

**CENTRO PAULA SOUZA**



---

**FACULDADE DE TECNOLOGIA DE AMERICANA**  
**Curso Superior de Tecnologia em Jogos Digitais**

**Rodolfo Helfenstein Bulgam**

**Kouri: Desenvolvimento de Jogos Mobile com Unity**

**Americana, SP**  
**2015**

**CENTRO PAULA SOUZA**



---

**FACULDADE DE TECNOLOGIA DE AMERICANA**  
**Curso Superior de Tecnologia em Jogos Digitais**

**Rodolfo Helfenstein Bulgam**

**Kouri: Desenvolvimento de Jogos Mobile com Unity**

Trabalho monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Jogos Digitais da Fatec Americana, sob orientação do Prof. Kleber de Oliveira Andrade.

**Americana, S. P.**  
**2015**

**FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS**  
**Dados Internacionais de Catalogação-na-fonte**

B952k	Bulgan, Rodolfo Helfenstein Kouri: desenvolvimento de jogos Mobile com Unity. / Rodolfo Helfenstein Bulgan. – Americana: 2015. 44f.  Monografia (Graduação em Tecnologia em Jogos Digitais). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza. Orientador: Prof. Me. Kleber de Oliveira Andrade  1. Jogos digitais I. Andrade, Kleber de Oliveira II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.  CDU: 681.6
-------	--

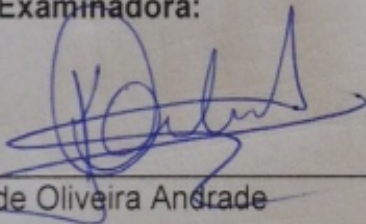
Rodolfo Helfenstein Bulgam

## Kouri: Desenvolvimento de Jogos Mobile com Unity

Trabalho de conclusão de curso  
apresentado à Faculdade de  
Tecnologia de Americana como parte  
dos requisitos para obtenção do título  
de Tecnólogo em Jogos Digitais

Americana, 12 de dezembro de 2015.

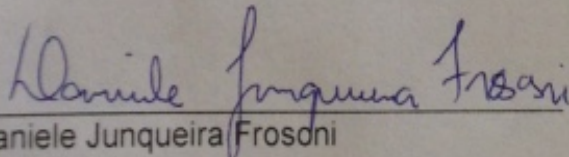
**Banca Examinadora:**



Kléber de Oliveira Andrade

Mestre

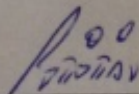
FATEC – Faculdade de Tecnologia de Americana



Daniele Junqueira Frosoni

Especialista

FATEC – Faculdade de Tecnologia de Americana



Rodrigo Viviani

Especialista

FATEC – Faculdade de Tecnologia de Americana

## RESUMO

Atualmente, o mercado de entretenimento está aquecido devido ao crescimento do público e a grande demanda por conteúdo. Um dos nichos que mais tem crescido em grande escala, é o de jogos digitais, em destaque, jogos para dispositivos mobile. Baseando-se nesse contexto, este trabalho tende a apresentar as fases de desenvolvimento de um jogo mobile de gênero casual para dispositivos Android.

Para o desenvolvimento deste trabalho, foi utilizado o motor gráfico Unity3D em conjunto a linguagem C#, onde é disponibilizado ferramentas de criação desenvolvimento 3D, também possuindo um ambiente completo para confecção de jogos 2D.

**Palavras-chave:** Jogos Digitais; Mobile; Desenvolvimento.

## **ABSTRACT**

Actually, the entertainment market is highly-heated because an increasing number of audience and high demand for content. One with the most growth, is the digital games market, for mobile devices. This paper will show the phases of a game development of a casual game, for a Android device.

For this project development, was used the Unity3D engine with C# language, which has the tool for the development of a 3D, but can offer too a complete environment for the development of a 2D games.

**Keyword:** Digital games; Mobile; Development

## SUMÁRIO

1. INTRODUÇÃO.....	9
2. PROCESSO DE DESENVOLVIMENTO DE JOGOS .....	11
2.1. METODOLOGIA DE DESENVOLVIMENTO.....	11
2.2. FERRAMENTAS UTILIZADAS NO DESENVOLVIMENTO.....	12
3. PROJETO DE JOGO PARA ANDROID .....	17
3.1. INTRODUÇÃO A TEN-PAGE SGDD.....	17
3.2. DETALHES DO PROJETO .....	17
3.3. CONTORNO DO JOGO.....	18
3.3.1. OBJETIVO .....	18
3.3.2. HISTÓRIA DO JOGO.....	18
3.3.3. FLUXO DO JOGO .....	18
3.4. PERSONAGEM.....	19
3.5. GAMEPLAY .....	19
3.6 – GAME WORLD .....	20
3.7. GAME EXPERIENCE.....	20
3.8. MECÂNICAS DE JOGO.....	20
3.9 - INIMIGOS.....	22
3.10. CUTSCENES.....	23
3.11. BÔNUS MATERIAL.....	23
4. DESENVOLVIMENTO DO JOGO KOURI .....	24
4.1- DIAGRAMA DE CASO DE USO .....	24
4.2. DIAGRAMA DE CLASSE .....	27
4.2.1. CLASSE LAUNCHER SCREEN .....	28
4.2.2. CLASSE MAIN MENU .....	28
4.2.3. CLASSE BACKGROUND SCROLLING .....	28
4.2.4. CLASSE GAMEMANAGER.....	30
4.2.5. CLASSE PLAYER CONTROLLER.....	32
4.2.6. CLASSE SPAWNCONTROLLER.....	33

4.2.7. CLASSE POWERUP .....	34
4.3. FLUXO DE TELAS.....	34
4.4. ARTE E SOM.....	35
4.5. SERVIÇOS.....	37
4.5.1. UNITY CLOUD BUILD .....	37
4.5.2. UNITY ANALITICS .....	39
4.5.3. UNITY ADS.....	40
4.6. PUBLICAÇÃO.....	41
4.7. VERSÕES DISPONÍVEIS .....	42
5. CONCLUSÃO .....	43



## LISTA DE FIGURAS

Figura 1 - Fluxo de Scrum aplicado a um projeto de 30 dias.....	12
Figura 2- Exemplo de Repositório de Git .....	14
Figura 3- Exemplo de Movimentação de Plataforma .....	21
Figura 4 - Fluxograma do Funcionamento das Plataformas .....	21
Figura 5 - Demonstração dos Obstáculos Elevados .....	22
Figura 6 - Demonstração dos Obstáculos Fendas .....	23
Figura 7 - Diagrama de Caso de Uso.....	24
Figura 8 - Diagrama de Classe .....	28
Figura 9 - Sistemas de Camadas Utilizando Eixo Z.....	29
Figura 10 – Componentes de Scroll do Background.....	29
Figura 11 - Propriedade Time das Configurações Gerais do Projeto .....	31
Figura 12 - Apresentação de Tela de Pause.....	31
Figura 13 - Demonstração do Obstáculo sendo Destruído .....	33
Figura 14 - Diagrama do Fluxo de Sequência de Telas.....	34
Figura 15 - Paleta de Cores usadas no Projeto .....	35
Figura 16 - Tela de Configurações do Unity Cloud Build .....	38
Figura 17 - Tela de Acompanhamento das Builds .....	39
Figura 18 - Gráfico de Jogadores Ativos por Período .....	39
Figura 19 - Gráfico da Propriest Custom Game Over .....	40
Figura 20 - Propriedades e Informações do Unity ADS .....	40
Figura 21 - Dashboard antes do Jogo ser Publicado .....	41
Figura 22 - Dashboard após Jogo ser Publicado .....	42

## LISTA DE TABELAS

Tabela 1 - Caso de Uso "Abrir Jogo" .....	25
Tabela 2 - Caso de Uso "Configurar Som".....	25
Tabela 3 - Caso de Uso "Iniciar Partida".....	26
Tabela 4 - Caso de Uso "Pular" .....	26
Tabela 5 - Caso de Uso "Pausar Jogo".....	27
Tabela 6 - Caso de Uso "Fechar Jogo".....	27
Tabela 7 - Tabela de Sons do Ambiente.....	36
Tabela 8 - Tabela de Sons Utilizados para Efeitos .....	37

## 1. INTRODUÇÃO

Segundo o estudo realizado pela NewZoo, o mercado de jogos tem como perspectiva movimentar US\$ 91,5 bilhões em 2015, uma diferença de até 9% em relação ao faturamento do ano passado. Com esse crescimento a demanda de novos jogos e a existência de mão de obra qualificada cresce em ritmo acelerado.

No mesmo estudo, é demonstrado que o setor de *smartphones* e *smartwatches*, que estão em constante crescimento, representam 23% do faturamento total da indústria o que significa US\$ 21,0 bilhões.

Com isso, existe uma grande quantidade de jogos entrando no mercado mobile, aliados a temas e mecânicas que abrangem todo tipo de publico, desde os consumidores assíduo até o publico que opta por jogos mais casuais.

Jogos casuais normalmente são desenvolvidos por pequenas empresas, onde não se tem um grande investimento, optando por mecânicas mais simples e intuitivas, que reduz a dedicação na qual o jogador tende a ter com aquele jogo, o tornando um entretenimento ocasional para diversas situações. Dentro do âmbito de jogos casuais, existe um gênero que chama a atenção do mercado, por sua praticidade e facilidade de desenvolvimento, o *Endless Runner*.

Jogos do estilo *Endless Runner*, popularmente conhecidos como jogos de corredor, tem como base, trazer a atenção do jogador para apenas uma partida, que é estendida conforme a determinação do mesmo, em conseguir uma pontuação maior, não impondo responsabilidades ao jogador, após o termino da mesma. Hoje, no mercado, existem vários jogos deste gênero, entre os mais populares, estão *Subway Surfers*, *Temple Run*, *Sonic Dash*. Os jogos em terceira dimensão são beneficiados por causar uma imersão maior do jogador com seus cenários detalhados, animações mais fluidas e realistas, porem, também existem jogos que utilizam mecânicas 2D, como *Zombie Tsunami*, *Sonic Runners*, que não ficam para trás na disputa, pois trazem outras características de jogabilidade aliado a estilos de arte que enriquecem a experiência do usuário.

Jogos desenvolvidos em 2D possuem alguns aspectos que facilitam a confecção do mesmo, como a não necessidade de modelos 3D, uma física de ambiente mais simplificada, desenvolvimento da arte através de sprite e na maioria

dos casos um ganho de desempenho por parte do aparelho, pois um ambiente 2D utiliza de menos cálculos e recursos mais leves. Esse estilo de jogo é bastante usado por empresas e desenvolvedores que não tem um valor alto para investirem em mecânicas e modelagem de objetos 3D, tendo assim apenas as texturas em formas de *sprites*, baseado na escolha do artista. Algumas técnicas artísticas, como, por exemplo *Pixel Art*, ganham bastante espaço no estilo de jogo 2D, pois remetem ao um jogo um característica *Retro*, que é bastante consumida pelo mercado até os dias de hoje.

Para o desenvolvimento deste projeto, que tem como base jogos 2D, foi escolhida a ferramenta Unity( baseando-se na linguagem C#), que disponibiliza um ambiente para criação de jogos desse estilo, com varias ferramentas já pré-moldadas, tendo como exemplo a física do ambiente, tratamento de sprites e spritesheets, gerenciamento de animações, entre outros, que tornam essa engine uma ótima opção.

