CENTRO UNIVERSITÁRIO UNIVATES
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE ENGENHARIA DE SOFTWARE

CARLOS COLOMBO

DESENVOLVIMENTO DE UM JOGO MULTIPLATAFORMA UTILIZANDO CROSS PLATFORM TOOLKIT HAXE

Lajeado, julho de 2017

CARLOS COLOMBO

DESENVOLVIMENTO DE UM JOGO PARA DISPOSITIVOS MÓVEIS UTILIZANDO CROSS PLATFORM TOOLKIT HAXE

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Tecnológicas do Centro Universitário UNIVATES, como parte dos requisitos para a obtenção do título de bacharel em Engenharia de Software.
Área de concentração: Desenvolvimento de Jogos

ORIENTADOR: Alexandre Stürmer Wolf

Lajeado, junho de 2017

RESUMO

Este trabalho apresenta o desenvolvimento de um jogo multiplataforma utilizando a biblioteca HaxeFlixel. O projeto foi especificado utilizando o Método Ágil Scrum. O jogo é constituído de duas modalidades com características diferentes de interação do jogador, onde a primeira modalidade possui características de RPG (*Role Playing Game*), onde tem-se a possibilidade de explorar cenários gerados aleatoriamente, já a segunda modalidade, consiste em enviar os heróis em uma série de missões de acordo com suas habilidades. Além disso, esse trabalho tem como objetivo compreender a complexidade do desenvolvimento de jogos, e as características e benefícios da linguagem Haxe para este ambiente.

Palavras-chave: Games, Jogos, Android, Dispositivos móveis, Haxe, Engenharia de Software.

ABSTRACT

This work presents the development of a multiplataform game using the HaxeFlixel library. The game project was specified using the Agile Method Scrum. The game is going to feature two modalities with different forms of interaction by the player, where the first presents RPG (Role Playing Game) characteristics where is possible to explore random generated levels. The second modality consists in sending the heroes in a series of missions according to their abilities. Furthermore, this work aims to comprehend the complexity of game development and the characteristics and benefits of the Haxe language in this environment.

Keywords: Games, Multiplataform, Android, Mobile devices, Haxe, Software Engineering.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

API – Application program interface – Interface de Programação de Aplicações

AS3 – ActionScript 3

ASCII – American Standard Code for Information Interchange – Código Padrão Americano para Intercâmbio de Informações

CETEC – Centro de Ciências Exatas e Tecnológicas

CRPG – Computerized Role-Playing Game – Jogo de Intepretação de Papel Computadorizado

GDD – Game Design Document – Documento de Definição de Jogo

GM – Game Master – Mestre do Jogo

IA – Inteligência Artificial

OpenFL – Open Flash Library – Biblioteca Aberta do Flash

POO – Programação Orientada a Objeto

REST – Representational State Transfer

RPG – Role-Playing Game – Jogo de Interpretação de Papel

RTS – Real-time Strategy – Estratégia em tempo real

SUMÁRIO

1 INTRODUÇÃO

O mercado de jogos eletrônicos demonstrou em 2015 uma receita total de 23,5 bilhões de dólares, o que representa um crescimento de 5% em relação ao ano de 2014. Da mesma forma, a venda de software na indústria de jogos eletrônicos demonstra um crescimento de 7% com relação ao faturamento de 16,5 bilhões de dólares, apresentado no ano anterior (MORRIS, 2016). Além disso, estatísticas preveem um crescimento de 30% na receita da indústria entre os anos de 2014 e 2019 (TAKAHASHI, 2015).

Além do crescimento geral da indústria, também ocorre a deflagração dos jogos eletrônicos dentro de suas plataformas. Como exemplo, é esperado que a receita gerada por jogos eletrônicos para dispositivos móveis ultrapasse, pela primeira vez, a receita gerada pelos jogos para computadores pessoais e consoles. Este crescimento pode ser associado a evolução dos processadores e recursos gráficos destes dispositivos (KHARPAL, 2016).

Devido à oportunidade de oferecer jogos eletrônicos em diversas plataformas, torna-se interessante o desenvolvimento de aplicações que sejam compatíveis com um maior número destas, permitindo ampliar o publico alvo dos aplicativos. Contudo, a manutenção de um software suportado em múltiplas plataformas pode tornar-se oneroso, e, em alguns casos, exige a subsistência de mais de uma base de código e de mão de obra especializada para cada linguagem ou *game engine* (seção 2.5) utilizada.

Portanto, surge a necessidade de encontrar linguagens, bibliotecas ou *game engines* que sejam projetadas para desenvolver jogos para múltiplas plataformas. No contexto deste projeto, será utilizada a biblioteca HaxeFlixel escrita na linguagem Haxe. O Haxe, por sua vez, é uma linguagem de alto nível, com código aberto e que compila aplicações para plataformas como Windows, MacOS, Android, Flash e HTML 5.

Um dos jogos desenvolvidos em Haxe, que acabou por surpreender a indústria de jogos chama-se Papers, Please, que tem como objetivo colocar o jogador na perspectiva de

um inspetor de imigração e decidir a quem deve ser permitida entrada em um país fictício. Além disso, o HaxeFlixel está sendo utilizado em outros projetos como The Enchanted Cave 2, Defender's Quest II, Cardinal Quest 2, Werewolf Tycoon, entre outros jogos multiplataforma.

1.1 Metodologia

O desenvolvimento do presente projeto é documentado e executado seguindo o Método Ágil Scrum, adaptado para projetos pessoais. Além disso, foi realizada a pesquisa bibliográfica das melhores práticas de *design* de jogos, com intuito de desenvolver um jogo que seja fonte de entretenimento, e também ferramenta de pesquisa. Por fim, serão tomadas as medidas necessárias para que a base de código possa ser utilizadas nas plataformas Windows e Android.

1.2 Objetivo geral

Esse trabalho tem como objetivo compreender a complexidade do desenvolvimento de jogos multiplataforma, e as características e benefícios da linguagem Haxe para este ambiente.

1.3 Objetivos específicos

- Estudar a linguagem Haxe e sua utilização no desenvolvimento de jogos multiplataforma;
- Desenvolver um módulo de jogo de exploração de cenários gerados aleatoriamente;
- Desenvolver um módulo de jogo de realização não interativa de missões.

1.4 Organização do trabalho

Este trabalho apresenta no Capítulo 2, a revisão bibliográfica relacionada a jogabilidade, gêneros de jogos. Também apresenta jogos que foram utilizados como referência para o conceito do jogo e um relato sobre *game engines.*

O Capítulo 3, contém alguns aspectos básicos de classificação de linguagens de programação, como o Haxe é enquadrado nesse contexto, e também sintetiza as bibliotecas e ferramentas que serão utilizadas no desenvolvimento.

O Capítulo 4, apresenta o os requisitos elencados para o projeto, assim como a documentação do processo de desenvolvimento do jogo.

2 *DESIGN* DE JOGOS

Neste capítulo serão abordadas algumas das principais características e elementos importantes no desenvolvimento de jogos eletrônicos. Primeiramente, na seção 2.1, serão abordados quais os elementos que definem a jogabilidade de um jogo e, na seção 2.2, quais os conceitos que ele utiliza para entreter o jogador. Em Seguida, na seção 2.3, será vista uma relação dos principais gêneros de jogos, e por fim, na seção 2.5, a definição de uma *game engine*.

A definição de jogo é ampla, devido à variedade de atividades que são enquadradas nesta categoria. Brincar, por si só, é uma forma de entretenimento, assim como jogar, e pode possuir uma definição ainda mais ampla do que a definição de jogo. Ambos conceitos podem ser considerados como derivações da necessidade humana de fingir, interpretar ou fazer de conta (do inglês *pretend*), contudo, um jogo diferencia-se de uma brincadeira pela presença de regras e objetivos (ADAMS, 2013). Por fim, é importante ressaltar que a definição de jogo, assim como as definições apresentadas nas próximas seções abordando jogabilidade, entretenimento e os gêneros de jogos, varia entre autores e *designers*, porém, neste trabalho, foram utilizadas como base as definições contidas na bibliografia de Adams (2013), e complementada em alguns casos utilizando as considerações de outros autores.

2.1 Elementos de jogabilidade

Segundo Adams (2013), a definição de jogabilidade depende dos conceitos de desafios e ações. Neste contexto, desafio é qualquer conjunto de tarefas dada aos jogadores que não seja considerada como trivial. Além disso, os desafios são definidos a partir das regras do jogo, porém, nem sempre de forma precisa, exigindo que o jogador utilize pensamento lógico ou tentativa e erro. As regras são responsáveis por definir as ações que o jogador pode ou não

tomar em ordem a superar os desafios do jogo e atingir o objetivo final. No entanto, algumas ações podem ter apenas o objetivo de entreter o jogador. Por fim, jogabilidade pode ser definida como os desafios oferecidos ao jogador para atingir o objetivo do jogo, e as ações que o mesmo pode executar para superá-los.

Nas próximas seções serão abordados os principais elementos de jogabilidade definidos na bibliografia de Adams (2013).

2.1.1 Regras justas

Em jogos clássicos, não computadorizados, é comum a ocorrência de modificação das regras a partir dos jogadores que as consideram injustas. Por este motivo, pode-se considerar que estes jogos possuem apenas "meta regras", que servem como base instruir o jogador. No entanto, a importância de regras justas em jogos digitais deve-se ao fato de o jogador não possuir o poder de modificá-las, e por este motivo, o jogador que sentir-se injustiçado, irá abandoná-lo (ADAMS, 2013).

2.1.2 Simetria e Assimetria

Jogos simétricos são aqueles onde todos os jogadores envolvidos submetem-se as mesmas regras, possuem o mesmo estado inicial e a mesma condição de vitória. Como exemplo, pode-se mencionar o xadrez, que apesar do fato de um dos jogadores ser o primeiro a movimentar sua peça, ainda assim é considerado como simétrico (ADAMS, 2013).

Segundo Adams (2013), jogos assimétricos remontam situações onde diferentes jogadores seguem regras diferentes para atingir objetivos distintos, e em alguns casos possuem estados inicias díspares. Portanto, torna-se complexo mensurar se o jogo é justo para todos participantes.

2.1.3 Competição e cooperação

Uma característica comum entre os jogos e uma das principais motivações para pessoas engajarem-se nesta atividade, é a socialização. Neste contexto, pode-se considerar a necessidade das pessoas socializarem e cooperarem, assim como, a vontade de competir (ROUSE, 2005). Segundo Adams (2013), a competição ocorre entre os jogadores quando

existe um conflito entre os objetivos dos mesmos. Cooperação, caracteriza-se pelos jogadores trabalharem juntos para alcançarem o mesmo o objetivo.

2.2 Entretenimento

Jogos eletrônicos podem ser utilizados em treinamentos ou como ferramenta de ensino, porém seu principal objetivo e entreter o jogador. Portanto, os mesmos não podem conter apenas atividades abstratas e desconexas, e sim um conjunto de experiências que consiga atingir seu objetivo (ADAMS, 2013).

Nas próximas seções serão abordadas as mecânicas e ferramentas utilizadas para criar uma experiência consistente em um jogo.

