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

Luiz Felipe Moumdjian Girotto

Ensino de Lógica e Conceitos Matemáticos por meio de Jogos Digitais

São Paulo
Dezembro de 2019

Ensino de Lógica e Conceitos Matemáticos por meio de Jogos Digitais

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Marco Dimas Gubitoso

São Paulo
Dezembro de 2019

Agradecimentos

A minha esposa, por sempre me incentivar na minha escolha de carreira, e por me dar forças para seguir com minhas ambições, por mais difícil que fosse o ambiente para fazê-lo.
Aos meus sogros, por sempre cuidarem de mim, e me apoiarem em todos os momentos, difíceis ou não.
E a minha família próxima, pelo grande amor, único de cada um, que sempre esteve presente na minha vida, e fez grande papel em modelar a pessoa que hoje sou.

Resumo

O objetivo deste trabalho foi desenvolver um protótipo de um jogo de quebra-cabeças. Estes quebra-cabeças foram escolhidos de forma a incentivar o desenvolvimento de lógica matemática nos jogadores, e aumentar o interesse dos mesmos em tópicos relacionados. O jogo foi desenvolvido com a acessibilidade em mente, e portanto pode ser jogado tanto em computadores quanto celulares. Ele conta com uma história que pode ser aproveitada por públicos de todas as idades, de forma a envolver todos os tipos de jogadores igualmente. Utilizando-se de técnicas modernas de desenvolvimento, e noções atuais sobre as tendências de usuários, foi desenvolvido um ciclo completo de jogabilidade, capaz de demonstrar o potencial de um jogo como ferramenta de ensino.

Palavras-chave: Ensino, Aprendizagem Baseada em Jogos, Mobile, Lógica Matemática.

Abstract

This work's objective was to develop a prototype of a puzzle game. The puzzles were picked in a manner as to develop the player's mathematical logic, and further interest them in related topics. The game was developed with accessibility in mind, so it can be played both in regular computer and mobile platforms. The game also features a seemingly simple, yet underlyingly complex story that can be enjoyed by all ages, as a means to engage all kinds of players. The game was made using modern development techniques, as well as current notions of user tendencies, resulting in the development of a complete gameplay cycle, capable o demonstrating the potential of a game as an educational tool.

Keywords: Education, Game-Based Learning, Mobile, Mathematical Logic.

Sumário

Capítulo 1

Introdução

1.1 Motivação

A motivação deste trabalho originou-se da noção de que o estudante brasileiro encontra-se cada vez menos conectado com os estudos nas áreas exatas. Estudos de 2017 mostraram que, por mais que mais da metade dos estudantes brasileiros aleguem gostar da disciplina de matemática (Tokarnia, 2019), apenas 9,1% destes estudantes apresentaram o aprendizado adequado dos conteúdos desta disciplina, no mesmo ano (Oliveira, 2019). Este problema pode ser atribuído ao fato dos estudantes ainda enxergarem a matemática apenas como uma disciplina, pertinente apenas ao âmbito escolar, e não como algo presente em todos os momentos de sua vida. Esta noção é então carregada para momentos posteriores da vida deste estudante, e a noção que este mesmo construiu é então disseminada para a geração seguinte, criando um ciclo vicioso.
A solução para a quebra deste ciclo vicioso, então, consistiria em encontrar uma plataforma que permita tornar a interação com conceitos de matemática e lógica algo acessível a todas idades. Ao mesmo tempo, esta interação deve também ser de caráter não explicitamente e singularmente educacional, de forma a evitarmos retornar ao problema original.
Os telefones celulares apresentam-se como a plataforma perfeita para o propósito deste trabalho: não somente são comuns para indivíduos de todas as faixas etárias, como também estão inseridos completamente nas rotinas de seus usuários, para os mais diversos propósitos. Adicionalmente, aplicamos o conceito de aprendizagem baseada em jogos no desenvolvimento de ideia, com o intuito de tornar o contato do usuário com os conceitos de matemática mais dissociado do ambiente de ensino, e mais atrelado ao lazer do usuário. Sendo assim, devido à combinação dos fatores anteriormente mencionados, foi motivada a criação de um jogo de quebra-cabeças matemáticos para dispotivos móveis.

1.2 Objetivos

O objetivo deste trabalho é a criação do protótipo de um jogo de quebra-cabeças, os quais possuem conceitos de lógica e matemática presentes em si. O jogo é projetado com dispositivos móveis em mente, mas pode ser utilizado em computadores regulares.
O objetivo final deste trabalho é a implementação de um ciclo completo de jogabilidade (conceito este que será desenvolvido posteriormente). No entanto, um ciclo de jogabilidade é suficiente para estimar a eficácia do jogo completo, uma vez que este é formado pela composição de múltiplos similares ciclos de jogabilidade.
O jogo, especificamente, possui o objetivo de familiarizar o jogador com conceitos de lógica

matemática, e interessá-lo nos mesmos, fora do âmbito exclusivamente de ensino, incluindo-os em sua forma de visualizar situações e problemas cotidianos.

1.3 Capítulos

O capítulo "Aprendizagem Baseada em Jogos"introduz o conceito que inspirou a criação de um jogo para o propósito deste trabalho. Além disso, são apresentados alguns conceitos relacionados a este tema, e resultados de trabalhos já existentes sobre o mesmo.
O capítulo "O Jogo"mostra uma visão geral do jogo, começando por seu planejamento (teórico e técnico), abordando os propósitos por trás de cada decisão de projeto, e eventualmente tratando das decisões concretas tomadas na implementação final do jogo.
O capítulo "Plataforma de Desenvolvimento"apresenta uma introdução à *Godot Engine*, a *engine* de desenvolvimento de jogos utilizada na criação do trabalho.
Por fim, o capítulo "Processo de Desenvolvimento"descreve em detalhes os processos e dificuldades envolvidos na criação do jogo.

Capítulo 2

Aprendizagem Baseada em Jogos

2.1 Conceito

O conceito de Aprendizagem Baseada em Jogos consiste da utilização de jogos em contextos de ensino, de forma a proporcionar maior engajamento do estudante com o conteúdo sendo ensinado. Esta prática, em si, não pode ser considerada uma prática nova. Por exemplo, a prática de competições de soletrar, os chamados *spelling bees*, são uma forma de transformar em jogo o aprendizado de gramática correta por estudantes de diferentes partes do mundo. A competição proposta pelo jogo é um grande motivador para que os alunos aprendam a gramática correta da maior quantidade possível de palavras, consequentemente impulsionando a aprendizagem destes alunos.
O termo Aprendizagem Baseada em Jogos, no entanto, é utilizado atualmente para referir-se principalmente à utilização de jogos digitais no ambiente de ensino. Esta prática possui origens no começo da década de 1970, sendo *Oregon Trail* o primeiro jogo notável deste gênero. Lançado em 1971, o jogo trata-se de guiar uma família de pioneiros de Missouri até Oregon, no ano de 1847, utilizando-se de uma carroça. Dado o cenário do jogo, o jogador interagia com diversos conceitos: matemática (as diferentes quantidades de variados suprimentos para a viagem), geografia nacional (o jogador passava por diversas localidades reais nos Estados Unidos), história (o jogo em si tratava-se do contexto histórico da emigração para o oeste dos Estados Unidos) e biologia (o jogador deveria lidar com possíveis doenças que poderiam aflingir a família de pioneiros). Este jogo foi extremamente bem recebido, sendo utilizado na maioria das escolas norte-americanas de 1971 até hoje, e foi o marco do reconhecimento da Aprendizagem Baseada em Jogos pelo público geral.

2.2 Histórico

Após o grande sucesso de *Oregon Trail*, várias companhias tentaram reproduzir o seu sucesso, gerando o gênero de jogos atualmente conhecido como "jogos educacionais". Alguns exemplos notáveis de jogos que sucedem *Oregon Trail* neste mesmo gênero são: *Math Blaster* (1983), um jogo de computador que mistura elementos de ação com cálculos de matemática primária; *Sim City* (1989), um jogo que simula a construção de uma cidade, e serve para ensinar tanto a estrutura e necessidades básicas de uma cidade, quanto a administração de recursos finitos; e *Reader Rabbit* (1983), conhecido no Brasil como Coelho Sabido, uma série de jogos que ensinavam conceitos do maternal até o segundo grau.
Após a virada do milênio, o conceito de jogos educacionais havia perdido grande parte de

sua potência. As grandes companhias de desenvolvimento de jogos haviam tentado ingressar neste mercado, sem obter o sucesso esperado, possivelmente pela percepção do público geral de que estes jogos não gerariam o mesmo nível de entretenimento que jogos regulares. Além disso, educadores haviam perdido a crença no potencial real de jogos como ferramentas educacionais, e grandes discussões sobre as influências negativas dos jogos digitais começavam a ganhar espaço na mídia.

Assim, por mais que o gênero dos jogos educacionais não tivesse sido completamente extinto, a comercialização de jogos acompanhados por este termo tornou-se uma prática extremamente incomum. Jogos como *Brain Age* (2005) e *Big Brain Academy* (2005) são um exemplo desta prática: ambos sendo comercializados como "treinamentos para o cérebro", ao invés de jogos explicitamente educacionais. O primeiro destes foi um grande sucesso comercial, vendendo mais de 20 milhões de cópias globalmente, sucesso este atribuido pelo marketing, e pela plataforma (sendo esta o *Nintendo DS*, atualmente o segundo console mais vendido de todos os tempos). A escolha desta plataforma também é bastante relevante, pois o *Nintendo DS* é um console portátil, que era bastante popular dentre múltiplas faixas etárias. A filosofia da companhia Nintendo, na época, era justamente abranger o maior grupo de jogadores possíveis, proporcionando experiências variadas e acessíveis para indivíduos com diferentes familiaridades com jogos digitais. O diferencial do portátil da Nintendo, na época, era sua tela de toque (bastante inovador para a época, ocorrendo concomitantemente com a popularização de *smartphones* com a mesma funcionalidade), que permitia interações mais simples do jogador com o jogo. Além disso, a portabilidade do *Nintendo DS* permitia que até mesmo usuários mais ocupados pudessem ter pequenas sessões de jogabilidade durante intervalos em suas rotinas. Isto é refletido também na duração de sessões/ciclos de jogabilidade, que dificilmente passavam de alguns minutos.