O objetivo deste projeto, é demonstrar de forma geral, a confecção de um jogo casual 2D do gênero *Endless Runner* para Android, utilizando o motor gráfico Unity, apresentando processos, ferramentas e serviços que auxiliam o desenvolvedor tanto na criação com na gestão geral do projeto e levantando pontos a serem analisados como positivos ou negativos.

## 2. PROCESSO DE DESENVOLVIMENTO DE JOGOS

### 2.1. METODOLOGIA DE DESENVOLVIMENTO

Para o desenvolvimento do projeto, foi utilizada a metodologia de desenvolvimento *Scrum* (PRESSMAN, 2006), onde o desenvolvimento é fracionado em etapas chamadas *sprints*. Cada *sprint* possui uma listagem de *story* que é um grupo de tarefas focado em uma determinada característica da aplicação.

Keith (2010) afirma que a primeira aparição do *Scrum* como uma metodologia aplicada ao desenvolvimento foi na obra *Wicked Problems, Righteous Solutions* (DEGRACE E STAHL, 1990). O mesmo modelo mencionado foi aplicado em "Easel Corporation." por Jeff Sutherland e Ken Schwaber que deram início na utilização desta prática no mercado.

O *Scrum* é dividido em algumas etapas que tem características específicas. São elas:

*Product Backlog*: Etapa em que é levantado os requisitos da aplicação e são divididos nas *story*, analisando cada funcionalidade que será executada e a forma na qual desenvolver a mesma. O tempo total da *story* é importante ser bem definida nessa etapa, pois a métrica da duração é utilizada para compor qual *story* entra no determinado *sprint*.

*Sprint Planning*: Nessa etapa é definido qual *story* compõe o *sprint* a ser desenvolvido no momento, o *sprint* possui um tempo definido e a duração total das *story* tem que conseguir caber no mesmo. Nesta etapa também é levantado como será o desenvolvimento de modo técnico, pois se define um tempo para realização da tarefa baseado na dificuldade.

*Execução do Sprint*: Este é o momento que é desenvolvido as funcionalidades e realizado os testes. Caso em tempo de desenvolvimento é analisado que algo que não foi estimado no *Sprint Planning* é necessário fazer uma reestimativa do mesmo e avaliar se consegue concluir no tempo planejado para o primeiro *sprint*, caso não, o mesmo será realocado e executado no *sprint* seguinte.

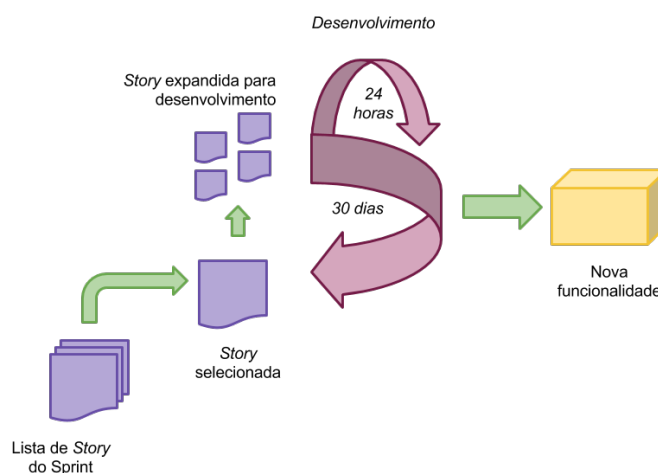
*Retrospectiva do Sprint*: Com o *sprint* finalizado essa é a etapa para avaliar o processo de desenvolvimento e ver o que foi executado como planejado,

dificuldades encontradas e novas características interessantes para serem desenvolvidas. Com a análise pronta, é criado um planejamento caso exista alguma mudança, que a mesma seja implantada no próximo *sprint* caso possível.

*Demonstração:* Este momento é apresentado o que foi desenvolvido no *sprint* e avalia se esta tudo conforme planejado e foi executado com sucesso.

O fluxo de desenvolvimento utilizado no projeto pode ser visto na Figura 1 com *sprint* de 30 dias corridos. Como resultado, a utilização do método ajudou, principalmente, devido à maleabilidade na escolha do que deve ser desenvolvido em cada *sprint*, bem como a liberdade de mudar a ordem do desenvolvimento das histórias, conforme notava se que seria mais eficiente a troca. Além disso, a visão de todo que o método fornece ajuda para saber exatamente o quanto falta para que tudo esteja pronto e de acordo com o que deve ser feito.

Figura 1 - Fluxo de Scrum aplicado a um projeto de 30 dias



Este projeto foi estimado em 21 dias sendo 7 o período de cada *sprint*, quando finalizado o *sprint* será enviado ao orientador uma demonstração do projeto para o orientador para análise. O *sprint* 1 teve o foco na mecânica do jogo, pois a mesma demandaria mais esforço e caso existisse algum atraso, poderia ser escalado para os próximos *sprints*, diminuindo o impacto de atraso do projeto. No *sprint* 2 foi desenvolvido e anexado a arte e o som do jogo, realizando toda parte artística e os efeitos, deixando o jogo com a aparência próxima da desejada no final do projeto. No último *sprint* é corrigido os bugs remanescentes dos *sprints* anteriores e a implementação de novas features como por exemplos os Unity Services.

## 2.2. FERRAMENTAS UTILIZADAS NO DESENVOLVIMENTO

As ferramentas utilizadas para o desenvolvimento foram selecionadas para suprir as necessidades do projeto. Para o desenvolvimento foi utilizado a *Engine* gráfica Unity, que como *Michelle Menard* descreve em seu livro *Game Development with Unity (2012)* é uma poderosa ferramenta de desenvolvimento que mescla o uma *Engine* poderosa com um editor que supre a maioria das necessidades do desenvolvedor. A Unity disponibiliza uma vasta e detalhada documentação e uma comunidade de desenvolvedores ativa onde existe uma troca de conhecimento grande. Além desses aspectos a Unity possui a *Asset Store* que é uma loja virtual onde os desenvolvedores podem disponibilizar gratuitamente ou monetariamente os componentes, artes e qualquer outra ferramenta que o mesmo tenha criado.

Outra característica relevante para a escolha da ferramenta que *Michelle* também cita em seu livro, é a linguagem que é utilizada pela Unity, conhecida como *Mono* que é uma linguagem de programação *open-source* baseada em .NET que possibilita gerar a aplicação final para várias plataformas, no caso de aplicações *mobile*, ele consegue exportar para iOS, Android e Windows Phone.

O Kouri primeiramente foi desenvolvido e exportado para Android com versões 4.0 até as mais novas.

Além de ferramentas de produção, foram utilizados outros recursos para outros pontos do desenvolvimento do jogo, como por exemplo, o GitHub.

Como *Richard E. Silverman* descreve em seu livro *GIT : A Pocket Guide*, o Git é uma ferramenta que auxilia o desenvolvedor a localizar mudanças realizadas em um bloco de arquivos, e consegue tratar essas alterações de forma separada, o que nos chamamos de controle de versionamento.

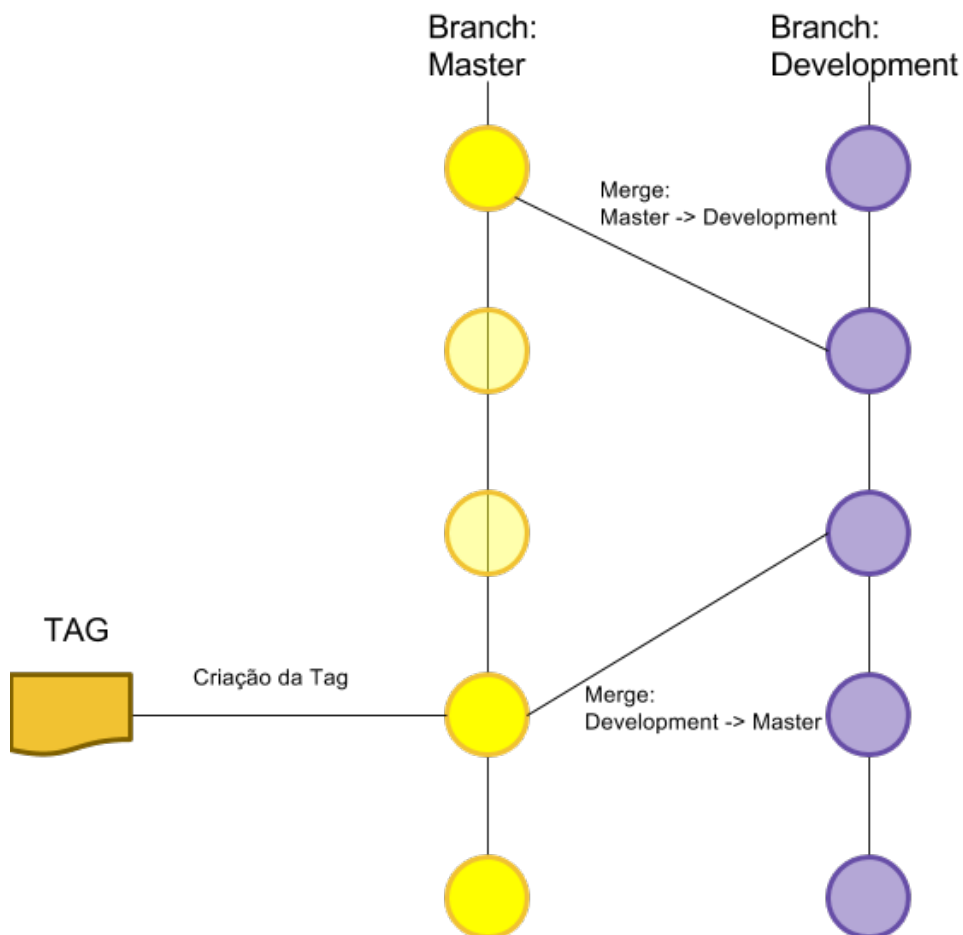
O Git auxilia o desenvolvedor a trabalhar de forma organizada as versões do seu código, podendo alterar de esquema de arquivos para o outro, que são chamados de *Branch*. Um repositório Git é constituído de várias *branches*, cada uma contendo uma versão do código.

A Figura 2 representa como é constituído o repositório do Kouri, onde existem duas *branches*: *master*, *development*. Quando finalizado um pacote de envio para produção, o processo é passar todo código existente na *branch* de *development* que é a *branch* na qual realizamos o desenvolvimento do *sprint* para a *branch master* que fica responsável por ter o código estável do jogo tendo o papel de *branch*

principal no repositório, esse processo e migração de código é chamado de *Merge*. No momento do *merge*, o Git avalia os arquivos alterados e quais pontos existem mudanças, o mesmo no primeiro momento tenta mesclas os dois códigos sem que exista conflito, caso isso não seja possível a ferramenta alerta ao usuário a necessidade de uma resolução manual do conflito.

Com o código final na *branch* principal atualizado, é interessante gerar uma Tag daquela versão. A Tag como próprio nome já diz, é uma Etiqueta da versão atual, quando gerada ela vai guardar toda a estrutura dos arquivos e o estado dos mesmo daquele momento, possibilitando e auxiliando assim, a alteração e correção de alguma versão específica que for para produção.

