapa.
- Geração de fechaduras entre salas de ranques diferentes.
- Criação de ciclos (quando possível) entre salas de mesmo ranque.

4.4 Protótipo final

Com o fim do terceiro gerador, o desenvolvimento do jogo foi retomado. Além da implementação do algoritmo de geração, algumas mudanças foram feitas, baseadas em parte no *feedback* do protótipo 2.1:

- Foram adicionados menus dentro do jogo, como principal, de pausa e configuração de controles.

Figura 4.9: *O menu principal do jogo.*

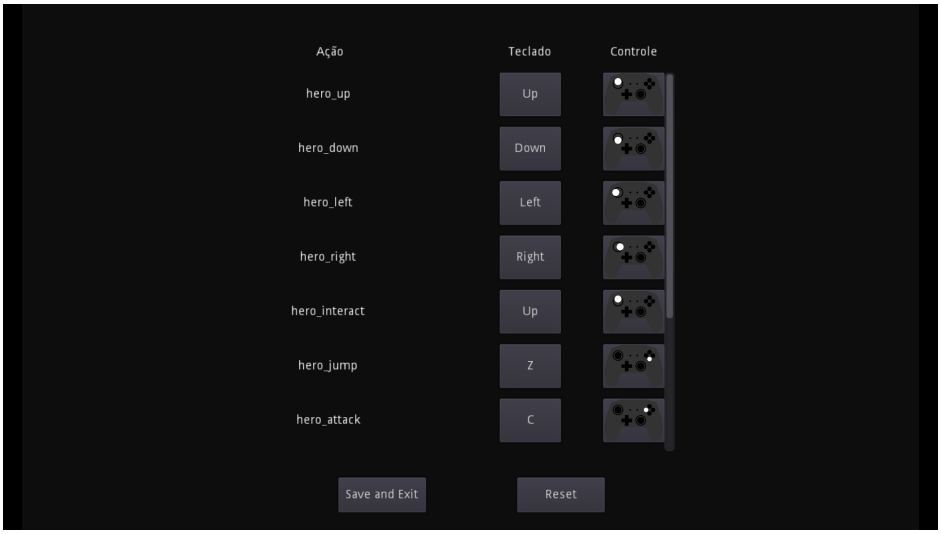

Figura 4.10: *Este menu permite o jogador alterar os botões do jogo, uma adição bem requisitada.*

- A parte audiovisual foi renovado: as cores do "Mundo dos Vivos"foram alteradas, os cenários foram feitos utilizando arte poligonal e diversos efeitos visuais e sonoros foram adicionados. Devido a essa mudança, o jogo ocorre inteiramente no "Mundo dos Vivos". Uma nova área de tutorial foi criada para compensar a remoção da antiga.
- O combate recebeu um balanceamento geral, com alguns golpes ficando melhores e alguns inimigos mais fracos. O chefe final do protótipo anterior foi reutilizado, mas recebeu novos golpes.