Atualmente, o jogo mais relevante que possui usos de Aprendizagem Baseada em Jogos é o popular *Minecraft*. Originalmente lançado em 2011, o jogo baseado em voxels é focado em estimular a criatividade de seus jogadores, acima de tudo. Assim, a grande liberdade de construção de ambientes permite que sejam criados ambientes educacionais, para que alunos aprendam sobre os mais diversos temas. A grande popularidade do jogo com o público infantil incentivou a empresa (*Mojang*) a lançar uma versão educacional do jogo, entitulada *Minecraft: Education Edition*. Esta edição conta com ferramentas para auxiliar o professor a melhor observar o progresso de seus estudantes, e permite que sejam dadas aulas de biologia, matemática, e lógica computacional.

2.3 Cenário Atual

Atualmente, as expectativas com relação à Aprendizagem Baseada em Jogos estão em ascenção. Por mais que o mercado conte com poucos produtos com alto reconhecimento, como o anteriormente mencionado *Minecraft: Education Edition*, as condições para uma ressurgência de popularidade do conceitos estão presentes. A popularização dos *smartphones* como elementos presentes no cotidiano de indivíduos de todas as faixas etárias trouxe consigo uma oportunidade única para os jogos educacionais. Esta oportunidade, naturalmente, é a de atingir um público abrangente, de forma acessível, por meio de uma plataforma que é considerada *mainstream*.

Este cenário difere de todos os outros anteriormente existentes por dois principais motivos. Primeiramente, por tratar-se de uma plataforma de múltiplos usos, os jogos educacionais nela presentes podem dissociar-se, em parte, do preconceito de serem somente jogos, destinados a um público específico. Assim, é proporcionado um cenário em que a Aprendizagem Baseada em Jogos deixa de ser invalidada por sua plataforma ou conceitos pré-formados sobre si.

Adicionalmente, por tratar-se de uma plataforma móvel, o acesso aos jogos que utilizam-se de Aprendizagem Baseada em Jogos em diversos momentos do dia é facilitado, em oposição a ser restrita a ambientes específicos, como salas de aula ou a residência do usuário. Este conceito é especialmente importante para o público jovem, que encontra-se intimamente ligado com seus dispositivos móveis, seja para lazer, comunicação, ou até mesmo estudos.

Capítulo 3

Sobre o Jogo

3.1 Metodologia de Planejamento

O trabalho no desenvolvimento de um jogo inicia-se com seu planejamento formal. As ideias propostas por um ou mais desenvolvedores são compiladas em um documento chamado *Game Design Document*. O *Game Design Document* (comumente abreviado como *GDD*) é um documento que contém a descrição do jogo a ser desenvolvido, com o objetivo de guiar os desenvolvedores com relação a direção e aspectos deste mesmo jogo. Em se tratando da indústria de jogos, o *Game Design Document* é produzido em um esforço coletivo entre *designers*, artistas e programadores de uma mesma empresa, e é útil tanto para reforçar as escolhas da equipe para cada indivíduo da mesma, como facilitar o entendimento das premissas necessárias para o desenvolvimento do jogo para um potencial novo membro desta empresa.
Os *Game Design Documents* são documentos altamente descritivos e flexíveis, podendo conter em si texto, imagens, diagramas, e até mesmo arte conceitual. Ademais, são também documentos conhecidos como "documentos dinâmicos"ou (*living documents*), ou seja, são documentos que podem ser atualizados a qualquer momento do ciclo de vida de um projeto, para refletir possíveis ajustes na direção do projeto, ou a adição de novos conceitos ao mesmo.
Ainda que a escrita de um *Game Design Document* seja mandatório por várias empresas de desenvolvimento de jogos, a indústria de desenvolvimento de jogos não encontrou um padrão para o formato desses documentos. Ainda assim, existem várias similaridades entre os diversos *Game Design Documents* publicados. Todos eles possuem o mesmo propósito: a descrição do jogo, seu apelo, seu público-alvo, e diversos aspectos conceituais e técnicos do mesmo. O documento é dividido em seções, de forma a facilitar a manutenção e consulta de seções pertinentes por desenvolvedores do jogo. Algumas seções comuns são: história, jogabilidade, *design* de níveis, arte, música, interface/controles do jogo. No entanto, nenhum dos itens desta lista deve obrigatoriamente estar presente em todos *Game Design Documents*, assim como esta lista não contém todos os possíveis itens presentes nestes documentos.
A criação de um *Game Design Document* foi o primeiro passo deste trabalho, também, e pode ser conferido em sua integridade na página principal do mesmo. Assim como outros documentos do mesmo tipo, o *GDD* deste trabalho sofreu algumas alterações durante o processo de desenvolvimento. No entanto, neste texto trataremos da descrição dos aspectos da versão mais atual do *Game Design Document*, para favorecer a escrita de um texto mais direto, e evitar confusão.

3.2 *Game Design Document*

Esta seção será composta da explicação e detalhamentos das escolhas presentes no *Game Design Document*. Ela seguirá a mesma estrutura de tópicos presente no documento, uma vez que esta estrutura é a mais natural para a compreensão dos conceitos que formam o jogo.

3.2.1 Estrutura e Progressão

Na primeira seção do *Game Design Document*, definimos os conceitos mais fundamentais do jogo. Utilizamos a palavra "cena"para descrever diferentes momentos do jogo, que podem diferir em interface, jogabilidade e propósito. O jogo é formado pela combinação de dois tipos de cenas. O primeiro tipo é chamado de "locais", e o segundo tipo é chamado de "quebra-cabeças". Ambos tipos de cenas serão explicados em maior detalhe posteriormente. O jogador começa sua aventura em um determinado local, e deve interagir com os elementos de cada mapa para eventualmente encontrar quebra-cabeças. Ao resolver estes quebra-cabeças, o jogador obtém progresso. O jogo possui uma história, e portanto possui começo e fim bem definidos. A história foi implementada como uma fonte de motivação para a resolução dos quebra-cabeças. É parte do planejamento a adição de múltiplos quebra-cabeças, de dificuldades variadas. Os quebra-cabeças mais simples devem ser ligados à história, enquanto os mais complicados devem ser opcionais, deixando para o jogador a decisão de resolvê-los. No entanto, em seu estado atual, o trabalho apenas contém quebra-cabeças ligados à história.
Adicionalmente, resolver quebra-cabeças garante ao jogador uma determinada quantidade de "experiência". Esta quantidade de experiência é variável, dependendo de certos critérios (quantidade de tentativas para resolver o quebra-cabeça, se a resolução do quebra-cabeça foi ótima, etc). Se o jogador ganhar experiência o suficiente, ele pode subir de nível. No estado atual do projeto, subir de nível não garante nenhuma recompensa, porém é planejado que estas existam. O objetivo da existência destas mecânicas será detalhado posteriormente.

3.2.2 Jogabilidade

Como foi especificado anteriormente, o jogo possui duas cenas distintas: os locais e os quebra-cabeças.

Os locais são o meio pelo qual o jogador interage com os elementos do mundo do jogo. O jogador deve explorar o ambiente e conversar com PNJs (personagens não-jogáveis) para descobrir mais sobre o mundo do jogo e progredir a história. Este modo é apresentado em uma visão superior (*top-down*). O jogador pode andar e interagir com os elementos de cada local. Estas interações podem gerar tanto uma descrição mais detalhada do que está sendo examinada, quanto conversas com os PNJs. Estas interações podem levar o jogador a um quebra-cabeça.
Existem dois modos de controle aplicáveis para este tipo de cena. No modo "direto"o jogador pode tocar diretamente no local para onde deseja mover-se, e para interagir com PNJs, basta tocar nos mesmos, uma vez que se esteja próximo o suficente. Já no modo "*gamepad* virtual", botões direcionais e um botão de interação são providenciados para o jogador, para que seja possível a movimentação e interação com PNJs, respectivamente.

Os quebra-cabeças são quebras-cabeça que devem ser resolvidos pelo jogador. Estes quebra-cabeças são de natureza diversa, mas em geral são quebra-cabeças matemáticos. Os controles nos quebra-cabeças são diferentes para cada um deles, dependendo da necessidade de cada um. Os contextos dos quebra-cabeças podem ser levemente adaptados para serem relacionados ao momento da história em que se encontram. Os quebra-cabeças foram baseados nos enigmas presentes nos livros *Of Course! The Greatest Collection Of Riddles & Brain Teasers For Expanding Your Mind* (Guido, 2013) e *101 Puzzles* (Perelman, 2013).