Figura 2- Exemplo de Repositório de Git





A última etapa antes de se iniciar o desenvolvimento, é definir a organização das tarefas que precisam ser realizadas: desenvolvimento de mecanismos, arte, som, e outras características do jogo. Para auxiliar nesse processo, é interessante utilizar o GDD (Game Design Document).

Como Rogrigo L. Motta explica em seu artigo, o GDD é uma ferramenta onde de forma textual é descrito as características do jogo, desde as características básicas do jogo como desenvolvimento de mecanismos, ações importantes dentro do jogo, até criação de cenários, e pontos mais fundos nas características do jogo.

Esse documento é a premissa de todo enredo e roteiro do jogo, nele deve estar contido todas características do mesmo, para ser um ponto de referencia na equipa na etapa de desenvolvimento.

O GDD costuma ser aplicado a desenvolvimento de jogos de grande porte, que torna a existência do mesmo obrigatória para que a organização do projeto não exista falhas. Porém, quando o jogo que está em pauta é um jogo de porte pequeno, normalmente feito por uma pequena equipe ou até mesmo por uma única pessoa, o contexto de uma necessidade de um documento tão específico se perde, e para esses casos existe a alternativa de um realizar um SGDD (Short Game Design Document). O Kouri se baseou na premissa desse estilo de documento para fazer o seu GDD com o template de 10 páginas.

No seu livro *Level Up!* Scott Rogers a necessidade da comunidade de desenvolvimento em geral tornar o GDD um documento mais enxuto, onde com poucas e precisas informações consigam demonstrar como completo a ideia do jogo. Rogers também descreve bem três formas de ser fazer o game design document de seu jogo, sendo eles: One-Page SGDD, Ten-Pages SGDD e o GDD.

O One-Page SGDD tenta demonstrar a ideia do jogo como próprio nome já diz em uma única página, contendo informações sobre os aspectos levantados para a elaboração da ideia inicial, algumas como análise de mercado, plataformas atingidas e algumas prototipação do personagem e ambiente onde ele se encontra. Já o Ten-Page SGDD vem com a premissa de detalhar vários aspectos importantes do jogo, sendo assim um documento um pouco mais amplo que o One-Page porém não chega a ser um GDD completo. Nele já é detalhado alguns aspectos do jogo como jogabilidade, características de cenário mais aprofundadas, interações entre o personagem com o jogo com um maior detalhamento entre outras. No Kouri, foi utilizado esse tipo de documento pois era necessário um bom detalhamento das

características do personagem e cenário porem por ser um jogo de pequeno porte, não há necessidade de um GDD completo.

### **3. PROJETO DE JOGO PARA ANDROID**

#### **3.1. INTRODUÇÃO A TEN-PAGE SGDD**

Como citado no capítulo anterior, o para o desenvolvimento da documentação de design do projeto, foi utilizada o template Ten-Page SGDD, descrito por Rogers.

Neste documento a principio é detalhado as informações gerais do jogo seguido de um contorno pela premissa do mesmo com sua história e fluxo. O personagem também é detalhado nesse documento, descrevendo suas ações e seu papel no jogo, dando assim informações que serão utilizadas no próximo passo que é a descrição da jogabilidade. Também é detalhado o mundo onde o jogo se passa, informando aspectos do mesmo e o que ele impacta para o jogo e na experiência do usuário, que também ganha uma parte nesse documento onde é descrito aonde o jogo deve atingir o usuário e com qual propósito, e como isso deve ser alcançado.

A mecânica do jogo também deve ser muito bem descrita, pois ela que se encarrega de especificar aos desenvolvedores como o jogo irá funcionar, quais obstáculos, dificuldades, recompensas que o jogador vai receber, e o que ele tem que fazer para que as mesmas aconteçam. Também é necessário descrever se no jogo vão existir inimigos ou não, não sendo necessariamente um inimigo tradicional, podendo tomar o papel de um obstáculo específico no jogo entre outras características apresentadas no projeto. Para alguns projetos que tenham um apelo muito grande na história, é interessante também descrever as "Cutscenes" que são alguns fragmentos das histórias contadas pelo jogo para introduzir o usuário aquela realidade na qual o jogo se passa. Por ultimo tempos o detalhamento de itens bônus, que são aspectos exclusivos do projeto que traga algum resultado positivo para o mesmo seja na forma de auxílio no desenvolvimento ou para a adição de uma característica peculiar no jogo.

#### **3.2. DETALHES DO PROJETO**

O nome escolhido para o jogo foi Kouri, do crioulo-haitiano "correndo", é um jogo casual do tipo Endless/Infinity Runner distribuido para Android. Possuindo classificação livre, o objetivo é atingir um publico de homens e mulheres na faixa de

8 a 16 anos, podendo também atingir o público acima desse alvo por tratar-se de um jogo casual. Possuindo hoje apenas o modo de apenas um jogador a data prevista para seu lançamento é 26 de outubro de 2015, entrando no mercado já com outros jogos como Subway Surf e Zombie Tsunami como concorrentes.

### **3.3. CONTORNO DO JOGO**

#### **3.3.1. OBJETIVO**

O jogo começa com o personagem sendo colocado no cenário que já está em movimento porém a contagem de pontos para, o mesmo já inserido no contexto, dá ao jogador o papel de desviar dos obstáculos possibilitando assim a continuidade no jogo e aumento na pontuação, que é baseada na distância percorrida somado as energias que aparecem aleatoriamente caso o personagem entre em contato.

#### **3.3.2. HISTÓRIA DO JOGO**

Tendo uma metrópole de aparência sombria com cores escuras, em Kouri o jogador assume o papel de uma esfera de energia que tem como objetivo continuar estável e em movimento, para isso ela precisa seguir em frente sem parar nos obstáculos, que são desintegrados quando em contato com o jogador porém a colisão tem um preço, que é reduzir a velocidade de movimento até a mesma ficar fraca a ponto de se desintegrar. Quanto maior a sua distância maior a velocidade, que também pode ser aumentada coletando alguns fragmentos de energia distribuídos aleatoriamente pelo cenário. A história do jogo foi criada para ser uma forma abstrata de demonstrar o comportamento social de superar obstáculos e barreiras encontrados no dia-a-dia.

#### **3.3.3. FLUXO DO JOGO**

Após iniciado o personagem já está em movimento e pontuando, a cada multiplicador da sua pontuação, a velocidade do personagem aumenta o que vai tornando maior a quantidade de obstáculos e fragmentos de energia.

Os obstáculos podem ser encarados de 2 maneiras, ou o jogador realiza uma ação de pulo para se esquivar ou ele pode entrar em contato, caso o contato for a opção, o obstáculo é desintegrado e o jogador sofre uma redução na movimentação, o que em excesso, também leva o mesmo a derrota.

Fragmentos de energia servem para aumentar sua pontuação rapidamente, sendo inseridos no cenário aleatoriamente conforme a velocidade do personagem.

### **3.4. PERSONAGEM**

Basicamente as ações do personagem se resumem a ele pular e colidir. Para realizar o pulo existe um botão no canto inferior direito da tela que quando pressionado dispara a ação para o personagem realizar o salto. A colisão acontece quando o personagem se esbarra com um obstáculo, que por sua vez é destruído e diminui a velocidade do jogador.

O personagem possui um formato esférico e cintilante, representando uma espécie de energia, que em movimento acaba deixando um pequeno rastro da emissão de suas partículas. A necessidade um personagem inanimado é devido ao tema, pois o jogo não tem a necessidade de criar um vínculo entre um personagem específico a mecânica do mesmo.

O personagem passou por algumas melhorias e alterações, tais como deixar o estilo de arte do mesmo mais "retro" utilizando de grandes pixels, que acaba entrando no estilo de arte do jogo que tem como premissa utilizar de objetos sem detalhamento baseados em figuras geométricas.

### **3.5. GAMEPLAY**

Kouri é um Endless-Runner e não foge muito da mecânica básica desse estilo de jogo. O jogador tem controle de um personagem que realiza saltos para se esquivar de obstáculos gerados aleatoriamente pelo cenário baseado na velocidade de movimentação. Porém foi implementado também algumas mecânicas diferenciadas de um runner convencional, tal como a possibilidade de colisão com o obstáculo resultar em destruição do mesmo e redução de movimento, o que em outros jogos do mesmo tema resultaria na impossibilidade do personagem seguir em frente sobrando apenas o pulo como ação.

Existem também os fragmentos de energia que são considerados "Power Ups" para o personagem, o que torna o mesmo mais rápido, pois são adicionados a sua pontuação que é o reflexo de da sua velocidade de movimentação.

Caso o personagem caia em alguma fenda, ou seja, empurrado para fora da tela devido as colisões em excesso com os obstáculo, o mesmo é desintegrado

resultado no final da partida. Sua pontuação é salva, realizando o controle da maior pontuação feita pelo jogador.

### **3.6 – GAME WORLD**

Para a realização do cenário, foi levado em consideração o estilo personagem, pois como o mesmo é uma esfera de energia que resulta para o usuário algo neutro e a ideia era criar um contraste contrario, com um ambiente mais escuro com tres camadas de profundidade. A necessidade dessa inversão de contraste é para demonstrar que o personagem não pertence aquele ambiente e precisa escapar, o que encaixa com a premissa do jogo de ser um runner onde o papel do personagem é fugir ou correr de ou para alguma coisa, neste caso, o ambiente.

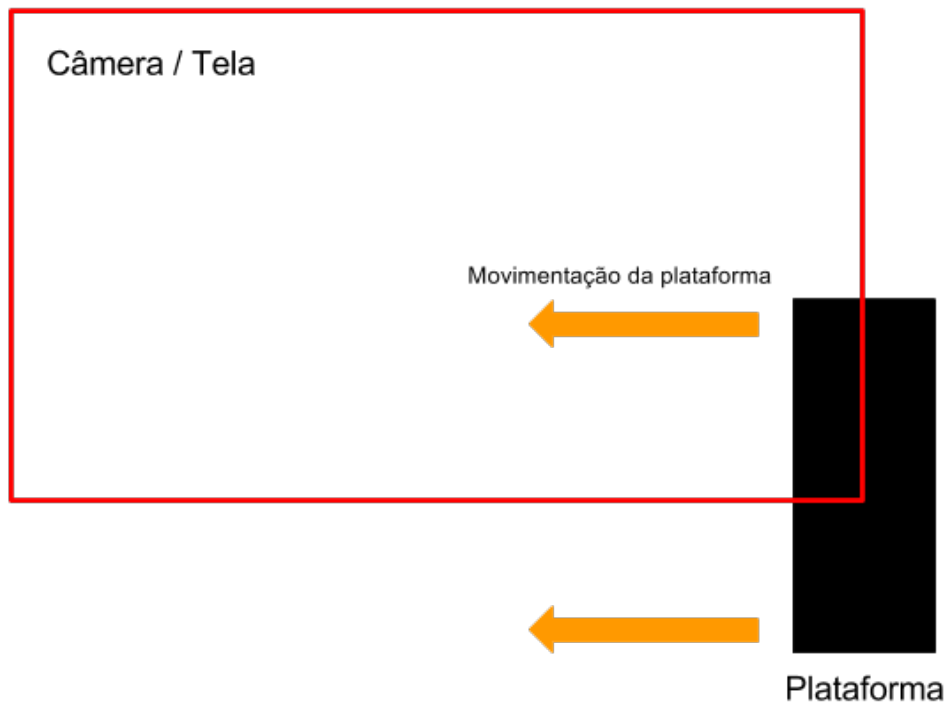
### **3.7. GAME EXPERIENCE**

O jogo tem uma aparência voltada para emissão de energia em um ambiente escuro, esse contraste de efeitos é uma forma de impor a necessidade do jogador de separa-los, assim como outros jogos utilizam-se de outros elementos, como, por exemplo, um inimigo perseguindo o personagem, ou um objetivo que se enquadra com as características do personagem. Alem disso o jogo conta com o sistema de pontuação e recordes, o que torna uma partida em um desafio para o jogador, que acaba dando o foco para o mesmo para tentar se superar.

