FUNDAÇÃO PRESIDENTE ANTÔNIO CARLOS - FUPAC
Engenharia de Computação

Luiz Eduardo do Valle

DESENVOLVIMENTO INDEPENDENTE DO JOGO KAYA

Conselheiro Lafaiete, 2016.

Luiz Eduardo do Valle

DESENVOLVIMENTO INDEPENDENTE DO JOGO KAYA

Relatório técnico apresentado como requisito parcial para obtenção de aprovação na disciplina TCC II, no Curso de Engenharia de Computação, na Fundação Presidente Antônio Carlos.

Prof. Me. Jean Carlo Mendes.

Conselheiro Lafaiete, 2016.

RESUMO

Um jogo pode ser definido como um sistema interativo, organizado ou dirigido por meio de regras implícitas ou explicitas, gerador de emoções e causador de diversão em seus participantes (SILVA, 2009).

Este relatório tem como objetivo apresentar algumas das etapas do processo de desenvolvimento do jogo digital *multiplayer* chamado Kaya, este que foi desenvolvido usando o *framework* Phaser IO juntamente com Node.js.

O Conteúdo deste relatório engloba muitas das fases do desenvolvimento do jogo Kaya, dentre elas a fase de planejamento do jogo, criação do *Game Design Document,* os conceitos de Inteligência artificial, animação de sprites, detecção de colisão, eventos usando Socket IO e análise da lógica de desenvolvimento da estrutura cliente-servidor utilizada no jogo.

Este documento contém também algumas das ferramentas, processos e procedimentos mais utilizados para a conclusão do projeto.

Palavras-chave: JavaScript; jogo Kaya; Node.js; Phaser IO.

Lista de Figuras

Sumário

1 INTRODUÇÃO

Um jogo digital é um software que possui elementos de várias áreas do conhecimento como Computação Gráfica, Inteligência Artificial, Redes de Computadores, Multimídia, entre outros (CLUA e BITTENCOURT, 2005). Estes trabalham em conjunto de forma a oferecer a melhor experiência para o usuário, pois a característica mais importante para um jogo é a diversão que ele proporciona, tendo em vista que seu principal objetivo é o entretenimento.

Neste relatório serão expostas algumas das fases da criação do jogo kaya, dentre elas criação do *Game Design Document*, os conceitos de Inteligência artificial do jogo e também a lógica de desenvolvimento da estrutura cliente-servidor utilizada, abordando alguns dos procedimentos e a aplicação de algumas ferramentas gratuitas e livres, como Phaser IO e Node.js, desta forma facilitando o acesso e o uso pela comunidade.

Espera-se que este projeto estimule novas pesquisas na área da Computação aplicada ao setor de Entretenimento Digital, pois mesmo que este relatório não trate de todos os aspectos sobre o desenvolvimento de jogos, o mesmo poderá servir como base para futuros projetos e pesquisas nessa área.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Aplicar os *frameworks* Phaser IO e Node.js de forma eficiente para cumprir todas as etapas de desenvolvimento da primeira fase do jogo Kaya, de forma a proporcionar uma boa experiência para o jogador.

2.1.1 OBJETIVOS ESPECÍFICOS

- Criar um jogo multiplayer aplicando o motor de jogo Phaser IO de maneira a proporcionar uma melhor experiência para o jogador.
- Através da ferramenta Node.js e Socket IO, implementar um meio de comunicação em tempo real satisfatório entre Cliente e Servidor.
- Documentar as fases percorridas para o desenvolvimento do jogo neste relatório.

3 METODOLOGIA

Para o início de qualquer projeto de desenvolvimento de um jogo digital, é preciso separa-lo em diversas fases, visto que precisamos concluir cada uma delas antes de avançar para a próxima, estas fases abrangem desde o planejamento do jogo até a finalização do produto. Segundo Thadani (2010), podemos dividir o projeto em fase conceitual, pré-produção, fase de produção e pós-produção.

3.1 FASE CONCEITUAL

Na fase Conceitual é onde tudo começa, é onde as ideias de criar um jogo começam a tomar forma, aqui define-se o tema do jogo, a plataforma para a qual o jogo irá funcionar, mecânicas, enredo, jogabilidade, entre outros. Segundo Thadani (2010), a fase conceitual precisa ser bem eficiente, e todos da equipe devem ter a mesma visão do projeto que será desenvolvido.

Para o jogo Kaya foi escolhido o estilo RPG, onde o jogador acumula experiência para ficar mais forte e evoluir suas habilidades. A história é contada nos tempos medievais, sendo assim, as fases do jogo terão ambientes de cenários da mitologia antiga. Os protagonistas serão três personagens disponíveis para escolha do jogador, uma Arqueira, uma Feiticeira e um Bárbaro. O enredo se passa no mundo de Kaya, mundo este que está sendo invadido por seres de imenso poder e origem desconhecida cujo objetivo é a escravização de povos e roubo de poder.

O jogo apresentará dificuldade crescente em seus desafios, aumentando de acordo com que o jogador evolui, além deste conseguir aplicar mais dificuldade escolhendo modos mais difíceis dos cenários que irá jogar.

Os equipamentos dos personagens irão mudar constantemente de acordo com a evolução do personagem. Os inimigos irão aparecer aleatoriamente de acordo com a movimentação do personagem pelo cenário, ao final de cada fase, o jogador enfrentará um inimigo bem mais forte que os enfrentados no mapa, esses

serão descritos como “*Boss*” ou “Chefe da fase”. Alguns inimigos com força equivalente a de um Chefe poderão aparecer eventualmente a qualquer momento para o jogador enfrentar, este ao ser derrotado irá oferecer experiência e recompensas muito melhores que as dos inimigos normais enfrentados.

3.1.1 GAME DESIGN DOCUMENT

Definindo o conceito do jogo, inicia-se a criação do GDD (Game Design Document), que segundo o site Make Indie Games (2015), é um documento que “tem como objetivo descrever por meio de palavras, tabelas e diagramas as características do jogo, como controles, interfaces, personagens, armas, golpes, inimigos, fases e todos os aspectos gerais do projeto”, podendo até incluir alguns protótipos deste. Por esse e outros motivos o GDD não possui um formato padrão.

No GDD define-se as regras do jogo e a sua jogabilidade, por isso esse documento é muito importante e deve ser elaborado com cuidado, pois é aqui que define-se como será a personalidade do jogo e possivelmente de seus jogadores, além disso, esse documento é o guia principal no desenvolvimento prático do jogo.