3.2.3 História

No início do jogo, o jogador depara-se com uma tela escura, em que são feitas diversas perguntas para o jogador. A maioria destas perguntas são apenas para contextualizar o jogador, e criar um senso de mistério, mas dentre estas perguntas haverá uma oportunidade para o jogador inserir seu próprio nome, nomeando o personagem principal.
O personagem principal encontra-se, então, perdido em uma floresta, desconhecida para o mesmo. Após explorar um pouco, o personagem principal nota que sua aparência foi alterada para a de um cãozinho, e que ele encontra-se também incapaz de comunicar-se, podendo apenas emitir latidos. O personagem principal supõe que encontra-se em um sonho, e tenta voltar a dormir, com a esperança de acordar na forma regular em que se identifica. No entanto, ao tentar fazê-lo, é interrompido por uma libélula, que afirma entender as dificuldades pelas quais o personagem principal encontra-se passando, e assegura a este que a situação em que o mesmo se encontra não é nenhum tipo de sonho ou ilusão. Frustrado, o personagem principal encontra-se obrigado a aceitar a condição em que se encontra, e consequentemente a ajuda oferecida pela libélula.
Os dois, então, dirigem-se para uma comunidade próxima, na qual o personagem principal é incentivado a interagir com os habitantes lá presentes. Ainda que contrariado, o personagem principal interage com diversos dos habitantes locais, alguns dos quais requerem ajuda com problemas pessoais. O personagem principal dispõe-se a ajudar com estes problemas (o que leva o jogador a resolver quebra-cabeças), e é recompensado com gratidão por seus esforços. Ao ajudar os diversos habitantes, o personagem principal percebe-se cada vez mais experiente (o que é transmitido também de forma numérica pela barra de experiência). Esta experiência traduz-se também em uma maior facilidade em compreender as intenções e sentimentos dos outros indivíduos que o cercam. Nos momentos próximos ao fim de sua aventura, o personagem principal recupera sua capacidade de falar, utilizando-a para alguns últimos desafios cruciais. Superando estes, o personagem principal finalmente recupera sua aparência original (humana), e percebe que todos os seres com quem interagiu durante sua aventura eram também seres humanos.

A história do jogo apresenta uma representação metafórica da entrada na vida adulta. O jogador começa atordoado e perdido, em um ambiente em que não reconhece, e percebe-se incapaz de comunicar-se com os demais indivíduos que lá habitam. O jogador logo percebe que sua forma também não é como a que ele recordava ser, e isto faz com que ele desmotive-se. No entanto, com a ajuda de um companheiro mais experiente, que ajuda o jogador a compreender os funcionamentos deste novo ambiente e situação em que se encontra, o mesmo passa a recuperar suas noções regulares. Pelo fim do jogo, o jogador recupera sua capacidade de fala, e no fim, percebe que tem se aventurado por lugares conhecidos, e então recupera sua forma anterior (humana).

3.3 Objetivos do *Design*

As decisões de *design* aplicadas no desenvolvimento de cada uma das partes do jogo foram todas feitas com propósitos claros em mente. Nesta seção, analisaremos cada uma delas em detalhes.

A primeira decisão deliberada é relacionada aos quebra-cabeças. A existência de quebra-cabeças opcionais foi decidida no início do projeto, determinando que a estes seriam reservados os quebra-cabeças mais complexos do jogo. Esta decisão foi feita com dois objetivos. Primeiramente, ao permitir que os quebra-cabeças mais difíceis sejam opcionais, evitamos frustrar um jogador com engajamento menor, que interessa-se pela história mais do que a jogabilidade em si. Isto permite que este jogador tenha algum aprendizado com os quebra-cabeças mais simples, e possivelmente desenvolva interesse por alguns dos quebra-cabeças opcionais, sem prejudicar as chances do jogo ser completo por este jogador. Ademais, a existência de quebra-cabeças adicionais à história permite que jogadores mais interessados aprofundem-se com quebra-cabeças mais complexos. Estes quebra-cabeças também são grandes fontes de experiência para o jogador, sendo esta o tópico da próxima decisão de *design* a ser analisada.

A implementação da barra de experiência e os níveis de jogador também foi uma decisão consolidada nos primeiros momentos do projeto. Esta decisão, conjuntamente com a de tornar opcional os quebra-cabeças mais complexos, foram feitas em favor de aumentar a taxa de retenção (*retention rate*) do jogo. A taxa de retenção de um jogo é definida como a porcentagem dos jogadores que permanecem jogando o jogo após uma determinada quantidade de dias. Segundo uma pesquisa feita pelo site *Business of Apps* (Freier, 2018), menos de 15% dos aplicativos de jogos possuem uma taxa de retenção de 35% após o primeiro dia. O infográfico da figura 3.1 apresenta os resultados da pesquisa em detalhes:

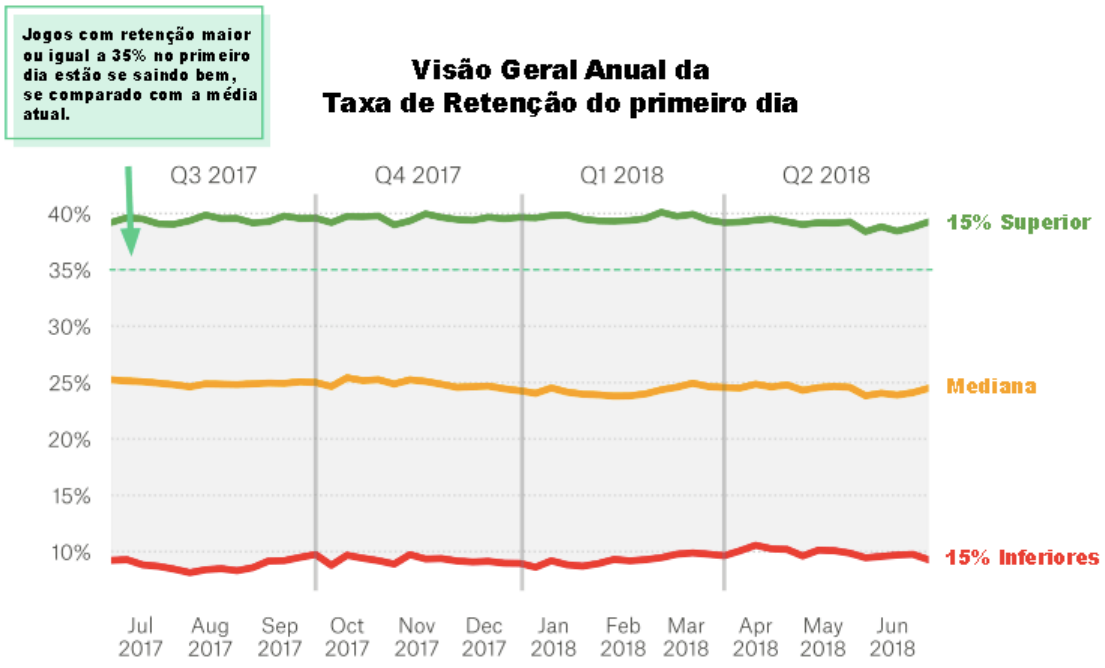

Figura 3.1: *Visão geral da taxa de retenção do primeiro dia de aplicativos de jogos*

Adicionalmente, o mesmo artigo reporta que, após o vigésimo-oitavo dia, apenas 6% dos jogadores permanecem jogando um mesmo jogo. É possível perceber, então, que por mais que o cenário atual de integração de dispositivos móveis permitam acesso a um maior público, a grande quantidade de aplicações disponíveis para acesso por este mesmo público significa que manter o usuário interessado e engajado é uma das grandes prioridades para o desenvolvedor de jogos.
A barra de experiência auxilia no aumento da taxa de retenção do jogo, uma vez que torna-se uma forma de quantificar o progresso do jogador, incentivando que o mesmo continue evoluindo. Adicionalmente, a barra de experiência também serve para evitar a resolução de quebra-cabeças por tentativa-e-erro, uma vez que a experiência recebida é reduzida para cada tentativa errada feita pelo jogador.

Por fim, a história também é um elemento essencial para a composição do jogo, e foi um dos aspectos melhor desenvolvidos durante o processo de criação do mesmo. Além de ser um elemento didático autônomo, ensinando o jogador por meio de metáforas, é também um dos elementos que auxilia no aumento da taxa de retenção dos jogadores. Existem diversos tipos de jogadores, que buscam os mais variados tipos de entretenimento nos jogos eletrônicos. Diversos pesquisadores utilizaram-se da psicologia aplicada a *games* para construir teorias sobre a categorização dos tipos de jogadores existentes. Uma das teorias mais conhecidas é a proposta por Bartle, e é chamada de Taxonomia de Bartle de tipos de jogadores. Esta teoria foi originalmente criada para aplicar-se apenas para jogos *MMO* (ou *Massive Multiplayer Online*), mas atualmente pode ser aplicada para jogos de jogador único. As implicações desta teoria estão representadas na figura 3.2.

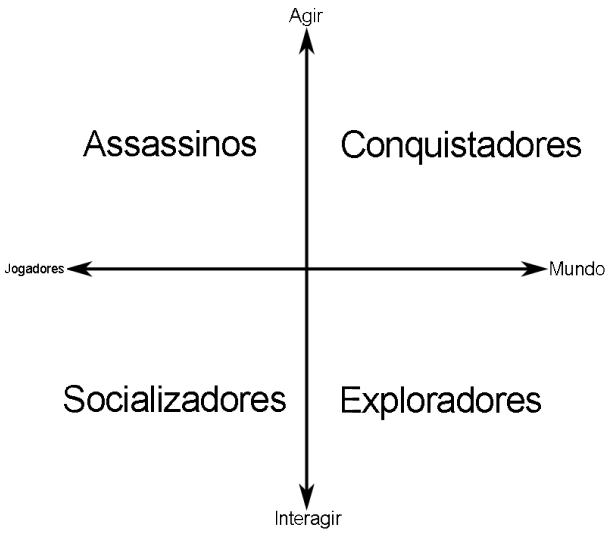

Figura 3.2: *Taxonomia de Bartle de tipos de jogadores*

Segundo esta taxonomia, jogadores do tipo *achievers* permaneceriam jogando o jogo pelo desejo de aumentar o nível de jogador e obter mais experiência, enquanto jogadores do tipo *explorers* interessariam-se pelo progresso na história do jogo e o desenvolvimento dos personagens, justificando portanto a inclusão de ambas no jogo.
Finalmente, a inclusão da possibilidade do jogador de inserir seu próprio nome é, naturalmente, em favor de maior engajamento do jogador com a história e evolução pessoal do personagem principal.