### **3.8. MECÂNICAS DE JOGO**

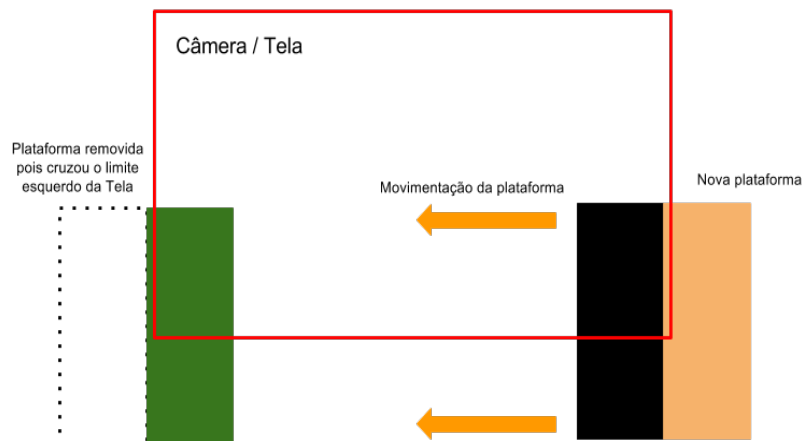
A mecânica principal do Kouri esta em gerenciar as plataformas a serem chamadas pelo cenário. Como padrão, a plataforma tem um tamanho fixo e movimentação da direita para a esquerda da visão do jogador conforme é demonstrado na Figura (3).

Figura 3- Exemplo de Movimentação de Plataforma



Quando a plataforma cruza por completo o limite direito da câmera é adicionado na lista uma nova plataforma que se inicia posicionada no final da plataforma anterior sem destruí-la. Quando qualquer uma das plataformas existentes cruzarem o limite esquerdo da câmera, ou seja, sair da visão do jogador, a mesma é removida da lista com intuito de economizar memória. A Figura (4) demonstra esse cenário:

Figura 4 - Fluxograma do Funcionamento das Plataformas



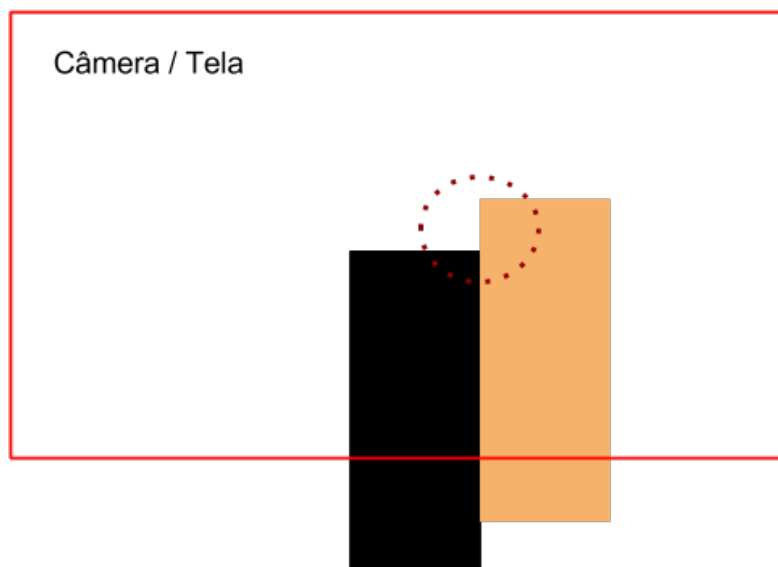
Quando inserido uma nova plataforma a mesma pode vincular um obstáculo ou um fragmento de energia a ela, sendo que a chance disto ocorrer, é baseada na propriedade definida dentro do componente.

A velocidade de movimento das plataformas é baseada na pontuação atual do jogador, quanto maior a pontuação, a movimentação das plataformas é aumentada resultando em uma maior velocidade de criação de novas plataformas que por sua vez aumenta as chances de existirem obstáculos ou power-ups. Quando o jogador colide com um obstáculo a velocidade é reduzida porem a pontuação não, e o objeto no qual ele colidiu é destruído.

### 3.9 - INIMIGOS

Em Kouri, o único inimigo do jogador são os obstáculos. Existe dois tipos de obstáculos no jogo, um deles são as elevações (Figura 5) que podem ser evitadas caso o personagem salte por cima, ou destruídas caso colida.

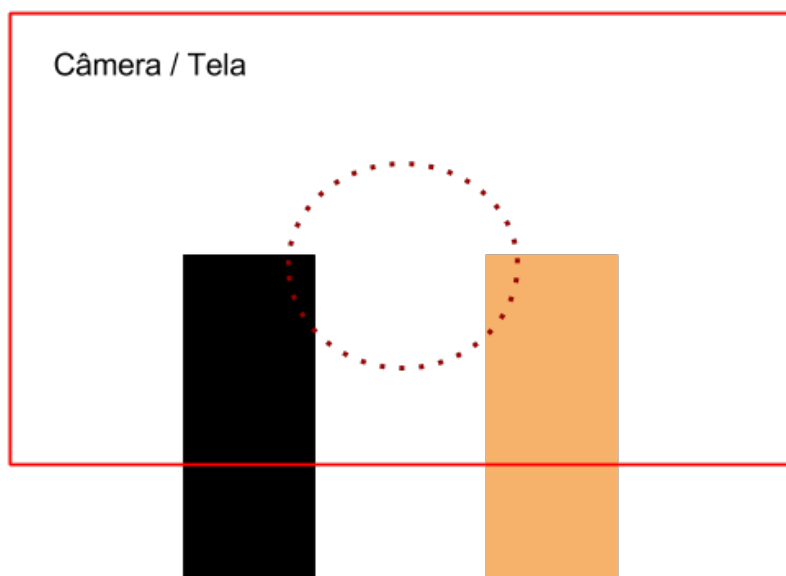
*Figura 5 - Demonstração dos Obstáculos Elevados*



Outra forma de obstáculo são as fendas (Figura 6), elas são espaços entre duas plataformas que pode ocasionar a queda do personagem entre elas, e por sua vez o fim da partida.



Figura 6 - Demonstração dos Obstáculos Fendas



### 3.10. CUTSCENES

Não haverá cutscenes em Kouri, pois como se trata de uma história com o contexto abstrato e aberto, o jogo não deve interferir na concepção que o jogador vai ter.

### 3.11. BÔNUS MATERIAL

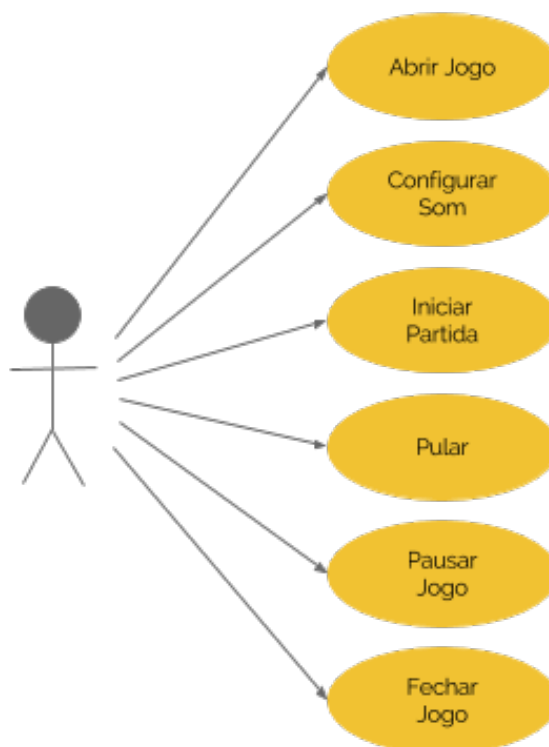
Em jogo, os maiores Bônus Material é o tratamento da arte com effects dinâmicos, dando uma aparência de luz de neon para o jogo criando um contraste com o ambiente e com o jogador. Com foco efeitos encima de um design minimalista com uma paleta de cores curta, realizando dessa forma a substituição de uma arte conceitual com muitos detalhes e optando pelo efeito do ambiente.

## 4. DESENVOLVIMENTO DO JOGO KOURI

### 4.1- DIAGRAMA DE CASO DE USO

No diagrama de caso de Uso identificamos os atores e suas interações com o a aplicação, detalhando assim as funções do sistema. Dentro desse contexto Larman (Larman, 2000) propõe utilizar o caso de uso essencial, que consiste em detalhar cada caso de uso além da demonstração do mesmo no diagrama. Na Figura 7 temos a o diagrama de caso de uso do Kouri, onde possui temos apenas 1 ator que é o usuário e 6 interações: Abrir Jogo, Configurar Som, Iniciar Partida, Pular, Pausar Jogo, Fechar Jogo.

Figura 7 - Diagrama de Caso de Uso



O Caso de Uso representa a interação do usuário com o sistema, no caso do jogo Kouri, onde o usuário realiza as ações do personagem através de interações com a interface do jogo. Abaixo segue os casos de uso essenciais, que são o detalhamento de cada ação.

Tabela 1 - Caso de Uso "Abrir Jogo"

Nome do caso de Uso	Abrir Jogo
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas para o jogador abrir o jogo
Pré-condições	Não existir nenhuma instancia do jogo aberta
Pós-condições	O jogo estará aberto
Ações do Ator	Ações do Sistema
1. O jogador clica no icone do jogo para o mesmo iniciar sua execução	2. O jogo vai carregar os seus recursos enquanto mostra ícones do fabricante e quando finalizado vai para a tela de inicial do jogo

Tabela 2 - Caso de Uso "Configurar Som"

Nome do caso de Uso	Configurar Som
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas para que o jogador configure o Som do jogo
Pré-condições	O jogo estar aberto na tela inicial ou na tela de pause
Pós-condições	O Som do jogo alterada entre ligado e desligado
Ações do Ator	Ações do Sistema

Tabela 3 - Caso de Uso "Iniciar Partida"

Nome do caso de Uso	Iniciar Partida
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas para iniciar uma partida
Pré-condições	O jogo aberto, e deve estar na tela inicial ou na tela de "Game Over"
Pós-condições	A partida ira começar, dando inicio a contagem
Ações do Ator	Ações do Sistema
1. O jogador clica no ícone de "Play" no caso da tela inicial, ou da dois cliques na tela no caso da tela de "Game Over"	2. O jogo abre a tela da partida já com o personagem e cenário em movimentos e a pontuação começa a contar.