2.2.1 Jogabilidade

Como já mencionado na seção 2.1, a jogabilidade é composta por desafios e ações, que acabam por entreter os jogadores. Pessoas costumam apreciar desafios e provar que podem resolvê-los, contanto que este seja superável e que possua uma recompensa a altura. As ações disponibilizadas pelo jogo, mesmo não relacionadas à resolução de desafios, também são fontes de entretenimento para os jogadores, em muitos casos, por tratarem-se de atividades que não são triviais na vida real, como pilotar aviões, construir cidades ou comandar exércitos. O Quadro 1 demonstra os tipos de desafios e algumas das ações associadas a cada um (ADAMS, 2013):

Quadro 1 – Tipos de desafios e suas respectivas ações

Tipo de desafio	**Ações**
Físico	Velocidade e tempo de reação Precisão (Pilotar, mirar, etc...) Temporização e ritmo Aprender combinação de movimentos
Lógico Formal	Dedução e codificação
Reconhecimento de padrões	Padrões estáticos Padrões de movimento e mudança
Pressão de tempo	Alcançar algo antes de terminar o tempo Alcançar algo antes de alguém
Memória e conhecimento	Trivialidades ou história Lembrança de objetos ou padrões

Exploração	Identificar padrões de espaço Achar chaves (desbloquear localidades) Achar passagens secretas Labirintos e espaços ilógicos
Conflito	Estratégia, tática e logística Sobrevivência Redução de forças inimigas Defender unidades vulneráveis Camuflagem
Econômico	Acumulação de recursos ou pontos Estabelecer sistemas eficientes de produção Alcançar balanço ou estabilidade em um sistema Cuidar de criaturas vivas
Razão conceitual	Achar pistas ou provas Detectar significados ocultos Entender ralações sociais Pensamento lateral
Criação e construção	Estética (Elegância ou beleza) Construção com objetivo funcional

Fonte: Do autor, adaptado de Adams (2013)

2.2.2 Estética

Os jogos são considerados como uma forma de arte, e como tanto necessitam de uma harmonia estética. Contudo, o jogo não precisa ser visualmente apelativo, e sim possuir uma identidade visual clara de forma a não confundir o jogador. Enfim, é importante ressaltar que a estética vai além dos cenários e personagens, englobando também a interface, a história, os sons e a forma como a estética do jogo reage as ações do jogador (ADAMS, 2013).

2.2.3 Harmonia

A harmonia nos jogos é também considerado um dos aspectos importantes para seu apelo estético. No entanto, ele tem como objetivo dar ao jogador a impressão de estar em um mundo paralelo e coerente, onde as ações executadas possuam consequências (ADAMS, 2013). Segundo Rouse (2005), é importante que os jogadores consigam prever o resultado de suas ações para criar um padrão de pensamento, caso contrário o jogo torna-se incoerente e desarmônico.

2.2.4 História

Independente da mídia utilizada as histórias servem, em grande parte dos casos, ao propósito de permitirem as pessoas fantasiarem e fugirem da vida cotidiana. Elas tem o poder de transferir a imaginação das pessoas para mundos alternativos, tanto realistas como fantasiosos e glamorosos. Além disso, quando histórias são contadas dentro de jogos eletrônicos se torna possível realizar as ações dos personagens e alterar o curso da mesma (ROUSE, 2005).

Outro grande atrativo da adição de fantasia aos jogos é a realização de atividades não permitidas socialmente, como atividades criminais, ou então atividades fisicamente impossíveis como voar (ROUSE, 2005). Por fim, a indústria de jogos utiliza histórias devido ao nível de entretenimento agregado, o aumento do público-alvo alcançado, por manter os jogadores interessados por mais tempo e pelo aumento nas vendas de jogos que as incorporam (ADAMS, 2013).

2.2.5 Riscos e recompensas

A ideia de riscos e recompensas em jogos como fonte entretenimento tem como exemplo jogos de apostas e também qualquer jogo competitivo, onde corre-se o risco de perder e, caso contrário, a recompensa da vitória. Nesse contexto, a sensação de risco em jogos é gerada através da incerteza projetada no jogador, seja através da aleatoriedade, dificuldade ou da oclusão de fatos. Contudo, é importante que as recompensas oferecidas sempre estejam a altura dos desafios de forma a incentivar o jogador (ADAMS, 2013).

2.2.6 Novidades

Um jogo torna-se mais interessante quando cria um padrão de revelar novos elementos que possam ser feitos, ouvidos ou vistos pelos jogadores durante seu decorrer. Em muitos casos, pode-se adicionar jogos secundários como novidades para os jogadores com o objetivo de manter a experiência menos monótona (ADAMS, 2013). Outro fator importante é a invenção de novas formas de desafiar os jogadores, sendo que o padrão da indústria é seguir as fórmulas conhecidas e que garantem o sucesso do jogo (ROUSE, 2005).

2.2.7 Aprendizado

O processo de aprendizado que acontece em jogos não tem relação com processos lúdicos, mas sim com a descoberta das regras do jogo e da resolução dos desafios envolvidos (ADAMS, 2013). Segundo Koster (2005), seres humanos são máquinas de reconhecimento de padrões, e também afirma que o processo de descobrimento e repetição de um padrão os traz satisfação e um senso de realização. Contudo, quando detectam apenas ruído e falham em detectar um padrão, sentem-se frustrados, e quando o padrão é repetido demasiadamente, sentem-se subestimados e até entediados.

2.2.8 Criação e expressão

Jogos também podem ser objetos de expressão e criatividade, devido ao fato de as decisões tomadas pelo jogador. Portanto, em alguns casos, é possível permitir a expressão da personalidade, seja pela mudança da aparência, voz, nome, escolhas ou atributos que definem uma personagem. Em outros, é possível instigar a criatividade dos jogadores através da construção ou da definição da cosmética de objetos do jogo. Por fim, a possibilidade de criar e se expressar ajuda ao jogador a se sentir parte do universo proposto pelo jogo (ADAMS, 2013).

2.2.9 Imersão

Jogos, assim como filmes, livros, músicas e outras formas de entretenimento tem como objetivo levar as pessoas a perder momentaneamente a noção do mudo real. Os jogadores podem ter a experiência de estarem imersos ao concentrar-se em um desafio, ao motivar-se a vencer uma partida, ou envolverem-se na história contada (ADAMS, 2013). Contudo, é importante que não aconteçam interrupções na experiência do jogador, como falhas no softwares, inconsistência na harmonia do ambiente ou violação das regras, pois, causam a quebra da imersão fazendo com que percebam que estão diante de um jogo (ROUSE, 2005).

2.2.10 Socialização

Segundo Rouse (2005), o principal motivo para a maioria das pessoas envolverem-se em jogos, é a socialização com amigos e familiares. Como ainda, a origem desta atividade

está atrelada a atividades sociais. Por isso, hoje existem jogos online (seção 2.2.2) onde os usuários interagem, dentro de um "mundo consistente", com milhares de outros jogadores com o objetivo de cooperar, competir e, principalmente, socializar. Segundo Adams (2013), ao tratar-se de jogos digitais, existe a vantagem de interagir com inteligências humanas, ou seja, não artificiais, pois aumentam o nível de dificuldade e diminuem a complexidade do desenvolvimento.

2.3 Gêneros de jogos

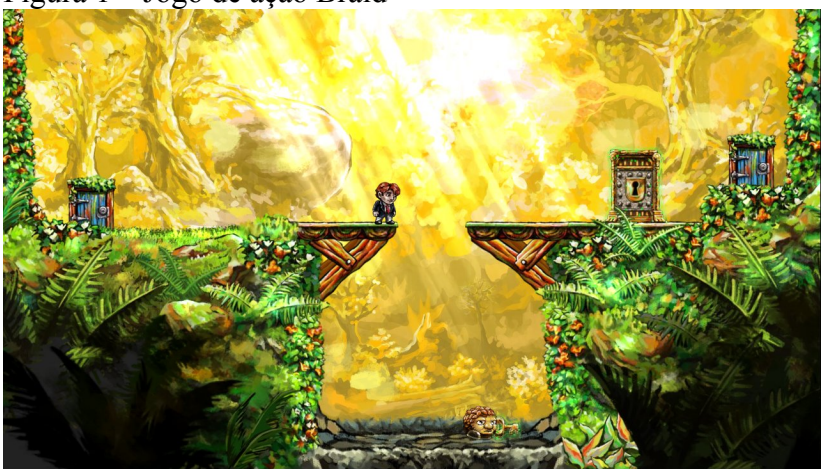
Em contraste com livros ou filmes, onde o seu gênero é definido pela trama da obra (ação, ficção científica, drama, romance, comédia) ou pelo ambiente onde ocorre (faroeste, espacias, épicos, medievais, futuristas), gêneros de jogos são definidos pelos desafios que são propostos aos jogadores através de sua jogabilidade (seção 2.2.1). Portanto, como exemplo, independente do ambiente ou tema, um jogo cujo objetivo é explorar e resolver desafios enquadra-se no gênero aventura (ADAMS, 2013).

Nas próximas seções serão apresentados os principais gêneros de jogos, segundo a bibliografia de Adams (2013). Para elucidar os gêneros, foram escolhidos alguns jogos que, segundo o autor, podem ser citados como exemplos em suas determinadas categorias.

2.3.1 Ação

Segundo Adams (2013), jogos de ação são considerados aqueles onde, os desafios encontrados testam as habilidades físicas e a coordenação dos jogadores. Outras características desse gênero incluem desafios táticos e de exploração e resolução de puzzles (enigmas ou quebra-cabeças). Os jogos de ação possuem muitos subgêneros como, por exemplo, jogos de plataforma como Braid (Figura 1), jogos de tiro (Doom, Space Ivaders, Call of Duty) e jogos de luta (Street Fighter, Soulcalibur).

Figura 1 – Jogo de ação Braid



Fonte: Do autor.

2.3.2 Estratégia

Jogos de estratégia são inspirados nos jogos mais antigos da humanidade, como, por exemplo, o xadrez. Em sua grande maioria envolvem temas de guerra, conflitos e conquista, sendo o principal desafio obter a vitória sobre um ou mais oponentes através de uma série otimizada de ações disponíveis dentro do jogo. Jogos como Age of Mythology, apresentado na Figura 2, são classificados como estratégia em tempo real (ou *Real-time strategy* – RTS) e estão entre os mais comuns (ADAMS, 2013).

Figura 2 – O RTS Age of Mythology

Fonte: Do autor.

2.3.3 RPG

Jogos de *Role Playing Game (*RPG), originalmente não computadorizados e conhecidos como RPG de mesa, consistem em imergir o jogador em um mundo imaginário e deixá-lo interpretar o papel de um personagem (ou avatar) também imaginário. Nos sistemas clássicos de RPG de mesa, na qual o exemplo mais popular é o Dungeons & Dragons, um grupo de pessoas criam seu personagem e formam uma companhia (ou *party*) de heróis que são motivados por objetivos determinados pelo GM (*Game Master*), e guiados pelas regras do sistema (ADAMS, 2013).

As versões computatorizadas de RPG, denominadas *Computerized Role-Playing Game* (CRPG), contrastam com outros jogos por permitir ao jogador evoluir um personagem simples e transformá-lo em um herói com superpoderes, enquanto em outros gêneros o herói já possui suas habilidades já determinadas. Além disso, a evolução dos personagens precisa ser alcançada através de sucesso nos objetivos do jogo, e permite ao jogador escolher os poderes e o estilo do seu personagem. Contudo, os jogos de CRPG, como Star Wars Knights of the Old Repuplic II: Sith Lords apresentado na Figura 3, compartilham muitas características de outros gêneros como Ação, Estratégia e Aventura (ADAMS, 2013).

Figura 3 – O CRPG Star Wars Knights of the Old Republic: Sith Lords

Fonte: Do autor.