Capítulo 4

Plataforma de Desenvolvimento

Para auxiliar o desenvolvimento do projeto, foi decidido utilizar-se de uma *game engine*. *Game Engines* (ou somente "*engines*"[1]) são ambientes de desenvolvimento de *software*, com foco em desenvolvimento de jogos. As *engines* normalmente providenciam algumas funcionalidades para o programador, para evitar que este precise programá-las antes de poder iniciar o trabalho no jogo própriamente dito. Dentre estas funcionalidades, destacam-se: um componente de renderização de gráficos (sejam eles 2D ou 3D), além de componentes de áudio, físicas, redes, entre outros. Algumas *engines* contam com editores de texto, com noções de sintaxe para as linguagens suportadas pela própria *engine*.
Para os propósitos deste trabalho, foi escolhida a *Godot Engine*.

4.1 *Godot Engine*: uma introdução

A *Godot Engine* é uma *engine* para desenvolvimento de jogos 2D e 3D. Seu desenvolvimento foi iniciado em 2007 por Juan 'reduz' Linietsky e Ariel 'punto' Manzur, e sua primeira versão foi disponibilizada no ano de 2014, ano em que a *engine* também tornou-se *open source*. O projeto iniciou-se com foco em desenvolvimento de jogos 2D, nas versões 1.0 (2014) e 2.0 (2016). A partir da versão 3.0 (2018), a *engine* passou a dar grande suporte para desenvolvimento de jogos 3D, também. A versão utilizada neste projeto é a 3.1.1.
A *engine* conta com diversas características interessantes, como o suporte a Programação Visual, para programadores com menos experiência, mas focaremos principalmente nas características relevantes para o entendimento da estrutura do projeto.

4.2 Elementos da *engine*

4.2.1 Elementos móveis

Uma vez feita a decisão de desenvolver para plataformas móveis, é necessário que haja compatibilidade da *engine* com o desenvolvimento para as mesmas. Felizmente, a *Godot Engine* satisfaz os diversos critérios necessários para o desenvolvimento de jogos nestas plataformas.
Primeiramente, a *engine* conta com suporte nativo para detectar telas de toque nos dispositivos, e permitir que o programador crie interações baseadas neste sinais. Outros elementos pertinentes aos dispositivos móveis, como suporte a sinais de acelerômetro presentes em

[1]Por mais que a tradução "motores de jogos"possa ser utilizada, as *Game Engines* são normalmente referidas somente por *engines*, e portanto este será o vocabulário utilizado nesta monografia.

muitos destes dispositivos, permite a criação de experiências que aproveitam ao máximo a plataforma de escolha.
Adicionalmente, a *engine* disponibiliza um recurso bastante utilizado por desenvolvedores de aplicativos móveis: a função de *One-Click Deploy*. Esta função permite que, feita a configuração apropriada, seja possível exportar o aplicativo por meio de um único clique para um dispositivo móvel conectado à máquina utilizada para o desenvolvimento do aplicativo. Existem limitações, no entanto: a exportação pode ser feita somente para dispostivos com o sistema *Android*, e para que esta instalação e execução seja possível, é muitas vezes necessário que o dispositivo em questão tenha em si habilitado o modo desenvolvedor, o que pode apresentar riscos de segurança para usuários menos experientes, ou mais desatentos.
Por fim, a *engine* disponibiliza também a possibilidade de simular sinais de toque utilizando-se do ponteiro do *mouse*, para que testes mais simples possam ser executados com facilidade na máquina de desenvolvimento, evitando a necessidade de executar o *One-Click Deploy* toda vez, uma vez que este é um processo razoavelmente demorado, pois requer a exportação do aplicativo toda vez que é executado.

4.2.2 Linguagem de Programação

A *Godot Engine* permite a utilização de diversas linguagens de programação para a escrita do código de jogos. A *engine* suporta oficialmente as linguagens C, C++, *VisualScript*, e *GDScript*, esta última sendo a linguagem oficial da *engine*, desenvolvida em conjunto com a mesma. A linguagem de programação escolhida para o desenvolvimento deste projeto é a *GDScript*.

A linguagem *GDSCript* é uma linguagem de alto nível de tipagem dinâmica, e possui algumas vantagens únicas[2], devido à sua integração com a *Godot Engine*:

- É uma linguagem simples e elegante, com elementos similares a linguagens conhecidas, como *Python*, *Lua* e *Squirrel*;
- É carregada e compilada rapidamente;
- É compatível com o editor, com a capacidade de reconhecer sintaxe da estrura de cenas específica da *Godot Engine*;
- Possui tipos de vetores inclusos em si, permitindo o eficiente uso de álgebra linear;
- Possui suporte a múltiplas *threads* tão eficientemente quanto linguagens de tipagem estática;
- Sua natureza dinâmica facilita a utilização de trechos em C++ (via *GDNative*) se melhor desempenho for requerido, sem a necessidade de re-compilar a *engine*.

Para a melhor compreensão e visualização da linguagem, segue um arquivo contendo um simples exemplo de um arquivo *GDScript* genérico, presente na documentação oficial da *engine*:

[2]Lista obtida da documentação oficial da *Godot Engine*: https://docs.godotengine.org/en/3.1/getting_started/step_by_step/scripting.html

```
# A file is a class!

# Inheritance

extends BaseClass

# (optional) class definition with a custom icon

class_name MyClass, "res://path/to/optional/icon.svg"

# Member Variables

var a = 5
var s = "Hello"
var arr = [1, 2, 3]
var dict = {"key": "value", 2:3}
var typed_var: int
var inferred_type := "String"

# Constants

const ANSWER = 42
const THE_NAME = "Charly"

# Enums

enum {UNIT_NEUTRAL, UNIT_ENEMY, UNIT_ALLY}
enum Named {THING_1, THING_2, ANOTHER_THING = -1}

# Built-in Vector Types

var v2 = Vector2(1, 2)
var v3 = Vector3(1, 2, 3)

# Function

func some_function(param1, param2):
    var local_var = 5

    if param1 < local_var:
        print(param1)
    elif param2 > 5:
        print(param2)
    else:
        print("Fail!")

    for i in range(20):
        print(i)

    while param2 != 0:
        param2 -= 1

    var local_var2 = param1 + 3
    return local_var2
```

Adicionalmente, o editor incluso na *engine* conta com diversas ferramentas para facilitar a depuração do código do jogo. Além de ressaltar a parte do código com problemas, no momento em que um erro é gerado, é possível adicionar *breakpoints* no próprio editor. Finalmente, existem diversas ferramentas separadas do editor para auxiliar na identificação e correção de erros, que serão explicadas com mais detalhes posteriormente.

4.2.3 Estrutura Lógica

Os jogos digitais são compostos de um ou mais modos de jogabilidade, aplicados em canários variados, que são conectados de forma a construir um jogo completo. O mais clássico exemplo deste conceito é o jogo *Super Mario Bros.*, lançado para o *Nintendo Entertainment System* em 1985.

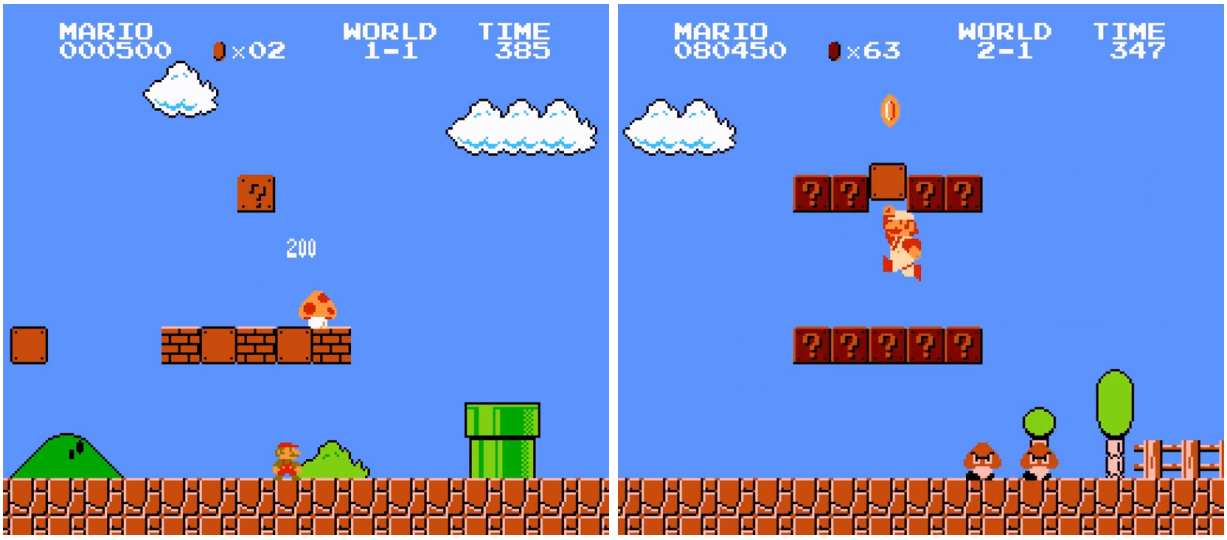

Figura 4.1: *À esquerda: primeira fase do jogo Super Mario Bros. À direita: quinta fase do jogo Super Mario Bros.*

Observando as capturas de tela presentes na figura 4.1, podemos notar que, ainda que sejam cenários diferentes, elementos do jogo preservam-se: o personagem controlado é sempre o mesmo, e os controles e formas de interação com os diversos elementos do jogo preservam-se entre os diferentes cenários propostos para o jogador.