Tabela 4 - Caso de Uso "Pular"

Nome do caso de Uso	Pular
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas para o jogador conseguir realizar o pulo do personagem
Pré-condições	O jogo aberto, e com a partida iniciada e o personagem deve estar no chão.
Pós-condições	O jogador ira realizar o salto
Ações do Ator	Ações do Sistema
2. O jogador clica no icone em formato de seta no canto direito inferior para realizar a ação de saltar.	2. O jogo calcula se o personagem pode saltar naquele momento, se sim ele realiza a ação.

Tabela 5 - Caso de Uso "Pausar Jogo"

Nome do caso de Uso	Pausar Jogo
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas para pausar o jogo
Pré-condições	O jogo aberto, e com a partida em andamento.
Pós-condições	O jogo entrara no estado de pausado
Ações do Ator	Ações do Sistema
3. O jogador clica no icone de pause no canto superior esquerdo.	2. O jogo realiza o pausamento da partida e abre a tela de pause, onde ele pode voltar a partida, configurar o som ou voltar para tela inicial.

Tabela 6 - Caso de Uso "Fechar Jogo"

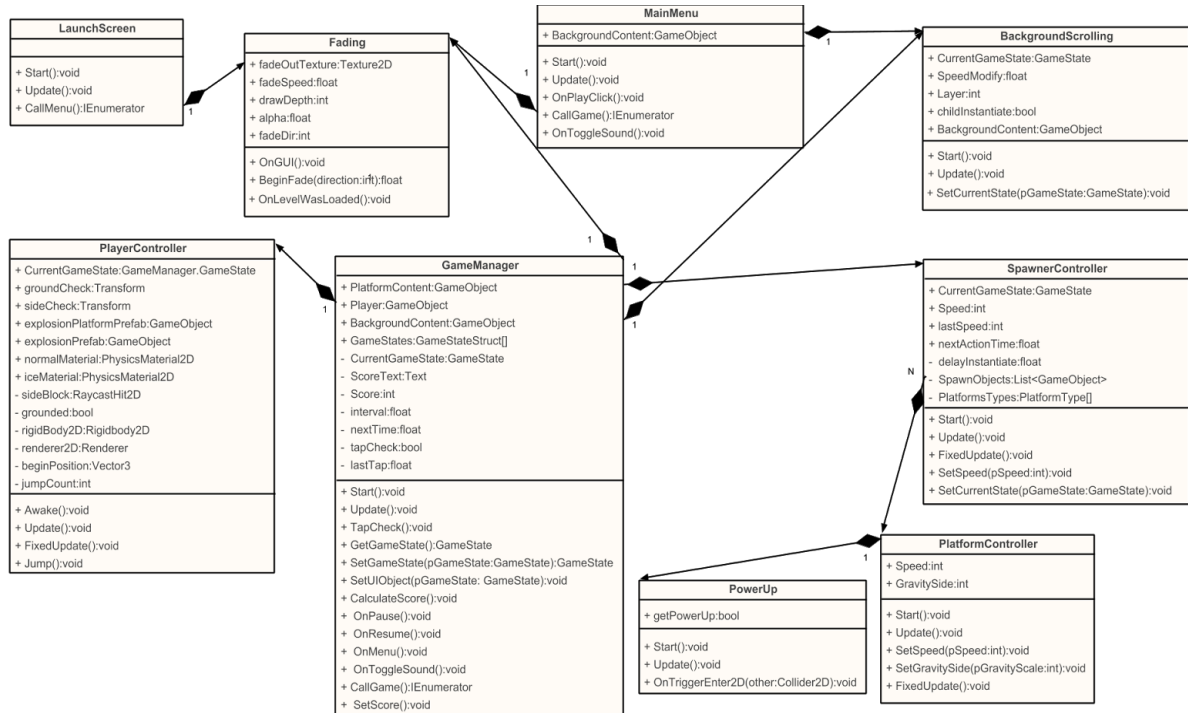
Nome do caso de Uso	Fechar Jogo
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas para o jogador fechar o jogo
Pré-condições	O jogo aberto e estar na tela inicial
Pós-condições	O jogo será fechado
Ações do Ator	Ações do Sistema
4. O jogador clica no botão de Voltar nativo do aparelho	2. O jogo finaliza seus processos e é fechado.

## 4.2. DIAGRAMA DE CLASSE

Segundo os autores BOOCH, RUMBAUGH e JACOBSON (2005, p. 108) "Um diagrama de classe é um diagrama que mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos."

Segue a figura do diagrama de classe proposto para este projeto:

Figura 8 - Diagrama de Classe



#### 4.2.1. CLASSE LAUNCHER SCREEN

Esta é a primeira classe instanciada do jogo, é responsável por exibir a tela com o logo da empresa, ela também já carrega a classe Fading que vai ser utilizada para as mudanças de telas ao longo do jogo. Após o delay de três segundos a tela realiza a chamada *CallMenu()* tem o papel de instanciar e exibir a tela de menu principal.

#### 4.2.2. CLASSE MAIN MENU

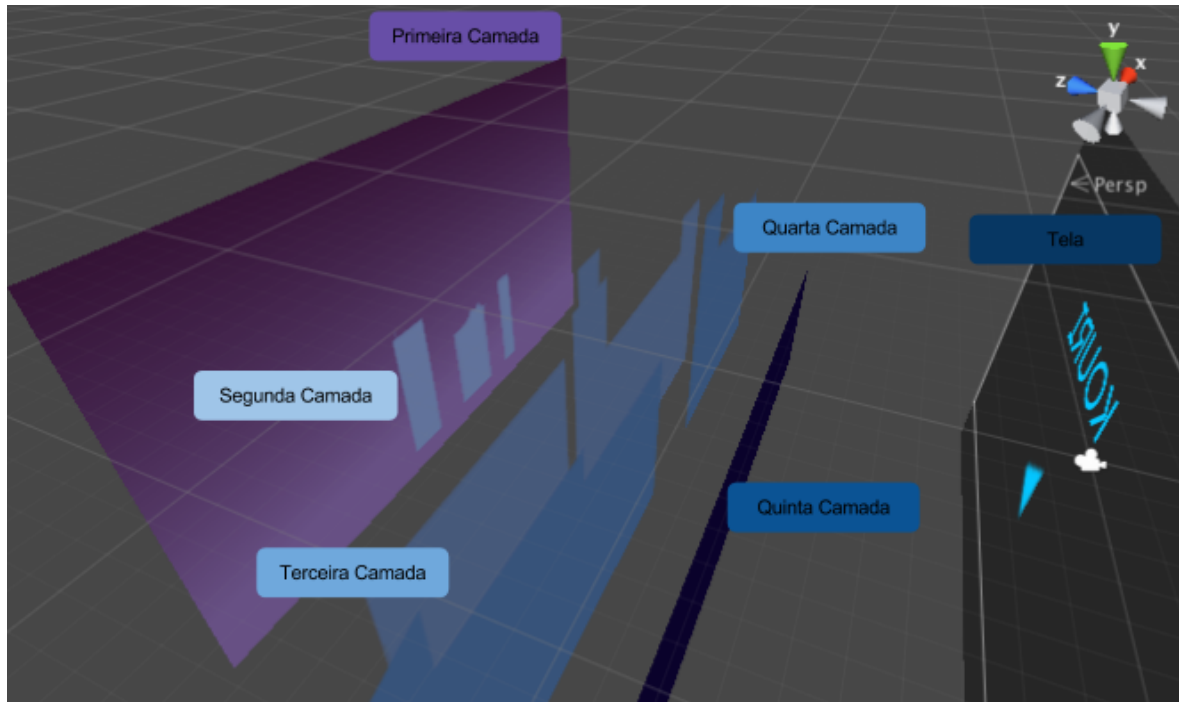
É responsável por exibir o menu principal do jogo, onde você pode alterar o estado do som com o metodo *OnToggleSound()* ou iniciar a partida com o *CallGame()*. Essa classe também tem o papel de instanciar o objeto de BackgroundScrolling, que é responsável por toda arte de fundo do jogo, tanto no menu principal quanto na partida.

#### 4.2.3. CLASSE BACKGROUND SCROLLING

Tem o papel de gerenciar o plano de fundo do jogo, criando as camadas de fundo e as movimentando em parallax , se baseando na velocidade atual do personagem no caso da tela da partida ou com um valor fixo como é o caso do

menu inicial. Para realizar o efeito de camadas utilizamos o eixo Z do espaço, pois como em jogos 2D utilizamos apenas os eixos X e Y para orientação da tela, o eixo Z fica com o papel de dividir a profundidade dos objetos, conforme o exemplo da imagem da figura abaixo:

Figura 9 - Sistemas de Camadas Utilizando Eixo Z



Conforme a camada se distancia da tela a movimentação da mesma diminui, aplicando assim modificadores de movimento a cada camada em referencia ao jogo, a imagem a seguir demonstra o exemplo da segunda camada, onde definimos na propriedade "Speed Modify" que a mesma vai se movimentar em um décimo da velocidade do jogo:

Figura 10 – Componentes de Scroll do Background



#### 4.2.4. CLASSE GAMEMANAGER

A classe `GameManager` como o próprio nome já diz, é responsável por gerenciar todos aspectos do gameplay: Velocidade, Score, Estado da partida, checagem dos toques na tela, som e etc...

Para gerar as plataformas da partida, essa classe gera uma instancia da `SpawnerController` e envia parâmetros através dos métodos `setCurrentState()` e `setSpeed()`.

O método `setCurrentState()` recebe como parâmetro uma propriedade enum de `GameState` e a envia para a `SpawnerController`, a mesma utiliza esse estado para controlar a movimentação das plataformas, bem como o `setSpeed()` que recebe um numero inteiro, altera a velocidade das mesmas.

Como utilizado na `MainMenu`, a `GameManager` utiliza também a instância do `BackgroundScrolling`, porem nesse caso a velocidade do plano de fundo é baseado em uma propriedade da existente na `SpawnerController` que pode ser obtida na `GameManager`, o *speed*.

Essa propriedade é principal responsável pela dinâmica de dificuldade e recompensa do jogo. Seu valor é iniciado em um, e vai sendo acrescentado mais um para cada mil pontos adquiridos pelo jogador. O aumento dessa propriedade resulta em uma velocidade ampliada do personagem, background e dos obstáculos que aparecem em cena. O jogador tem duas maneiras de aumentar seus pontos: com a sua distancia percorrida e coletando `PowerUps`.

O aumento do *score* pela distancia percorrida é baseado no calculo feito dentro do método `CalculateScore()`, que consiste em adicionar um ponto no score a cada um cem avos de segundo enquanto a partida estiver ativa. Outra forma de aumentar o score é coletar os `PowerUps`, que no caso do Kouri são fragmentos de energia que quando se entra em contato com o personagem são destruídos e somam cem ao score.

A `GameManager` também controla o pause do jogo, onde ele interrompe momentaneamente as ações de todos objetos do jogo através de uma propriedade geral da unity chamada *timeScale* que fica nas configurações do projeto como segue na Figura 11. O jogo em estado de pause exibe também uma tela onde o jogador pode optar por: voltar a partida, voltar ao menu principal e alternar o som entre desligado e ligado conforme segue na Figura 12.