2.3.4 Jogos de esporte

Jogos de esporte são caracterizados por simularem esportes reais ou imaginários, que podem ser focados tanto esporte em si ou no gerenciamento de times ou carreiras. No entanto, os jogos de esportes costumam ser realísticos devido ao nível de conhecimento dos jogadores em relação aos esportes simulados. Como exemplo, pode ser citado o jogo Grand Turismo Sport, demonstrado na Figura 4, que permite ao jogador montar carros de corrida e pilotá-los contra outros jogadores (ADAMS, 2013).

Figura 4 – Jogo de esporte Grand Turismo Sport

Fonte: Do autor.

2.3.5 Simulação de veículos

Jogos de simulação permitem aos jogadores a experiência de voar ou dirigir veículos que podem ser reais ou fictícios. Além disso, os jogos de simulação se caracterizam por desafiarem o jogador fisicamente, ou seja, exigem que ele domine a habilidade de controlar o veículo sendo simulado. Como exemplo, na Figura 5, o jogo X Wing Alliance desafia o jogador a pilotar uma nave espacial durante uma batalha, o que inclui destruir inimigos e evitar ser atingindo (ADAMS, 2013).

Figura 5 – Jogo de simulação de veículo Star Wars Alliance

Fonte: Do autor.

2.3.6 Construção e gerenciamento

Jogos de construção e gerenciamento tem como foco a criação ou o crescimento de empreendimentos ou civilizações. Traz como principal desafio ao jogador limites econômicos, eventos randômicos e evita trazer desafios físicos e de coordenação. Um exemplo clássico de jogo de construção e gerenciamento é o Roller Coaster Tycoon, demonstrado na Figura 6, onde o jogador deve construir e gerenciar um parque de diversões (ADAMS, 2013).

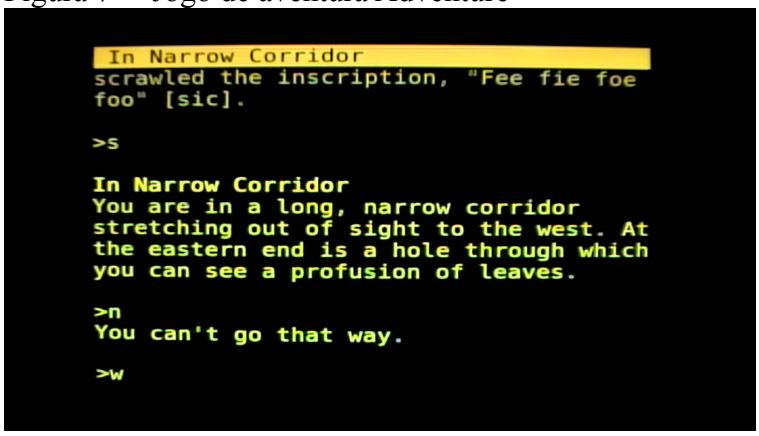
Figura 6 – Jogo de construção e gerenciamento Roller

Fonte: Do autor.

2.3.7 Aventura

Jogos de aventura são diferentes da maioria dos demais gêneros, e também considerados os de maior popularidade. Não oferecem um desafio específico ao jogador, apesar de poderem conter elementos variados de outros gêneros. Resumidamente, podem ser considerados histórias interativas, que apesar do nome, nem sempre oferecem uma aventura ao jogador. Este fato pode não ficar claro devido nome do gênero cuja origem, o jogo Adventure apresentado na Figura 7, serviu como inspiração para seus predecessores (ADAMS, 2013).

Figura 7 – Jogo de aventura Adventure



Fonte: Do autor.

2.3.8 Vida artificial e jogos de *puzzle*

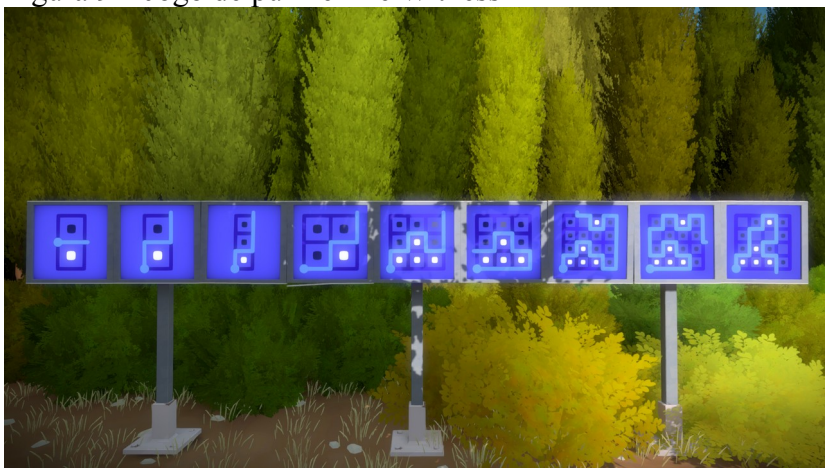
Vida artificial, além de ser um gênero de jogo, é uma área de estudo que tem como objetivo simular processos biológicos utilizando mecanismos biológicos bem conhecidos (como mutações e seleção natural, por exemplo). Porém, como em grande parte dos jogos, apenas uma aproximação destes processos é atingido dentro dos jogos, que limitam-se a simular pequenas populações de organismos, quando não apenas um organismo. Por fim, em alguns casos como no jogo The Sims apresentado na Figura 8, são simuladas as necessidades e interações sócias de seres humanos dentro de uma comunidade (ADAMS, 2013).

Figura 8 – Jogo de vida artificial The Sim 3

Fonte: Do autor.

Os jogos de puzzle, ou jogos de quebra-cabeça, são aqueles onde a principal atividade gira em torno da resolução de desafios. Contudo, elementos de quebra-cabeça são encontrados em muitos jogos de ação e aventura onde, por exemplo, o jogador pode ser bloqueado por uma porta trancada e deve executar uma série de ações de forma a avançar no jogo. Os quebra-cabeças oferecidos aos jogadores podem exigir habilidades como reconhecimento de padrões, deduções lógicas, coordenação física com restrição de tempo ou a compreensão de processos (ADAMS, 2013). Como exemplo, na Figura 9, está apresentado o jogo de puzzle tridimensional The Witness, onde o jogador é desafiado por um conjunto complexo de labirintos e quebra-cabeças após acordar em uma ilha sem lembrar da sua própria identidade.

Figura 9 – Jogo de puzzle The Witness



Fonte: Do autor.

2.3.9 Jogos online

Jogos *online* podem ser considerados qualquer jogo que permite a interação entre dois ou mais jogadores, sem a necessidade dos participantes compartilharem a mesma tela. Não só os jogos online permitem a socialização dos jogadores, mas também oferecem a experiência de competir entre si, dentro de um mundo persistente, onde o estado do jogo e dos jogadores ficam salvos na nuvem. Além disso, em muitos casos, estes jogos dispensam a utilização de Inteligências Artificial (IA) utilizando inteligência humana dos próprios jogadores, o que acaba por ser o principal desafio e atrativo deste gênero (ADAMS, 2013). Um exemplo de jogo online que foi responsável pela criação do conceito de esportes digitais (ou *e-sports*), é o Dota 2 (Figura 10), onde dois times de cinco heróis com habilidades distintas, competem pela vitória que é conquistada ao destruir a base inimiga.

Figura 10 – Jogo online Dota 2 Reborn

Fonte: Do autor.

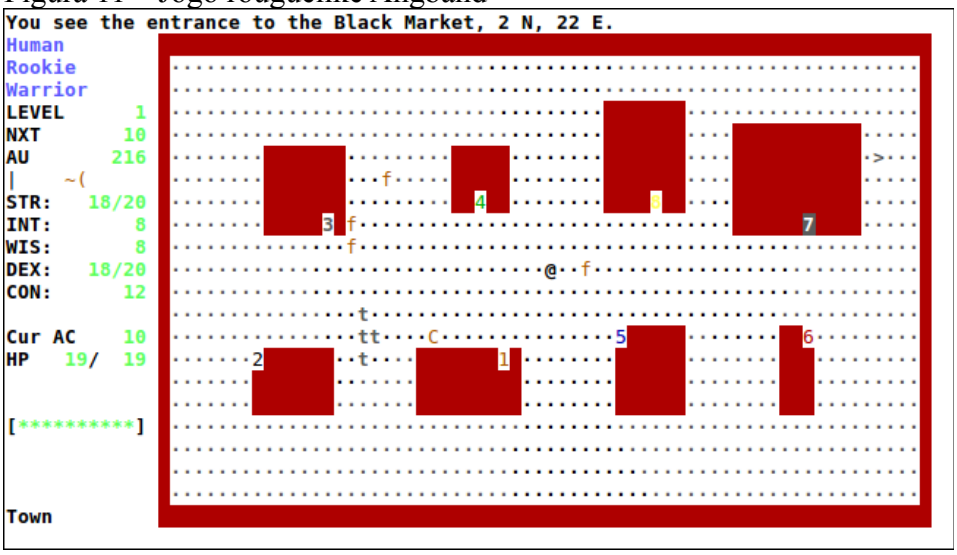
2.4 Jogos Similares

Nessa seção serão apresentados jogos que serviram de inspiração para o desenvolvimento do projeto. Angband é referência por conter exploração de cenários onde os mapas são gerados aleatoriamente, uma das características do jogo proposto neste trabalho. Knights of Pen & Paper tornou-se inspiração pelo seu estilo de batalhas, e por suportar as plataformas Windows e Android.

2.4.1 Angband

Angband é um jogo de exploração de cenários, classificado como *rouguelike*[1] por ser derivado do jogo Rogue, que começou a ser desenvolvido em 1990. Tem como tema as obras de J. R. R. Tolkien, e sua jogabilidade consiste em superar cem níveis gerados proceduralmente, até derrotar Morgoth no centésimo andar da masmorra. Como pode ser visto na Figura 11, o jogo utiliza caracteres *American Standard Code for Information Interchange* (ASCII), para representar os elementos do cenário, e sua utilização é baseado em comando de tecla como ‘l’ para olhar, ‘e’ para equipar itens entre outros.

Figura 11 – Jogo rouguelike Angband



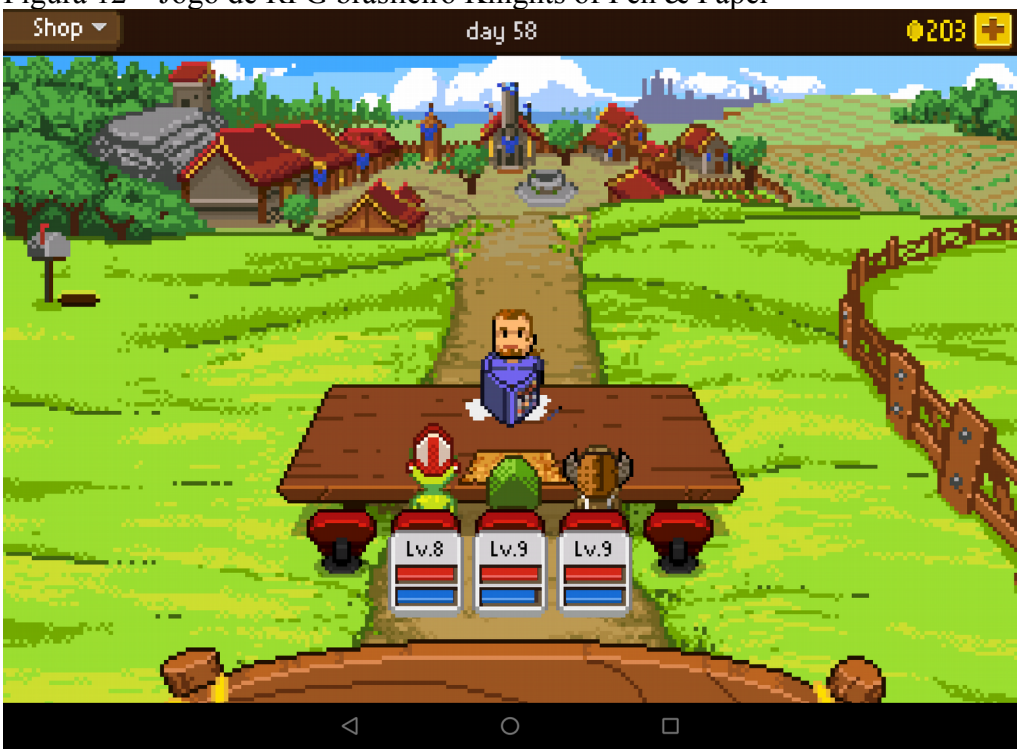
Fonte: Do autor.