Está presente também no GDD os objetivos apresentados no jogo, suas fases, como funcionará seu sistema de upgrades, sistemas de combate e definimos principalmente como será a jogabilidade (CICANC, 2013). Para o jogo Kaya, o jogador poderá se mover para todos os lados com mais quatro inputs para as habilidades, desta forma acredita-se que o jogador se adaptará rapidamente ao jogo.

Ao concluir o GDD, tem-se a possibilidade de perceber a relevância do jogo e até mesmo estipular o nível de interesse do público, sendo assim um documento muito importante para convencer possíveis investidores sobre o potencial de venda do jogo (THADANI, 2010).

3.2 FASE DE PRÉ-PRODUÇÃO

Aqui é onde se define os conceitos artísticos, as imagens dos personagens, os mapas, as principais características dos cenários e a descrição dos níveis do jogo. Nessa fase deve estabelecer se o projeto é realmente realista e exequível ou não. Se o objetivo não é alcançável, certos ajustes podem ser feitos para a proposta inicial (THADANI, 2010).

Para o desenvolvimento do jogo Kaya, foi preciso elaborar e reunir todo o tipo de material, desde desenhos, livros, informações, ideias, imagens, personagens, definições, histórias, músicas, vídeos, dentre outras mídias.

Nessa fase também definiu-se os conceitos referentes à programação, os quais facilitaram a execução do projeto e permitiram concluir os objetivos estabelecidos.

Este projeto se limita a um jogo 2D e a primeira e grande questão levantada foi a plataforma pra a qual o jogo Kaya seria desenvolvido. Como o objetivo é a facilidade de acesso, logo decidiu-se que o jogo deveria funcionar em navegadores de internet, pelo fato de ser uma plataforma onde vários dispositivos têm acesso.

Com a plataforma definida, escolheu se as linguagens de programação a qual o jogo Kaya seria desenvolvido, que foram JavaScript e HTML5. A escolha dessas linguagens não foi difícil visto que essas são voltadas para a internet e isso permite que a maioria dos dispositivos, com acesso a um navegador de internet que suporte HTML5, poderão executar o jogo, além dessas linguagens facilitarem sua compilação para aplicativos de smartphones.

3.2.1 NAVEGADORES DE INTERNET

Um navegador de Internet, em inglês *browser*, é o *software* que permite o usuário interagir com documentos virtuais da internet. Os Navegadores comunicam-se na maioria das vezes com servidores *web* usando principalmente o protocolo de comunicação HTTP (Protocolo de Transferência de Hipertexto), para

efetuar pedidos de arquivos e processar respostas vindas do servidor, mas aceitam vários outros protocolos de transferência (FREDIX, 2014).

A principal finalidade de um navegador é requisitar um determinado arquivo da Internet e providenciar a exibição do mesmo, quando o arquivo se trata de texto simples, linguagem de marcação ou imagens, o navegador tenta exibir o conteúdo.

3.2.2 HTML5

HTML (*HyperText Markup Language*) é a linguagem que descreve a estrutura e o conteúdo semântico de um documento da *web*, sendo este um padrão internacional onde suas especificações são mantidas pelo World Wide Web Consortium e o WHATWG (BIRCK, 2016). Sua versão mais atual é referida como HTML5.

O mais que mais interessa discutir por enquanto é o elemento HTML Canvas, pois este pode ser utilizado para desenhar, fazer composições de imagens ou fazer animações (TEOLI, 2014). Além disso, esse elemento tornou possível a implementação de uma API como o WebGL (*Web Graphics Library*), que oferece suporte para renderização de gráficos 2D e 3D.

3.2.3 JAVASCRIPT

JavaScript é uma linguagem dinâmica orientada a objetos. Seu uso mais comum é como scripts para páginas web, mas pode ser usada em vários outros ambientes sem *browser* como Node.js (FREITAS, 2016).

Sua sintaxe vem das linguagens Java e C, no entanto uma das principais diferenças é que o JavaScript não possui classes, no entanto essa funcionalidade é realizada por meio de protótipos de objetos. Outra diferença que pode ser notada é que suas funções são objetos, dando a essas a capacidade de armazenar código executável não precisando ser passados como parâmetro para algum outro objeto.

3.2.4 NODE.JS

Node.js é uma plataforma para desenvolvimento de aplicações do lado do servidor que são baseadas em rede utilizando JavaScript, ele utiliza um modelo de entrada/saída baseada a eventos não obstrutivos, tornando essa ferramenta ideal para aplicações em tempo real e com troca intensa de dados (COSME, 2014).

Uma coisa importante a se saber sobre Node.js é o fato de ser single thread, mesmo que isso pareça uma desvantagem isso é tratado de forma vantajosa pela ferramenta, de maneira que todos os recursos presentes no Node.js, e consequentemente na maioria das bibliotecas desenvolvidas, adotam um padrão não obstrutivo, ou seja, as operações são executadas de forma independente, não dependendo de nada que está sendo executado.

3.2.5 INTELIGÊNCIA ARTIFICIAL

O termo Inteligência Artificial (IA) em um jogo, tem um significado muito amplo, podendo variar desde a representação do controle de comportamento de personagens não controlados pelo jogador (*Non-player-character* - NPC) até a implementação da modelagem da física e detecção de colisão (MAXWELL, 2008).

Para a implementação do jogo Kaya, foi implementado o conceito de duas técnicas de IA, máquinas de estados finitas (FSM) e sistema baseado em regras (RBS).

A maneira de implementação do conceito de máquina de estados finitas foi basicamente em representar, através dos estados, as ações possíveis do agente, seja personagem jogável ou não, criando também regras de transição representando as condições que são avaliadas para que o agente mude de estado.

Em um RBS, o conhecimento é representado através de um conjunto de regras que trabalham em cima de variáveis, de modo que essas variáveis são levadas em consideração na hora de o sistema escolher sua próxima ação (Maxwell, 2008). Por exemplo, um NPC que receba os dados do jogador como parâmetros, pode escolher fugir da briga quando seus próprios pontos de vida

estão baixos, ou no caso contrário, atacar aquele jogador com menos pontos de vida.

3.2.6 BANCO DE DADOS

Um banco de dados é uma entidade na qual permite o armazenamento de dados de maneira estruturada, ou seja, é uma coleção de dados relacionados entre si, representando informações que tratam de um mesmo assunto.

As tabelas de um banco de dados podem representar entidades como usuários, personagem, veículos. Essas entidades possuem atributos que representam uma coluna da tabela e são os dados que se deseja armazenar.