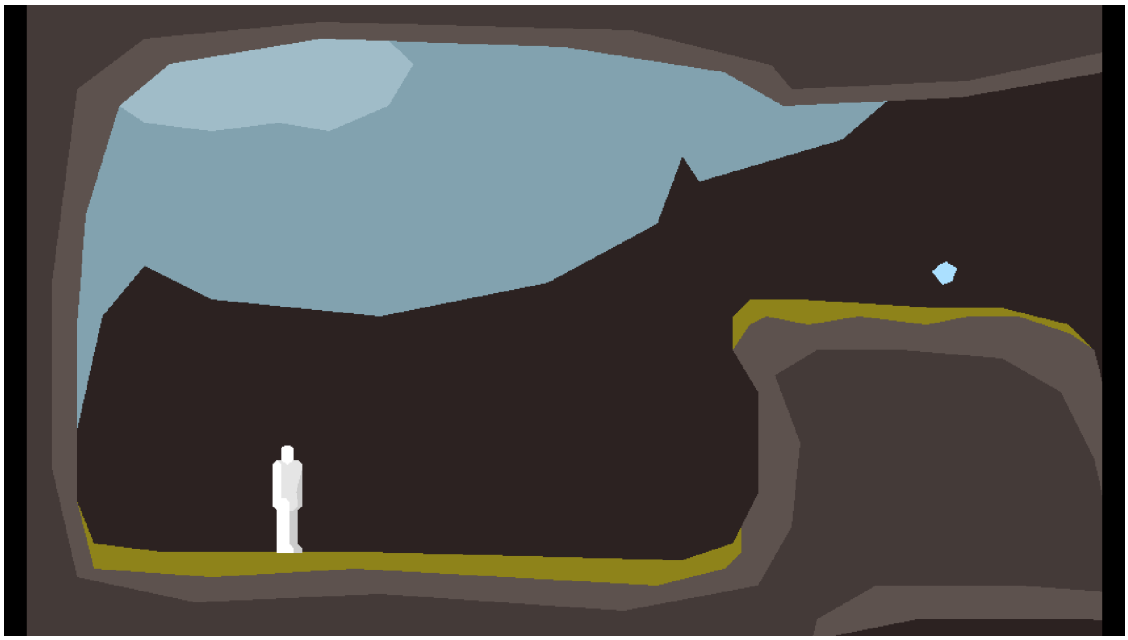

Figura 4.11: *Exemplo de um cenário utilizando arte poligonal.*

- Além disso, os inimigos reapareciam somente quando o jogador utilizasse um altar no protótipo anterior. Isso foi alterado para eles voltarem quando o personagem adentra a sala, na expectativa de deixar o ocasional *backtracking* menos monótono.
- O jogador pode gastar almas coletadas de inimigos em altares para melhorar a vida ou energia de seu personagem.
- Adição do poder "Dash", que permite o personagem se mover rapidamente uma pequena distancia. O jogador deve adquirir esse poder para conseguir terminar o jogo, pois existe uma barreira que é atravessável somente com ele.

Figura 4.12: *"Caverna de Gelo", uma nova área no jogo que serve de tutorial para o poder. A direita temos uma barreira que só pode ser atravessada durante um "dash".*

Para a aplicação do gerador, pouco foi mudado em relação ao estudo. Foram feitas refatorações (especialmente nas funções de troca de salas) e leves melhorais, como verificação de casos em que certo elemento do jogo não foi posicionado. A estrutura das salas também está equivalente, sendo alterado somente os "tipos"de salas:

- Normais: são as mais abundantes e servem de caminhos e exploração para o personagem percorrer entre outras salas especiais. Inimigos são encontrados somente em salas normais e cada uma possui pelo menos três *layouts* diferentes: eles definem que inimigos serão colocados se baseando no ranque da sala. Além disso, existem elementos "variantes"em cada sala, que são modificados aleatoriamente para minimizar a monotonicidade do jogo e dar uma sensação de unicidade em cada sala. Exemplos podem ser vistos nas figuras 4.13 e 4.14.