2.4.2 Knights of Pen & Paper

A ideia do jogo consiste em simular um grupo de amigos que reúnem-se para jogar RPG de mesa (Seção 2.3.3), e evoluir seus personagens. Sua interface é planejada para ser utilizado em dispositivos móveis e computadores pessoais, e é uma das referências de jogos desenvolvidos no Brasil. A Figura 12 mostra os personagens reunidos em frente ao GM, onde existe a opção de realizar missões e engajar os personagens em batalhas.

[1] *Rouguelike* são CRPGs onde o jogador explora cenários gerados aleatoriamente, normalmente baseado em turnos e com morte permanente dos personagens. Grande parte de jogos *rouguelike* são baseados em temas de fantasia medieval.

Figura 12 – Jogo de RPG brasileiro Knights of Pen & Paper



Fonte: Do autor.

2.5 *Game engine*

A definição de uma *game engine* surgiu a partir da evolução e do crescimento dos desenvolvedores de jogos, que perceberam a distinção entre os vários componentes que compõem um jogo. Deste modo, foram elencadas as principais funcionalidades de um jogo, como a renderização dos gráficos, detecção de colisões, entradas do usuário e sons, dos outros recursos como imagens, modelos tridimensionais e arquivos de áudio. Da mesma forma, houve um esforço para isolar-se a lógica específica de cada tipo de jogo, das funcionalidades comuns entre cada um (GREGORY, 2009).

Gregory (2009), ressalta que existe uma linha tênue entre as funções da *game engine* e das funções de um jogo específico. Isso ocorre devido às funcionalidades que são particulares de um tipo característico de jogo, e como consequência, existe uma *game engine* adequada para cada situação ou gênero. Nesse sentido, o emprego de uma *game engine* não otimizada pode causar retrabalho e perda de desempenho.

Existem bibliotecas de jogo como a HaxeFlixel (seção 2.2.6) que não são consideradas *game engines*. Contudo, pode-se defini-las como qualquer biblioteca que execute uma função específica necessária para o desenvolvimento de jogos, com áudio, simulação física,

renderização ou interação com usuário. No entanto, como visto anteriormente, uma *game engine* é formada por bibliotecas que implementem exatamente estes tipos de funções. Embora a HaxeFlixel seja considerada uma biblioteca de jogos, ela agrega muitas das funcionalidades necessárias em uma *game engine*, e por isso, pode-se concluir que não existe uma diferença entre os dois conceitos.

Neste capítulo forram abordadas algumas das principais características e elementos importantes no desenvolvimento de jogos eletrônicos, como a jogabilidade, as formas de entrenimento, os gêneros e *game engines*.

O próximo capítulo aborda as ferramentas que serão utilizadas no desenvolvimento e especificação do jogo.

3 ESPECIFICAÇÃO E TECNOLOGIAS

Neste capítulo será apresentado a pesquisa bibliográfica sobre a metodologia de apoio e gerenciamento de requisitos na seção 3.1, e a classificação das linguagens de programação na seção 3.2. Em seguida, na seção 3.3, será demonstrada a classificação do Haxe, assim como algumas das bibliotecas de apoio utilizadas no processo de desenvolvimento.

3.1 Método Ágil Scrum

O Scrum tem como origem nos métodos ágeis, que segundo Sommerville (2011), surgiram como alternativa ao método tradicional, onde todo o projeto era planejado minuciosamente antes de sua execução. Os problemas com esse tipo de abordagem surgiram principalmente no desenvolvimento de projetos pequenos ou médios, onde o tempo de planejamento acabava excedendo em muito o tempo de execução. Segundo Brod (2015), outro argumento importante para a utilização de métodos ágeis, é a constatação das discrepâncias entre as necessidades levantadas no início do projeto e a satisfação dos usuários com o produto final.

O Scrum, assim como os métodos ágeis, diferenciam-se dos métodos tradicionais por seguir os princípios de envolvimento do cliente, entregas incrementais, foco nas pessoas e não nos processos, aceite de mudanças durante a implementação do projeto, e prezar pela simplicidade na usabilidade do produto (SOMMERVILLE, 2011). O Manifesto Ágil, resume alguns desses princípios e aconselha segui-los em todas as etapas do projeto. Ele é constituído de quatro frases, segundo Pressman (2011):

> Indivíduos e interações acima de processos e ferramentas;
> Software operacional acima de documentação completa;
> Colaboração dos clientes acima de negociação contratual;
> Resposta a mudanças acima de seguir um plano.

3.1.1 Papéis do Scrum

Dentro de uma equipe que utiliza Scrum existem três papéis distintos. O *Product Owner* é aquele que pagará o projeto ou então é o responsável pelo mesmo, e será quem aprova as entregas e solicita mudanças no produto. O *Scrummaster* é um membro da equipe que, idealmente, troca a cada *Sprint*, e é responsável por manter os artefatos do Scrum (*Product Backlog, Sprint Backlog, Burndown Charts*), comandar o *Daily Scrum* e, principalmente, remover qualquer problema que possa ser obstáculo para o trabalho dos demais componentes da equipe. O último dos papéis é o de membro da equipe, cujos componentes são responsáveis pela execução do projeto, e não deve existir hierarquia entre estes (BROD, 2015).

3.1.2 Artefatos e ritos do Scrum

O planejamento de um projeto com Scrum inicia-se com o *Product Backlog*, que consiste em uma listagem de requisitos do sistema expressados em forma de *User Stories*. Estas consistem em relatos feitos pelo *Product Owner* em relação a tarefas que necessitam que o sistema ou produto realize. Uma *User Story* que engloba muitos aspectos do projeto é chamado de *Epic*, e deve ser subdividido em outras *User Stories*. Após o levantamento das *User Stories*, as mesmas são pontuadas e colocadas em ordem de prioridade (BROD, 2015).

No Scrum, um *Sprint* é um período que deve ser fixado entre uma a quatro semanas, onde uma parte do projeto será planejado, desenvolvido e testado. Portanto, o *Sprint Backlog* é formado pelas *User Stories* retiradas do *Product Backlog* de acordo com sua prioridade de execução, o que é realizado durante uma reunião de planejamento do *Sprint*. Cada *User Story* é subdividida em tarefas que devem ser executadas até o final do *Sprint*. É exigido que não exista mudanças nos requisitos durante a execução do *Sprint*, com intuito de não aumentar o tempo de implementação, o que causaria atrasos na entrega (BROD, 2015).

Ao final de cada *Sprint* é realizada a entrega de uma parte funcional do projeto, que é acompanhada por uma reunião (*Sprint Review*), com a participação do *Product Owner*, onde é realizado o aceite da entrega, ou então são propostas mudanças ou novas funcionalidades que devem ser executadas no próximo *Sprint*. Outra reunião, também executada ao final do *Sprint*, é o *Sprint Restrospective*, onde apenas o *Scrummaster* e a equipe realizam uma retrospectiva discutindo as dificuldades encontradas, os aspectos positivos e as ideias que podem aumentar a satisfação do *Product Owner* (BROD, 2015).

Além das reuniões já discutidas, existe o *Daily Scrum*, que é uma reunião curta onde cada um deve falar brevemente sobre as tarefas que executou no dia anterior, sobre as tarefas que executará no dia e quais dificuldades está encontrando na realização de suas tarefas. Nesta reunião é tarefa do *Scrummaster* mitigar os obstáculos encontrados pelos membros da equipe (BROD, 2015).

Na próxima seção serão abordadas algumas características importantes para categorizar o Haxe e apontar aquelas que o distinguem de outras linguagens de programação.

3.2 Linguagens de programação

As linguagens de programação e as linguagens naturais possuem o mesmo intuito, de padronizar e facilitar a forma como nos expressamos. Contudo, as linguagens de programação distinguem-se das linguagens naturais por serem limitadas pelo seu contexto, podendo auxiliar apenas a expressão de ideias computacionais. E, se diferenciam por permitirem que pessoas comuniquem ideias para um computador. Portanto, as linguagens de programação necessitam, em alguns casos, de regras divergentes as das linguagens naturais (TUCKER, 2008).

Nas próximas seções serão apresentados características importantes para a definição das principais características da linguagem Haxe.

3.2.1 Tipos de variáveis

Variáveis são utilizadas pelos programadores para associar um nome a um endereço na memória. O valor que é permitido armazenar em uma variável, no entanto, depende de seu tipo. Cada linguagem de programação possui suas particularidades com relação aos seus tipos básicos e a rigidez com que trata os tipos de suas variáveis (TUCKER, 2008).

De acordo com Tucker (2008), as linguagens de programação possuem tipos básicos de variáveis para armazenar números naturais, para números reais com ponto flutuante, para caracteres e conjuntos de caracteres (chamadas *strings*), para valores *booleanos*, entre outros. Contudo, em algumas linguagens, consideradas fortemente tipadas, o tipo da variável é definido obrigatoriamente no código, e possíveis erros são detectados antes de o programa ser executado. Enquanto em outras, denominadas dinamicamente tipadas, tem-se a possibilidade de não definir um tipo para variável e manipular os dados com maior flexibilidade. No caso

das linguagens dinamicamente tipadas os erros de tipo só podem ser detectados durante a execução do programa.

3.2.2 Paradigmas de programação

Segundo Tucker (2008), a definição de um paradigma, e de um paradigma de programação, é a seguinte:

> De modo geral, pensamos em um "paradigma" como um padrão de pensamento que guia um conjunto de atividades relacionadas. Um paradigma de programação é um padrão de resolução de problemas que se relaciona a um determinado gênero de programas e linguagens.

Dentro das linguagens de programação, quatro paradigmas distintos surgiram para fundamentá-las. Nas próximas seções serão apresentados os paradigmas imperativo e orientado a objetos por estarem relacionados ao escopo deste projeto, porém, existem outros paradigmas, como por exemplo o funcional e lógico, podem ser aprofundados em Tucker (2008).

3.2.2.1 Programação imperativa

A programação imperativa é considerado o paradigma mais antigo, tendo surgido na década de 40. Ela tem como característica que o programa e suas variáveis são armazenados juntamente na memória, sendo formado por comandos que permitem executar cálculos aritméticos, atribuir valores a variáveis, receber entradas de dados, mostrar saídas de dados e mudar o fluxo do programa para qualquer ponto do código (TUCKER, 2008).