3.3 FASE DE PRODUÇÃO

Esta fase se destaca das demais no quesito trabalho e dedicação, também é onde a maior parte do dinheiro é gato (THADANI, 2010). Deve se ter em mente que desde o início que é preciso deixar o código flexível, tanto para implementar alterações no futuro quanto para incluir novas funcionalidades, para isso pode-se usar alguns padrões de projeto (*designs patterns*) como MVC (*Model-View-Controller*). Segundo Schissato (2012) *design patterns* se usados com bom senso, podem se tornar ferramentas poderosas para qualquer desenvolvedor de software.

A criação da Interface Gráfica é, sem dúvida, uma das partes mais importantes no desenvolvimento do jogo, pois aqui é onde o jogador tem seus primeiros contatos com o jogo e é um dos motivos de a maioria de deles continuarem jogando, a primeira impressão é muito importante.

Ao escrever o código do jogo, o desenvolvedor deve ser orientado pelo GDD, pois nele contém todos os dados do processo de planejamento do jogo. O GDD não é definitivo, dessa forma pode receber mudanças constantes, pois algumas coisas no jogo provavelmente não irão sair como esperado, e neste caso mudanças são necessárias (CICANCI, 2013).

A parte de desenvolvimento da física inclui a programação do motor do jogo, onde normalmente utiliza-se um que já exista reduzindo assim o tempo de produção.Um motor de jogo ou *engine*, é um conjunto de códigos que executam todos os cálculos básicos para que um ambiente de jogo funcione, incluindo a simulação da física, colisões e movimento de objetos. Para o jogo Kaya não foi diferente, o motor escolhido para ser usado no projeto foi o Phaser IO, pois este se encaixou perfeitamente na proposta do jogo.

3.3.1 PHASER IO

Phaser IO é um motor para desenvolvimento de jogos em 2D gratuito e possui código aberto, isto é, qualquer um pode consultar, examinar ou até mesmo modificar o seu código, de forma a melhor atender suas necessidades. Este *framework* foi desenvolvido em JavaScript e recebe diversas atualizações frequentemente, mostrando que o projeto está em e em contínua melhora (DAVEY, 2013).

O foco desse *framework* é para o desenvolvimento de jogos que rodem em navegadores de desktops e de dispositivos móveis, possuindo um sistema para identificar e qualificar a melhor tecnologia a ser utilizada no dispositivo entre WebGL e Canvas. O Phaser IO possui um sistema de carregamento de arquivos bem completo, suportando os mais variados tipos como:

- Imagens, sprite sheets, Texture Atlases (incluindo pacotes de texturas, JSON Hash, JSON Array, Flash CS6/CC, e formato XML Starling);
- Arquivos de áudio e vídeo;
- Arquivos de dados como JSON, XML, texto;
- Arquivos JavaScript (permitindo o carregamento de partes do um jogo ou recursos baseados em JS);
- Tilemaps (CSV e Tiled map);
- Fontes Bitmap.

Esse *framework* possui também um sistema de física e emissão de partículas completo e que funciona muito bem, permitindo a criação ambientes simulados e efeitos de partículas facilmente, como explosões ou fluxos constantes para efeitos como chuva ou fogo.

3.4 FASE DE PÓS-PRODUÇÃO

A última fase descrita nesse projeto será a Fase de Pós-produção, onde pessoas são destacadas para jogar e apesentar todos os erros encontrados, além disso os gráficos e a jogabilidade são refinados e polidos (THADANI, 2010). Apesar de os testes estarem presentes durante todo o processo de desenvolvimento do jogo, é nessa parte do projeto que eles são mais relevantes.

Segundo Alves (2014), essa fase consiste em reunir pessoas diferentes para testar as funcionalidades do jogo, estes são chamados de testers e devem apontar e documentar todos os erros, falhas, defeitos ou até melhorias para serem aplicadas nos sistemas do jogo.

A Fase de Testes garante a qualidade do produto finalizado, portanto dever ser levada muito a sério, requerendo experiência e grande capacidade analítica dos *testers*, garantindo assim que o objetivo do jogo seja cumprido sem falhas, tanto em suas funcionalidades como também em seu objetivo principal que é o entretenimento.

Essas são as fases principais do desenvolvimento de um jogo digital onde primeiro é definida a “forma” do jogo, ou seja, parte conceitual, regras, jogabilidade, plataforma a qual o sistema será disponibilizado. Depois, passa-se à produção do jogo em si, ou seja, o jogo é desenhado e programado de fato, e por último o jogo se testado, tem suas eventuais falhas corrigidas e é aplicado os retoques finais deixando assim o jogo pronto para ser distribuído.

4 RESULTADOS

Os jogos *multiplayer* são aqueles em que vários usuários interagem ao mesmo tempo, seja ajudando ou competindo entre si, sendo que todos os jogadores se comunicam em tempo real (Filho, 2015). Vendo pela perspectiva de desenvolvimento do jogo Kaya, o sistema foi divido em duas partes denominadas Cliente e Servidor, onde o Servidor é responsável pelo por executar um serviço ou processamento de informações e o Cliente é responsável por requisitar esse serviço ou dados e oferece-lo ao jogador. Estes necessitam de comunicação em tempo real, de forma a oferecer uma boa experiência.

Para o jogo Kaya, a parte responsável pelo Servidor foi desenvolvida com Node.js e o Cliente desenvolvido utilizando Phaser IO. O Servidor é o responsável por maior parte do trabalho, porque precisa cuidar de todas as mensagens enviadas por todos os Clientes conectados e, só depois de fazer o devido tratamento, repassar as mensagens necessárias para os outros jogadores (Clientes).

4.1 CRIANDO O CLIENTE DO JOGO

O Cliente de um jogo *multiplayer* será responsável por fornecer a interface gráfica e também pelo envio mensagens para o Servidor, essas mensagens praticamente dizem respeito à ação a ser executada. O fluxo de ações do jogo é o seguinte:

- Cada Cliente terá sua tela com foco em seu personagem, podendo movê-lo para qualquer parte do cenário;
- O personagem pode utilizar suas habilidades sempre que quiser, contanto as mesmas já estejam disponíveis para uso.
- Inimigos irão aparecer conforme os jogadores percorrem o cenário, e os jogadores precisarão abatê-los para continuar avançando.

Segue uma imagem do jogo:

Figura 1: Imagem demonstração do jogo Kaya

Como foi dito, o Cliente do jogo foi desenvolvido usando Phaser IO, este possui uma estrutura muito concisa, possibilitando desenvolver protótipos de forma muito rápida.