Tendo em mente este conceito, a *Godot Engine* propõe uma estrutura que acomoda a necessidade modular dos desenvolvedores de jogos: a estrutura de cenas e nós.
Na *Godot Engine*, nós são os blocos básicos para a construção de cada jogo. Cada nó pode realizar uma variedade de funções especializadas, mas todos possuem alguns atributos em comum. Estes atributos são os seguintes:

- Todos nós possuem um nome;
- Todos nós possuem propriedades editáveis;
- Nós podem ser estendidos (para terem mais funções)
- Todo nó pode ser adicionado a outro como seu filho

É notável a existência da estrutura necessária para a criação de uma hierarquia de nós. Este tópico será abordado em mais detalhes posteriormente. É importante também ressaltar que nós possuem tipos, os quais definem as funcionalidades padrões de cada nó. Alguns tipos de nós são apresentados na figura 4.2:

Figura 4.2: *Interface de adição de um novo nó presente na engine.*

Na figura, podemos notar alguns dos tipos mais básicos de nós presente na *engine*. Naturalmente, o tipo mais básico de nó chama-se *Node*. Todos os demais tipos de nós herdam as funcionalidades básicas deste. A forma com que os nós herdam funções e parâmetros de outros tipos de nós assemelha-se muito à noção de classes e subclasses do paradigma de Programação Orientada a Objetos. Ainda na figura, podemos observar os dois tipos de nós mais importantes para o desenvolvimento de jogos 2D: *Control* e *Node2D*. Os nós do tipo *Control* são elementos básicos de interface de usuário (*User Interface*, ou, como é comumente abreviado, *UI*), e dentre suas especializações encontram-se etiquetas (nós de texto), botões, *popups*, barras de progresso, entre outros. Já os nós do tipo *Node2D* são a base de todos os objetos presentes em um jogo 2D. Nós deste tipo possuem posições, rotações, escala, entre outros atributos relevantes. Especializações de nós deste tipo são compostas por *sprites*, cameras, corpos físicos, partículas, entre outros.

A composição de um ou mais nós e seus filhos gera o que chamamos de uma cena. As cenas, na *Godot Engine*, são utilizadas para representar momentos ou partes do jogo. A composição de cenas pode gerar cenários complexos e vivos, que eventualmente tornam-se em partes completas do jogo. Estas então são concatenadas, para gerar o jogo completo. A compreensão deste conceito é facilitada ao observar a cena do personagem principal deste projeto, apresentada na figura 4.3:

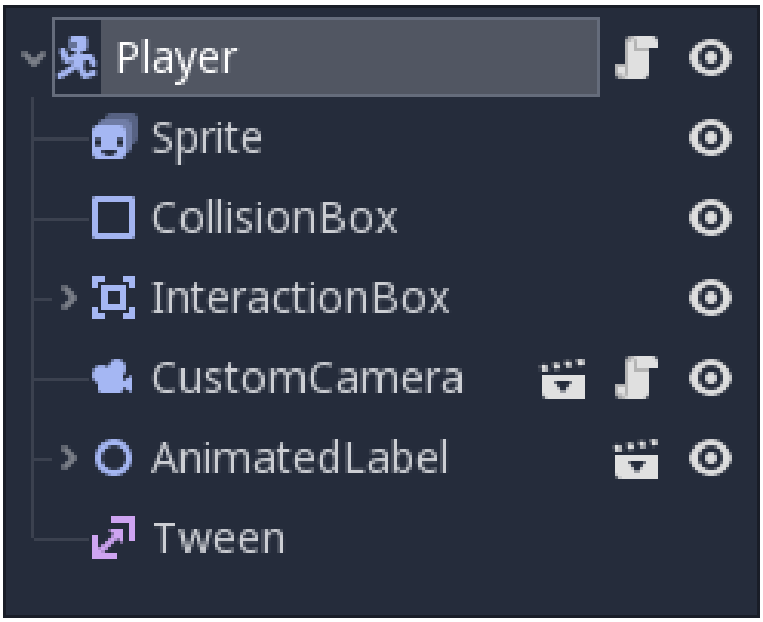

Figura 4.3: *Estrutura de nós que constituem a cena do personagem controlado pelo jogador.*

O nó-raiz do personagem principal é do tipo *KinematicBody2D* (cujo nome é *Player*), o que garante a propriedade de interagir com outros objetos que possuam física atrelado a eles. Esta propriedade permite programar colisão e interação com os limites dos mapas, objetos, e PNJs (Personagens Não-Jogáveis) presentes no jogo. O nó que define a forma e limites desta colisão é do tipo *CollisionShape2D*, e a interação com os PNJs é determinada por um nó do tipo *Area2D*, o qual também possui uma *CollisionShape2D* como nó filho. Adicionalmente, o personagem principal possui um *AnimatedSprite*, que define sua aparência e animações, por via de uma série de *sprites* exibidas em sequência, uma *Camera2D*, que faz com que o personagem controlado pelo jogador esteja sempre em foco durante o jogo, e define os limites da camera, para que o jogador veja apenas as porções desejadas de cada mapa. Os nós *AnimatedLabel* (que é, na verdade, um *Node2D* em sua raíz) e *Tween* são utilizados para animações mais complexas do personagem principal.
É notável a complexidade de apenas um elemento do jogo: a composição de vários nós é utilizada para a criação de um componente complexo do jogo. É possível notar, também, ícones de pergaminhos ao lado de determinados nós. Estes ícones sinalizam que existe um *script* associado a este nó. Cada um destes *scripts* são escritos na linguagem *GDScript*, e servem para garantir comportamentos e funcionalidades mais complexas do que apenas as funcionalidades padrões de cada nó. Por exemplo, o *script* ligado ao nó-raiz do personagem principal permite que este mova-se, tenha os mais diversos tipos de interações com PNJs, e assuma outros comportamentos de controle, estes importantes para a criação de momentos cinemáticos no jogo.

No entanto, o personagem principal não é suficiente para a composição de um jogo completo. Assim, criamos cenas para os diversos mapas, personagens, e elementos que constituem os cenários presentes na narrativa do jogo. A composição destes elementos é feita por meio da criação de instâncias destas cenas. A seguir, observe a estrutura da cena do primeiro mapa do jogo, apresentada na figura 4.4:

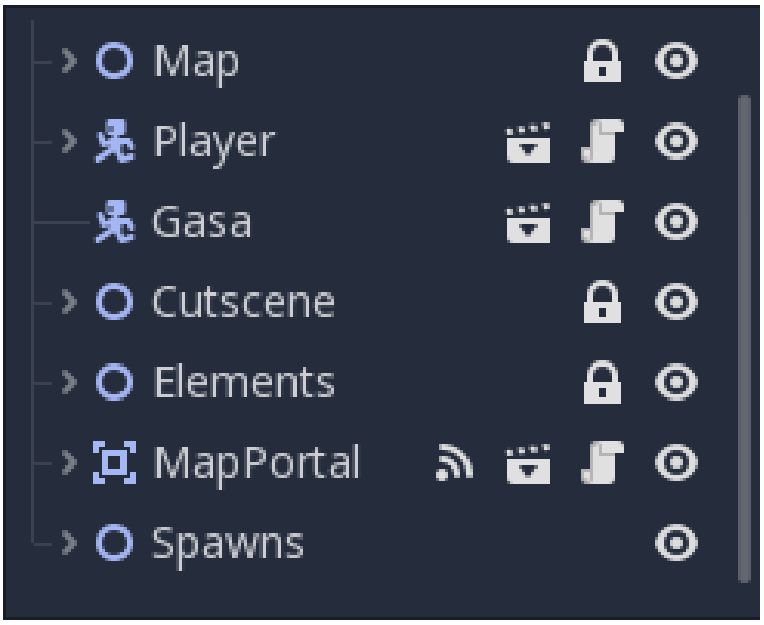

Figura 4.4: *Estrutura de nós que constituem a cena do mapa entitulado "Forest Wake".*

É possível notar a presença do nó do jogador principal que vimos anteriormente. Adicionalmente, temos nós que geram o mapa da cena atual (*Map*), mais um personagem (chamado Gasa), um nó que contém os elementos interagíveis desta cena (*Elements*), a saída do mapa, que gera uma transição para outra cena (*MapPortal*), e dois nós contendo elementos de controle: o nó *Spawns* determina as posições em que o jogador é colocado, a depender do contexto de sua entrada na cena, enquanto o nó *Cutscene* contém todos os elementos necessários para gerar as cenas cinemáticas da história, como balões de fala, e geradores de animações, por exemplo.

Em conclusão, a estrutura lógica que rege a composição de um jogo na *Godot Engine* é feita da seguinte maneira: nós são agrupados para formar cenas, as quais podem ser consideradas árvores de nós; estas, por sua vez, são agrupadas, e formam cenas mais complexas. Estas cenas complexas são, então, instanciadas conforme a necessidade do jogo, de forma a criar uma sequência coerente de eventos, que constituem este mesmo jogo. A instanciação e manutenção destas cenas é feita utilizando-se da árvore de cenas (*SceneTree*), que encontra-se sempre presente desde a inicialização de cada jogo feito na *Godot Engine*, e cuida de algumas funcionalidades básicas do jogo.

4.2.4 Comunicação entre cenas

Existem diversos cenários existentes em jogos que requerem a comunicação entre dois de seus elementos. Esta comunicação pode ser tão simples quanto o apertar de um botão fazer seu personagem pular, por exemplo, mas pode também constituir comportamentos complexos, em que diversos elementos comunicam-se entre si para gerar um único comportamento.