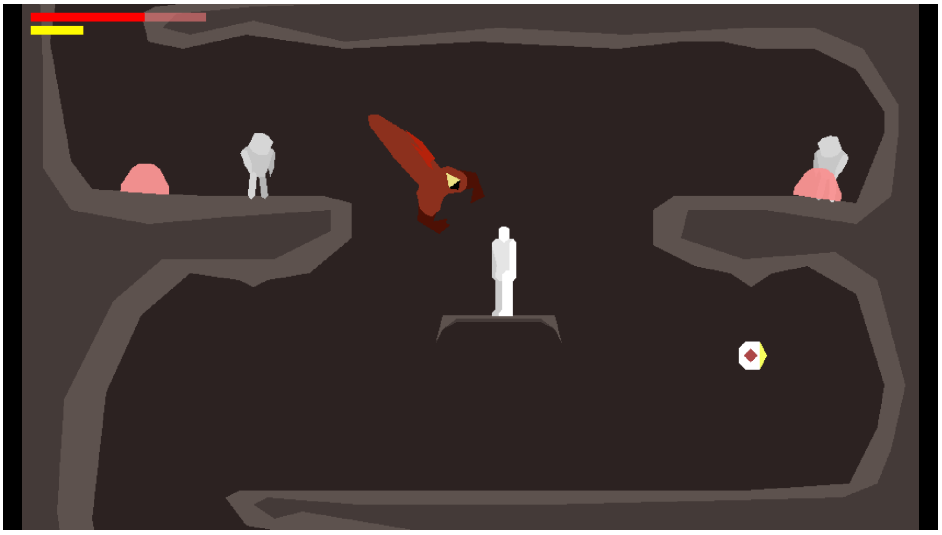

Figura 4.13: *Um exemplo de sala normal.*

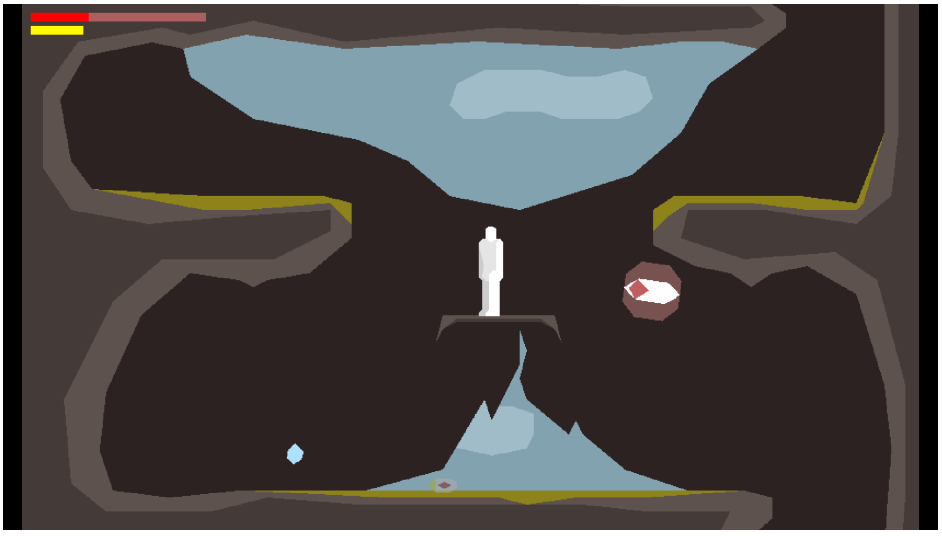

Figura 4.14: *A mesma sala normal anterior, mas com variantes diferentes.*

- *Checkpoint*: são pequenas salas com um altar, permitindo que o jogador se cure e gaste suas almas para evoluir. Caso o personagem morra, ele reaparece no ultimo altar tocado. Todas essas salas tem a mesma aparência, mostrada na figura 4.15.

Figura 4.15: *A sala Checkpoint.*

- Poder: Uma pequena sala com um poder coletável no seu centro (figura 4.16). Ao interagir com ele, o jogador é levado a uma sala separada do mapa, onde deve realiza o tutorial do poder (a "Caverna de Gelo"citada anteriormente e visível na figura 4.12).