Para o iniciar qualquer jogo usando Phaser IO, é preciso implementar três funções ‘*preload’*, ‘*create’* e ‘*update’*, sendo estas pré-requisito para seu funcionamento. Na Figura 2 é demonstrado a implementação básica da estrutura do Phaser IO, onde foi incluído uma imagem de fundo, um personagem juntamente com suas configurações de física e controles de movimento.

```
// Iniciando o uso do Phaser IO.
game = new Phaser.Game(1200, 600, Phaser.AUTO, 'Jogo Kaya',
        {preload: preload, create: create, update: update});

// Carrega os arquivos e as mídias do jogo.
function preload (){
    game.load.image('imgemFundo', 'assets/fundo.jpg');
    game.load.image('personagem', 'assets/personagem.png');
}

// Função responsável por criar os elementos do jogo.
function create (){
    //Criando o jogador e adicionando no jogo.
    jogador = game.add.sprite(50, 250, 'personagem');

    //Adicionando a física no jogador.
    game.physics.arcade.enable(jogador);
    jogador.body.gravity.y = 1500;

    cursors = game.input.keyboard.createCursorsKeys();
}

// Responsável por atualizar os comportamentos do jogo.
function update (){
    // Verifica se a tecla de seta esquerda está pressionada.
    if(cursors.left.isDown){
        // Move o jogador pra esquerda.
        jogador.body.velocity.x = -250;

    // Verifica se a tecla de seta direita está pressionada.
    } else if (cursors.right.isDown){
        // Move o jogador pra direita.
        jogador.body.velocity.x = 250;
    }
}
```

Figura 2: Implementação das funções principais do Phaser IO.

As animações de todos os personagens são feitas através de sprites, que são imagens com diversos quadros de um personagem com uma pequena variação entre si, estes quando disposto numa ordem coerente e obedecendo o padrão sequencial, geram ao observador uma ilusão de movimento quando exibidas sucessivamente (SILVA, 2014).

Figura 3: Exemplo de sprite utilizado no jogo Kaya.

O trabalho de efetuar a animação dos sprites ficou a cargo do Phaser IO, de forma que a imagem é carregada como "*spritesheet*" pela função "*preload*" (linha 11 da Figura 4), suas animações adicionadas na função "*criate*" (linha 18 e 19 da Figura 4) e por fim, a execução dessas é feita na função "*update*" (linha 26 e 31 da Figura 4).

```
1  // .. Função pleload presente na Figura 1...
2
3  // Função responsável por criar os elementos do jogo.
4  function create (){
5      // .. Código da função criate presente na figura 1...
6
7      // Define duas animações 'esquerda' e 'direita'.
8      jogador.animations.add('esquerda', [0,1,2,3], 10, true);
9      jogador.animations.add('direita', [4,5,6,7], 10, true);
10 }
11
12 // Responsável por atualizar os comportamentos do jogo.
13 function update (){
14     // Verifica se a tecla de seta esquerda está pressionada.
15     if(cursors.left.isDown){
16         // Move o jogador pra esquerda.
17         jogador.body.velocity.x = -250;
18         jogador.animations.play('esquerda');// inicia animação.
19
20     // Verifica se a tecla de seta direita está pressionada.
21     } else if (cursors.right.isDown){
22         // Move o jogador pra direita.
23         jogador.body.velocity.x = 250;
24         jogador.animations.play('direita');// inicia animação.
25     }
26 }
```

Figura 4: Animação de sprites com Phaser IO.

Todas as animações dos personagens do jogo Kaya, foram desenvolvidos tendo 15 quadros cada, tudo para que todos os movimentos dos personagens

sejam mais sutis, isto é, não havendo mudanças muito bruscas de um quadro pro outro, o que deixou seus movimentos mais fluidos.

Os cenários são criados a partir de imagens, essas que se movimentam tendo por base como o personagem se move, criando assim uma ilusão de movimento da paisagem. E para que houvesse sensação de profundidade nos cenários do jogo, utilizou-se um sistema de sobreposição de camadas de fundo dos elementos principais na ação decorrente.

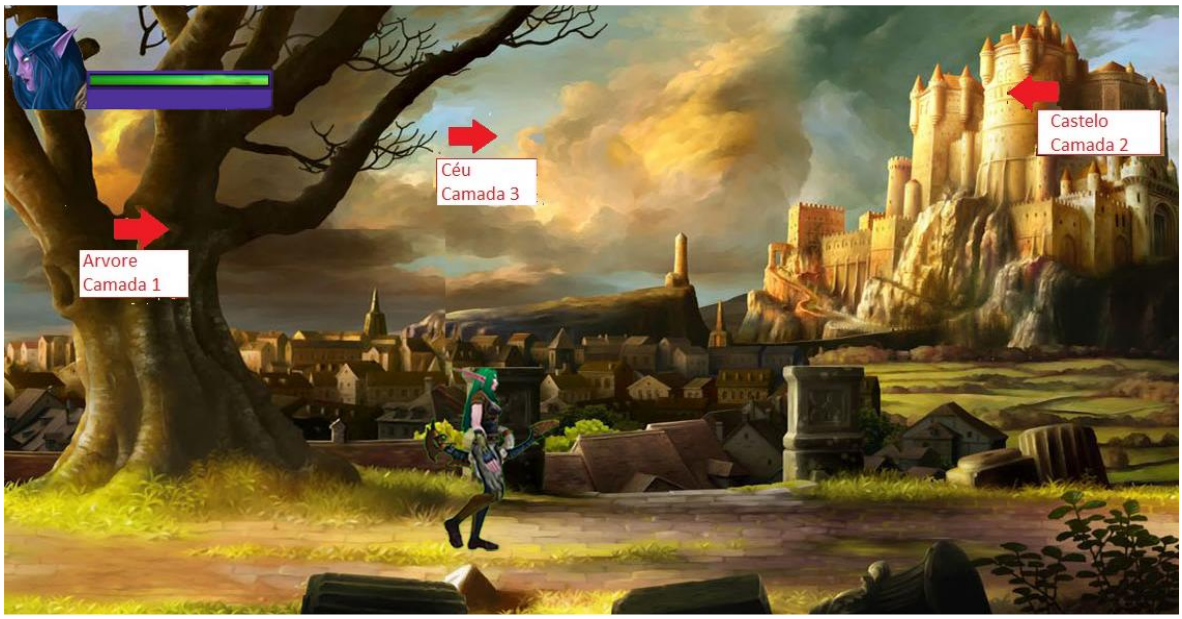

Figura 5: Demonstração de sobreposição de camadas.

Na imagem acima o céu, o castelo e a arvore são imagens separadas e se movimentam de forma diferente de acordo com a movimentação do personagem, além do castelo e o céu estarem mais “ofuscados” o que causa a sensação de profundidade.