Figura 11 - Propriedade Time das Configurações Gerais do Projeto

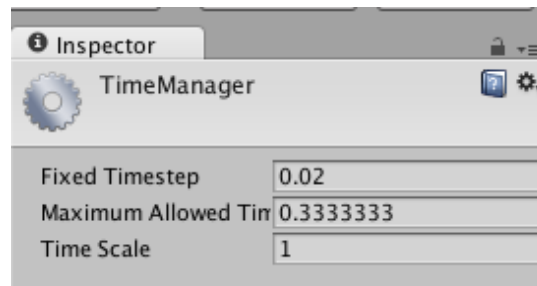
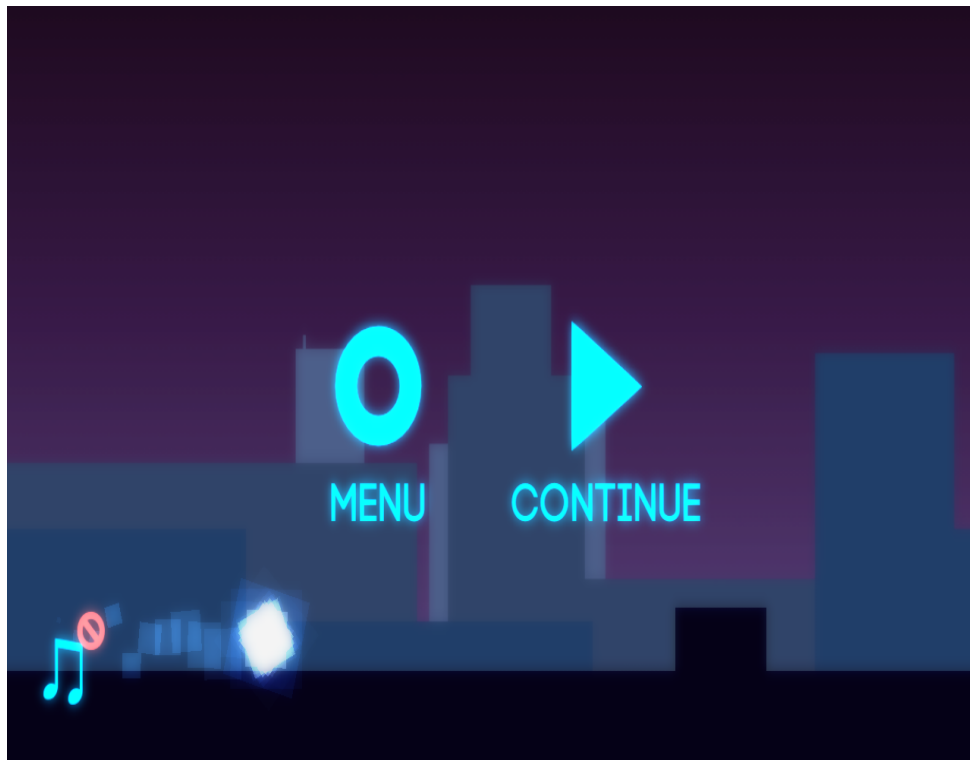


Figura 12 - Apresentação de Tela de Pause



A entrada de dados através de toques na tela também é gerenciada na GameManager, que utilizando o método TapCheck() verifica se a tela foi ou não tocada e quantas vezes, para o caso de "Double Tap".

A checagem de toque também se encarrega de verificar aonde o usuário tocou para determinar qual evento chamar, como, por exemplo, o pulo do personagem. No canto inferior direito existe um ícone direcional que representa a ação pulo, quando acionado ele chama o método TapCheck() que verifica qual ação deve ser tomada para aquele caso, neste exemplo, ele vai até a instância da classe

Player que esta no GameManager e chama o metodo Jump() que faz o personagem realizar o salto.

#### **4.2.5. CLASSE PLAYER CONTROLLER**

A Unity possui um mecanismo de fisica baseado em Box2D onde existe quatro tipos de componentes de colisao 2D que podem ser aplicados a um GameObject: BoxCollider2D, CircleCollider2D, EdgeCollider2D e PolygonCollider2D. Cada um deles interpreta a colisao entre objetos de formas diferentes, no caso do personagem, foi utilizado um CircleCollider2D, que era o mais adequado, devido ao formato do mesmo. Já nas plataformas é aplicado o BoxCollider2D já que por ter o mesmo formato, consegue alcançar todos limites do objeto.

No personagem, alem de aplicar o componente de colisao também é utilizado o Rigidbody2D que transforma o objeto em um corpo físico e assim permitindo que o mesmo interaja com outras variáveis do espaço como gravidade, massa entre outras, e alem disso o possibilita de receber interações como a adição de força em uma determinada direção.

A PlayerController fica encarregada de gerenciar todas essas informações e utiliza o que é necessário. Para o Pulo do personagem é acionado o método Jump() que primeiramente verifica a atual situação do personagem: Se o mesmo está apto a pular ou não. Caso sim, utilizamos um método do Rigidbody2D chamado AddForce() que recebe como parâmetro um vetor bidimensional informando para qual direção em que a força deve ser aplicada, por exemplo para o pulo do personagem foi utilizado o vetor com as direções 0 e 800 para x e y respectivamente.

Alem de trabalhar com o Rigidbody, essa classe também gerencia o sistema de colisao entre objetos, onde utilizamos o método do mecanismo de física 2D da Unity, o Raycast() para identificarmos colisões entre dois ou mais objetos que contem algum componente de colisao. No caso do personagem, ele pode interagir com dois tipos de objetos: o chão, os obstáculos.

A colisao com o chão serve para habilitar ou não o pulo para o personagem sendo que o mesmo só pode realizar o salto quando estiver em contato, já a colisao obstáculos é detectada quando o personagem colide sua lateral, implicando na destruicao do obstaculo que sera removido do cenário e dando espaço para a animação de destruicao, conforme exemplo da figura abaixo:

Figura 13 - Demonstração do Obstáculo sendo Destruído



Outra consequência da colisão com os obstáculos, é a redução repentina de velocidade do personagem que e arrastado um pouco para trás. Quando fora de sua posição padrão é acionado dentro do método `FixedUpdate()` que aos poucos aplica força horizontal ao personagem para que o mesmo volte para sua posição e velocidade.

#### 4.2.6. CLASSE SPAWNERCONTROLLER

Essa classe fica responsável por gerenciar as instancias da classe `PlatformController`, enviando informações para as mesmas através dos métodos `SetSpeed()` e `SetCurrentState()`. Ela realiza a ponte entre a `GameManager` e todas as plataformas instanciadas em cena, como por exemplo a mudança de velocidade pelo método `SetSpeed()` que é chamado pelo `GameManager` e em sua lógica percorre todas instancias das plataformas e altera a velocidade das mesmas. Quando já instanciado a plataforma, a `SpawnerController` verifica se o mesmo vai conter um `PowerUp` com uma chance de 10% para que apareça.

#### 4.2.7. CLASSE PLATFORMCONTROLLER

A `PlatformController` realiza apenas a movimentação da plataforma, baseando-se nas informações enviadas pela `SpawnerController` como a velocidade de movimento e o estado atual do jogo.

#### 4.2.7. CLASSE POWERUP

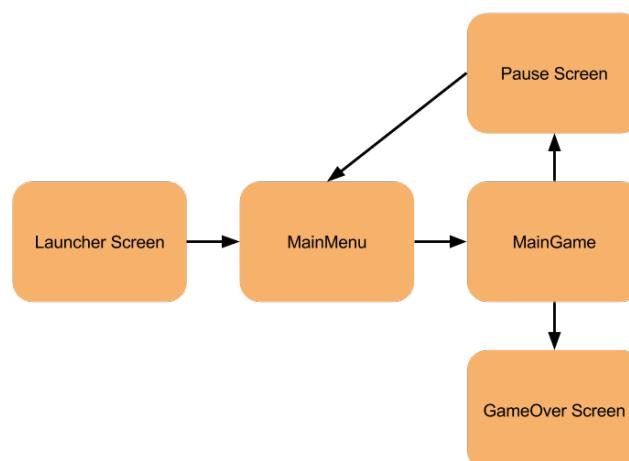
Esta classe gerencia os objetos de PowerUp que no contexto do projeto são representados por fragmentos de energia. Quando o personagem entrar em contato ativa o método `OnTriggerEnter2D()` que é responsável pela colisão entre esses objetos, e quando acionado ele executa sua rotina que consiste em iniciar a animação do PowerUp e somar cem pontos ao score da partida.

#### 4.3. FLUXO DE TELAS

O fluxo de telas serve para demonstrar os estados do jogo e quais caminhos que podem ser tomados entre eles. Quando iniciado a primeira tela apresentada é o logo da desenvolvedora que após o carregamento dos recursos necessários para a execução automaticamente faz a chamada do menu. No menu principal, o jogador tem dois caminhos: Iniciar a partida ou sair do jogo.

A próxima tela é a `MainGame`, onde todo jogo ocorre tendo sub-caminhos a tela de pause ou caso perca a partida a tela de game-over. Na tela de pause o jogador pode optar por voltar a partida ou ir ao menu inicial. Quando o jogador acaba perdendo a partida lhe é apresentada a tela de game-over que permite iniciar outra partida. Segue abaixo o fluxo de telas do projeto de forma grafica:

*Figura 14 - Diagrama do Fluxo de Sequência de Telas*








#### 4.4. ARTE E SOM

Para a arte do projeto foi utilizado o conceito de Sprite, que consiste em colocar a imagem em uma camada separada dos outros objetos do jogo, facilitando assim a criação das animações, pois como a imagem esta em um escopo separado do jogo o artista pode trabalhar com o foco nela sem precisar configurar o ambiente em paralelo.

No projeto foi utilizado Sprites somente para o cenário, que foram aplicadas ao sistema Parallax gerando a impressão de profundidade entre camadas, e como o projeto necessitava de uma aparência mais "Sombria" foi utilizado uma paleta de cores curta e levando para um contraste mais escuro, para assim entrar em contraposição com o personagem que é uma esfera de energia.

Figura 15 - Paleta de Cores usadas no Projeto

<b>Shadow Blue</b> by yroggregory					
					
0B032A	11,3,42	0	1	19	1
HEX	RGB	COMMENTS	FAVORITE	VIEWS	LOVE
<b>just above the roof</b> by trivenia					
					
37537A	55,83,122	0	0	9	0
HEX	RGB	COMMENTS	FAVORITES	VIEWS	LOVES
<b>scared</b> by Dell					
					
341235	52,18,53	1	2	22	2
HEX	RGB	COMMENT	FAVORITES	VIEWS	LOVES
<b>x</b> by eighteyed					
					
675084	103,80,132	0	0	0	0
HEX	RGB	COMMENTS	FAVORITES	VIEWS	LOVES
<b>smile for me</b> by luffy					
					
CDB5D9	205,181,217	0	0	17	0
HEX	RGB	COMMENTS	FAVORITES	VIEWS	LOVES

Para o personagem, foi utilizado um componente da Unity de sistema de partículas. Com ele é possível configurar o formato em que as partículas serão emitidas como também a cor, direção, tamanho entre outras propriedades. O personagem tem três componentes de partículas, o primeiro é em formato esférico e têm as direções estáticas deixando a emissão para fora do seu formato parado, o segundo tem o papel de criar o rastro deixado pelo personagem, tendo uma direção baseada no primeiro componente. O terceiro realiza o brilho emitido pelo personagem dando a impressão de a partícula esta instável.