Ainda, de acordo com Tucker (2008), dois conceitos adicionais são utilizados na formação dos algoritmos, sendo elas a abstração procedural e o refinamento gradual. A primeira consiste em uma série de funções e operações básicas disponíveis ao programador, de forma que o mesmo não precise preocupar-se como os cálculos são executados, ou até esmo, como o programa deve fazer para imprimir resultados na tela. Por fim, o refinamento gradual é a divisão de um problema complexo em funções menores, utilizando-se da abstração procedural, e com objetivo mais específico.

3.2.2.2 Programação orientada a objeto

A programação orientada a objeto (POO), resume-se em um paradigma onde a codificação do programa é formado por um conjunto de objetos, que interagem através de mensagens e formam o estado do programa. O que difere a POO do paradigma imperativo é o envio de mensagem entre seus objetos, e, também, a utilização do encapsulamento, herança e polimorfismo (TUCKER, 2008).

Na POO os tipos de objetos são definidos dentro de uma classe. O encapsulamento é uma técnica que permite ao programador criar tipos, e baseia-se na ideia de abstração procedural mencionada na seção 3.2.2.1. Um tipo pode agrupar variáveis, funções e outros tipos dentro da classe, e, ainda consegue definir a visibilidade de cada elemento. Esta funcionalidade permite abstrair os detalhes de sua implementação e garantir a integridade dos dados e do funcionamento de um objeto (TUCKER, 2008).

A herança é uma das técnicas de reutilização de código da POO, ela permite a criação de subclasses de outra classe, nesse contexto chamado de superclasse. A subclasse, então, herda os elementos declarados como visíveis na superclasse. A POO, em algumas linguagens, também permite a herança múltipla, que consiste em uma subclasse poder herdar as funcionalidades de uma ou mais superclasses (TUCKER, 2008).

A POO permite a superclasse forçar a implementação de determinados métodos por parte de suas subclasses, o que é chamado de método abstrato. Ao criar-se um objeto considerado superclasse, é possível atribuir o valor desse objeto com uma instância de qualquer uma de suas subclasses, que por sua vez, possuem diferentes implementações dos métodos abstratos. Portanto, como o comportamento do método depende da subclasse que será atribuído ao objeto, e este recurso é chamado de polimorfismo (TUCKER, 2008).

3.3 Haxe

Haxe é uma linguagem de programação de alto nível e também um compilador. Além de ter código aberto, permite a compilação de programas para múltiplas linguagens e plataformas, o que possibilita, quando utilizadas as devidas práticas, a manutenção de apenas uma base de código. É fortemente tipada, contudo suporta tipagem dinâmica, e possui um sistema de averiguação de erros de tipo que só poderiam ser detectado em tempo de execução, quando rodados nas linguagens alvo (KRAJEWSKI, 2016).

A linguagem Haxe foi desenvolvida para suportar os paradigmas tanto estático como orientado a objetos. Entre as linguagens suportadas pelo seu compilador estão JavaScript, Neko, PHP, Python, C++, ActionScript 3, Java e C#. Mais informações sobre as particularidades do Haxe podem ser encontradas em seu manual, acessível em https://haxe.org/manual (KRAJEWSKI, 2016).

De forma a ampliar os recursos da linguagem e agilizar o desenvolvimento da aplicação, foram utilizadas ferramentas e bibliotecas que serão explicitadas nas próximas seções.

3.3.1 Flixel

O Flixel é uma biblioteca de código aberto para desenvolvimento de jogos, que foi desenvolvida em ActionScript 3 (AS3) por Adam Saltsman. O início do desenvolvimento ocorreu em 2008, após tentativas falhas de desenvolver jogos utilizando C++, Python e OpenGL. Segundo o desenvolvedor, a cada jogo desenvolvido com o AS3, maior era a quantidade de código reutilizado e mais produtivo era o desenvolvimento, e, portanto decidiu criar um projeto separado chamado Flixel (SALTSMAN, 2016).

3.3.2 Lime

O Lime é um *framework* de código aberto básico para desenvolvedores que utilizam Haxe. O Lime possui rotinas de criação de janelas, recebimento de entradas do usuário (teclado, mouse, *touchscreen*, controles, etc…), eventos, áudio, renderização, comunicação via rede e gerenciamento de recursos. Em sua versão 3.0.3 o Lime tem suporte para Windows, MacOS, Linux, Neko, Android, HTML5 e Flash (GITHUBb, 2016).

3.3.3 OpenFL

O Flash foi a principal ferramenta para criação de conteúdo interativo para páginas web, chegando a alcançar 92% dos usuários da Internet. No entanto, ele necessita a instalação de um *plugin* adicional ao navegador. *Open Flash Library (*OpenFL) é uma plataforma para criação de aplicações 2D, e tem como foco disponibilizar a API do Flash sem a necessidade de instalação do seu *plugin* (GRANICK, 2013). Por fim, o OpenFL foi desenvolvido

utilizando Haxe e portanto, tem código aberto e pode ser compilado para as plataformas Windows, Mac, Linux, iOS, Android, Firefox OS, HTML 5 e Flash (GITHUBa, 2016).

3.3.4 HaxeFlixel

O HaxeFlixel é uma implementação da biblioteca de jogos Flixel (seção 3.3.1), utilizando como base a biblioteca OpenFL (seção 3.3.3). Seu idealizador, e até hoje, principal desenvolvedor e líder de projeto é Alexander Hohlov, porém, conta com a colaboração de membros da comunidade. Apesar de ter como base o Flixel, o HaxeFlixel tornou-se um projeto independente e oferece suporte as plataformas BlacBerry, iOS, Android, WebOS, Linux, Mac, Windows, Neko, Flash e HTML 5 (GITHUBc).

3.4 MongoDB

MongoDB é um banco de dados orientado a documentos, desenvolvido com foco em escalabilidade. Oposto aos bancos de dados relacionais que armazenam dados em linhas de tabelas, o MongoDB armazena os dados em formato de documentos, que possibilitam representar estruturas complexas como vetores e objetos serializados. Além disso, os documentos são organizados em coleções, cujas estruturas são equivalentes as tabelas do modelo relacional, que não necessitam de um esquema fixo, o que diminui a complexidade na adição novos campos e permite que regras da estrutura dos documentos sejam definidas via código (CHODOROW, 2013).

3.5 Google Sign-In

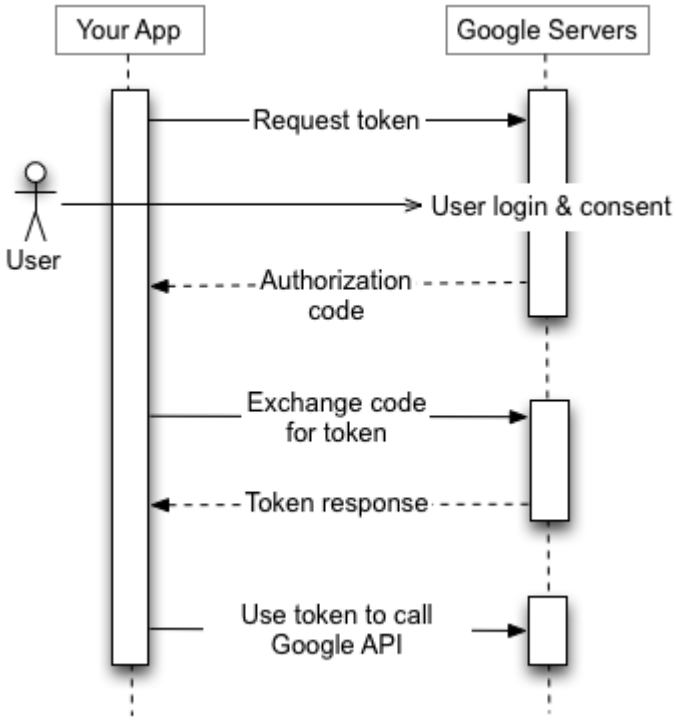
O Google Sign-In é um sistema fornecido pelo Google, que facilita a autenticação de usuários permitindo a utilização de suas contas ao acessar outras aplicações de forma segura. Além disso, após o usuário estar autenticado é possível acessar serviços do Google, como Google Play Games, que permite a socialização e classificação de jogadores e o armazenamento de dados (GOOGLE, 2017).

A segurança do Google Sign-In é baseado na troca de *tokens* de acesso do Oauth 2.0. O OAuth 2.0 é regido pela RFC 6749, e consiste em um *framework* de autorização que permite o acesso de terceiros a um *web service*. Um *token* de acesso é uma *string* que representa uma autorização fornecida a um cliente, o que inclui a validade do mesmo e seu

escopo de acesso. Ao contrário de outros métodos de autorização que necessitam o armazenamento de senhas e credenciais, o OAuth 2.0 utiliza as informações de autenticação do usuário para trocar *tokens* (RFC 6749, 2012).

Como pode ser observado na Figura 13, o fluxo de autorização inicia-se quando o aplicativo realiza uma requisição de um *token* aos servidores do Google. Nesse momento os usuários são encaminhados a uma tela de autenticação e autorização do acesso a informações da conta, ou APIs específicas do Google. Em caso de sucesso na autenticação e autorização por parte do usuário, a aplicação recebe como retorno um código de autorização. Em seguida, a partir do código de autorização, a aplicação obtém um *token* que pode ser utilizado como autenticação das requisições do usuário, ou para acesso às APIs do Google (GOOGLE, 2017).

Figura 13 – Fluxo de autorização do Google Sign-In



Fonte: Google (2017)

3.6 Flask-RESTful

Segundo Pautasso et al. (2014), a definição de *Representational State Transfer* (REST), é seguinte:

> *Representational State Transfer (REST)*is an architetural style that defines the architectural quality attributes of the World Wide Web, seen as an open, loossly coupled, massively dstributed and decentralized hypermedia system.

A Flask-RESTful, é uma biblioteca escrita em Python para o desenvolvimento de serviço REST. A Flask-RESTful é uma extensão do *framework web* Flask, também desenvolvido em Python.

3.7 Git

O Git, criado por Linus Torvalds para realizar o versionamento do código do Linux Kernel, é uma ferramenta de controle de versão distribuído. Seu desenvolvimento foi motivado pela falta de um software livre capaz de suportar um projeto de grande escala, e mantido por milhares de colaboradores trabalhando de partes variadas do mundo. Outras características do Git incluem a confiabilidade e rastreabilidade de todas as mudanças, o incentivo ao desenvolvimento paralelo e a imposição da responsabilidade da autoria de mudanças (LOELIGER, 2009).

3.8 HaxeDevelop

Em função do Haxe ser uma linguagem ainda pouco utilizada e um projeto relativamente novo, existe um número pequeno de opções de *Integrated Development Enviroment* (IDE) para utilização no desenvolvimento. O HaxeDevelop foi a IDE utilizada, por possuir a melhor integração com o Lime, que auxilia no processode compilação, e, também por agilizar o desenvolvimento com a funcionalidade de autocompletar código.

Esse capítulo abordou os fundamentos básicos do Scrum, e a estrutura e bibliotecas do Haxe. Além disso, foram explicitadas as demais ferramentas e bibliotecas utilizadas no desenvolvimento do projeto de jogo documentado neste trabalho.

O próximo capítulo trata sobre os requisitos (*Product Backlog*) que foram elencados para o projeto, assim como o processo de desenvolvimento e a jogabilidade do aplicativo.

4 PROJETO E DESENVOLVIMENTO DO JOGO

O desenvolvimento do jogo proposto neste trabalho, tem como foco a utilização das ferramentas disponibilizadas pelo HaxeFlixel e pela linguagem Haxe. As interfaces foram projetadas com o intuito de serem utilizadas tanto em computadores, como em dispositivos móveis. Além do aplicativo, realizou-se o desenvolvimento de um serviço RESTful para armazenagem de informações dos jogadores, e também garantir a consistência das interações no módulo de missões.