Os inimigos aparecem de forma aleatória de acordo com que o jogador se move pelo cenário, e seu objetivo é atacar o jogador mais próximo, há situações em que o jogador deverá eliminar todos os inimigos para poder avançar no cenário. No caso de Chefes, cada um poderá ter um objetivo diferente, desde atacar jogadores com menos pontos de vida, até mesmo se esquivar de algumas habilidades do jogador.

A escolha do Phaser IO foi muito assertiva, visto que existem muitos outros motores para jogos no mercado. A escolha deste para a utilização do projeto se deu ao fato de ser voltado para navegadores, o qual era a proposta do projeto, além de sua API se muito extensa e completa, facilitando o entendimento de suas ferramentas e agilizando o processo de desenvolvimento.

4.2 CRIANDO O SERVIDOR DO JOGO

O Servidor de um jogo *multiplayer* centralizará a lógica de troca de mensagens e segurança de todas partidas jogadas. Este será apenas um e necessitará se comunicar com diversos clientes que poderão se comunicam de maneiras bem diferentes. Como já citado, utilizou-se Node.js para a implementação do servidor, em conjunto com uma biblioteca chamada Socket.IO.

Socket.IO é uma biblioteca que oferece uma API escrita em JavaScript tanto para front end quanto para back end, com base em eventos bidirecionais que permite a comunicação entre o Servidor e o Cliente sem esforço e em tempo real.

```
1  // Importando alguns pacotes para serem usados.
2
3  const io = require('socket.io'),
4        http = require('http');
5
6  // Criando o servidor no endereço 127.0.0.1:8080
7  const server = http.createServer((req, res) => {
8
9      res.statusCode = 200;
10     res.setHeader('Content-Type', 'text/plain');
11     res.end('Conectado!!');
12
13 }).listen(8080, '127.0.0.1', () = > {
14     console.log("Listen on 127.0.0.1:8080");
15 });
16
17 var socket = io.listen(server);
18
19 socket.sockets.on('connection', function(socket){
20
21     // Esperando por eventos vindos do cliente.
22     socket.on('mover', function(dado){
23
24         // Eventos podem vir com dados enviados junto com a mensagem.
25         console.log(dado);
26
27         // Emitindo um evento de volta para o cliente com os dados.
28         socket.emit('', {x: 10, y: 5});
29     });
30 });
```

Figura 6: Demonstração do Socket.io

A Figura 6 mostra uma maneira de se configurar um servidor Socket.IO para ouvir o Cliente, dando ênfase na facilidade de fazê-lo. Ao observar a linha 19, pode-se descrever que esta função está apenas esperando um evento de conexão vindo de um cliente. Ao receber este evento, é criado um objeto socket que é passado para a função de retorno. Este objeto é usado para se comunicar diretamente com este cliente que se conectou.

Dentro deste *callback*, cria-se mais ouvintes usando '*socket.on'*. Na linha 22, foi implementado um outro evento que aguarda o cliente para emitir a mensagem "mover", quando esta é enviada pelo Cliente, este evento será chamado. O argumento de 'dado' contém os dados enviados junto com a mensagem pelo cliente.

Para emitir uma mensagem de volta para o um cliente, usa-se o *'socket.emit'*, como demonstrado na linha 28, onde o primeiro argumento é o nome

da mensagem, que é também o nome de evento que o cliente deve ouvir, juntamente com todos os dados que desejamos enviar com a mensagem.

Para utilizar Socket.IO, é preciso apenas escutar as mensagens vindas do Cliente, reagir a elas e enviar mensagens de volta para ele.

Aplicando esse conceito ao servidor do jogo Kaya, pode-se concluir que quando um Cliente tenta se conectar ao Servidor do jogo, este primeiro recebe sua conexão, valida seus dados, e emite de uma mensagem aceitando ou recusando sua conexão. Caso o Cliente, já conectado ao Servidor, tenta entrar em um cenário, este emite esse evento para o Servidor do jogo, que responde de volta para com informações e dados necessários para que tudo funcione. Esses dados podem conter por exemplo, quantos jogadores já estão conectados naquele cenário, qual o nome de cada um, pontos de vida, além dos dados dos inimigos que estão no cenário.

Para a validação de ações dos jogadores, foi implementado um conceito de máquina de estados finitas, onde é aplicado um conjunto de estados e regras de transição no jogador, que refletem algum evento no mundo do jogo. Sendo assim, quando um jogador está parado, é aplicado a ele o estado “PARADO”, quando ele se move, este recebe o estado “MOVENDO”, o mesmo é aplicado quando ele ataca, quando está desorientado ou até mesmo quando está morto. Suas animações e ações giram em torno desses eventos, de modo que é iniciado a animação ou ação referentes ao estado desse jogador. Isso tudo acontece após passar por validações do Servidor para garantir que seja possível executar tal ação naquele momento.

Os inimigos também utilizam do conceito de máquina de estados, pois suas ações também são definidas por esta, com a diferença que alguns dos inimigos também seguem o modelo de representação de sistemas baseados em regras, ou seja, a situação atual do mundo é avaliada para se determinar qual é a melhor ação a ser executada.