A Unity possui sua própria loja de componentes chamada Asset Store, onde desenvolvedores podem distribuir suas diversas criações. Para a realização do som utilizamos dois pacotes disponibilizados na loja: Dark Future Music para a musica do jogo, e SciFi UI Sounds FX para os sons de efeitos dentro do jogo.

O pacote Dark Future Music distribuído, pela Affordable Audio 4 Everyone, contem musicas com uma ideia mais obscura aliado um tom futurístico que se encaixa na temática do jogo. Dos sons disponíveis, foram utilizados os seguintes:

*Tabela 7 - Tabela de Sons do Ambiente*

Nome do Som	Duração	Formato	Loop	Onde é Utilizado
FutureWorld_Dark_Loop_3	2 minutos e 23 segundos	ogg	Sim	Na tela inicial do Jogo
FutureWorld_Resolution_Loop	48 segundos	ogg	Sim	Durante a partida

Outro pacote de som que foi utilizado é o SciFi UI Sounds FX, distribuído pela Bright Shining Star, onde utilizamos sons para representar efeitos como o pulo do personagem, obstáculos destruídos e a partida finalizada. Segue abaixo as especificações dos sons utilizados:

Tabela 8 - Tabela de Sons Utilizados para Efeitos

Nome do Som	Duração	Formato	Loop	Aonde é Utilizado
Slider1	45 milésimos de segundo	ogg	Não	Pulo do personagem
Slider3	89 milésimos de segundo	ogg	Não	Morte do personagem
Message2	1 segundo e 52 milésimos	ogg	Não	Explosão do obstáculo

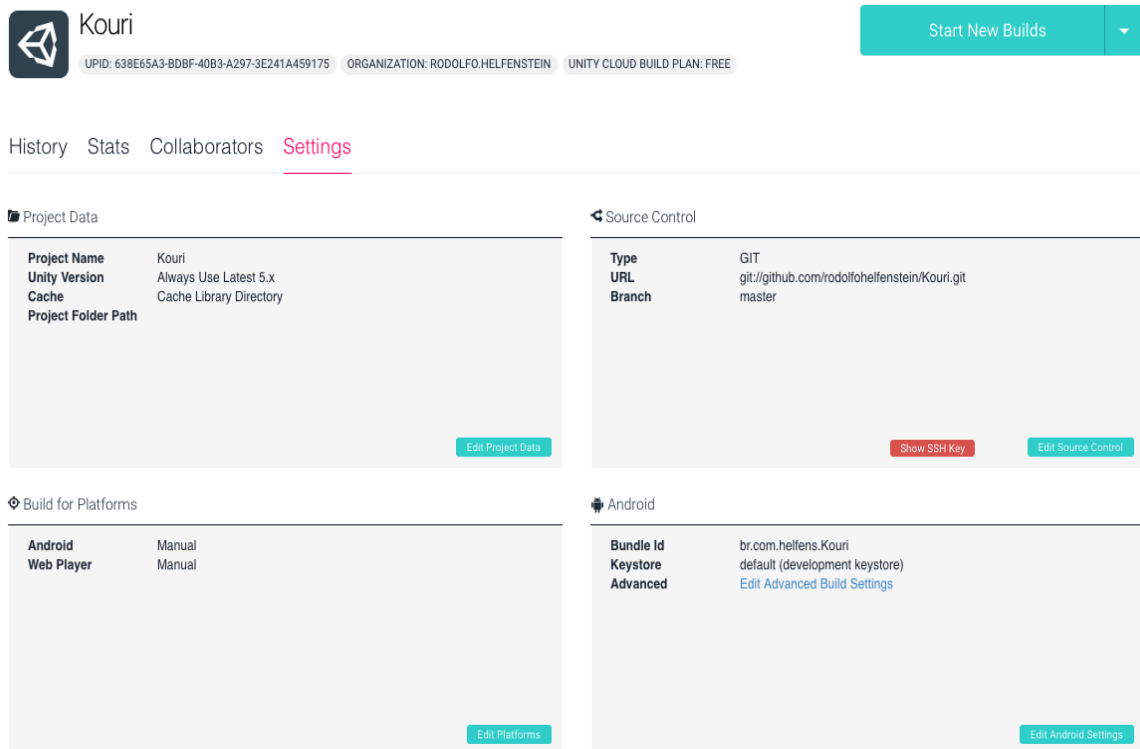
## 4.5. SERVIÇOS

A Unity disponibiliza para os desenvolvedores alguns serviços que podem desde auxiliar o desenvolvimento do projeto, como ferramentas que podem ser inseridas dentro do jogo de forma prática. Para este projeto, foi utilizado o Unity Cloud Building, Unity ADS e o Unity Analytics.

### 4.5.1. UNITY CLOUD BUILD

O Unity Cloud Build auxilia o desenvolvedor no processo de integrações contínuas, onde após configurado um projeto na ferramenta o usuário o vincula com o projeto em desenvolvimento utilizando algum servidor de versionamento e pode gerar versões da aplicação para Android, iOS e Desktop. No caso desse projeto, foi configurado um projeto na ferramenta e vinculado ao GitHub do projeto.

Figura 16 - Tela de Configurações do Unity Cloud Build



The screenshot shows the Unity Cloud Build settings page for a project named "Kouri". The page is divided into four main sections: Project Data, Source Control, Build for Platforms, and Android. At the top, there is a "Start New Builds" button. The navigation menu includes "History", "Stats", "Collaborators", and "Settings" (which is currently selected).

**Project Data**

Project Name	Kouri
Unity Version	Always Use Latest 5.x
Cache	Cache Library Directory
Project Folder Path	

[Edit Project Data](#)

**Source Control**

Type	GIT
URL	git://github.com/rodolfohelfenstein/Kouri.git
Branch	master

[Show SSH Key](#) [Edit Source Control](#)

**Build for Platforms**

Android	Manual
Web Player	Manual

[Edit Platforms](#)

**Android**

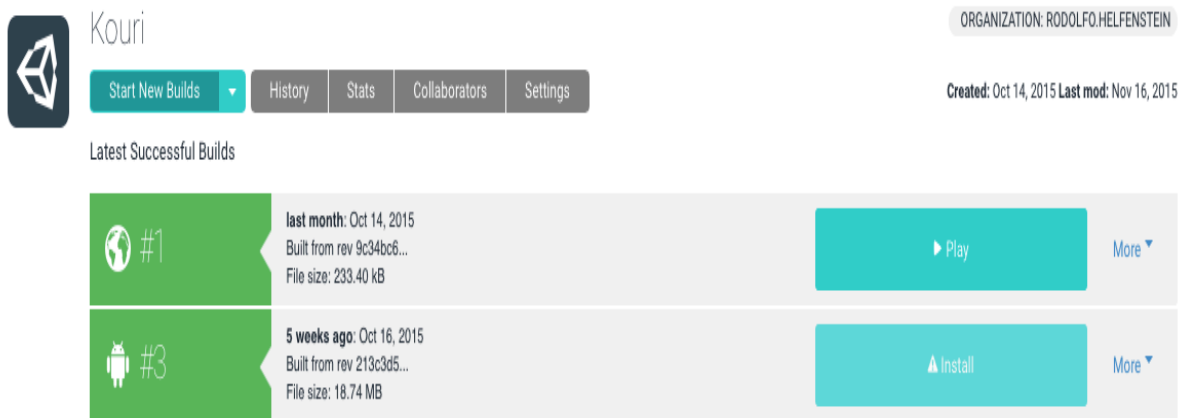
Bundle Id	br.com.helfens.Kouri
Keystore	default (development keystore)
Advanced	<a href="#">Edit Advanced Build Settings</a>

[Edit Android Settings](#)

Com o projeto configurado, através do Dashboard da ferramenta conseguimos iniciar novas builds que quando finalizadas, enviamos para os e-mails cadastrados no projeto a última versão do projeto que se encontra no GIT. Também conseguimos ter controle de quais builds tiveram sucesso e quais falhas, que são apontadas no log de erro gerado.



Figura 17 - Tela de Acompanhamento das Builds



#### 4.5.2. UNITY ANALITICS

O Unity Analytics serve para obter um relatório com as análises e informações do jogo tais como jogadores ativos, seções iniciadas, retenção do público, e outras variáveis na qual o desenvolvedor pode criar. No caso desse projeto, foi criada uma variável para obter a pontuação final do jogador quando a partida acaba, tendo controle assim das médias de pontuações feitas pelo jogadores o que pode representar algumas dificuldades ou facilidades encontradas por eles. Segue abaixo alguns exemplos das análises feitas pelo Unity Analytics

Figura 18 - Gráfico de Jogadores Ativos por Período

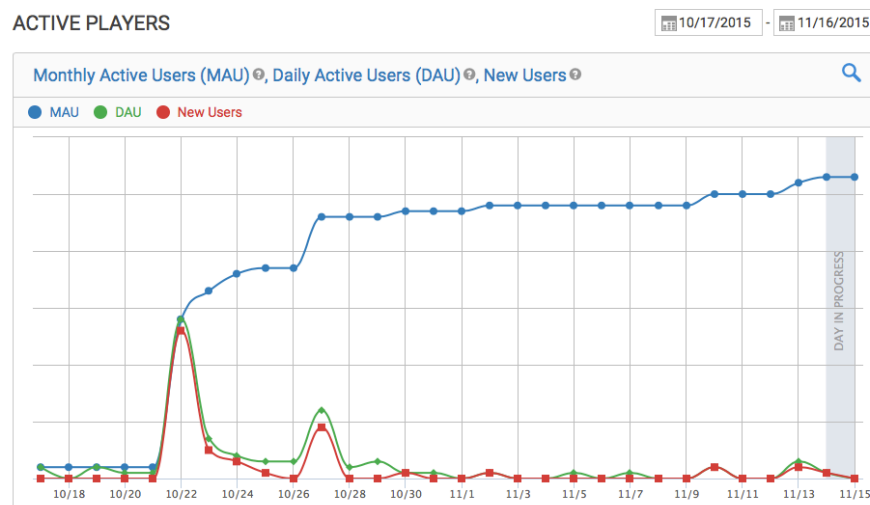
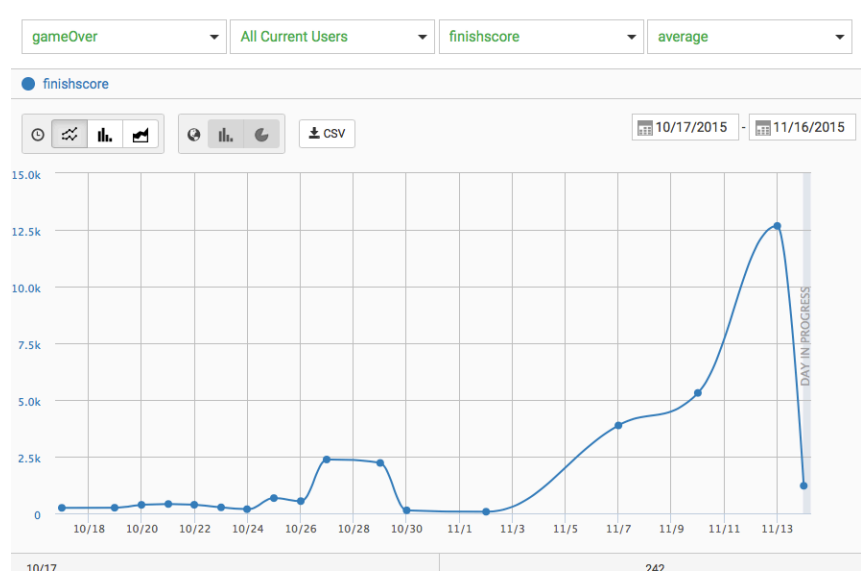