Nesse capítulo serão listados, em formato de *Product Backlog* os requisitos para o desenvolvimento jogo naseção 4.1. Na seção 4.2 será apresentado o relato do processo de desenvolvimento do aplicativo e as ferramentas utilizadas no mesmo. Por fim, na seção 4.3 será demonstrada a jogabilidade dos módulos desenvolvidos.

4.1 Product Backlog

Para o desenvolvimento do *backlog* foi realizado uma análise para identificar quais os principais requisitos de jogos enquadrados no gênero RPG. O Quadro 2 apresenta as *User Stories* e *Epics* elencados para o desenvolvimento do jogo.

Quadro 2 – Product Backlog

Assunto	**Tipo**
Gerador aleatório de *dungeon*	*Epic*
Geração aleatória de salas, corredores e portas	*User story*
Gerar armadilhas aleatoriamente	*User story*
Gerar recompensas aleatórias	*User story*
Adquirir traços	*User story*

Adquirir habilidades	*User story*
Pontos de habilidade/traço a cada nível	*User story*
Evolução de heróis	*User story*
Heróis e inimigos	*Epic*
Pontos de vitalidade	*User story*
Habilidades	*User story*
Traços	*User story*
Armadura e defesa	*User story*
Armas e ataque	*User story*
Personagens afetados por efeitos	*User story*
Morte de personagens	*User story*
Módulo de missões	*Epic*
Seleção de heróis para participação na missão	*User story*
Receber recompensas pela execução das missões	*User story*
Disponibilizar missões online	*User story*
Resolver missões apenas online	*User story*
Menu principal	*Epic*
Criar um novo jogo	*User story*
Carregar jogo	*User story*
Definir configurações do jogo	*User story*
Criar novo herói	*User story*
Menu de acesso	*User story*
Inventário para armazenamento de itens	*Epic*
Equipar itens de armadura e armas aos personagens	*User story*
Limitar a capacidade de armazenamento de itens	*User story*
Transferir, utilizar e remover itens	*User story*
Trocar itens por dinheiro	*User story*
Exploração de *dungeon*	*Epic*
Selecionar heróis para exploração da *dungeon*	*User story*
Desafio a partir de batalhas	*Epic*
Gerar encontros aleatórios	*User story*

Combate	*User story*
Utilizar habilidades em batalha	*User story*
Pontos de transição entre *dungeons*	*User story*
Contratação de heróis	*User story*
Batalha com inimigos	*User story*
Batalha com chefões	*User story*
Recompensas por vitória na batalha	*User story*

Fonte: Do autor.

4.2 Implementação do jogo

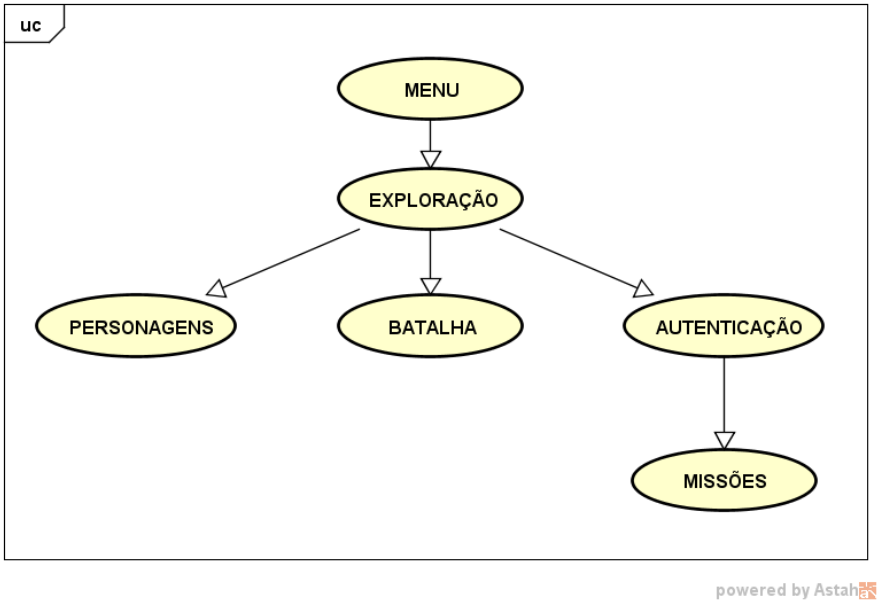
A implementação do jogo foi realizado utilizando a linguagem Haxe, auxiliado pela API das bibliotecas HaxeFlixel e OpenFL. Como já mencionado, o Haxe é utilizado com o objetivo de desenvolver uma aplicação multiplataforma (Windows e Android), com base em apenas uma base de código.

A principal área de iteração do jogador com o jogo é o módulo de exploração. Este módulo foi desenvolvido de forma a poder ser jogado sem necessidade de conexão com a Internet. No entanto, o módulo de missões exigirá a comunicação do aplicativo com um *web service* RESTful, que foi desenvolvido em Python utilizando a biblioteca Flask-RESTful. O acesso aos dados do *web service* são restritos, e dependem da autenticação de um usuário via Google Sign-In. Os dados necessários para as operações do módulo de missões são armazenados em um banco de dados orientado a documentos MongoDB.

4.3 Jogabilidade do aplicativo

Na Figura 14, é apresentado o modelo que representa o fluxo de jogabilidade da aplicação. Nas próximas seções serão abordadas os aspectos considerados de maior significância na jogabilidade, como o menu principal, o módulo de exploração, o menu disponível durante o jogo e o módulo de missões.

Figura 14 – Visão geral do jogo

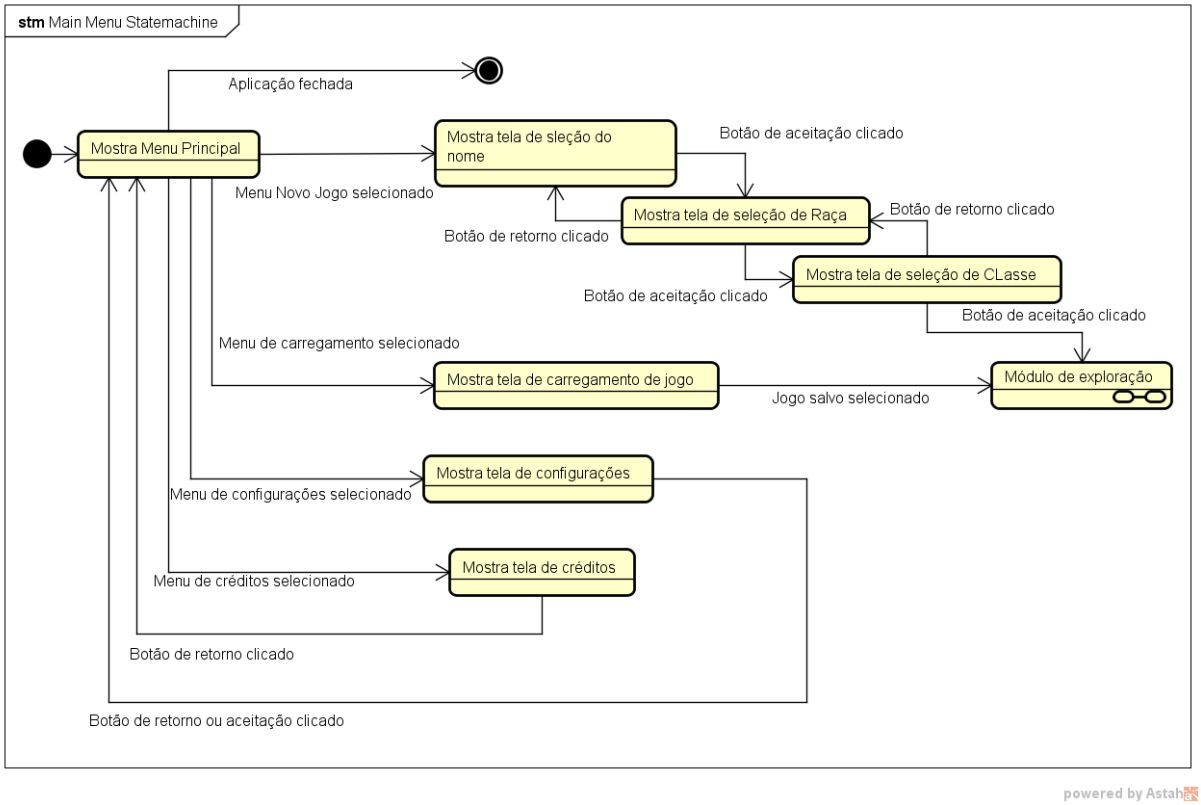


Fonte: Do autor.

4.3.1 Menu principal

O menu principal é a primeira tela a ser visualizada, e segue um modelo bem estabelecido, em outros jogos, de estrutura e opções aos jogadores. Entre estas, estão as opções de criação de um novo jogo (novo personagem), o carregamento de um jogo previamente salvo, configurações do jogo e, por fim, a tela de apresentações dos créditos do jogo. A Figura 15, apresenta a máquina de estados das possíveis interações do jogador com o aplicativo até o acesso ao módulo de exploração (seção 4.3.2), representado na figura pelo estado *Dungeon State*.

Figura 15 – Máquina de estados do menu principal



Fonte: Do autor.

A Figura 16 apresenta a interface exibida ao jogador. A ação de maior escopo deste menu é a criação de um novo jogo (*New Game*), e nela será necessária a escolha de um nome para o personagem, assim como a raça e a classe, consideradas características importantes para as mecânicas do jogo e também utilizadas no *role playing*.

O menu de carregamento do jogo (*Load Game*), possibilita ao jogador continuar com um jogo previamente criado, sendo que existe um limite de 4 jogos a serem armazenados, e neles constarão apenas dados referentes ao módulo de exploração. Também, neste menu, será possível a remoção dos jogos salvos e a seleção de um jogo vazio que remeterá o jogador a tela de criação de um novo jogo. O menu de configurações (*Settings*), possibilitará a configuração da dificuldade do jogo, e permitirá ao jogador desligar os efeitos sonoros.

Por fim, a tela de créditos (*Credits*) apresentará os nomes dos envolvidos no projeto e o nome de artistas responsáveis pela autoria de músicas e do *design* gráfico do jogo.