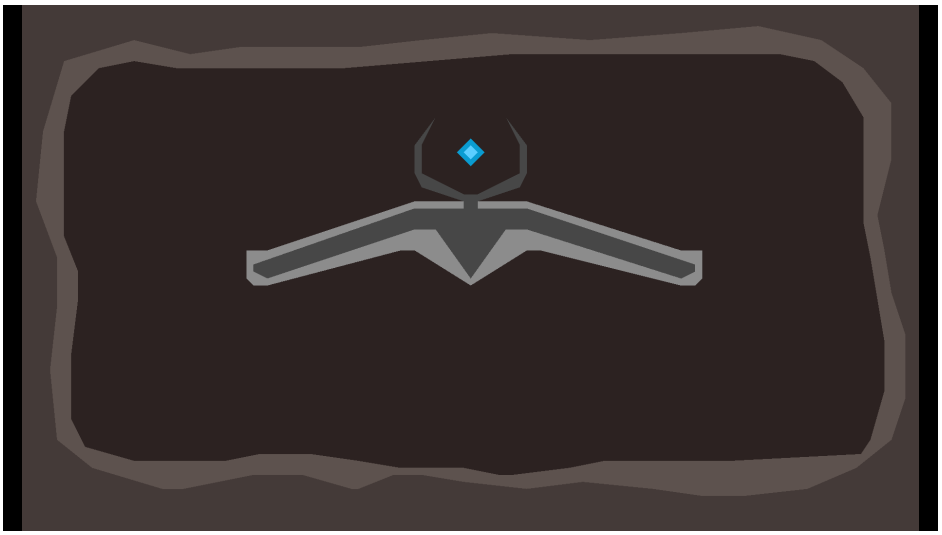

Figura 4.16: *A sala de Poder.*

- Bônus: Parecidas com as salas de poder, elas possuem um coletável em seu centro que concede uma grande quantia de almas para o personagem (figura 4.17). Elas aparecem no fim de ramificações opcionais do mapa.
- Fechadura: Aparecem entre salas de ranques diferentes. Para atravessa-la, o personagem precisa utilizar o "dash"adquirido na sala de Poder. Na figura 4.18, por exemplo, o jogador viria pela direito e só conseguiria acessar o lado esquerdo utilizando o poder.
- Chefe: O objetivo do jogo. Ao interagir com uma saída no centro da sala (figura 4.19), o jogador é levado a uma área separada do mapa, onde pode lutar com o chefe do protótipo (figura 4.20).

Figura 4.17: *A sala bônus.*

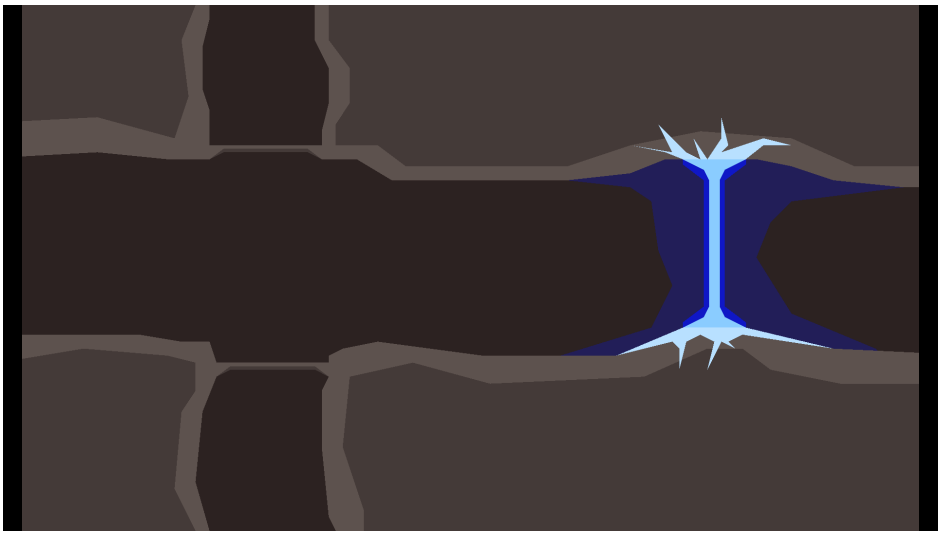

Figura 4.18: *A sala fechadura.*

O questionário dessa versão foi levemente modificado. A seção de perfil do usuário foi reduzida para duas perguntas (uma sobre o habito de jogar jogos e outra opcional, sobre seus jogos favoritos), pois ela não foi muito utilizada nas respostas anteriores. Além disso, a seção de avaliação foi alterada, sendo composta de uma pergunta sobre a movimentação e combate, outra sobre a estrutura geral do mapa e uma sobre as salas em si. Cada uma dessas perguntas possui 4 possíveis respostas ("gostei", "falta algo", "precisa de melhorias", "odiei"), tendo sido escolhido um número par de alternativas para evitar que o usuário selecione uma opção neutra. Essas modificações foram feita para verificar a eficacia do gerador.

