

Electro Jam

Gabriel Pacheco Cunegato^{1*}

João Ricardo Bittencourt^{1†}

¹Universidade do Vale do Rio Dos Sinos, Curso de Jogos Digitais, Brasil

RESUMO

Este artigo traça um breve histórico de jogos dos gêneros *platformer* e *runner* até os dias de hoje, em plataformas *mobile*, juntamente de uma análise deste mercado. Descreve então algumas das principais funcionalidades esperadas para o desenvolvimento de *Electro Jam*, um jogo próprio do gênero *platformer*, que tem como objetivo trazer uma jogabilidade diferenciada dos outros desta área. Discorre também sobre a metodologia utilizada na execução deste projeto, levando em consideração certas necessidades de um pequeno time independente. Expõe então passos e implementações realizadas no desenvolvimento deste jogo, descrevendo a forma utilizada para alcançar diversos efeitos visuais presentes neste, tal qual algumas das etapas realizadas na otimização de várias destas implementações, necessárias por conta das plataformas-alvo. Por fim, relata testes efetuados com intuito de validar a qualidade do jogo como produto, também demonstrando uma análise de seus resultados, onde é possível constatar que no geral as implementações e o design planejados foram bem-sucedidos em seus respectivos objetivos.

Palavras-chave: platformer, runner, dispositivos móveis, shaders, technical art, otimizações

1 INTRODUÇÃO

O mercado *mobile*, por conta de seu baixo custo de desenvolvimento e fácil entrada, é inundado com inúmeros clones de jogos bem-sucedidos, com conceitos e *assets* copiados, que caem no esquecimento à medida que são lançados. Os títulos de sucesso, porém, conseguem se destacar em meio à estas imitações e jogos inacabados trazendo consigo uma experiência polida e utilizando de uma jogabilidade interessante, com mecânicas bem executadas, sejam elas novas ou revisitadas. Neste sentido, os *runners* vem evoluindo criando adaptações de conceitos já estabelecidos dentro de jogos presentes em outras plataformas, em especial do gênero *platformer*. O gênero *runner* se popularizou nas plataformas *mobile* por conta de sua simplicidade para controlar o jogo, permitindo que se use facilmente a *touchscreen* como meio de entrada. Ainda, o visível aumento na profundidade de mecânicas e jogabilidade presentes em títulos que vão sendo lançados neste mercado, serve como uma indicação do amadurecimento dos consumidores de jogos do gênero em dispositivos móveis.

1.1 Objetivo Geral

Desenvolver um jogo independente e original do gênero *platformer*, para plataformas *mobile*, com potencial comercial, que atraia a atenção de jogadores *core* e além, distinguindo-se de outros jogos disponíveis no mercado.

1.2 Objetivos Específicos

Criar uma jogabilidade diferente do que é disponibilizado atualmente para jogos do gênero nas plataformas escolhidas, utilizando

mecânicas únicas que possam acrescentar à experiência de jogo sem aumentar diretamente a dificuldade de compreensão deste para o jogador. Desenvolver uma estética visual atraente e chamativa que em conjunto com uma trilha sonora autoral, permita entregar à quem joga uma experiência distinta à cada fase. Se adaptar acerca das limitações das plataformas escolhidas para que estas não prejudiquem a experiência de jogo como um todo.

1.3 Estrutura do Artigo

Este artigo contém sete capítulos, com o primeiro sendo a presente introdução. O segundo capítulo oferece uma análise referente ao mercado no qual se encaixa o jogo aqui apresentado, traçando um paralelo com jogos similares, passados e atuais, com suas diferenças e semelhanças. O terceiro capítulo contém uma descrição do produto, detalhando seu funcionamento e escolhas de design. O quarto capítulo descreve a metodologia utilizada ao longo do projeto, oferecendo uma análise de suas etapas e discorrendo sobre sua forma de utilização. O quinto capítulo por sua vez contém elementos importantes acerca do desenvolvimento dos principais aspectos do jogo. Também tem como objetivo contribuir academicamente com a área de Jogos Digitais, detalhando soluções que possam vir a ser úteis para outros projetos. O sexto capítulo relata os testes realizados a fim de validar a qualidade do jogo como produto, servindo para atestar sobre seu potencial comercial. Por fim, o sétimo capítulo contém as considerações finais do autor, analisando a importância das decisões descritas ao longo deste artigo, como também avaliando os resultados decorrentes dos testes realizados.

2 MERCADO

Desde seu surgimento, a Indústria de jogos tornou-se uma indústria bilionária, ultrapassando inclusive a Indústria de filmes em diversas frentes, segundo pesquisa publicada pelo grupo SuperData Research[1]. Ainda, segundo mesmo estudo, é possível observar que dentro deste mercado, o setor *mobile* vem crescendo exponencialmente, se tornando o segmento que mais movimenta dinheiro dentro desta indústria, tendo gerado mais de 40 bilhões de dólares em 2016, um aumento de 18 por cento em relação ao ano anterior. Já dentro do segmento *mobile*, o gênero *platformer* popularizou-se com a ajuda dos *runners*, um subgênero dos jogos de plataforma, onde o personagem corre por conta própria, e dos *endless runners*, uma variação do anterior, onde não há progressão de dificuldade dividida entre níveis, mas sim ao longo de um único caminho percorrido pelo jogador. Permitindo com que o jogador jogue apenas com um dedo, jogos deste subgênero normalmente apostam na simplicidade dos controles para atrair e manter público. Contudo, muitas vezes, a vontade em simplificar seus jogos para atrair um público mais abrangente acaba por tornar as mecânicas destes rasas e desinteressantes. A medida que este mercado cresce seu público também amadurece, passando cada vez mais a esperar por jogos com mais conteúdo e mecânicas mais profundas.

2.1 Antecessores

Sendo um gênero muito explorado desde o advento dos jogos digitais, e depois, de certa forma reinventado em subgêneros predominantemente criados para dispositivos móveis, é possível observar diversas abordagens para a criação de jogos do gênero *platformer*. Esta seção irá descrever certos jogos que servem como influência

*e-mail: gabriel.cunegato@gmail.com

†e-mail: joaorb@unisinos.br

para o projeto neste artigo, assim como suas qualidades e seu lugar no mercado.

Lançado em 2009 em formato Flash e tempos depois em diversas outras plataformas, Canabalt é conhecido por muitos como o precursor dos *endless runners*[2]. Neste jogo criado por Adam Saltzman, você controla um personagem que pula entre prédios de uma cidade em ruínas, com o único objetivo de percorrer uma maior distância possível. Em Canabalt a única interação possível com o jogo é saltar, com os obstáculos sendo os prédios no cenário e a queda que há entre eles. Sua jogabilidade simplificada e a velocidade na qual o jogador é colocado de volta ao jogo caso perca são facilmente transportados para dentro de dispositivos móveis e são os pontos fortes do título.

Em *Jetpack Joyride*, um *endless runner* desenvolvido pela Halfbrick Studio e lançado em 2011 para dispositivos móveis, o jogador controla um personagem com uma mochila à jato que percorre um corredor sem fim, ultrapassando obstáculos, pegando *power ups* e acumulando pontos. A interação com o dispositivo faz com que o personagem voe mais alto enquanto o jogador tocar na tela, e caiá quando solta o toque. Ao longo do percurso são coletadas moedas que servem para comprar melhorias e itens. Com uma variação maior de obstáculos que Canabalt e seu sistema de *power ups* que mudam a forma de controle quando coletados, *Jetpack Joyride* foi bem-sucedido em expandir a jogabilidade proposta pelo jogo citado anteriormente.