Na *Godot Engine*, esta comunicação pode ser realizada de algumas maneiras.
O *script* de um nó, inicialmente, "enxerga"apenas a si mesmo e seus filhos, e possui fácil acesso a árvore de cenas.
A primeira maneira com a qual um *script* pode obter fácil acesso a outro *script*, é se este

último for um *Singleton*. *Singletons* são *scripts* ou cenas carregados globalmente, que possuem um identificador único (geralmente seu nome), o qual permite que sejam acessados a qualquer momento, por qualquer outro *script*. Estes *Singletons* são sempre carregados no início da execução do jogo, e portanto estão prontamente disponíveis para qualquer cena ou *script* que os necessitem. Ademais, é possível notar que a nomenclatura e descrição dos *Singletons* remete ao padrão de *design* de mesmo nome.
A segunda maneira presente na *Godot Engine* para a realizar a comunicação entre processos é um pouco mais complexa, e utiliza-se de um sistema chamado de "sinais"(*signals*). Os sinais são um dos grandes motivos que permitem a modularização dos componentes de jogos desenvolvidos na *Godot Engine*. Elas são a versão da *engine* do padrão *observer*, e permitem que nós enviem mensagens, as quais outros nós podem receber, e responder de acordo com a mensagem recebida. Desta forma, ao invés de forçar que elementos do jogos esperem necessariamente a presença de outros elementos especificos para seu funcionamento, é possível que estes apenas emitam sinais, os quais podem ser captados por todos demais elementos interessados.
Um exemplo da utilização de sinais encontra-se na funcionalidade do controle virtual presente no jogo, e sua relação com o personagem principal. A seguir, são destacados trechos relevantes do código do personagem principal (*Player.gd*) e do controle virtual (*VirtualGamepad.gd*):

Listing 4.1: *Player.gd*

```
1 ...
2 func _ready():
3   if (Global.control_mode == Global.ControlModes.direct):
4     set_process_input(true)
5   elif (Global.control_mode == Global.ControlModes.virtual_gamepad):
6     set_process_input(false)
7     var status = []
8     status.append(VirtualGamepad.connect("up_pressed", self, "
          determine_gamepad_movement", [Vector2(0, -1)]))
9     status.append(VirtualGamepad.connect("down_pressed", self, "
          determine_gamepad_movement", [Vector2(0, 1)]))
10    status.append(VirtualGamepad.connect("left_pressed", self, "
          determine_gamepad_movement", [Vector2(-1, 0)]))
11    status.append(VirtualGamepad.connect("right_pressed", self, "
          determine_gamepad_movement", [Vector2(1, 0)]))
12    status.append(VirtualGamepad.connect("confirm_pressed", self, "
          determine_and_trigger_interaction"))
13    status.append(VirtualGamepad.connect("up_released", self, "
          determine_gamepad_movement", [Vector2(0, 1)]))
14    status.append(VirtualGamepad.connect("down_released", self, "
          determine_gamepad_movement", [Vector2(0, -1)]))
15    status.append(VirtualGamepad.connect("left_released", self, "
          determine_gamepad_movement", [Vector2(1, 0)]))
16    status.append(VirtualGamepad.connect("right_released", self, "
          determine_gamepad_movement", [Vector2(-1, 0)]))
17
18    for code in status:
19      if code != 0:
20        print("Something went wrong with signal connection in Player!")
21 ...
```

Listing 4.2: *VirtualGamepad.gd*

```
1 ...
2 func _on_Up_pressed():
3   emit_signal("up_pressed")
4
5
6 func _on_Right_pressed():
7   emit_signal("right_pressed")
8
9
10 func _on_Left_pressed():
11   emit_signal("left_pressed")
12
13
14 func _on_Down_pressed():
15   emit_signal("down_pressed")
16
17
18 func _on_Confirm_pressed():
19   emit_signal("confirm_pressed")
20
21
22 func _on_Up_released():
23   emit_signal("up_released")
24
25
26 func _on_Right_released():
27   emit_signal("right_released")
28
29
30 func _on_Left_released():
31   emit_signal("left_released")
32
33
34 func _on_Down_released():
35   emit_signal("down_released")
36 ...
```

Listing 4.3: *Player.gd*

```
1 ...
2 func _physics_process(delta):
3   if Global.control_mode == Global.ControlModes.direct and moving:
4     move_to(get_viewport().get_mouse_position())
5   elif Global.control_mode == Global.ControlModes.virtual_gamepad and
        virtual_gamepad_direction != Vector2(0, 0):
6     move_to_absolute(virtual_gamepad_direction)
7 ...
8 func determine_gamepad_movement(direction):
9   virtual_gamepad_direction += direction
10  if (virtual_gamepad_direction != Vector2(0, 0)):
11    $Sprite.set_animation("walk")
12  else:
13    $Sprite.set_animation("idle")
14
15 func move_to_absolute(pos):
16  var direction = (pos).normalized()
17  set_sprite_and_interaction_direction(direction)
18  speed = (speed * ACCEL).clamped(MAX_SPEED)
19  move_and_slide(direction * speed)
20 ...
```

Considerando que o *script VirtualGamepad.gd* é um *Singleton*, se atrelássemos a cena do controle virtual diretamente à cena do personagem principal, todo momento de jogo em que a cena do jogador não se encontrasse presente (como, por exemplo, na maioria das cenas dos quebra-cabeças), o jogo simplesmente não funcionaria, pois o *script* do controle virtual queixaria-se sobre a ausência da cena do personagem principal. Em contraste, ao utilizarmos sinais, a ausência da cena do personagem principal não teria nenhuma implicação direta para a cena do controle virtual: a cena emitiria seus sinais normalmente, mesmo sem ter quem os escutasse, e o processo geral do jogo não seria pausado por este problema.

Capítulo 5

Processo de Desenvolvimento

5.1 Visão Geral

Este projeto foi desenvolvido utilizando-se do *Github*, uma plataforma que permite a criação de repositórios de desenvolvimento de *software* com controle de versão. Adicionalmente, o *Github* mantém o controle de diversos dados relevantes para acompanhar o progresso do desenvolvimento de cada projeto, os quais serão utilizados extensivamente neste capítulo.
Este projeto foi produzido ao longo de 119 *commits*, os quais ocorreram no decorrer de um período total de 7 mêses. Este período é constituído pelos mêses de Abril até Novembro, seguindo o cronograma proposto, com uma pequena extensão da finalização do trabalho no mês de Novembro, ao invés do mês estimado de Outubro. O progresso dos *commits* pode ser analisado com o gráfico da figura 5.1, obtido utilizando-se das métricas do *Github*:

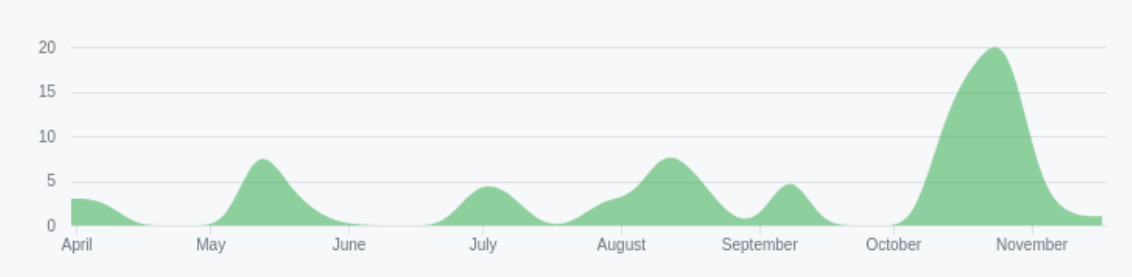

Figura 5.1: *Número de* commits *por semana, no período do desenvolvimento do projeto.*

É notável a existência de picos e vales de *commits* neste gráfico. Isto ocorre pois, em conjunto com o desenvolvimento deste projeto, outros projetos de faculdade estavam sendo desenvolvidos. Desta forma, cada momento dedicado ao desenvolvimento deste projeto implicava no desenvolvimento completo de alguma funcionalidade para o mesmo. Desta forma, era possível reservar tempo para os demais projetos da faculdade, ao mesmo tempo que refletir sobre decisões de *design* e ajustar a direção do jogo, caso necessário. É possível perceber, também, que o mais robusto pico de desenvolvimento ocorreu do começo do mês de Outubro, até o início do mês de Novembro. Isto deve-se, naturalmente, à finalização de diversos trabalhos da faculdade, abrindo maior tempo para o desenvolvimento deste projeto. Ademais, neste ponto, todos os quebra-cabeças haviam sido selecionados para inclusão no jogo, e a história do mesmo finalmente havia tomado forma concreta.
Finalmente, este projeto tem a si atrelado a licensa GLP-3.0, o que torna este projeto *software* livre. Esta licensa garante a liberdade de distribuir, comercializar, e modificar o *software*, e o torna livre para uso pessoal e comercial, ao mesmo tempo que requer que projetos que surjam da modificação deste projeto possuam esta mesma licensa.

5.2 Linha do Tempo

Esta linha do tempo tem como marcos pontos importantes no desenvolvimento do código e funcionalidades do projeto. Ainda que avanços envolvendo a pesquisa de quebra-cabeças e as ponderações relacionadas à história e *design* do jogo sejam, também, um grande aspecto do desenvolvimento deste, elas são mais dificilmente documentadas, e portanto não serão abordadas nesta linha do tempo.