De maneira geral, as avaliações foram positivas. Mesmo que a maioria das criticas tenham sido negativas em alguns aspectos, as respostas se concentraram entre "gostei"e "falta algo", mostrando mais interesse que aversão dos usuários. A analise posterior levantou os seguintes pontos:

- O tema mais comentado e elogiado foi o audiovisual, devido a nova ambientação e musicas do jogo.

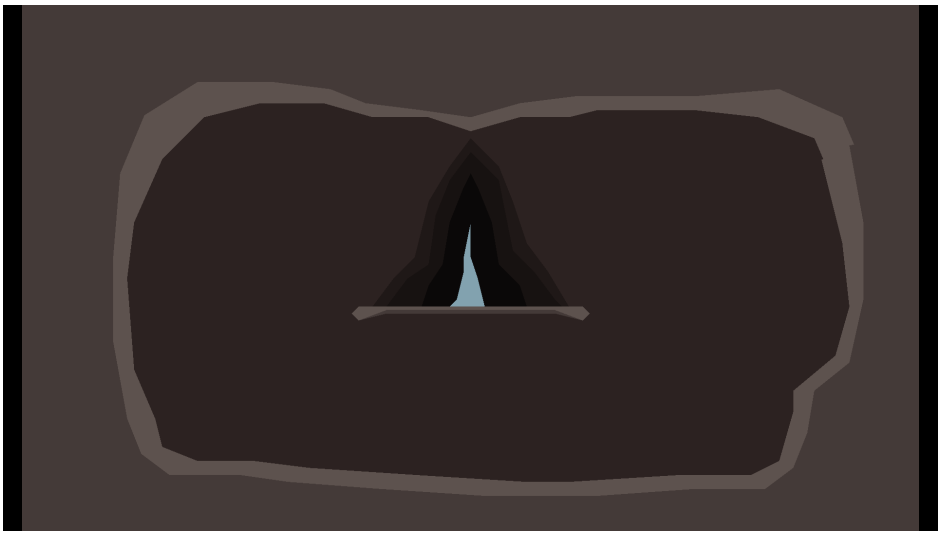

Figura 4.19: *A sala do chefe, com a saída no meio.*

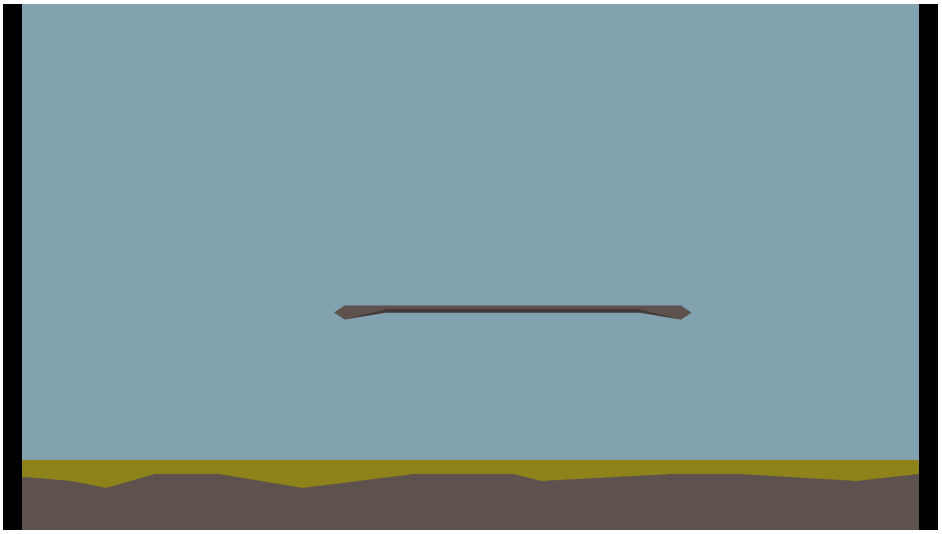

Figura 4.20: *Parte da arena onde o jogador batalha com o chefe.*

- A nova habilidade de Dash não se mostrou interessante, raramente tendo uma aplicação em combate.
- Consequentemente, a maior parte dos usuários não viu motivo para aumentar a barra de energia do personagem e focaram em aumentar a vida, o que facilitou consideravelmente a progressão. Mesmo assim, isso não parece te-los desmotivado a coletar almas.
- Quase nenhum jogador precisou utilizar o guia de ajuda, indicando que as informações do jogo estão claras. Por outro lado, todos eles responderam que tinham hábitos de jogar, o que poderia ter facilitado o entendimento.
- Os golpes também foram criticados, principalmente em relação ao alcance. Enquanto isto era algo já comentando na versão passada, acredita-se que a introdução de terreno irregular, como rampas e declives, tenha intensificado esse fator.
- Em relação ao mapa, boa parte das pessoas acharam interessante as ramificações,

enquanto outras reclamaram do *backtracking*, especialmente dos inimigos voltando a vida. Sobre o posicionamento das salas, as respostas variam bastante: alguns acharam muito repetitivo ao passo que outros gostaram da variedade.

- Sobre as salas, o fato mais comentado foi a falta de segredos, que existiam no protótipo passado.
- Também houveram comentários sobre *bugs* no jogo.

Capítulo 5

Conclusões

Neste trabalho foi desenvolvido um jogo do gênero *metroidvania* com a aplicação de geração procedural em seu mapa, partindo da suposição que essa técnica contribuiria para a experiência de jogo.