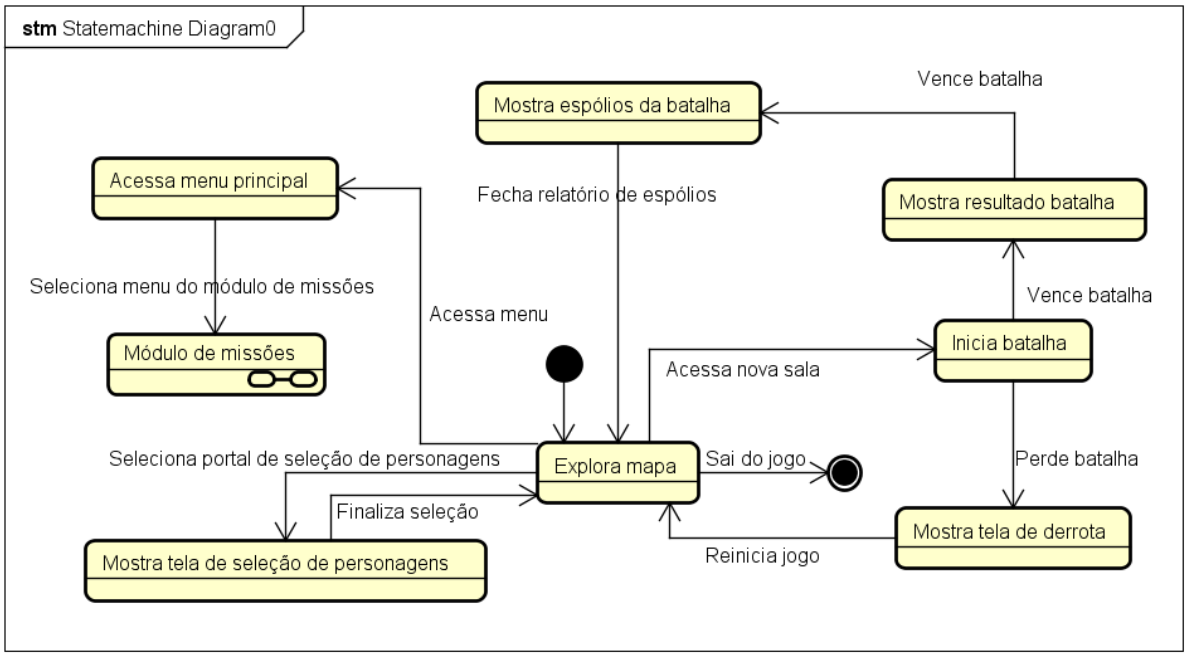
Figura 16 – Tela do menu principal

Fonte: Do autor.

4.3.2 Módulo de exploração

O módulo de exploração será o modo de jogo com maior nível de interação por parte dos jogadores. Nele será possível explorar mapas formados por salas, corredores, portas, armadilhas e baús gerados aleatoriamente. O jogador terá um limite de 4 personagens para comandar durante a exploração, e estes serão utilizados em batalhas geradas, também aleatoriamente, sempre que os personagens entrarem em uma das salas do mapa. Como recompensa pelas batalhas e ao achar baús, os personagens receberão espólios, como armas, armaduras, poções, alimentos e ouro. Além disso, ao vencer uma batalha os personagens envolvidos receberão pontos de experiência, que, podem ser utilizados para melhorar suas habilidades em batalha ou quando utilizados no módulo de missões (seção 4.3.4). A Figura 17 demonstra o fluxo da jogabilidade do módulo de exploração.

Figura 17 – Máquina de estados do módulo de exploração

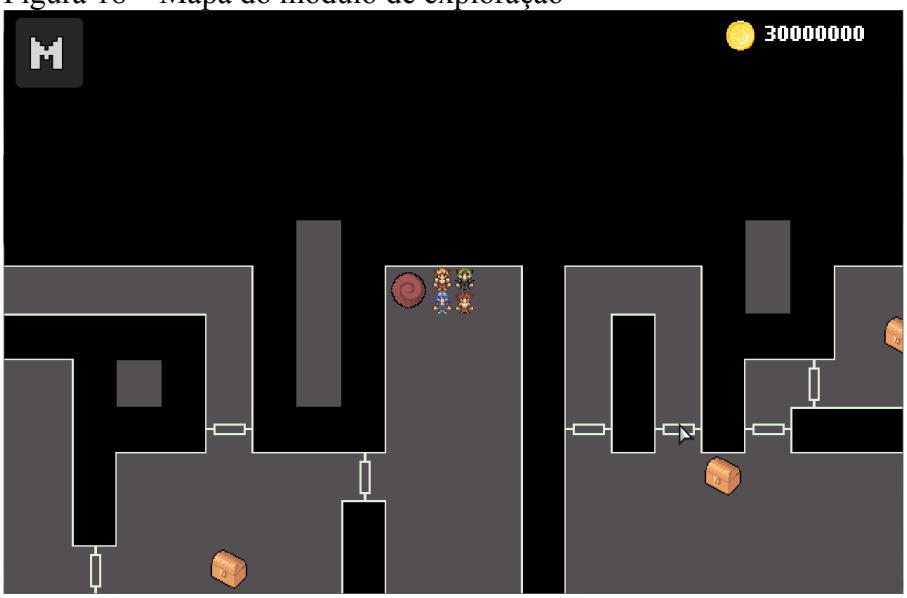


Fonte: Do autor.

Inicialmente, o jogador possuirá apenas seu próprio personagem para realizar a exploração do mapa. Os demais personagens serão adquiridos aleatoriamente pelo jogador durante a exploração ou durante a realização de missões, porém, sempre será respeitado o limite de 4 participantes na exploração. Quando o jogador possuir mais de 4 personagens, ele terá a possibilidade de selecioná-los (4 deles), de acordo com seus critérios no início da exploração de um mapa.

O mapa disponibilizado para exploração terá um tamanho limitado. No entanto, possuirá uma sala de entrada e outra de saída, onde encontrará o acesso para outro mapa aleatoriamente gerado, criando assim uma experiência continua para o jogador. Além do acesso a outro mapa, a sala de saída oferecerá um desafio extra aos jogadores por gerar uma batalha "mais desafiadora" (conhecida como *boss fight* em jogos do mesmo gênero). Durante a exploração o jogador poderá deparar-se com armadilhas, que causarão danos aos personagens, caso não sejam detectadas e desarmadas. Por fim, os cenários conterão baús de tesouro, onde poderão ser encontrados ouro e equipamentos que podem ser utilizados pelos personagens ou trocados por mais ouro. A Figura 18 apresenta a tela principal do menu de exploração.

Figura 18 – Mapa do módulo de exploração



Fonte: Do autor.

As batalhas, ou encontros, que são gerados durante a exploração do mapa, serão automaticamente parametrizados de acordo com o nível de experiência dos personagens, o tipo de mapa (cada mapa possuirá um tema), a quantidade de mapas explorados e o nível de dificuldade escolhido pelo jogador. Os inimigos encontrados em batalha pelos personagens terão características específicas, como a habilidade de envenenar, distrair ou derrubar oponentes e, da mesma forma, os personagens terão técnicas de contra-ataque. Para tanto, o jogador necessitará utilizar de estratégias específicas para cada inimigo, o que adiciona o elemento de aprendizado na jogabilidade.

O desempenho dos personagens durante as batalhas também dependerá de suas características. Os atributos básicos de cada personagem são o seu nível, força, agilidade e inteligência. O nível terá efeito na maioria das ações realizadas pelo personagem, e pode ser evoluída ao ganhar experiência. Cada nível obtido exige uma quantidade maior de experiência, o que é fornecido em maior quantidade de acordo com o número de cenários explorados, e adiciona pontos de treinamento para serem gastos pelo jogador nas habilidades e traços de cada personagem. Outra característica importante do personagem é seu bioma de origem (ex: Montanhas, Cidade, Deserto, Floresta), que será utilizado principalmente no módulo de missões.

Cada personagem pertencerá a uma classe, que definirá sua função dentro do grupo e qual o atributo mais importante para seu desempenho. Como exemplos, podem-se usar o *Warrior* (guerreiro), o *Rougue* (ladino) e o *Mage* (mago). O *Warrior* tem como função

principal ser a frente do combate e utilizar sua força física para vencer batalhas, e, portanto, o atributo força é o mais importante para essa classe. O *Rougue*, por sua vez, tem como função ser furtivo, desarmar e detectar armadilhas, atacar das sombras de forma ágil e precisa, e, neste caso seu principal atributo é a agilidade. Por fim, o *Mage* utiliza do conhecimento para defender seus aliados utilizando magias de ataque, defesa ou cura, e, consequentemente, a inteligência é o atributo que lhe é mais útil.

4.3.3 Menu do jogo

O menu do jogo pode ser acessado através do ícone localizado no canto superior esquerdo da tela principal do módulo de exploração, como pode ser visto na Figura 18. Nele, como demonstra a Figura 19, o jogador tem acesso ao menu de gerência de personagens (*Heroes*), ao módulo de missões (*Missions*), configurações do jogo (*Settings*) e o botão de retorno ao menu principal (*Main Menu*).

Figura 19 – Menu do jogo

Fonte: Do autor.

Ao acessar o menu de gerência de personagens, é apresentada uma página de navegação entre todos os personagens disponíveis para agilizar a seleção. A mesma tela pode ser acessadaa qualquer momento durante a gerência de personagens.

A Figura 20 apresenta a interface da tela de gerência de personagens. Como é possível observar, o jogador pode consultar informações básicas sobre o personagem, como seu nome, nível, pontos de experiência necessários para troca de nível, força, agilidade e inteligência.

Também é possível modificar os equipamentos do personagem a partir do inventário (*Inventory*), e associar os pontos de treinamento a alguma habilidade ou traço.

O menu de configurações possui opções similares as encontradas no menu principal, que possibilita a configuração de dificuldade e permite ao jogador desabilitar os efeitos sonoros.

Ao utilizar o botão de acesso ao módulo de missões, o jogador será direcionado para uma tela de autenticação. Nesta tela será solicitado a autenticação via Google Sign-In, que obtém o *token* de segurança, que é necessário para realizar as transações com o servidor REST. Caso o usuário já tenha realizado a autenticação previamente, o *token* é automaticamente renovado e o acesso ao módulo de missões liberado.

Figura 20 – Tela de gerenciamento de personagens

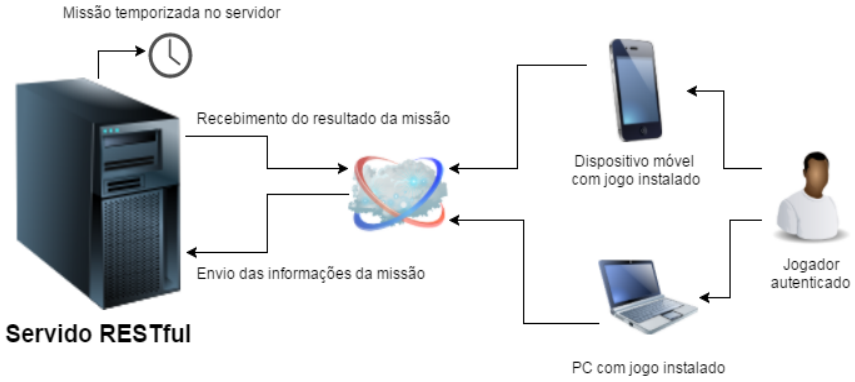
Fonte: Do autor.

4.3.4 Módulo de missões

O módulo de missões tem como objetivo exigir menos interação por parte do usuário, e utilizar os personagens não empregados no módulo de exploração. No entanto, trará certos desafios em relação a estratégia e os personagens utilizados em cada missão. Além disso, para prevenir que jogadores burlem as mecânicas do jogo modificando o relógio do dispositivo utilizado, o módulo de missões exige uma conexão com a Internet, e, uma autenticação via

Google Sign-In. Como já mencionado, um *web service* RESTful é responsável por fornecer e gerenciar as missões realizadas pelo jogador, e garantir a consistências das regras. A Figura 21 ilustra a troca de dados entre a aplicação e o web service RESTful.

Figura 21 – Comunicação do módulo de missões com o web service



Fonte: Do autor.

Cada missão fornecida para o jogador tem um nível como requisito, um custo em ouro para sua realização e um tempo de duração. Além disso, associado a cada missão, há o bioma da região onde a missão será realizada, e requisitos importantes para a realização da mesma. Os requisitos são atendidos pelos traços associados pelo jogador a cada personagem. Da mesma forma, personagens cujo o bioma de origem coincidem com o da missão terão melhores chances de sucesso. Tal qual no módulo de exploração, apenas quatro personagens são permitidos na participação de uma única missão.