5.2.1 Primeiro Semestre

A criação do repositório, e início do desenvolvimento do jogo, ocorreu nos primeiros dias do mês de Abril. As primeiras prioridades foram a criação do personagem controlável pelo jogador, e um outro personagem com que o jogador pudesse interagir. Neste momento, os gráficos estavam bastante distantes de finalizados, como é possível notar na figura 5.2.

Figura 5.2: *Capturas de tela da primeira cena de teste implementada no jogo.*

A noção de que este projeto seria um jogo para plataformas móveis estava presente desde estes primeiros momentos: a ação de andar já estava associada à ação de toque na tela (traduzida para cliques do ponteiro, para testes na máquina de desenvolvimento). Uma das primeiras dificuldades foi justamente fazer este movimento ser algo agradável. Descobrir como a *engine* entende *inputs* de toque, e como utilizar os dados deste *input* foram desafios encontrados neste passo do desenvolvimento.
Em seguida, ao notar o possível desconforto gerado pela ação de tocar diretamente no local para o qual deseja-se mover, foi desenvolvida uma segunda opção de controle: um controle virtual. Este componente é um controle semi-transparente que é gerado em uma camada acima de toda cena de jogo, que permite que o jogador movimente-se e interaja com elementos presentes no jogo sem ter que mudar o posicionamento de suas mãos.

Após o desenvolvimento destas funcionalidades, o foco tornou-se para um elemento imperativo para facilitar a transmissão de elementos de história: as caixas de texto. O desenvolvimento da caixa de texto foi bastante direto, devido a seu conceito simples, e foi seguido pelo desenvolvimento de "caixas de escolha". Estas últimas permitem que o jogador responda perguntas de alternativas feitas por PNJs, e é geralmente utilizada para gerar opções do tipo "sim ou não"para o jogador.
Ambos os elementos estão presentes em grande parte da jogabilidade fora das cenas de quebra-cabeças. A capacidade de aumentar a velocidade em que o texto é exposto respeita a capacidade de cada jogador, evitando aborrecer ou entediar jogadores que possuam maior velocidade de leitura. Adicionalmente, as "caixas de escolha"podem ser utilizadas para

quebra-cabeças simples, que podem trazer desafios com perguntas de múltiplas escolhas. Alguns destes quebra-cabeças foram selecionados, mas nenhum deles encontra-se presente no estado da entrega deste trabalho.

5.2.2 Segundo Semestre

A entrada no segundo semestre trouxe mudanças para a estrutura do repositório, que passou a conter não só o código-fonte do jogo, mas também a documentação de conceitos pertinentes para o desenvolvimento do trabalho. Adicionalmente, nas primeiras semanas, foi criada a cena base para todos os quebra-cabeças que seriam posteriormente adicionados no jogo, e em seguida uma cena de teste de um quebra-cabeça básico, este observável na figura 5.3.

Figura 5.3: *O quebra-cabeça de testes criado.*

Após a criação da cena de testes do modo de jogo "locais", e uma cena de teste do modo de jogo "quebra-cabeças"(como descrito anteriormente), foi feito o processo de ligação destes dois diferentes modos de jogo. Este processo envolveu aprender sobre a árvore de cenas da *Godot Engine*, e como manipular cenas na mesma. O término desta ligação marcou o fim de mais um ciclo de desenvolvimento.

No mês seguinte, a história havia tomado um forma mais concreta, e portanto o desenvolvimento de mapas além do de teste foi iniciado. O primeiro "mapa"a ser desenvolvido foi a *cutscene* de introdução. Ela é composta somente de texto, o qual é exibido em instâncias da caixa de texto alteradas para esta cena específica. Em seguida, foi necessário entender o conceito de *tilemaps*, e seu funcionamento na *engine*. *Tilemaps* são, como o nome implica mapas compostos de diversas partes de natureza diferente, mas de tamanhos uniformes. Os *tilemaps* são construídos a partir de *tilesets*, que são grande imagens que comprimem todos os elementos de um determinado tipo de mapa. Como um exemplo: no *tileset* de floresta presente no jogo, encontram-se gráficos do piso de grama, das árvores que envolvem os ambientes, de água, de flores, troncos, e muitos mais elementos. Estes diferentes elementos são

dispostos em uma ou mais matrizes, cada uma destas chamadas *tileset*, para construir diferentes ambientes utilizando-se do mesmo conjunto de elementos.

O trabalho no primeiro mapa no qual o jogador possui controle do seu personagem foi o mais extenso, comparado a todos outros mapas. Este fato deve-se, principalmente, à quantidade de componentes novos que precisaram ser feitos para este mapa, e que futuramente foram reaproveitados para os demais mapas.
Primeiramente, a criação do mapa iniciou-se com o desenho de seu *tileset*, e o posicionamento do personagem principal. A primeira adição a este mapa foi uma extensa *cutscene*, que introduz o jogador ao personagem principal, e sua motivação. Em seguida, foram criadas as *flags* globais de progresso do jogador, e foi criada a cena para objetos genéricos com os quais o jogador pode interagir. Adicionalmente, foi feita a ligação entre a introdução e este primeiro mapa, e em seguida foi criado o objeto genérico que permite a transição do personagem principal entre diferentes mapas.

Finalizado este primeiro mapa, foi feito mais um mapa, com mais uma *cutscene*, seguida pela adição de mais uma destas no mapa anterior, para seguir com a sequência de história. O próximo componente relevante adicionado foi o companheiro do protagonista. Este companheiro é uma libélula, e segue o protagonista durante a história. Por ser uma libélula, seus movimentos são esporádicos e rápidos, separados por algumas pequenas pausas.

Finalizado o trabalho neste último elemento, mais componentes relacionados à história foram desenvolvidos, até abrir caminho para a implementação dos quebra-cabeças, os quais serão tratados com mais detalhe posteriormente. Por fim, foram desenvolvidos a função de gravar o progresso do jogo, e carregá-lo da tela de inicio do mesmo, e a barra de experiência do jogador. A figura 5.4 apresenta uma captura de tela da versão do jogo submetida para este trabalho.

Figura 5.4: *Captura de tela de uma cena to tipo "local", na versão submetida para este trabalho.*

5.3 Quebra-Cabeças

5.3.1 O remédio de Shimmi

Este quebra-cabeças foi baseado no quebra-cabeças "*Water Buckets*", do livro *Of Course! The Greatest Collection Of Riddles & Brain Teasers For Expanding Your Mind* (Guido, 2013). A versão implementada consiste de um poço, e dois baldes: um com a capacidade de cinco litros de água, e outro com a capacidade de três litros de água. O objetivo é fazer com que o balde com maior capacidade contenha um total de quatro litros de água em si. O quebra-cabeças permite que os baldes sejam reposicionados tocando e arrastando-os com a tela de toque.
O quebra-cabeças chama-se "O remédio de Shimmi", pois este personagem encontra-se em apuros, e necessita que seu remédios seja diluído em exatamente quatro litros de água para que tenha o efeito desejado. Esta situação é representada nas capturas de tela presentes na figura 5.5.

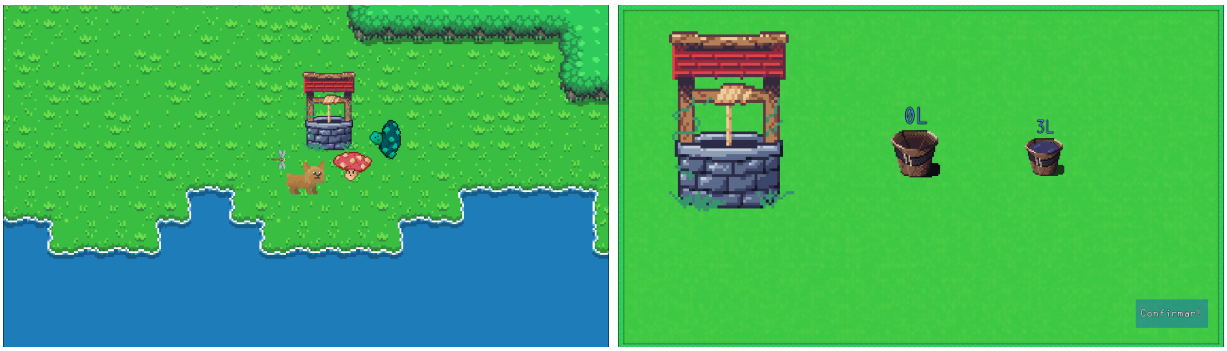

Figura 5.5: *À esqueda: a cena que motiva a resolução do quebra-cabeças. À direita: cena do quebra-cabeças em questão.*

5.3.2 Tempo sem Relógio

Este quebra-cabeças foi baseado no quebra-cabeças "*Rope Burning*", do livro *Of Course! The Greatest Collection Of Riddles & Brain Teasers For Expanding Your Mind* (Guido, 2013). Nestes quebra-cabeças, o objetivo é calcular a passagem de tempo de 45 minutos, utilizando-se de duas cordas, ambas as quais demoram uma hora para serem queimadas completamente, ateando fogo a uma de suas pontas. A queima de segmentos da corda não resulta em tempos exatos, uma vez que a corda apresenta espessura irregular. O jogador pode iniciar a queima das cordas tocando em qualquer uma de suas pontas, e deve queimar ambas cordas para que possa submeter o resultado.
A motivação do quebra-cabeças surge ao encontrar a namorada do personagem titular do quebra-cabeças anterior, a qual encontra-se esperando-o para um encontro, porém não possui relógio para a medição do tempo. Uma instância deste quebra-cabeças é representada na figura 5.6.