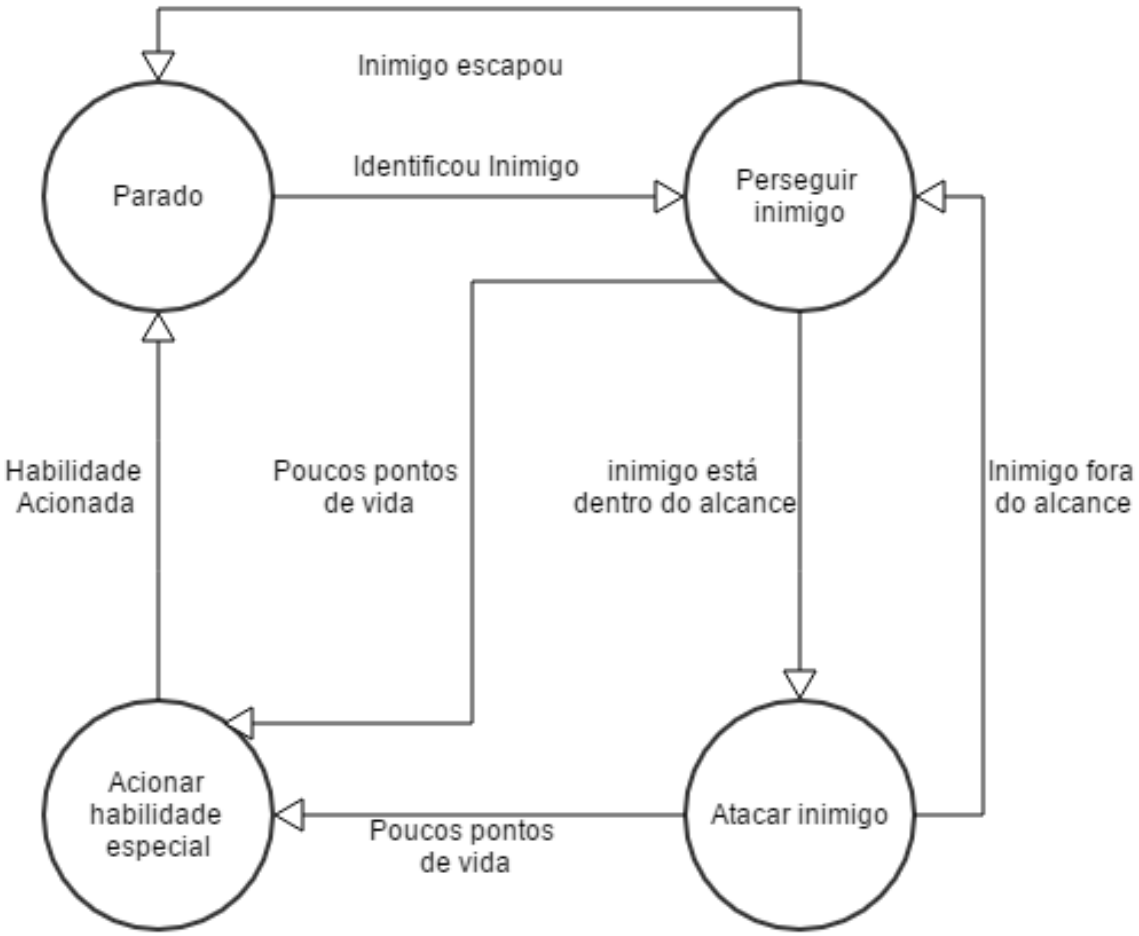

Figura 7: Máquina de estados de um inimigo do jogo Kaya.

Na representação do comportamento mostrada na Figura 7, a máquina de estados encontra-se estado inicial, nesse caso o ‘Parado’. A cada evento disparado, as regras de transição são avaliadas e, caso as condições atendam aos requisitos, o estado corrente é alterado para aquele que mais se encaixa na determinada situação. Com isso, as ações associadas ao estado serão executadas pelo agente.

O fluxo de execução de uma ação no jogo Kaya pode ser simplificado da seguinte maneira, primeiro o Cliente (jogador) emite um evento para o Servidor do jogo requisitando uma ação, os dados do Cliente são avaliados de forma a garantir que o mesmo possa executar tal ação, no caso de permissão, o Servidor dispara um evento de volta para o Cliente permitindo que o mesmo execute a ação requisitada.

Tomando como exemplo a ação de execução de uma habilidade, primeiro o Cliente (jogador) envia um evento requisitando a execução de uma determinada habilidade, o Servidor recebe esse evento, verifica se essa habilidade já está disponível para ser executada por esse jogador, no caso da aprovação de execução, o Servidor cria essa habilidade e responde enviando um evento para o Cliente, contendo todos os dados necessários para a execução da habilidade requisitada.

Todas as ações realizadas pelo Cliente são, antes de serem executadas, avaliadas pelo Servidor, desta forma cria-se uma camada de segurança a fim de garantir que as regras do jogo serão cumpridas por todos os jogadores.

O servidor Node.js foi a melhor escolha para o desenvolvimento desse projeto, visto que as operações e respostas oferecidas eram instantâneas, mesmo quando envolvia banco de dados. Uma de suas propostas é agilizar o desenvolvimento de aplicações e por mais que ele não seja a melhor solução para todos os casos, para o desenvolvimento do jogo Kaya ele atendeu perfeitamente.

4.3 DETECÇÃO DE COLISÃO

Basicamente a ideia de detecção de colisão é criar um polígono imaginário em torno daquilo que se deseja testar a colisão, ou seja, define-se pontos em volta dessas figuras que ao serem ligados formam um polígono. As coordenadas destes significam seu posicionamento no cenário, sendo assim pode-se tratar a colisão entre eles (ZANARDO, 2005).

Figura 8: Representação dos polígonos que envolvem os personagens.

Na Figura 8 é mostrado como como são representados os polígonos nos personagens, mas praticamente tudo no jogo Kaya tem sua estrutura física delimitada por um polígono, sendo personagens, magias, projéteis, objetos, entre outros. Essa estrutura de detecção de colisão é fornecida pelo Phaser IO.

No jogo Kaya, as estruturas físicas podem ser separas por grupos, estes que podem ou não sofrer colisões com outros grupos, dependendo de sua implementação.

Para ilustrar essa situação criou-se seguinte exemplo, tem-se dois grupos de estruturas físicas diferentes denominadas ‘estrutura_inimigos’, que engloba a estrutura física de todos os inimigos, e a ‘estrutura_habilidades _jogadores’, que engloba a estrutura física de todos as habilidades dos jogadores. Utilizando a estrutura do Phaser IO, pode-se definir que há colisão entre esses dois grupos, ou seja, uma habilidade do jogador reagirá ao entrar em contato com um inimigo, podendo resultar em uma ação como reduzir pontos de vida do inimigo alvo. Implementação ilustrada na Figura 9.

```
function criate (){
    // Habilitando a física no jogo.
    game.physics.startSystem(Phaser.Physics.P2JS);

    // Ativar eventos de impacto.
    game.physics.setImpactEvents(true);

    // Criando 2 grupos de colisão, como citado no exemplo.
    var estrutura_inimigos = game.physics.p2.createCollisionGroup();
    var estrutura_habilidades_jogadores = game.physics.p2.createCollisionGroup();
    game.physics.p2.updateBoundsCollisionGroup();

    // Criando Habilidade.
    var habilidade = game.add.sprite(200, 200, 'habilidade');
    game.physics.p2.enable(habilidade, false);
    habilidade.body.setCircle(28); //Criando um poligono em torno do sprite.

    // Criando Inimigo.
    var inimigo = game.add.sprite(200, 200, 'inimigo');
    game.physics.p2.enable(inimigo, false);
    inimigo.body.setRectangle(40, 40); //Criando um poligono em torno do sprite.
    inimigo.pontosVida = 10;

    // Definindo o grupo que o inimigo e a habilidade pertencem.
    inimigo.body.setCollicionGroup(estrutura_inimigos);
    habilidade.body.setCollicionGroup(estrutura_habilidades_jogadores);

    // Definindo a colisão entre a habilidade e o inimigo;
    habilidade.body.collides([estrutura_inimigos]);

    // Definindo a colisão entre a inimigo e a habilidade com uma função de ação.
    habilidade.body.collides(estrutura_habilidades_jogadores, reduzirPontosVida, this);
}

function reduzirPontosVida (body1, body2){
    inimigo.pontosVida--;
}
```

Figura 9: Implementação de grupos de física utilizando Phaser IO.