O sucesso de uma missão dependerá da dificuldade selecionada pelo jogador, o nível dos personagens participantes em relação ao nível da missão, o número de personagens participantes e, se o bioma e os requisitos da missão são atendidos. Cada requisito não cumprido causa um ônus nas chances de sucesso da missão, assim como o inverso também ocorre, e, cada requisito cumprido causa um bônus nas chances de sucesso. O jogo indica as chances de sucesso da missão através de um percentual que adapta-se aos personagens selecionados. Ao final de cada missão, em caso de sucesso, o jogador receberá uma recompensa na forma de equipamentos e ouro.

A Figura 22 apresenta a tela principal do módulo de missões, onde o jogador pode navegar entre as missões disponíveis. Um maior número de missões será disponibilizado dependendo da quantidade e do nível médio dos personagens controlados pelo jogador. Como pode ser observado, as informações básicas de cada missão estão presentes nessa tela, e, para

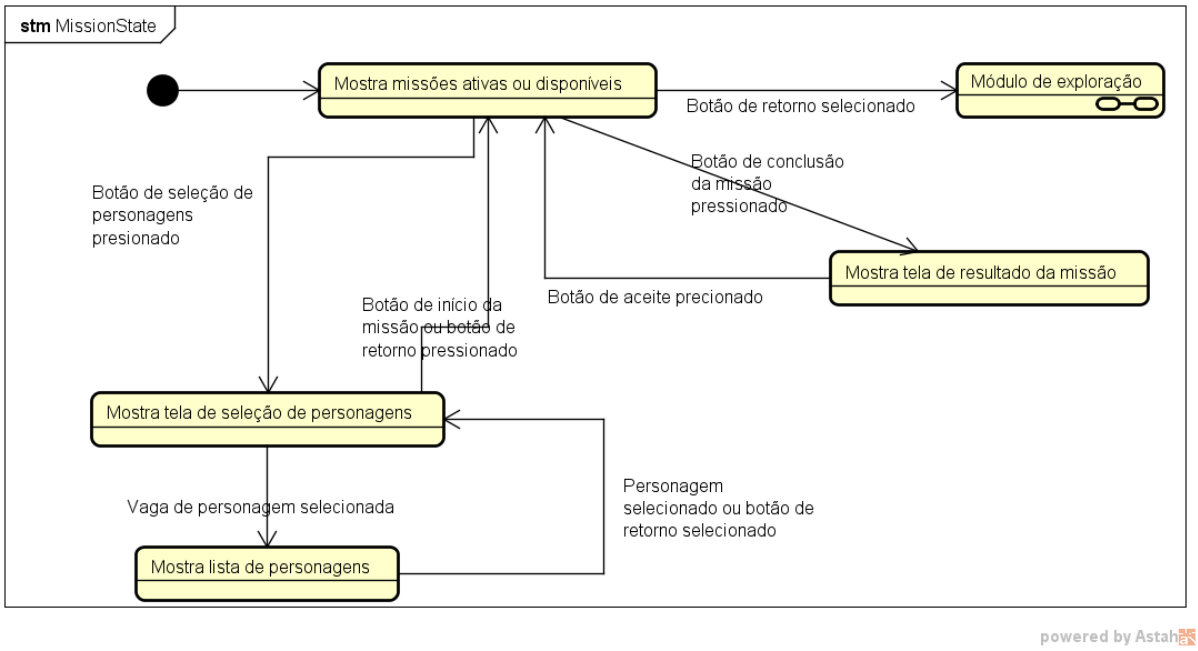
sua realização, o jogador deve pressionar o botão “Select heroes”, selecionar os personagens que julga mais adequado e iniciar a missão.

Figura 22 – Tela principal do módulo de missões

Fonte: Do autor.

Ao final do tempo de realização da missão, o jogador poderá verificar o sucesso da mesma. Em caso positivo o jogador será encaminhado a uma tela mostrando as recompensas obtidas e os pontos de experiência ganhos pelos personagens. Em caso de negativo apenas os pontos de experiências, reduzidos consideravelmente, serão apresentados. A Figura 23 explicita a máquina de estados do módulo de missões.

Figura 23 – Máquina de estados do módulo de missões



Esse capítulo apresentou os requisitos elencados para o desenvolvimento do jogo, como as ferramentas foram utilizadas no seu desenvolvimento e, por fim, os principais elementos da jogabilidade do aplicativo.

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de um jogo multiplataforma utilizando a linguagem Haxe, com auxílio da biblioteca HaxeFlixel. O Scrum foi utilizado como metodologia de apoio e gerenciamento de requisitos. O jogo desenvolvido possuí dois módulos com níveis diferentes de interação por parte do jogador: no módulo de exploração, baseado em um *rouguelike* RPG, é exigida a contante interação, já o módulo de missões exige que o jogador crie uma estratégia ao iniciar uma missão, com tempo de realização pré-determinado, e que ao final obtenha os resultados da missão, quando existentes.

Dentre os objetivos do trabalho estavam o estudo do Haxe na criação de jogos multiplataforma, o desenvolvimento de um módulo de exploração e o desenvolvimento de um módulo de missões. O Haxe e suas bibliotecas mostraram-se eficazes no desenvolvimento de jogos, possuindo funcionalidades que auxiliaram na implementação da interação com o usuário, o carregamento de imagens e áudios e a comunicação com o servidor RESTful. Por fim, o desenvolvimento de ambos os módulos de exploração e de missões foram concluídos conforme seus requisitos, com exceção da autenticação em aplicativos Android utilizando Google Sign-In.

Com relação ao design do jogo, foi mantido o foco em proporcionar ao jogador uma experiência de riscos e recompensas através das missões, batalhas e o recebimento de itens. Adicionalmente, existe um aprendizado crescimento para o jogador conforme compreende as mecânicas do jogo e percebe a evolução de seus personagens. Por fim, os principais desafios proporcionados aos jogadores são de natureza conflitiva, como uso de táticas e estratégias, sobrevivência e redução de forças inimigas.

Considerando os resultados obtidos durante o desenvolvimento do jogo, verificou-se que a utilização do Haxe para aplicações multiplataforma possui limitações. Na utilização de funções da API do Haxe, HaxeFlixel ou OpenFL foi possível manter a mesma base de código

para múltiplas plataformas, porém, quando houve a necessidade ao acesso de funções específicas (como a autenticação utilizando Google Sign-In) não foi possível a implementação utilizando o Haxe, e, exigindo o desenvolvimento e manutenção de funções para cada plataforma.

A utilização do Scrum como metodologia de apoio ao gerenciamento de requisitos, mostrou-se adequada na realização do projeto, considerando que o mesmo foi realizado por apenas um desenvolvedor. Porém, pode-se concluir que escopo do projeto mostrou-se desproporcional ao tempo e recursos disponíveis para o seu desenvolvimento.

5.1 Trabalhos futuros

Como a conclusão das tarefas elencadas no *project backlog* surgiram novos requisitos e ideias para melhoria do projeto. Além disso, o jogo ainda requer uma quantidade maior de conteúdo e variedade de itens, criaturas e missões. Por conseguinte, foram elencadas os seguintes quesitos como importantes trabalhos futuros neste projeto, são eles:

- Melhoras na estética e variedade de itens;
- Melhoras nos efeitos sonoros do jogo;
- Melhorar inteligência das batalhas;
- Criar um sistema de batalhas com maior complexidade;
- Implementar autenticação via Google Sign-In na plataforma Android;
- Melhorar a geração de mapas, de forma a tornar a experiência do jogador mais dinâmica.

REFERÊNCIAS

ADAMS, E. **Fundamentals of Game Design**. 3. ed. San Francisco: New Riders, 2013.

BROD, C. **Scrum: guia prático para projetos ágeis**. 2. ed. São Paulo: Novatec, 2015.

CHODOROW, K. **MongoDB: The Definitive Guide**. 2. ed. Sebastopol: O'Reilly Media, Inc, 2013.

GITHUBa. **GitHub - openfl/openfl: The "Open Flash Library" for fast 2D development**. Disponível em: <https://github.com/openfl/openfl> Acesso em: 18 de ago. 2016.

GITHUBb. **GitHub - openfl/lime: A foundational Haxe framework for cross-platform development**. Disponível em: <https://github.com/openfl/lime> Acesso em: 23 de ago. 2016.

GITHUBc. **GitHub - HaxeFlixel/flixel: Free, cross-platform 2D game engine powered by Haxe and OpenFL**. Disponível em: <https://github.com/HaxeFlixel/flixel> Acesso em: 06 de nov. 2016. 2016.

GOOGLE. **Google Identity Plataform | Google Developers**. Disponível em: <https://developers.google.com/identity/> Acesso em: 19 de mai. 2017. 2017.

GRANICK, J. **Introducing OpenFL**. Dísponível em <http://www.joshuagranick.com/2013/05/30/introducing-openfl/>. Acesso em 31 de out. 2016. 2013

GREGORY, J. **Game Engine Architecture**. 1. ed. Boca Raton: A K Peters, 2009.

KHARPAL, A. **Mobile game revenue to pass console, PC for first time**. CNBC, 22 de abr. 2016. Disponível em: <http://www.cnbc.com/2016/04/22/mobile-game-revenue-to-pass-console-pc-for-first-time.html>. Acesso em: 03 de out. 2016.

KOSTER, R. **A Theory of Fun for Game Design**. 1. ed. Scottsdale: Paraglyph Press, Inc., 2005.

KRAJEWSKI, S. et al. **Introduction – Haxe – The Cross-plataform Toolkit**. Disponível em: <https://haxe.org/manual/>. Acesso em : 31 de out. 2016.

LOELIGER, J. **Version Control with Git**. 1. ed. Sebastopol: O'Reilly Media, Inc, 2009. 310 p.

MORRIS, C. **Level Up! Video Game Industry Revenues Soar in 2015**. Disponível em: <http://fortune.com/2016/02/16/video-game-industry-revenues-2015/>. Fortune, 16 de fev. 2016. Acesso em: 3 de out. 2016.

PAUTASSO, C; ALARCON, R; WILDE, E. **REST: Advanced Research Topics and Pratical Applications**. 1. ed. New York: Springer, 2014.

PRESSMAN, R. S. **Engenharia de Software: Uma Abordagem Profissional**. 7. ed. Porto Alegre: AMGH Editora Ltda., 2011.

RFC 6749. **The OAuth 2.0 Authorization Framework**. Disponível em: <https://tools.ietf.org/html/rfc6749> Acesso em: 19 de mai. 2017. 2012.

ROUSE, R. **Game Desing: Theory and Pratice**. 2. ed. Texas: Wordware Publishing, 2005. 704 p.

SALTSMAN, A. **About Flixel**, 2016. Disponível em: <http://flixel.org/about.html>. Acesso em: 18 de ago. 2016.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

STANTON, R. **Knights of Pen & Paper review**. Disponível em: <http://www.eurogamer.net/articles/2013-01-25-knights-of-pen-and-paper-review>. Acesso em: 03 de nov. 2016.

TAKAHASHI, D. **U.S. games industry forecast to grow 30 percent to $19.6B by 2019**. VentureBeat, 2 de jun. 2015. Disponível em: <http://venturebeat.com/2015/06/02/u-s-games-industry-forecast-to-grow-30-to-19-6b-by-2019/>. Acesso em: 3 de out. 2015.

TUCKER, B. T.; NOONAN, R. E. **Linguagens de Programação: Princípios e Paradigmas**. 2 ed. São Paulo: Bookman, 2008.