Já mais recentemente, desenvolvido e publicado pela RobTop Games em Agosto de 2013, *Geometry Dash* é um *runner* onde o jogador controla um quadrado que pula sobre obstáculos que vão surgindo em sua frente. Com uma progressão com níveis predefinidos, pode expandir o *level design* em comparação aos jogos anteriores, criando fases mais elaboradas e utilizando de mecânicas diferentes dos outros títulos citados. Exige muita habilidade do jogador e é extremamente punitivo, sendo necessário voltar ao início da fase quando se perde. Além disso cria sequência de pulos e comandos sem dar tempo de reação ao jogador, forçando o mesmo a ter conhecimento prévio de segmentos específicos para passar de certas fases. Esta combinação de elementos torna comum repetir centenas de vezes o mesmo nível até que o jogador consiga vencê-lo.

Nos exemplos acima utilizados, é possível observar que as mecânicas do gênero *platformer* foram sendo adaptadas e trazidas de jogos de outras plataformas ao longo do tempo. Neste sentido, *Electro Jam* tem como objetivo não só contribuir para este processo de polimento do gênero em plataformas móveis, como também se propõe a criar uma reflexão diferente acerca de certos conceitos de *gameplay* já consolidados.

2.2 Monetização

Dentro do mercado *mobile* existem diferentes formas de monetização utilizadas pelos aplicativos disponíveis. As duas principais são os modelos *premium* e o *freemium*. No modelo *premium* de monetização o usuário paga um valor antes de realizar o download e recebe o jogo por completo. Já no *freemium*, ou *free-to-play*, os downloads são realizados gratuitamente, com a monetização ocorrendo através de anúncios e de venda de itens secundários ao jogo como *skins*, *power ups*, entre outros. Normalmente itens adquiridos desta forma não são necessários para experienciar o jogo por completo, mas por sua vez, dão algum tipo de vantagem ou diferencial à quem os adquire, mesmo que este diferencial seja apenas estético.

A escolha do modelo de negócios correto na hora de lançar um aplicativo pode ser um fator decisivo no sucesso do produto, especialmente para desenvolvedores independentes. Um exemplo recente e de repercussão foi o valor arrecadado por *Fire Emblem Heroes* em comparação à *Super Mario Run*, ambos desenvolvidos pela Nintendo, que com menos downloads e utilizando uma franquia com muito menos reconhecimento está trazendo um retorno financeiro

maior para a empresa[3]. Exemplos como este só reforçam a noção de que o sucesso de um jogo raramente depende apenas do seu processo de desenvolvimento.

Embora somente 5 por cento dos usuários de aplicativos grátis realizem alguma transação *in-app*, estes usuários gastam em média 20 vezes mais do que o usuário que paga por aplicativos do modelo *premium*[4]. Está previsto que em 2017 o total de vendas dentro de aplicativos movimentará aproximadamente 37 bilhões de dólares, contra 29 bilhões do modelo *premium*. Ainda, é esperada que a renda principal de aplicativos *mobile* em 2017 seja decorrente de transações *in-app*[4]. Isto define uma tendência de mercado que aponta para a superioridade do modelo *freemium* neste meio. Além dos dados acima, é válido destacar que o modelo *freemium* permite com que os usuários possam testar jogos e aplicações de desenvolvedores desconhecidos sem se comprometerem em investir previamente algum valor, sendo uma boa escolha para equipes independentes que pretendam entrar no mercado. Sendo assim, e na tentativa de melhor monetizar o jogo desenvolvido neste projeto, escolheu-se o modelo *free-to-play*, se utilizando da monetização através de anúncios e *microtransactions*.

2.3 Exposição e Marketing

Em um mercado onde as duas principais plataformas de vendas de aplicativos para dispositivos móveis, Google Play Store e App Store, tem mais de 2 milhões de aplicativos cada[4], a necessidade de expor seu produto ao mundo se torna cada vez mais presente. É inviável e ingênuo esperar que aconteça crescimento orgânico na divulgação de um aplicativo sem que haja algum tipo de publicidade por trás. Esta necessidade foi reconhecida desde o início do desenvolvimento deste projeto. Por conta disto, a partir do momento que o jogo começou a ter conteúdo suficiente para ser exposto e compartilhado, foram criadas contas em redes sociais para promovê-lo, além de ter sido desenvolvido um site¹ com domínio próprio para centralizar informações sobre o jogo. Já em paralelo ao projeto principal foi criada uma demonstração da versão alfa do jogo e publicada em portais como GameJolt e Itch.io, com intuito de gerar exposição ao mesmo, juntamente de um trailer desta mesma versão. Por fim, foi também realizada a submissão desta ao festival de jogos da SBGames 2017.

3 Electro Jam

Electro Jam é um *arcade platformer* 2D desenvolvido inicialmente para plataformas *mobile*. Em *Electro Jam* o jogador controla um disco de vinil que se aventura através de terras musicais, mergulhando e pulando em plataformas feitas de geleia. A temática do jogo é ligada à diferentes gêneros musicais eletrônicos, resultando em fases compostas por cenários coloridos e abstratos. Cada fase terá aproximadamente três minutos de duração e irá desafiar os jogadores criando um ambiente de complexidade emergente, unindo diferentes mecânicas para criar desafios variados e cada vez mais complexos.

3.1 Desafios

Como *Electro Jam* tem suas raízes no gênero *platformer*, a maior parte dos seus desafios testarão as capacidades motoras e de reação do jogador, assim como domínio das mecânicas e do *level design*. Os jogadores irão encontrar diversas formas de testarem suas habilidades através de níveis variados com diferentes dificuldades, cada um com abordagens inusitadas às mecânicas do jogo. Além disso, itens coletáveis espalhados ao longo das fases convidam os jogadores a se aventurarem por seções mais complexas em busca de segredos e recompensas. Por fim, o jogo tomará nota do tempo levado para completar cada fase, adicionando ainda mais uma camada de desafio e exploração à este, incentivando quem joga à buscar ataques e rotas cada vez mais rápidas.

¹<http://www.electrojamgame.com>

3.2 Jogabilidade e Mecânicas

Dentro de um jogo digital, os diferentes aspectos de jogabilidade tem o objetivo de engajar e desafiar os jogadores. Fazem parte do conjunto de regras implícitas do jogo. No caso de *Electro Jam*, as mecânicas principais consistem na movimentação diferenciada do personagem que faz com que o jogador precise do *timing* correto para escolher a sua rota e na capacidade deste de mergulhar nas plataformas. A primeira altera o tipo de jogabilidade vista nos outros jogos do gênero e exige que o jogador divida sua atenção entre o seu personagem e o meio que o cerca. Já a segunda permite que o jogador siga por diferentes rotas, criando uma camada extra de exploração e descobrimento. Além dos aspectos acima citados, também foram planejadas outras com o objetivo de diversificar a jogabilidade. Com o intuito de não sobrecarregar o jogo com elementos à esmo, estas mecânicas secundárias trabalham em cima das duas primeiras já mencionadas, seja alterando como o personagem se movimenta de alguma forma ou mudando a funcionalidade esperada das plataformas. Isto cria uma sinergia entre os elementos de jogo e permite com que estes sejam facilmente compreendidos pelo jogador, sem a necessidade de instruções adicionais. A seguir serão apresentadas tais mecânicas e suas principais formas de interação com o jogador.

Movimentação: Quando dentro da geleia, o personagem irá se mover horizontalmente por si só em um formato de onda. O jogador então deve tocar na tela no momento certo para fazer com que o personagem se mantenha na rota desejada. O tempo certo para o toque depende se o jogador deseja que o disco vá para cima ou para baixo. A velocidade horizontal padrão do personagem pode mudar dependendo da fase.

Pulo: O personagem dará um pequeno pulo quando sair de uma plataforma. Se o jogador continuar tocando na tela no momento do pulo então o disco usará de sua eletricidade para saltar mais alto e permanecer mais tempo no ar. Soltar a tela no meio do pulo faz com que o personagem caia consideravelmente mais rápido.

Plataformas de Geleia: Principal meio de locomoção do personagem. Quando dentro delas o personagem não é afetado pela gravidade. Cair nos espaços entre uma plataforma e outra significa a morte.