Durante o desenvolvimento, o *feedback* adquirido através dos testes foi essencial para a progressão dos protótipos. Além de elas comunicarem a existência de erros, grande parte das escolhas de *design* foram feitas e priorizadas com base nos comentários dos jogadores. A analise final das respostas revelou que a maioria dos usuários se divertiram e conseguiram terminar o jogo, além de conter avaliações positivas quanto a jogabilidade e a ambientação.

De maneira geral, as mecânicas implementadas se mostraram competentes o suficiente para causar uma boa experiencia de jogo. O algoritmo do gerador sucedeu em gerar mapas aplicando o conceito de "chave e fechadura"do gênero *metroidvania*, ramificações e áreas bônus sem prejudicar gravemente o entretenimento dos jogadores.

A versão final possui 7 salas normais (sendo uma usada somente em casos *fallbacks*) e uma para cada caso especial (começo, poder, *checkpoint*, chefe, fechadura) e 6 inimigos diferentes (excluindo o chefe), além de um único poder adquirível. Enquanto esses números se mostraram o suficiente para os protótipos, com poucos usuários relatando repetição, não foi possível verificar se eles seriam o bastante para estimular novas sessões de jogo.

5.1 Próximos passos

Considerando as criticas do ultimo questionário, muito ainda pode ser melhorado e feito. Entre elas, temos:

- Realizar um teste mais amplo onde os jogadores devem terminar o jogo mais de uma vez. Seu objetivo é verificar se a geração procedural foi efetiva em reduzir a repetitividade do jogo quando jogado diversas vezes.
- Tornar o combate mais rápido e dinâmico. Atualmente, os jogadores tendem a aumentar muito a vida de seu personagem e não se defender. Seria interessante reduzir a vida do personagem e deixar as habilidades defensivas mais viáveis.
- Expandir o sistema de gasto de almas. A ideia original (descrita no *GDD*) é uma arvore de habilidades onde o jogador deve escolher uma das duas opções de melhoria

por nível. Essas mudanças devem engajar e motivar mais os jogadores que o estado atual.