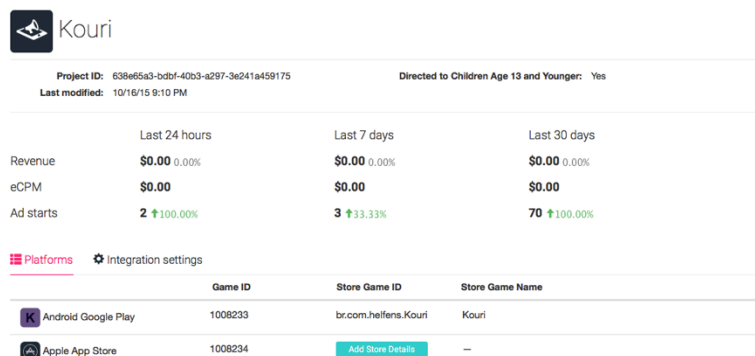
Figura 19 - Gráfico da Propriest Custom Game Over



### 4.5.3. UNITY ADS

Para jogos que são distribuídos gratuitamente, a propaganda é um meio de se obter lucros com a jogo, e o Unity ADS serve para gerenciar essas propagandas de forma pratica. Com o jogo configurado e vinculado ao Dashboard do serviço, basta chamar o método `Advertisement.Show()` aonde é necessário que aparece a propaganda, assim a Unity se encarrega de exibir essa propriedade em forma de imagem ou vídeo, o que pode ser configurado junto ao projeto. Segue na figura abaixo o Dashboard do Unity ADS do projeto, nele temos informações como o valor da receita que o jogo arrecadou a quantidade de propagandas iniciadas entre outras.

Figura 20 - Propriedades e Informações do Unity ADS



## 4.6. PUBLICAÇÃO

A Play Store permite que a distribuição dos aplicativos desenvolvidos seja feita de forma direta para usuários de Android. A Play Store é aberta para todos desenvolvedores que querem criar aplicações ou jogos para Android. Para a publicação de software é necessário que seja realizado o pagamento de uma taxa anual no valor de \$25,00 aceitando todos termos de distribuição e preparar seu seguindo a documentação de publicação (MARKET, 2011)

Com o cadastro feito o desenvolvedor pode controlar todos aspectos da sua conta bem como configurações dos jogos existentes na conta, como informações de download, classificação do publico, comentários entre outros. Segue abaixo algumas imagens com passos da publicação onde temos no primeiro exemplo a aplicação criada na conta de publicação, porem sem nenhuma configuração ou algum pacote gerado, e no segundo temos já o Dashboard da nossa aplicação com as configurações de imagens concluída e a o pacote do jogo já salvo e apenas esperando aprovação da Google para que entra no ar.

Figura 21 - Dashboard antes do Jogo ser Publicado

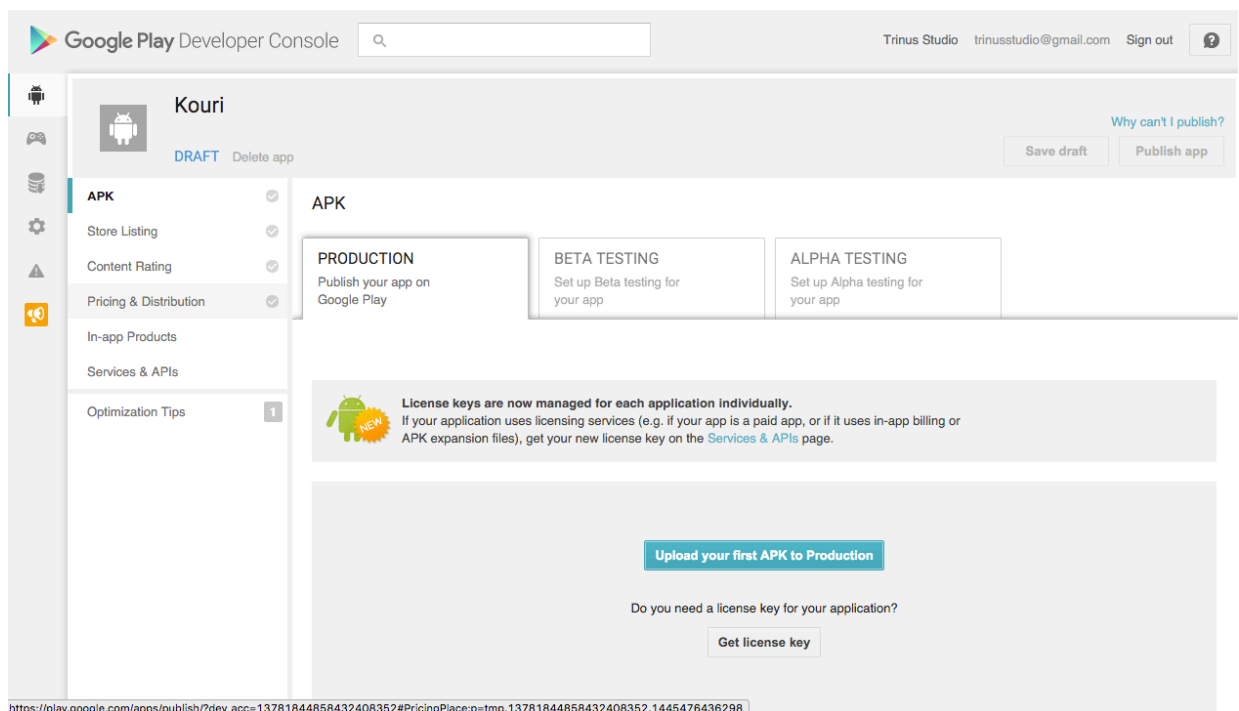


Figura 22 - Dashboard após Jogo ser Publicado

The screenshot shows the Google Play Developer Console interface for the app 'Kouri' (br.com.helfens.Kouri). The app is currently in a 'DRAFT' state. The dashboard is divided into several sections:

- APK Section:** Contains 'PRODUCTION', 'BETA TESTING', and 'ALPHA TESTING' options. The 'PRODUCTION' version is '1'. There are buttons for 'Save draft', 'Publish app', and 'Switch to advanced mode'.
- PRODUCTION CONFIGURATION:** Includes an 'Upload new APK to Production' button.
- CURRENT APK:** Shows the app was uploaded on Oct 21, 2015, at 6:37:05 PM.
- Supported devices:** 3450 devices. A link 'See list' is provided.
- Excluded devices:** 0 devices. A link 'Manage excluded devices' is provided.
- Table of Versions:**

VERSION	UPLOADED ON	STATUS	ACTIONS
1 (1.0)	Oct 21, 2015	Draft in Prod	

At the bottom of the dashboard, the text 'APP TRANSITION SERVICE' is visible.

#### 4.7. VERSÕES DISPONÍVEIS

Nos dias de hoje, existem sete versões do Android, e para ajudar a escolher em qual delas desenvolver, a Google disponibiliza em seu site oficial (<http://developer.android.com/about/dashboards/index.html>), um Dashboard com todas as versões e a percentagem que estão sendo utilizadas no mercado. Nesse gráfico é possível observar que a versão do Android que possui mais compatibilidade e estão em maior número no mercado é a 4.4 KitKat, que foi a versão escolhida para este projeto, sendo que o mesmo irá rodar em todas as versões do Android a partir da KitKat.

## 5. CONCLUSÃO

Este trabalho teve como intenção apresentar todos os passos para o desenvolvimento de um jogo. Com relação à ferramenta de desenvolvimento, a Unity disponibiliza ao desenvolvedor, vários componentes prontos e mecanismos que tomariam tempo do projeto para serem feitos, como por exemplo, o gerenciamento de telas e a parte da física dos objetos em jogo. Outro benefício da Unity, é a possibilidade de distribuição para várias plataformas, utilizando o mesmo projeto, o que economiza mão de obra e tempo dos desenvolvedores, que ao invés de terem que fazer dois projetos diferentes, consegue concentrar esforços em apenas um. Os serviços da Unity também são um grande ponto forte da ferramenta, pois de forma eficiente, eles conseguem gerenciar informações e incluir mecanismos no seu jogo, que acabam gerando um ganho de tempo no projeto.

Durante o desenvolvimento, alguns contratempos surgiram e acabaram tomando um tempo precioso do desenvolvimento, o que acabou atrasando algumas estimativas do projeto. Um dos contratempos encontrados, que exigiu maior tempo empregado, foram os problemas de colisão lateral do personagem com o cenário, pois como a ferramenta de física da Unity é muito fechada, quando é necessário realizar algo específico para o jogo acaba resultando em algumas falhas, como, por exemplo, quando o personagem que tem um corpo rígido entra em contato com um obstáculo que não tem, dependendo da velocidade de movimento, a colisão será ignorada e o personagem continuará seu percurso.

Um dos trabalhos futuros é inserir no jogo os serviços da Google, onde o usuário pode entrar com sua conta do Google Play, receber troféus e recompensar por jogar e criar o Ranking de melhores pontuações através dele, isso melhoraria a interação entre jogadores. Obter uma conta de desenvolvedor da Apple para publicar o jogo em sua plataforma também é interessante, pois é um segundo mercado com uma demanda grande de aplicativos e jogos de qualidade.

Kouri é um jogo com mecânica simples que opta por atender um público mais casual que não quer ter um vínculo com jogo, onde o utiliza-se em maioria das vezes para passar o tempo e se distrair, utilizando a pontuação do jogo como desafio para o jogador tentar se superar. Com uma aparência simples, o foco artístico do jogo é na emissões de partículas o que torna o jogo um diferencial no seu meio.

## 6. REFERÊNCIAS

PRESSMAN, Roger S. Engenharia de software. 6. ed. Rio de Janeiro: McGraw-Hill, 2006.

KEITH, Clinton. Agile Game Development with Scrum. Addison-Wesley Professional, 2010.

DEGRADE, P. and L.H. STAHL, Wicked Problems, Righteous Solutions: a Catalogue of Modern Software Engineering Paradigms. Yourdon Press, 1990.

MENARD, M., Game Development with Unity. Cengage Learning, 2012.

SILVERMAN, R. E., Git: Pocket Guide, O'Reilly Media, 2013.

Scott Rogers. Level Up. Um guia para o design de grandes jogos. São Paulo: Blucher, 2012.

MOTTA, Rodrigo L., JUNIOR, J. T., Short game design document (SGDD) :

Documento de game design aplicado a jogos de pequeno porte e advergames Um estudo de caso do adverggame Rockergirl Bikeway. Art & Design Track, 2013.

Disponível em [http://www.sbgames.org/sbgames2013/proceedings/artedesign/15-dt-paper\\_SGDD.pdf](http://www.sbgames.org/sbgames2013/proceedings/artedesign/15-dt-paper_SGDD.pdf). Acesso em: 11 de out., 2015.

MARKET, Android Market Publish em: <<http://market.android.com/publish>>. Acesso em: 10 de out. 2015.