Geleias sólidas: Similares às plataformas que são padrão no gênero, pode parar o movimento do personagem em ambos os eixos, dependendo do sentido que ele colidir com a plataforma. É possível que apareça dentro de outras geleias.

Bounce Pads: Uma geleia mais rígida que rebate o jogador ao invés de deixá-lo entrar. Se tocado na vertical, quica o personagem de volta. Se tocado na horizontal, rebate o disco fazendo-o viajar no sentido contrário. Permite explorar mais a verticalidade no *level design*.

Geleia Reversa: Similar às plataformas de geleia, porém inverte o efeito da gravidade quando o jogador entra nela até que o mesmo pule para dentro de uma plataforma normal.

Booster Pad: Impulsiona o personagem aumentando sua velocidade na direção apontada pelo *Booster Pad*. Quando dentro de geleia o efeito da impulsão é mais breve, com o personagem retornando à sua velocidade normal logo após.

Death Pads: Principais obstáculos do jogo, destroem o personagem quando tocados. Aparecem de diversas formas e podem interagir com o jogador não só como obstáculos estáticos mas também o perseguinto, caindo ou surgindo quando quem joga menos espera.

Raios de Ruído: Se parecem e se comportam como um raio laser. Atravessam a tela destruindo o personagem caso o mesmo os toque. Um aviso aparece na tela no canto de onde os raios irão surgir, dando a oportunidade do jogador mudar sua rota para escapar do perigo.

"*Tron*": Um inimigo que viaja pelo cenário na frente do personagem deixando barreiras perigosas que devem ser evitadas.

3.3 Recursos

Consistem em objetos coletáveis ou utilizáveis pelo jogador. Podem compor os objetivos primário ou secundário do jogador nas fases. Se o recurso for coletável e possuir valor suficiente para quem joga, pode se aliar ao *level design* e ajudar a influenciar as escolhas do jogador, sendo um fator importante na hora de criar diferentes desafios ou, de ajudar o mesmo à compreender melhor o jogo. A seguir serão descritos os recursos presentes neste jogo.

Jelly Bits: Moeda do jogo, espalhadas através de diferentes fases e usadas para comprar itens diversos disponibilizados na loja do jogo. Os *Jelly Bits* são um recurso renovável e após coletados aparecerão novamente caso o jogador reinicie a fase. Entretanto, os *Jelly Bits* coletados só são contabilizados para uso do jogador quando este termina a fase. Também podem ser adquiridos através de microtransações.

Tokens Musicais: Apenas três disponíveis por fase, normalmente escondidos ou em alguma seção mais difícil do jogo. Ao pegar os três tokens espalhados em uma determinada fase o jogador desbloqueia uma versão mais difícil desta.

Farol de Checkpoint: Ativar um Farol de *Checkpoint* (quando disponível) durante a fase irá utilizar de sua localização como *checkpoint* caso o jogador morra em algum momento. São distribuídos em média 4 ao longo das fases e normalmente estão posicionados logo após ou antes de alguma seção desafiadora.

Checkpoint Flare: Ativa um farol presente na tela. Seu uso reduz o número de *Flares* do jogador. Podem ser comprados na loja utilizando a moeda do jogo.

Skins: Alteram a estética visual do personagem controlado pelo jogador de diversas formas. Adquiridos através da moeda do jogo que é coletada ao longo das fases ou completando algum desafio específico.

Portal: O objetivo principal do jogo é alcançar o portal em segurança. Significa o final da fase e a vitória do jogador, abrindo uma nova fase.

3.4 Fases e Trilha Sonora

Com seus níveis intimamente ligados à sua música, *Electro Jam* contará com uma trilha sonora única, feita especificamente para o jogo. A estética de cada nível irá variar e será associada ao gênero musical de cada fase. Além disso, boa parte das animações e efeitos visuais estarão em sincronia com a música.

Além do tutorial, o jogo terá dez diferentes níveis, cada um com um gênero musical e estética visual diferentes. Ainda, cada gênero musical contará com uma versão mais difícil de sua fase, chamado de *B-Side*, em referência à parte traseira de discos de vinil.

O modelo de fases permite um controle mais fino do *level design* e daí que é experienteido pelo jogador. Cada uma das fases iniciais tem como objetivo introduzir alguns conceitos do jogo ao jogador, enquanto as fases mais avançadas visam misturar estes conceitos para criar situações inusitadas. No que diz respeito à *build* disponibilizada² junto à este artigo, as fases disponíveis (além do tutorial) para testes da banca avaliadora serão as seguintes: *Electro Bossa Nova*, *Electro Swing* e *Ciptune*. Estas fases servirão para apresentar alguns elementos dentro do jogo como *Death Pads*, Geleias Sólidas, Raio de ruído e *Bounce Pads*.

3.5 Experiência Visual

Já com a ideia de explorar um gênero musical exclusivo em cada fase, veio o interesse em conceber também uma experiência visual diferente entre cada nível de jogo. Sendo assim, cada cenário conta com elementos visuais únicos e sua própria paleta de cores. Ainda, parte dos elementos de *gameplay* que aparecem em múltiplas fases (como *Death Pads*) tem suas aparências trocadas entre cada nível. Para que um visual diferente de algum elemento não corra o risco

²Disponível em: <https://goo.gl/xoX8Py>

de confundir o jogador, foi adotada uma regra para as cores utilizadas em cada fase. Foi definido que, ao selecionar uma cor base para estas, utilizaria-se cores análogas para a composição de seu cenário e componentes visuais, enquanto os elementos que representam alguma ameaça ao jogador seriam coloridos com cores complementares às cores do cenário, como pode ser visto na Figura 1.



Figura 1: Paleta base escolhida para uma das fases. Fonte: Autor.

Isto não só viabiliza uma mudança estética entre níveis sem correr o risco de confundir quem joga, como também acaba salientando visualmente os perigos distribuídos ao longo das fases, independente de sua aparência. Finalmente, visando trazer uma experiência fluída, o jogo apresenta no mínimo 30 quadros por segundo, sendo possível jogá-lo em 60 quadros por segundo dependendo do dispositivo.

3.6 Interface Gráfica

Em jogos digitais, a interface desempenha o papel de informar o jogador e guiá-lo através de diferentes interações disponíveis, podendo afetar negativamente a experiência do usuário caso seja mal planejada. Além do mais, em dispositivos móveis, os requisitos de uma interface são substancialmente diferentes de outros tipos de dispositivos. Isto porque além de se apresentar em uma tela pequena, a interface deve dividir espaço com a tela de jogo e (talvez o maior diferencial) com o meio de entrada, com as mãos do usuário bloqueando parte da área visível. Também é preciso levar em consideração a menor precisão do tipo de input presente nestes dispositivos, a fim de planejar o tamanho da área de interação dos elementos da interface, como botões por exemplo, para que não haja a possibilidade do usuário errá-los na hora da interação. Ainda, é necessário considerar o alcance limitado que os dedos do jogador tem à certas áreas da tela, de acordo com a orientação desta.

Em *Electro Jam*, foi dada prioridade ao uso de símbolos ao invés de texto sempre que possível. A UI (*user interface*) foi planejada pensando em esconder algumas de suas partes quando não for necessário utilizá-las, desobstruindo a tela de jogo. Além disso, houve a intenção de sempre deixar a interface em concordância com a temática musical do jogo.

A cena que mais destaque à interface é a tela de introdução do jogo. Isto porque, visando trazer uma experiência mais fluída ao usuário, esta tela foi planejada para agir como um tipo de *hub*, englobando todas funcionalidades que possam vir a ser utilizadas pelo jogador entre uma fase e outra, dispensando a necessidade de carregar novas telas.



Figura 2: Seleção de fases. Os painéis retráteis mostram o progresso do jogador em uma das fases. Fonte: Autor.