- Melhorar o *backtracking* do jogo. O fato que os inimigos reaparecem sempre que o jogador volta para uma sala tinha como ideia reduzir o tédio de repetir seu caminho adicionando algo para se fazer. Uma ideia seja talvez não voltar com os inimigos: por mais que ele tenha que retornar, a falta de oponentes deve acelerar o processo. Outra ideia seria implementar o algoritmo de geração cíclica descrito por Dormans (2017).

- Criar mais conteúdo. O protótipo implementa uma parte bem pequena do que é descrito no *GDD*. Diversas outras áreas e poderes foram planejados, além do interesse em analisar se a capacidade engajar por longos períodos de tempo.

Apêndice A

Game Design Document

O *Game Design Document* (GDD) é um documento onde todas as ideias e decisões de projeto são registradas. Considerando modelo de *design* Diamante Duplo descrito por Norman (2013), por exemplo, é de grande importância ter um local onde todas as decisões feitas durante a fase de convergência possam ser rapidamente consultadas, garantindo a consistência do desenvolvimento e evitando que tempo seja perdido rediscutindo os mesmos assuntos.

O GDD deste trabalho pode ser acessado por por este link. É importante informar que maior parte das ideias escritas neste documento acabaram não sendo usadas devido a limitações de tempo, enquanto outras foram "riscadas"para indicar que não seriam utilizadas.

Apêndice B

Respostas dos Formulários

Nessa seção podem ser acessadas as respostas dos formulários de teste de cada versão, como também os painéis montados.

Respostas do primeiro formulário (Versão 1.0): link. Painel montado: link.

Respostas do segundo formulário (Versão 2.0): link. Painel montado: link.

Resposta do terceiro formulário (Versão 2.1): link. Painel montado: link.

Respostas do quarto formulário (Versão 3.0.1): link. Painel montado: link.

Apêndice C

Repositórios Git

Neste apêndice estão os repositórios Git tanto deste trabalho quando do estudo de geradores procedurais. Ambos projetos foram desenvolvidos utilizando o motor de jogos *Godot*, então este programa é necessário para edita-los corretamente.

Trabalho: link.

Estudo de Algoritmos: link.

Apêndice D

Versões do Trabalho

Essa seção contém o link para todas as versões compiladas do jogo disponibilizadas nos formulários. Os jogos estão disponibilizados para Windows, OSX e Linux.

Versão 1.0:

- Para Windows: link.
- Para OSX: link.
- Para Linux:link.

Versão 2.0:

- Para Windows: link.
- Para OSX: link.
- Para Linux:link.

Versão 2.1:

- Para Windows: link.
- Para OSX: link.
- Para Linux:link.

Versão 3.0.1:

- Para Windows: link.
- Para OSX: link.
- Para Linux:link.

Referências Bibliográficas

Dormans(2017) Joris Dormans. Cyclic generation. Em *Procedural Generation in Game Design*, chapter 9. Routledge. Citado na pág. 17, 29

Norman(2013) Don Norman. *The Design of Everyday Things - Revised and Expanded Edition*. Basic Books. Citado na pág. 5, 30

Nystrom(2014) Robert Nystrom. *Game Programming Patterns*. Genever Benning. Citado na pág. 5

Schell(2019) Jesse Schell. *The Art of Game Design: A Book of Lenses, Third Edition*. A K Peters/CRC Press. Citado na pág. 4, 5, 12, 13

Shaker *et al.***(2016)** Noor Shaker, Julian Togelius e Mark J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. Citado na pág. 6, 18

Short e Adams(2017) Tanya X. Short e Tarn Adams. *Procedural Generation in Game Design*. Routledge. Citado na pág. 6

Yu(2016) Derek Yu. *Spelunky (Boss Fight Books)*. Boss Fight Books. Citado na pág. 6, 18