Aplicando isso à estrutura de cliente-servidor do Jogo Kaya, quando há uma colisão entre dois grupos, é disparado um evento para o Servidor com as informações necessárias, este recebe esse evento, calcula o dano que esse inimigo sofreu e reduz seus pontos de vida, em seguida ele responde enviando um evento para o Cliente, contendo todos os dados já calculados e prontos para a renderização pelo Cliente.

O sistema de física e detecção de colisão oferecida pelo Phaser IO funcionou perfeitamente dentro do que é a proposta do jogo, agindo melhor que o esperado ao reagir aos eventos vindos do servidor.

4.4 ESTRUTURA DO BANCO DE DADOS

Representar um jogo RPG pode ser muito trabalhoso, visto que além de existir muitas entidades que possuem muitos atributos, a maioria destas dependem uma da outra.

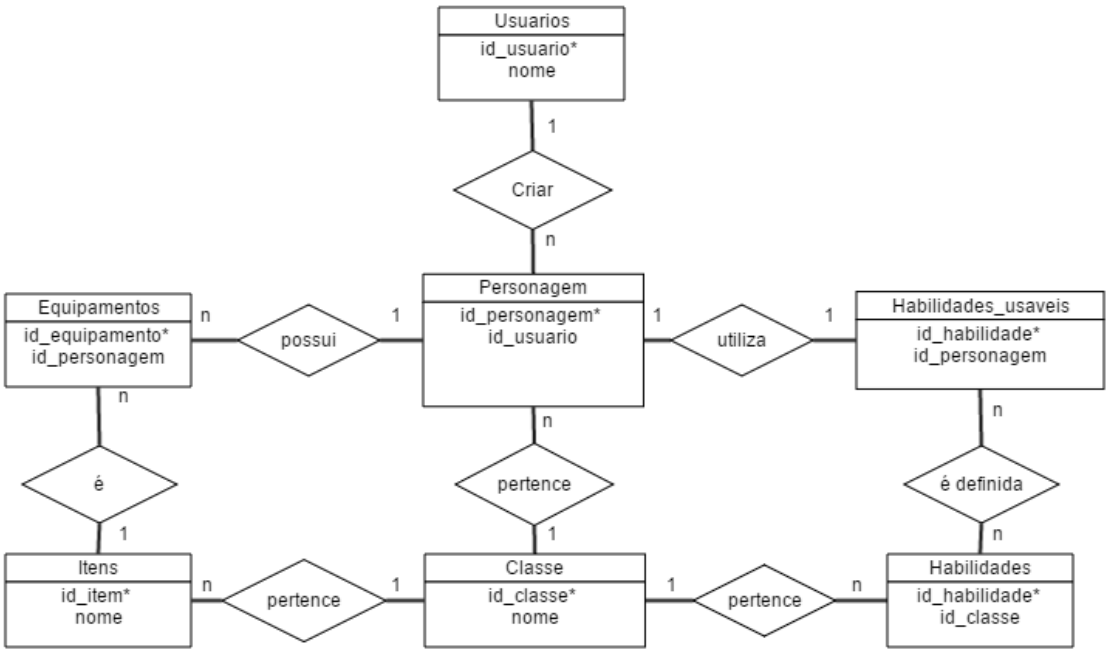

Figura 10: Diagrama de entidade e relacionamento de um usuário.

Na Figura 10, tem-se representado pelo diagrama as seguintes entidades e relacionamentos:

- Usuário pode criar vários Personagens e um Personagem pode ser criado apenas por um Usuário.
- Personagem pertence a uma Classe e uma Classe pode ser pertencida a vários Personagens.
- Personagem possui vários Equipamentos e um Equipamento pode ser possuído por apenas um Personagem.
- Um Equipamento é um Item e um Item pode definir vários Equipamentos.

- Um Item pertence apenas a uma Classe e uma Classe pode pertencer a vários Itens.
- Personagem possui várias Habilidades-usáveis e uma Habilidade-usável pode ser possuída por apenas um Personagem.
- Habilidades-usáveis são definidas por uma Habilidade e uma Habilidade pode definir várias Habilidades-usáveis.
- Uma Habilidade pertence apenas a uma Classe e uma Classe pode pertencer a várias Habilidades.

Para representar as entidades e relacionamentos que envolvem um mapa, criou-se o diagrama representado na Figura 11, nele será possível guardar todos os dados referentes aos mapas, fases e cenários.

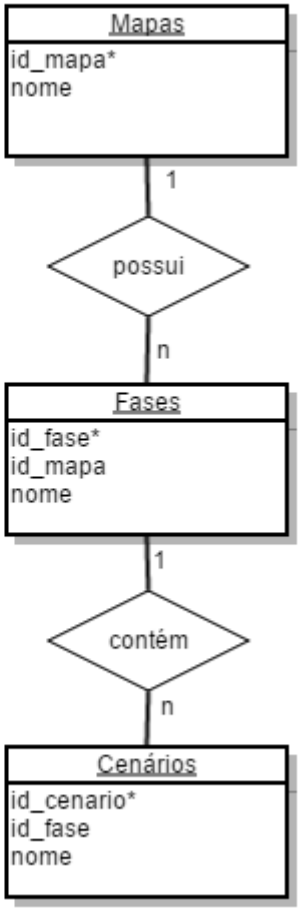

Figura 11: Diagrama de Entidade e Relacionamento de um Mapa.

A partir do diagrama representado pela Figura 11, pode-se concluir que:

- Um Mapa possui várias Fases e uma Fase pode ser possuída por apenas por um Mapa.
- Uma Fase possui vários Cenários e um Cenário pode ser possuído por apenas uma Fase.

Por fim, na Figura 12 tem-se a representação das entidades e relacionamentos que envolvem um inimigo, o qual vai ser responsável por armazenas todos os dados dos inimigos, como nome, dano, habilidades usadas, dentre outras informações.