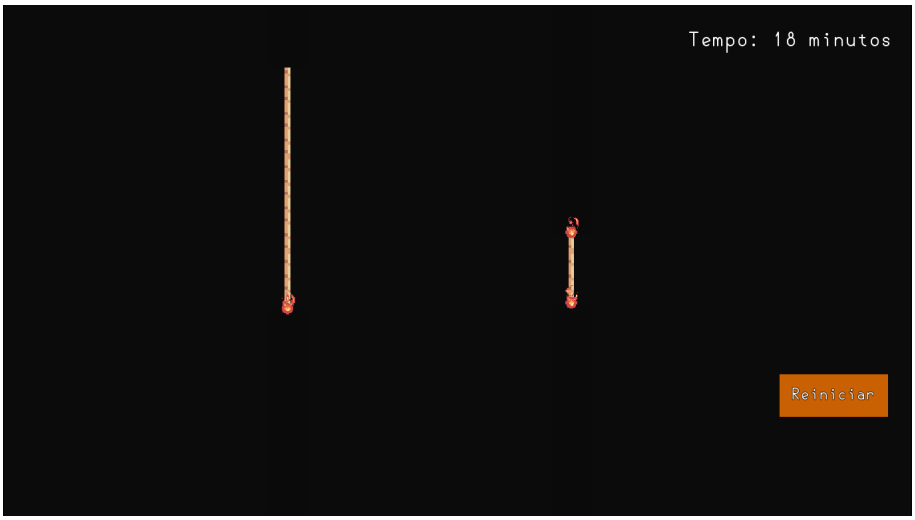

Figura 5.6: *Captura de um momento com ambas cordas queimando, na cena deste quebra-cabeças.*

5.3.3 O jogo de Domi

Este quebra-cabeças foi baseado no quebra-cabeças "*King Arthur's Coin Game*", do livro *Of Course! The Greatest Collection Of Riddles & Brain Teasers For Expanding Your Mind* (Guido, 2013). Este quebra-cabeças consiste de um jogo, em que, em uma mesa redonda, o jogador e seu oponente posicionam moedas em turnos. O primeiro jogador incapaz de colocar uma moeda na mesa perde o jogo. O jogador pode colocar moedas na mesa em seu turno tocando a posição desejada na tela de toque. Ao tocar e segurar, o jogador pode visualizar uma prévia da posição na qual a moeda será colocada. Caso não seja exibida a prévia da moeda, é impossível colocar uma moeda naquele ponto.
Este quebra-cabeças é proposto pelo personagem Domi, como uma espécie de teste para determinar se o personagem principal encontra-se pronto para progredir. Uma instância deste quebra-cabeças é representada na figura 5.7.

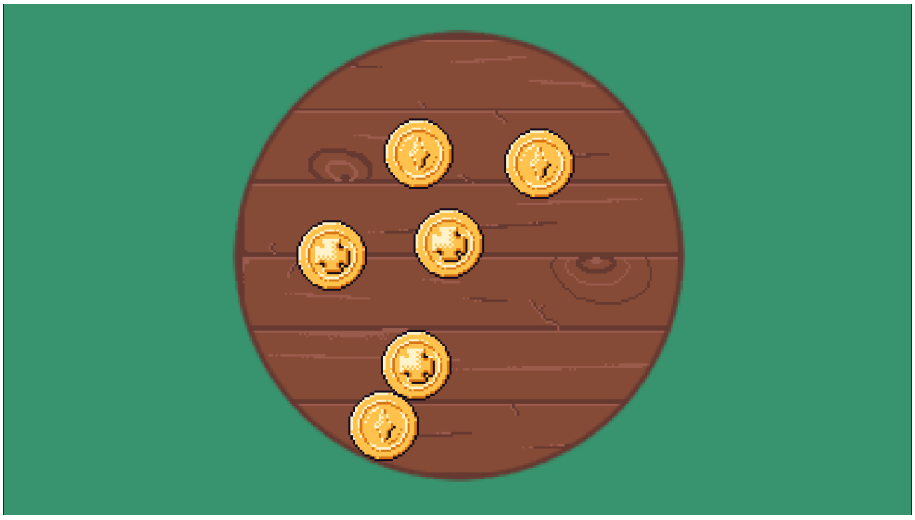

Figura 5.7: *Captura de tela de um momento neste quebra-cabeças.*

Capítulo 6

Conclusões

Com a finalização do trabalho, temos como resultado final um protótipo do jogo, no qual se encontra um ciclo de jogabilidade completo. O jogo segue a filosofia de *design* estabelecida na documentação que precede seu desenvolvimento, proporcionando uma experiência acessível, com foco no engajamento e aprendizagem de seus jogadores.

6.1 Considerações

Por mais que os objetivos de *design* do jogo tenham sido atendidos, a eficácia do mesmo com relação às ambições de ensino e engajamento de usuário ainda requer testes, para que possa ser definitivamente afirmada.
A experiência proposta por um jogo é o resultado da junção de cada uma de suas partes, e de quão bem estas partes se complementam para formar um produto coeso e satisfatório. Assim, por mais que testes tenham sido feitos ao longo do desenvolvimento do jogo para melhor direcionar o mesmo, a adição de novos conteúdos e funcionalidades geram uma experiência de jogo nova, significativamente diferente da experiência anterior, que deve ser analisada de forma singular.
De qualquer forma, para os propósitos desta monografia podemos extrapolar estas limitações, e inferir algumas considerações positivas e negativas com relação ao presente e futuro do projeto.

6.1.1 Positivas

Os jogadores que testaram o projeto durante seu desenvolvimento expressaram, de forma unânime, a satisfação com a apresentação do jogo, e em geral demonstravam agrado com relação ao fato de poderem controlar um cãozinho como o personagem principal. Ademais, a possibilidade de controlar o jogo de duas formas diferentes tornou o jogo mais inclusivo, uma vez que houveram usuários que preferiram tipos diferentes de controles. Jogadores de diferentes faixas etárias apreciaram os quebra-cabeças, elogiando a natureza intuitiva da jogabilidade implementada para a maioria deles. Por fim, foi perceptível o acerto na proporção entre história e quebra-cabeças. O jogador passa os primeiros minutos do jogo sendo introduzido à história, para então ser colocado em uma situação em que ele pode tentar resolver três quebra-cabeças na ordem em que desejar. A função de salvar o jogo posicionada logo após a grande seção de história fez com que jogadores que precisavam jogar em curtos intervalos de tempo pudessem jogar o jogo em múltiplas seções, o que é também um dos grandes objetivos do *design* do projeto.

Em termos de desenvolvimento futuro, o jogo encontra-se em seu melhor estado. O tempo dedicado para tornar cada aspecto do jogo modular e autônomo faz com que criar e testar novos locais e quebra-cabeças, e ligá-los aos já existentes, seja um processo extremamente simples. Finalmente, o fato da história tomar uma forma mais definida durante o desenvolvimento das demais funcionalidades do jogo permite agora um progresso mais acelerado na criação das cenas que a compõem.

6.1.2 Negativas

Durante os testes do jogo, foi possível observar e apontar algumas falhas no mesmo. Primeiramente, alguns jogadores, especialmente de faixas etárias mais jovens, entediaram-se durante os momentos mais prolongados de exposição de história, ainda que estes não durassem mais do que alguns minutos. Este problema é especialmente alarmante, pois a primeira impressão de um jogador com um jogo para plataformas móveis pode ser o fator determinante do mesmo continuar ou não jogando o mesmo. Ainda com relação à história, jogadores apontaram que o modo de escrita não é muito interessante, e poderia ser melhor estruturado, de forma a criar maior suspense e engajamento dos jogadores com os personagens.
Muitos jogadores apontaram que o jogo parece se prolongar demasiadamente, devido tanto pela velocidade de movimento do personagem principal, quanto pela velocidade em que o texto é exposto (ainda que o jogador tocar a tela para avançar mais rapidamente o texto, esta ação é limitada por um tempo de $\frac{8}{10}$ segundos). Por fim, por mais que os jogadores achassem a maioria dos quebra-cabeças intuitivos, o quebra-cabeças "Tempo sem Relógio"deixou muitos jogadores confusos em seu funcionamento, apesar das instruções que o precedem.

6.2 Próximos passos

Para impulsionar o sucesso do desenvolvimento futuro deste projeto, alguns pontos-chave devem ser resolvidos. Primeiramente, o jogo requer muito mais conteúdo do que se encontra presente no estado deste projeto. Múltiplos ciclos de jogabilidade devem ser concatenados, aumentando o número de lugares exploráveis, adicionando novos quebra-cabeças, e expandindo a história do jogo.
Adicionalmente, devem ser feitas melhorias estéticas no jogo, como a criação de arte para novos locais, a adição de animações para todos os personagens, e, principalmente, a adição de sons e música no jogo.
Por fim, e não menos importante, a realização de testes com maior público e caráter mais objetivo deve ser realizada, de forma a direcionar o desenvolvimento do jogo, e melhorar o que já encontra-se presente no mesmo.

Referências Bibliográficas

Bartle(1996) Richard Bartle. Bartle taxonomy of player types. Citado na pág. 11

Freier(2018) Anne Freier. Less than 15% of game apps retain 35% of players after day 1. *Business of Apps*. Citado na pág. 10

Guido(2013) Zack Guido. *Of Course! The Greatest Collection Of Riddles & Brain Teasers For Expanding Your Mind*. Zack Guido, primeira edição. Citado na pág. 9, 27, 28

Oliveira(2019) Elida Oliveira. Cai aprendizado de matemática no último ano do ensino médio, aponta levantamento. *G1*. Citado na pág. 1

Perelman(2013) Yakov Perelman. *101 Puzzles*. Prodinnova, primeira edição. Citado na pág. 9

Tokarnia(2019) Mariana Tokarnia. Maioria dos alunos gosta de estudar português e matemática. *Agência Brasileira*. Citado na pág. 1