A tela de seleção de fases exibida na Figura 2 visa ser intuitiva para não obstruir o caminho dos jogadores até a experiência de jogo, em especial dos jogadores iniciantes. Por conta disto, e se aproveitando da temática musical do projeto, a interface dispõe cada fase como se fosse um disco de vinil guardado em sua capa. A navegação nesta parte da interface se dá através de um modelo de *overflow*, um conceito de navegação muito difundido em plataformas móveis e aplicativos reprodutores de músicas. Por sua vez, cada disco contém dois painéis retráteis que mostram informações sobre o progresso do jogador em suas respectivas fases. Por fim, um botão localizado logo abaixo de onde estão os discos inicia a fase referente ao disco centralizado na tela.

A tela de seleção de skins demonstrada na Figura 3 utiliza duas linhas de janelas em disposição horizontal, que exibem tanto as skins que podem ser selecionadas e as que ainda estão bloqueadas, com estas últimas estando obscurecidas. Entretanto, apesar do formato convencional, foi optado por manter a comunicação com o jogador através de algo mais visual, sendo exibidas skins animadas ocupando maior parte do espaço em suas janelas individuais.



Figura 3: Seleção de skins, mostrando skins disponíveis e bloqueadas. Fonte: Autor.

A interface da loja é separada em duas partes, como visto na Figura 4, com uma lista de itens disposta ao lado esquerdo, como também uma janela que exibe uma prévia animada e uma breve descrição do item na porção direita. Ao selecionar algum item da lista da esquerda, esta muda o conteúdo exibido na janela de prévia, mostrando ao jogador informações sobre o item selecionado. Abaixo desta janela foi posicionado um botão de compra e a informação do valor do item selecionado, enquanto acima da janela é possível visualizar a quantia de moeda de jogo possuída pelo jogador. Caso o jogador tente comprar um item com dinheiro suficiente, surge uma janela de confirmação antes de finalizar a compra, caso contrário, o botão de compra fica desabilitado.



Figura 4: Interface da loja, mostrando preview do item selecionado à direita. Fonte: Autor.

4 METODOLOGIA DE DESENVOLVIMENTO

Com o objetivo de aperfeiçoar o processo de concepção deste jogo, foi adotada uma metodologia para seu desenvolvimento. A finalidade de se utilizar uma metodologia é de planejar e estruturar as etapas que guiarão, do início ao fim, o desenvolvimento de um projeto em particular. Este capítulo discorrerá sobre o método utilizado ao longo do desenvolvimento de *Electro Jam*, bem como cada uma de suas etapas.

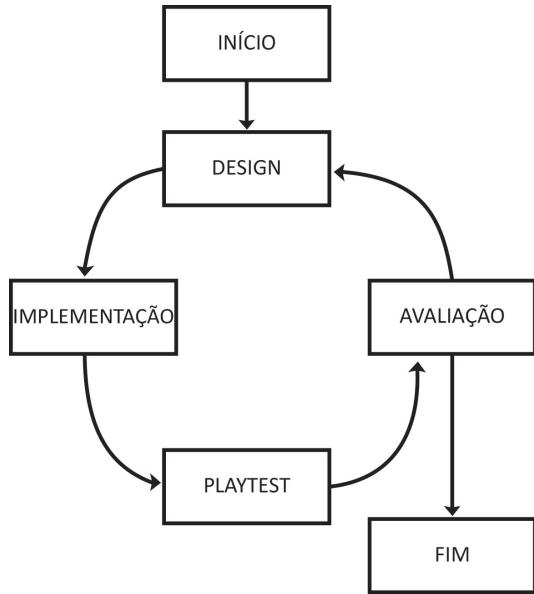


Figura 5: Metodologia de *Game Design* Iterativo como apresentado por Schreiber. Fonte: Autor.

A metodologia utilizada se baseia na Metodologia Iterativa (Figura 5) exposta por Ian Schreiber em seu curso *Game Design Concepts*[5], cujo processo é dividido em quatro etapas, sendo estas (em ordem de execução), Design, Implementação, *Playtest* e Avaliação. A metodologia é considerada iterativa pois sua última etapa, a de avaliação, não implica no fim do desenvolvimento, mas sim em um novo ciclo do processo, voltando novamente à etapa de Design, permitindo com que as informações obtidas ao longo da iteração passada sejam utilizadas para aperfeiçoar o jogo. Ainda, segundo Schreiber, um maior número de iterações contribui para o resultado final do jogo, sendo que "...qualquer coisa que permita com que você itere mais rápido geralmente resultará em um um jogo melhor no final." [5]. Sendo assim, é inequívoca a necessidade de se adotar uma metodologia que melhor comporte as peculiaridades de cada projeto.

A realidade presente no desenvolvimento de jogos independentes muitas vezes impõe o acúmulo de funções ao longo do projeto, fazendo com que uma única pessoa desempenhe diversos papéis que em outra situação não estariam relacionados. No caso de *Electro Jam*, a função de *Game Designer* estava ligada à função de Programador. Considerando que a metodologia acima apresentada visa acolher apenas a Etapa de *Game Design* do processo de desenvolvimento de um jogo, e pensando em melhor suprir as necessidades do projeto levando em conta o tempo e recursos disponíveis, foram propostas algumas alterações na metodologia mostrada por Schreiber. Estas mudanças tem a finalidade de incorporar diferentes etapas da implementação que sucedem à Etapa de Design. A modificação principal consiste em dar à etapa de Implementação seu ciclo iterativo próprio independente do ciclo de *Game Design*, como pode ser observado na Figura 6.

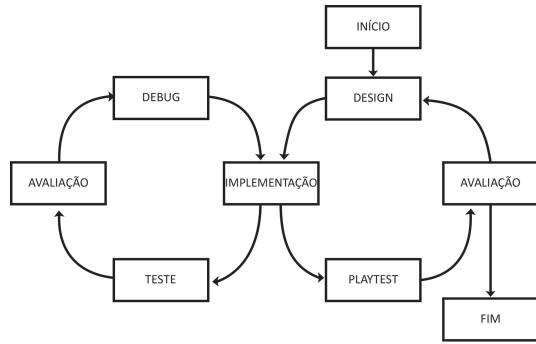


Figura 6: Metodologia alterada para comportar melhor a parte de implementação. Fonte: Autor.

Este ciclo pode ser todo realizado pelos desenvolvedores, resultando em iterações menores e mais rápidas que as de *Game Design*. Isto beneficia a execução do projeto pois permite gerar uma implementação mais completa e livre de erros antes que esta seja enviada para a Etapa de *Playtest* no período de *Game Design*, ocasionando uma diminuição em iterações redundantes dentro deste ciclo, repassando a responsabilidade de certas iterações para um ciclo mais rápido que ocorre dentro das etapas de implementações.

4.1 Etapa de Design

A Etapa de Design que ocorre na primeira iteração é utilizada para definir as principais mecânicas e regras do jogo. Aspectos importantes como interação dos jogadores, progressão de jogo e condições de vitória são definidos aqui. Já nas outras iterações, utiliza-se das informações colhidas na etapa de avaliação para redefinir o que for necessário no jogo.

No ciclo interno da Implementação esta etapa é chamada de Etapa de *Debug*, e tem como objetivo replanejar implementações para corrigir as falhas identificadas ao longo deste ciclo. Também comporta o planejamento de *refactoring* do código quando necessário, reestruturando certas implementações para se adaptarem melhor ao crescimento do projeto, sem alterar seu resultado ou afetar seu desempenho negativamente.

4.2 Etapa de Implementação

Parte voltada ao desenvolvimento do código do jogo, visando honrar o planejado na Etapa de Design da forma mais fiel possível. Quando sucede a Etapa de *Debug*, busca implementar as soluções concebidas nesta etapa a fim de fazer que códigos com problemas retornem os resultados esperados. É importante nesta etapa não se afastar das funcionalidades planejadas na Etapa de Design, a fim de evitar uma mistura nas implementações, o que pode dificultar a manutenção do código e causar resultados indesejados.