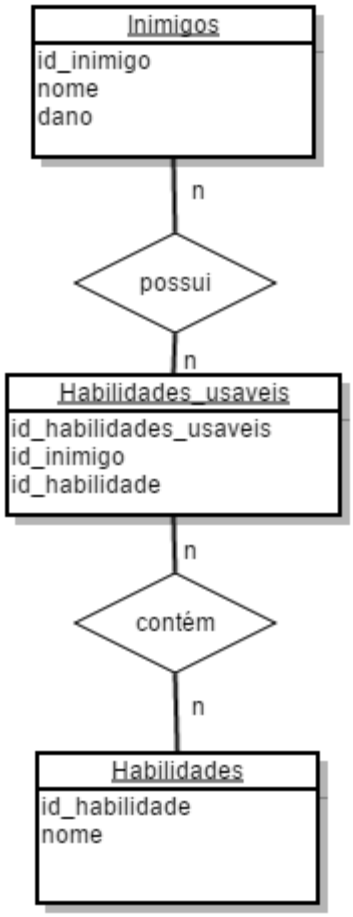

Figura 12: Diagrama de Entidade e Relacionamento de um Inimigo.

- Um Inimigo possui várias Habilidades_usáveis e uma Habilidades_usável pode ser possuída por apenas por um Inimigo.
- Habilidades_usáveis contém várias Habilidades e várias Habilidades estão contidas em Habilidades_usáveis.

A partir das informações abstraídas dos diagramas apresentados, foi possível desenvolver o banco de dados propriamente dito, já que eles forneceram todas as informações sobre os aspectos de relacionamentos entre as entidades do jogo.

5 CONCLUSÕES

A realização deste projeto nos permite tirar várias conclusões acerca do processo de desenvolvimento de um jogo digital 2D. Permitiu o entendimento de cada uma das fases de desenvolvimento e como cada uma delas é importante para a conclusão do produto final. Principalmente a Fase do Game Design, onde é criado o GDD, pois além de estar presente em todo o processo de criação do jogo, pode até servir como ferramenta para conseguir patrocínio para o projeto.

Permitiu também ganhar conhecimento com ferramentas incríveis como Node.js e Phaser IO, de modo conseguir aplicar essas ferramentas em conjunto de forma a conseguir desenvolver com sucesso o projeto que foi proposto aqui neste relatório. Concluindo também que Node.js foi a escolha perfeita para a proposta desse jogo, visto que foi ideal para a aplicações do conceito de “tempo real” dentro do jogo.

JavaScript foi a principal linguagem em todo o projeto, e o uso de uma linguagem como essa, tanto pro Cliente quanto para o Servidor, permitiu um ganho de tempo inestimável, principalmente pelo fato dela ser extremamente dinâmica. Outro fato a se considerar é que não há instaladores, para rodar o jogo desenvolvido basta apenas ter um navegador compatível com HTML5, poupando o trabalho de compilação e instalação.

Phaser IO simplesmente cumpriu o que prometeu, é uma ferramenta leve e extremamente prática de usar, deixando o trabalho duro apenas para as lógicas de funcionamento do jogo. A física do jogo funcionou melhor que o esperado e as animações ficaram perfeitas, além de reagir aos eventos do servidor com extrema sutileza.

Ao final do projeto foi possível entregar a primeira fase do jogo Kaya, este pode ser nomeado de versão de demonstração, o qual proporcionou a oportunidade de entender mais como funciona o processo de criação de jogos digitais, tanto sua documentação, modelagem e processos de desenvolvimento.

Para sugestão de trabalhos futuros será a implementação de um servidor para o jogo em uma linguagem como Java ou C++, de forma a fazer uma análise

de desempenho entre Node.js que é uma ferramenta single *thead* não obstrutivo, com uma linguagem *multithread* como as citadas.

6 REFERÊNCIAS

CLUA, Esteban Walter Gonzalez, Bittencourt, João Ricardo. Desenvolvimento de Jogos 3D: Concepção, Design e Programação. Disponível em: <http://bit.ly/2fLKmG7>. Acessado em: 21/04/2016.

CICANCI, Bruno. Como ser um programador de jogos: Game Design Document. Disponível em: <http://bit.ly/2eZ9koc>. Acessado em: 11/07/2016.

CICANCI, Bruno. Fim do TCC da pós-graduação: Oniro. Disponível em: <http://bit.ly/2gb9F7f>. Acessado em: 11/07/2016.

MAKE INDIE GAMES. GAME DESIGN DOCUMENT. Disponível em: <http://bit.ly/2geKKgs>. Acessado em: 10/10/2016.

THADANI, Rahul. Video Game Development Process. Disponível em: <http://bit.ly/2geNlXL>. Acessado em: 10/10/2016.

BIRCK, Renan. HTML. Disponível em: <https://mzl.la/2gCuQQW>. Acessado em: 21/09/2016.

TEOLI. CANVAS. Disponível em: <https://mzl.la/2gvmIR8>. Acessado em: 21/09/2016.

FREITAS, Marcelo Melo. JAVASCRIPT. Disponível em: <https://mzl.la/2fV5p8H>. Acessado em: 21/09/2016.

SCHISSATO, Jéssica. O que são Design Patterns? Disponível em: <http://bit.ly/2f4G8e3>. Acessado em: 23/09/2016.

LOPES, Cosme. O que é Node.js e saiba os primeiros passos. Disponível em: <http://bit.ly/2fy4wTb>. Acessado em: 21/10/2016.

MAXWELL Vrac Puc Rio, Inteligência Artificial em Jogos. Disponível em: <http://bit.ly/2eZ94W7>. Acessado em: 21/09/2016.

DAVEY, Richard. Aprenda a Utilizar a Game Engine Phaser Para Desenvolvimento de Jogos HTML5. Disponível em: <http://bit.ly/2eG7pok>. Acessado em: 21/09/2016.

ALVES, João. Desenvolvimento de Jogos 2D Criação e distribuição de jogos 2D. Disponível em: <http://bit.ly/2geLoKW>. Acessado em: 21/09/2016.

FILHO, Sergio. Android Games: Desenvolvendo um jogo multiplayer com Java e Android. Disponível em: <http://bit.ly/2ghxRWE>. Acessado em: 21/09/2016.

SILVA, Maycon Prado Rocha et al. Jogos Digitais: definições, classificações e avaliação. Disponível em: <http://bit.ly/2fRdJHh>. Acessado em: 21/09/2016.

SOUZA, J.F. Sprite Sheets (Definição). Disponível em: <http://bit.ly/2fVFnTi>. Acessado em: 21/10/2016.

ZANARDO, Gustavo Russo. Técnicas de Detecção de Colisão para Jogos. Disponível em: <http://bit.ly/2fVBrBL>. Acessado em: 21/10/2016.