É imprescindível a boa enumeração e descrição do código criado e alterado nesta etapa. Isto porque enquanto a Etapa de Design conta com o *Game Design Doc*, o código-fonte necessita de uma solução similar para catalogar as mudanças realizadas ao longo do projeto. Para tal, além de implementações bem documentadas se utilizou uma ferramenta de controle de versão. Estas ferramentas permitem catalogar e retornar à versões anteriores do código-fonte quando necessário.

4.3 Etapa de *Playtest*

Tem objetivo de recolher informações sobre o uso e experiência relativos à diferentes aspectos do projeto, como interface, *gameplay*, *level design*, entre outros. É a etapa que mais difere entre o ciclo iterativo da implementação e o ciclo de *Game Design*. O formato destes é relatado a seguir.

4.3.1 Testes de implementação

Nos testes de implementação, são verificadas a qualidade do código gerado e sua capacidade para primeiramente reproduzir os resultados esperados, e em segundo plano, manter o desempenho em um nível aceitável. Sendo assim, fez-se necessário que desde o início do desenvolvimento fossem definidas as especificações mínimas requeridas de um dispositivo para executar o jogo. Para tal fim, ao longo do desenvolvimento, estes testes foram realizados majoritariamente em um dispositivo Sony modelo Xperia E3. Este dispositivo *low end* acabou por delimitar o mínimo de desempenho esperado para executar *Electro Jam* da forma planejada descrita mais no inicio deste artigo. Tendo dito isso, é plausível que outros dispositivos mais fracos também consigam executar o jogo, porém isto está fora do escopo destes testes.

Para garantir os resultados esperados da implementação, especialmente das mecânicas de jogo, foram realizados testes de forma iterativa ao longo do desenvolvimento, tanto pelo desenvolvedor como através de terceiros. Os testes consistiram em misturar mecânicas diferentes em pequenos ambientes controlados onde a interação entre estes elementos possa ocorrer de forma inesperada pelo desenvolvedor. Isto permitiu observar e identificar comportamentos inapropriados das implementações que não seriam possíveis de constatar tão facilmente em um ambiente normal de jogo.

Já os testes de desempenho foram realizados pelo desenvolvedor, e o procedimento consistiu em executar diferentes *builds* ao longo do desenvolvimento no dispositivo escolhido juntamente com uma ferramenta de *profiling* de código. Muitas destas ferramentas permitem analisar em tempo real o período levado pela execução de implementações de um software, viabilizando a análise de gargalos na programação e permitindo com que o desenvolvedor observe o impacto de desempenho proveniente de suas mudanças e reiterações no código com facilidade.

4.3.2 Testes de jogabilidade e usabilidade

Como citado anteriormente, o principal objetivo do desenvolvimento foi criar uma jogabilidade divertida com mecânicas engajantes. Assim, a etapa de testes do ciclo de *Game Design* teve grande importância durante todo o desenvolvimento. Nesses testes, foram utilizadas *builds* das versões mais atuais e sempre com participação de terceiros. É necessário deixar claro a importância deste processo não ser realizado por quem desenvolve o jogo. Isto porque à medida que o desenvolvimento avança, as pessoas em maior contato com o jogo tendem a desenvolver um domínio maior sobre o mesmo, podendo levar em consideração apenas o balançamento de uma jogabilidade de mais alto nível, negligenciando inconscientemente a experiência dentro de jogo para usuários iniciais. Além disso, usuários com uma experiência prévia diferente em jogos podem trazer à tona observações que de outra forma passariam despercebidas. Em decorrência desta etapa é que são e foram realizadas as maiores alterações do que foi descrito no documento de *game design*.

Os testes nesta etapa foram realizados de forma pessoal ou online, sempre dando prioridade para testar pessoalmente. Isso torna possível não só receber o feedback do jogador como permite perceber nuances na jogabilidade deste. Após o teste, tanto on-line como off-line, foram realizadas algumas perguntas ao jogador dependendo do objetivo do teste, além do desenvolvedor anotar algumas informações padrões, como tempo de jogo, experiência anterior, se visitou o tutorial, entre outros. Testes on-line também seguiram o mesmo processo, com pequenas alterações nas perguntas devido ao formato.

Além dos testes no formato acima, foi realizado um teste de maior abrangência. Para tal foi utilizado a demonstração pública mencionada no início deste artigo. Esta *build* publicada conta com um breve tutorial e com uma fase para fins de demonstração, contendo algumas das mecânicas planejadas para o jogo. Esta versão foi excepcionalmente criada para a plataforma HTML5, visando

seu envio para os portais independentes de publicação de jogos Itch.io e GameJolt. É importante notar que foi optado pela troca de plataforma para publicação em tais portais por conta da maior facilidade de convencer possíveis usuários a testar um jogo em formato que dispensa downloads ou instalações, eliminando qualquer tipo de esforço extra por parte do jogador em experimentar o jogo. Juntamente à demonstração, foi utilizado uma ferramenta de *Content Analytics* ou Análise de Métricas. Estas ferramentas permitem, à vontade do desenvolvedor, receber informações acerca da forma que o software está sendo utilizado. No caso deste projeto, alguns dos dados recolhidos continham informações referentes ao uso do tutorial, mortes dos jogadores, coletáveis encontrados, entre outros. Junto às informações recolhidas desta forma, também foi possível receber feedback mais específico utilizando um formulário que foi disponibilizado para os jogadores preencher voluntariamente, além do recebido também através das comunidades on-line nas quais o jogo foi compartilhado, como fóruns de games, Reddit, e dos próprios portais onde a demonstração foi publicada.

4.4 Etapa de Avaliação

Ocorrendo após a etapa de testes, consiste em interpretar os resultados obtidos, isolar os problemas identificados e propor soluções. No ciclo de *Game Design* encontram-se problemas de usabilidade, compreensão de mecânicas e aspectos gerais da experiência do jogo, como problemas de interface, curva de aprendizagem, dificuldade, entre outros. É importante também identificar a causa destes problemas, que podem ser resultantes de uma forma imprópria de apresentar algum fator do jogo ao jogador, como também podem decorrer de um problema inerente a algum elemento utilizado. Dependendo da origem pode significar que algum aspecto do jogo precisa ser refeito completamente ou deixado de fora do produto final.

No ciclo de implementação encontram-se defeitos na elaboração do código, causando efeitos indesejados que prejudicam a utilização do aplicativo, sendo que nos casos mais extremos podem impossibilitar que o jogador continue seu jogo. Também podem causar problemas de desempenho nos dispositivos utilizados. Por fim, podem resultar em efeitos inesperados nas mecânicas do jogo, criando regras inconsistentes e prejudicando a experiência do jogador. Para identificar mais facilmente as causas de problemas observados nesta etapa é necessário primeiramente assegurar um meio de reproduzir de forma consistente as falhas encontradas, a fim de restringir suas possíveis causas, facilitando o processo de depuração. Em casos onde não é possível encontrar um meio de replicar alguma disfunção encontrada, e caso esta ocorra de forma esporádica ocasionando apenas pequenos inconvenientes, recomenda-se que se tome nota da falha mas que a deixe de fora da próxima etapa de *Debug*. Isto porque à medida que outras iterações são realizadas é possível que uma forma constante de reproduzi-la seja encontrada. Por fim, quando identificados erros em funcionalidades que operavam corretamente em iterações anteriores, se utilizou da ferramenta de controle de versão mencionada na etapa de Implementação para precisar a versão do código-fonte que originou o problema, limitando assim consideravelmente o campo de busca por uma solução.

5 DESENVOLVIMENTO

Logo no início do projeto, é necessário escolher as plataformas para as quais o jogo será desenvolvido. Dependendo de algumas especificidades do documento de *Game Design* as possibilidades diminuem. Esta é uma etapa importante na concepção de qualquer jogo, visto que influencia outras decisões ao longo da execução do projeto. Uma destas decisões é a escolha das ferramentas utilizadas para desenvolver o jogo. No caso de *Electro Jam*, foi escolhida a *Unity* como motor de jogo para o projeto. Isto porque o suporte de portabilidade disponibilizado pela *engine* às nossas plataformas-alvo oferece uma facilidade na hora do desenvolvimento, elimi-

nando a manutenção de implementações variadas para plataformas diferentes. Isto significa também que é possível portar o jogo para outras plataformas no futuro se assim se desejar, expandindo seu alcance dentro do mercado. Além disso, *Unity* conta com algumas ferramentas integradas à si que trazem mais praticidade e agilidade ao desenvolvimento, como os já mencionados *Profiler* e o serviço *Unity Analytics*, dispensando o uso de soluções externas.

Durante a implementação do projeto, os seguintes objetivos foram elencados, em ordem de importância: implementar as mecânicas e elementos diversos da jogabilidade descritos na seção de *Game Design*, gerar uma estética visual que realce as individualidades do jogo, vincular elementos estéticos e de jogabilidade ao *sound design* e dar suporte para dispositivos *low end* sem sacrificar itens anteriores. Consequentemente, o desenvolvimento ao longo do projeto se mostrou complexo apesar das mecânicas relativamente simples do jogo. Isso se deve em especial às diversas soluções necessárias como *shaders* e efeitos customizados que tiveram de ser desenvolvidos para atingir a estética desejada. Esta complexidade também foi decorrente de algumas especificidades do *sound design* para implementação e tratamento da trilha sonora, e também devido à certas peculiaridades inerentes à dispositivos móveis. Sendo assim, esta seção divide-se em categorias com o intuito de detalhar melhor as diferentes áreas do desenvolvimento que se mostraram de alguma forma problemáticas, assim como as soluções desenvolvidas para tais.

5.1 Otimizações

Além de termos o desenvolvimento para plataformas *mobile* como foco principal, também se teve como objetivo abranger uma gama maior de dispositivos. Para tal, manter baixos os requerimentos mínimos para executar o jogo em dispositivos *low end* em uma taxa mínima de 30 quadros por segundo, mantendo-se fiel à experiência planejada para o jogo, sempre foi um objetivo paralelo à todo desenvolvimento. Consequentemente, foram necessários cuidados extras em algumas implementações, visando a otimização e melhora de desempenho. Esta seção relata, de modo geral, os principais aspectos do processo de otimização de algumas partes importantes ao desempenho do projeto.

5.1.1 Shaders

Os *shaders* utilizados no desenvolvimento do jogo são compostos por duas funções principais, o *vertex shader*, que é executado uma vez por vértice, e o *fragment shader*, que executa uma vez por fragmento (pixel) produzido por malhas. Além disso, pelo fato das instruções dos *shaders* serem executadas pela GPU simultaneamente, não é possível trocar informações entre ciclos de fragmentos. Isto faz com que muitas vezes os mesmos cálculos e fórmulas que retornam o mesmo resultado sejam resolvidos por inúmeros ciclos de fragmentos diferentes, desperdiçando tempo de processamento da GPU. Para diminuir ao mínimo possível o desperdício de ciclos da placa de vídeo, faz-se o uso do *vertex shader* para qualquer operação matemática sempre que possível. Isto porque não só o *vertex shader* é normalmente executado menos vezes, como também aproveita-se de uma interpolação de valores realizada em nível de hardware (leia-se: com custo quase nulo) que entrega aos fragmentos entre dois ou mais vértices uma média ponderada dos valores calculados no *vertex shader*.

No exemplo da Figura 7 os vértices tem as cores azul, verde e vermelho. Entretanto, devido à interpolação realizada pelo hardware, é possível notar que as cores entre os vértices se misturam. Caso o cálculo de cores fosse realizado dentro do *fragment shader*, a quantidade de operações dependeria da resolução da tela, repetindo-as para cada pixel preenchido pelo triângulo. Entretanto ao usar o *vertex shader*, essa quantidade se limita à apenas três.

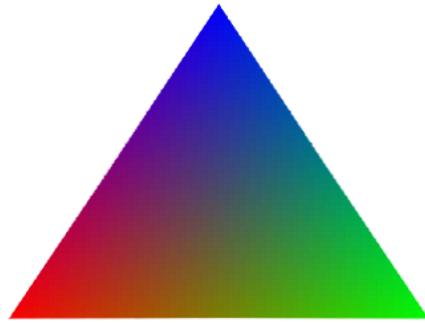


Figura 7: Interpolação de cores provenientes dos vértices de um triângulo. Fonte: Autor.

5.1.2 Blur

Independente da plataforma, efeitos de *blur* em tempo real normalmente prejudicam o desempenho. Ainda, como as peculiaridades estéticas deste projeto que serão discutidas mais à frente exigem o uso deste efeito em tempo de execução, foram levadas em consideração as limitações de poder de processamento das plataformas *mobile* na hora de implementação. Uma implementação de *Gaussian Blur* consiste em pegar uma amostragem das cores dos pixels vizinhos à um pixel central, aplicando um peso à estas amostras de acordo com a função gaussiana de distribuição e então recalculando e combinando os valores de cor deste grupo de pixels. Em um efeito com um raio de 5 por 5 pixels, que é o caso do efeito implementado neste projeto, temos a necessidade de ler 25 pixels, sendo estes pesados e somados ao pixel central, se repetindo para todos os pixels da imagem. Levando em conta que a resolução mais usada atualmente em dispositivos móveis é 720p[6] isso resultaria em mais de 23 milhões de operações por quadro renderizado.

A primeira e mais óbvia otimização neste caso é fazer o *downsampling* da imagem na qual será realizado o efeito. Neste caso em específico, a imagem usada é reduzida em quatro vezes antes de ser entregue ao efeito de *blur*, que depois de aplicado sofre *upsampling* de volta para a resolução original. Se tomarmos novamente o número de operações citado acima, apenas com este passo reduzimos o número de operações necessárias para aproximadamente 5.7 milhões. Além disso, por conta de características específicas dos dispositivos móveis, como resolução e tamanho da tela, podemos realizar um *downsampling* desta magnitude sem que a perda na qualidade do efeito seja muito perceptível, como pode ser visto na comparação feita na Figura 8.



Figura 8: Imagem comparativa com Zoom em 5x. Na esquerda com 4x de *downsampling*. Na direita sem *downsampling*. Fonte: Autor.

A segunda etapa na otimização do efeito de *blur* utilizado é a implementação de uma amostragem linear de pixels. Isto é possível porque a aplicação do efeito de *gaussian blur* é separável, podendo separar a amostragem dos eixos X e Y, realizando em dois passos as operações necessárias. Isto faz com que a quantidade de amostras necessárias por pixel no nosso caso (5×5) caia de 25 para 10. Mais uma vez usando de nosso exemplo, e após aplicadas ambas etapas de otimização recém citadas, o número de operações para reproduzir o efeito resulta em aproximadamente 2,3 milhões de operações, uma redução de quase 90 por cento.

5.1.3 Fillrate e overdraw

Uma das grandes limitações dos dispositivos móveis é sua *fillrate*, ou seja, a quantidade de pixels que estes conseguem preencher na tela em um determinado período de tempo. Por conta disto, a exibição de materiais transparentes e em quantidade podem apresentar impactos negativos no desempenho destas plataformas, isto porque o efeito de transparência faz com que o mesmo pixel seja escrito mais de uma vez para gerar a cor resultante da sobreposição entre objetos transparentes e sólidos. No caso deste projeto, este problema foi claramente visível na hora da implementação do efeito de portal presente no final das fases. O Portal consiste em um composto de diferentes efeitos de partículas com transparência, algumas cobrindo boa parte da tela no momento em que o Portal está visível. Ainda, muitas destas partículas se sobreponham com o objetivo de passar uma impressão de volume ao efeito. Isto causava quedas de até 10 quadros por segundo em certos momentos, prejudicando a fluidez do jogo. A solução adotada para este exemplo em específico foi simples: diminuir o número de partículas simultâneas presentes no efeito. Entretanto, para não ser necessário sacrificar o visual pretendido para o portal, as texturas das partículas foram sobrepostas em um programa de edição de imagens, de forma com que a aparência volumosa do efeito fosse produzida previamente, diminuindo a quantia de partículas presentes em tempo real, como mostrado na comparação da Figura 9.

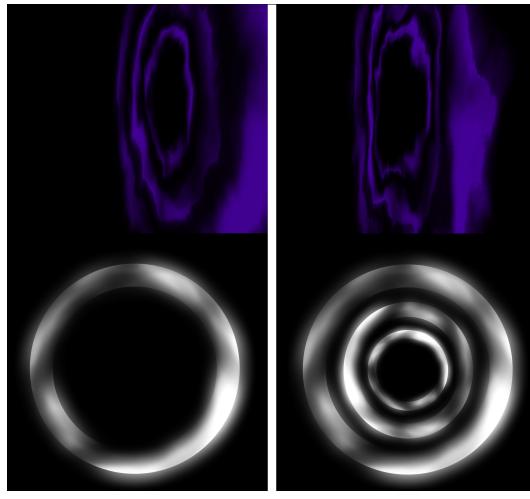


Figura 9: Na direita, partícula e efeito otimizado. Fonte: Autor

5.1.4 Draw Calls

Colocando de maneira simples, uma *draw call* acontece quando a CPU envia à GPU dados para serem renderizados. Esta operação tem seu custo para a CPU, impactando o desempenho caso a quantia de *draw calls* em uma cena seja alta. No motor de jogo utilizado é possível renderizar um conjunto de malhas com uma única chamada contanto que se cumpram alguns requisitos, sendo o mais importante deles que estas malhas compartilhem do mesmo material.

Na fase *Electro Bossa Nova*, por exemplo, é utilizada uma árvore formada de doze quadriláteros que compõem diferentes segmentos de sua estrutura. Como as malhas que compunham a árvore utilizavam uma textura diferente para cada segmento, esta acabava por gerar doze *draw calls* por árvore em tela. Em um jogo onde a média de chamadas do tipo fica em torno de vinte na maior parte do tempo, ter que arcar com doze *draw calls* por conta de apenas um elemento visual é algo inviável. Neste caso o problema foi resolvido criando um *texture atlas*, uma imagem que continha a arte de todos os segmentos da árvore, permitindo assim a utilização de apenas um material com textura de todos componentes de tal elemento. Ainda, utilizando um programa de modelagem 3D, foram criados quadriláteros cujo mapeamento UV apontava para diferentes partes do *texture atlas*. O resultado disso é visto na Figura 10, mostrando a mesma composição visual planejada, mas agora com o custo de apenas uma *draw call*.



Figura 10: *Texture atlas* e seu respectivo objeto finalizado. Fonte: Autor

5.2 Estética Visual

A estética visual foi trabalhada não só por ser um diferencial necessário neste mercado, mas também por ser diferente do que é visto em outros jogos do gênero, se tornando um dos grandes atrativos do projeto. Por conta de sua temática mais abstrata, foi necessário recorrer ao desenvolvimento de *shaders* e outros efeitos especiais. Estes efeitos por sua vez acabaram contribuindo para gerar uma identidade visual coesa ao jogo (Figura 11), servindo como guia na criação de diversos aspectos visual do mesmo.



Figura 11: Tela do jogo. Fonte: Autor.

5.2.1 Efeito de Geleia

Esta implementação cumpre um papel como atrativo estético e como feedback visual ao jogador quando o mesmo entra e sai das plataformas. O efeito foi alcançado utilizando-se uma implementação da Lei de Hooke[7] para a simulação de molas, e funciona aplicando os efeitos desta simulação aos vértices das malhas que compõe as plataformas. O algoritmo age aplicando uma força no

vértice mais próximo do ponto de colisão do personagem com a plataforma, após isto puxa os vértices vizinhos na mesma direção como demonstrado na Figura 12. Ainda, é possível alterar os valores atribuídos à cada plataforma, o que permite com que estas simulem diferentes comportamentos com consistências variadas.

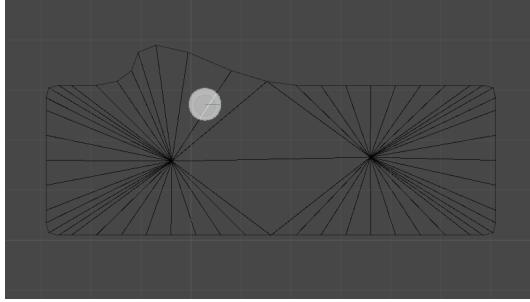


Figura 12: Malha de uma plataforma com vértices alterados. Fonte: Autor.

As malhas empregues, por sua vez, são oca e feitas utilizando vértices ordenados, de forma que o algoritmo que aplica o efeito de fluido sempre tem conhecimento sobre quais são os vértices vizinhos àquele que foi movido inicialmente. Para tal, foi necessário também a criação de um algoritmo próprio para geração das malhas, que fizesse a triangularização da forma correta sem sobrepor vértices e sem alterar a ordem dos mesmos. Este algoritmo funciona inicialmente permitindo com que o desenvolvedor desenhe a forma da malha como visto na Figura 13, clicando na tela para definir os pontos mais externos desta.

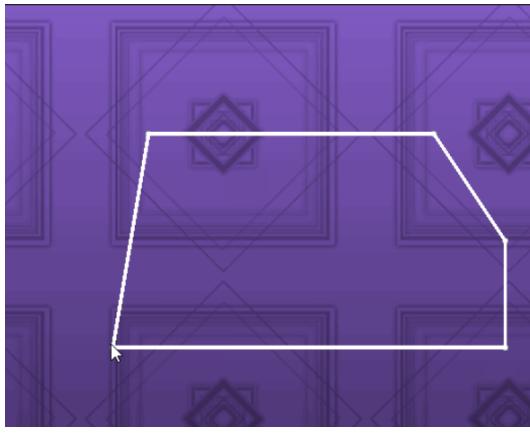


Figura 13: Definindo a forma da malha ao escolher os pontos mais externos. Fonte: Autor.

Logo então o algoritmo adiciona mais vértices entre os pontos antes definidos. Estes vértices extras determinam o quanto suave será a aparência das plataformas quando deformadas. Após adicionar os pontos extras, o desenvolvedor posiciona vértices que agirão como pontos centrais nas malhas, com objetivo de impedir a sobreposição de triângulos durante o efeito. O algoritmo utiliza destes vértices centrais definidos pelo desenvolvedor para realizar a triangularização das malhas, fazendo com que os triângulos destas sejam sempre formados por dois vértices em sequência e pelo ponto central mais próximo. Este conjunto de implementações por fim foi sendo incrementado até virar uma ferramenta utilizada no *level design*.

A técnica acima porém, mesmo com o uso de um algoritmo específico de triangularização resultava, em algumas situações, em

um problema de sobreposição de triângulos, em especial nos cantos das malhas onde os triângulos são menores e mais aglomerados. Somando isto à transparência das plataformas, ocorria a sobreposição dos valores de cores e de transparência de tais triângulos, como pode ser visto na Figura 14.

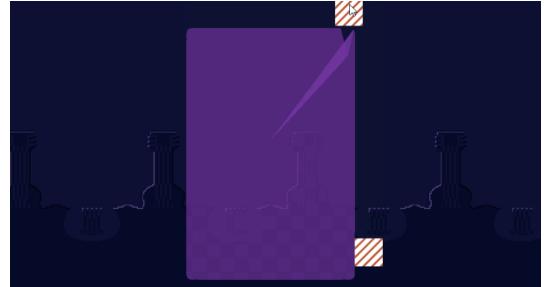


Figura 14: Triângulos sobrepostos acabam por resultar em uma combinação de cor e transparência diferente do esperado. Fonte: Autor.

Para resolver este problema foi implementado um *shader* que faz uso do *stencil buffer* para eliminar fragmentos sobrepostos. O *stencil buffer* é um buffer que guarda números inteiros por pixel, neste caso com tamanho de 1 byte. Enquanto o *depth buffer* (ou *Z-Buffer*) contém informações de qual pixel está mais próximo da tela, o *stencil buffer* é utilizado para criar máscaras de pixels. Amplamente disponível em GPUs atuais e antigas, permite limitar a renderização. Segue abaixo um breve exemplo e explicação de uso deste buffer dentro de um *shader* feito na linguagem HLSL (*High Level Shader Language*), utilizada pela *engine* deste projeto.

```

1 Stencil
2 {
3     Ref 2
4     Comp NotEqual
5     Pass Replace
6 }
```

Lista 1: Breve implementação de uso do *stencil buffer*.

Neste exemplo mostrado na Lista 1, o *shader* das plataformas escreve um número inteiro (de valor 2 neste caso) como referência no *stencil buffer* para cada pixel que ele renderiza. O pixel só é renderizado, porém, se o inteiro salvo no *stencil buffer* não for igual ao valor de referência, como explicitado pelo parâmetro *Comp* e seu atributo *NotEqual*. Por fim, o parâmetro *Pass* define o que é feito com o valor de referência caso o teste passe, neste caso substituindo o valor presente no buffer pelo valor de referência.

5.2.2 Deformação de Texturas

Esta implementação é utilizada por diversos dos *shaders* do jogo, e é responsável por muitos dos efeitos animados vistos em diversos elementos deste, como o fundo presente em algumas das fases, armadilhas ou inclusive a animação da textura das geleias. Faz uso de uma textura secundária, normalmente em escala de cinza, que é inicialmente lida e então tem seus canais de cores utilizados para alterar as coordenadas UV da textura primária. Além disso, as coordenadas da textura utilizada para a deformação são também alteradas incrementalmente pelo fator tempo, fazendo com que a deformação aplicada em um determinado ponto da textura principal esteja sempre sendo modificada. Isto permite assim a criação de diversos efeitos para fins variados, apenas usando padrões de deformação diferentes, como é demonstrado na Figura 15.

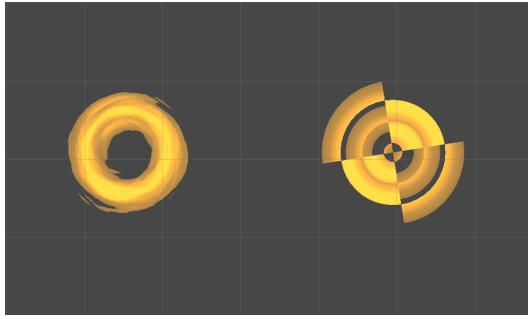


Figura 15: Duas malhas idênticas utilizando a mesma textura mas com texturas de deformação diferentes. Fonte: Autor

5.2.3 Brilho dinâmico

Outro importante efeito na composição da estética do jogo. Necessário para destacar as plataformas e dar um visual iluminado às mesmas. Como as malhas das plataformas são deformadas em tempo de execução, não foi possível utilizar uma camada de brilho composta previamente em algum programa de edição de imagens. Portanto este efeito é alcançado utilizando uma combinação de *shaders de post-image effects*. Tal efeito se adapta às formas tomadas pelas plataformas mesmo quando elas sofrem deformações. Também pode mudar de tonalidade, cor ou padrão de animação em tempo de execução, fazendo deste um efeito muito versátil que pode ser amplamente usado no jogo sem ficar repetitivo.

A implementação utiliza uma câmera secundária para renderizar a silhueta dos objetos que devem ter o efeito de brilho, com a cor utilizada pela silhueta inicial definindo a cor base do efeito. Também calcula a distância da posição do jogador até os pixels destes objetos, e então incrementa igualmente os canais de cores das devidas silhuetas baseado nesta distância, gerando um sutil efeito de "aura" que serve para aumentar suavemente a intensidade do brilho por onde o jogador passa. O resultado disto é então renderizado em uma textura. Nesta se aplica um efeito simplificado de *gaussian blur* para expandir gradativamente a silhueta dos objetos. Ocorre então uma subtração entre a silhueta com *blur* e a original, resultando em uma terceira textura na qual sobra apenas a linha utilizada para o brilho. São aplicadas então alterações feitas através de uma técnica de deformação de texturas, como explicado anteriormente. Por fim o resultado do processo descrito acima é adicionado à tela do jogo, reproduzindo um efeito de linha brilhante e animado que envolve os objetos escolhidos.

Além do conjunto de técnicas detalhado acima, também foi implementado uma solução alternativa utilizando MRT (*Multiple Render Targets*). O objetivo desta solução era principalmente a otimização do efeito de brilho, eliminando a necessidade de uma segunda câmera. Para uso desta técnica foi necessária a implementação de um *fragment shader* que realizava a renderização em duas texturas diferentes com parâmetros de cor distintos. A primeira textura consistia no *color buffer* padrão do objeto em cena, e era imediatamente renderizada na tela de jogo. Já a outra imagem contia as silhuetas que no método acima eram geradas pela segunda câmera. De posse desta segunda imagem, a finalização do efeito procedia da forma já descrita. Os resultados de desempenho com este método foram mais que satisfatórios, aumentando a taxa de *frames* por segundo em torno de 15 por cento. Apesar dos bons resultados, houve a necessidade de utilização da implementação anterior. Isto se deveu ao fato de que o suporte para MRT em alguns dispositivos móveis só foi implementado a partir da versão 3.0 da OpenGL ES, o que limita a compatibilidade do jogo com dispositivos mais antigos.

5.2.4 Gradiente Dinâmico

Este efeito cria um gradiente de duas cores de acordo com as coordenadas UV da malha renderizada. Visto que alguns objetos, como o fundo das fases, são estáticos e apenas tem suas coordenadas UV animadas para dar impressão de movimento, permite criar uma transição de cor que acompanha o jogador de acordo com a sua localização nas fases como é visto na Figura 16. A implementação se dá dentro do *fragment shader* e faz com que a cor resultante de um pixel seja a cor lida da textura, multiplicada pelo resultado da interpolação linear entre duas cores escolhidas, enquanto utiliza as coordenadas UV como fator t da interpolação. Ainda, foi feita uma variação deste efeito para objetos que não tem suas coordenadas animadas durante o decorrer do jogo, utilizando-se neste caso da posição deste objeto no eixo Y como fator t .

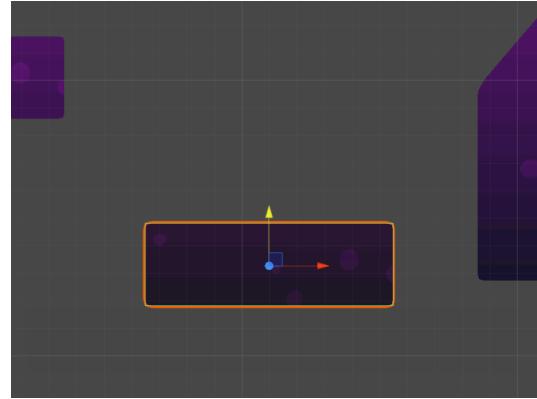


Figura 16: Três objetos utilizando a mesma textura, porém com tonalidades diferentes. Fonte: Autor.

5.3 Interface Gráfica

A singularidade da interface planejada para o jogo, em particular na tela de introdução, fez necessário implementações que vão além das soluções já presentes no motor de jogo utilizado. Além de utilizar extensivamente o *stencil buffer* para criar máscaras, como é possível observar por exemplo nos painéis que contêm as estatísticas de cada fase, foram necessários criar alguns efeitos distintos para melhor se comunicar com o jogador.

5.3.1 Efeito de desbloqueio de fases

Como dito anteriormente, a seleção de fases é disposta em formato *coverflow*, com a capa das fases desbloqueadas mostrando alguma arte referente à fase, enquanto as fases bloqueadas mostram um ponto de interrogação. Sendo assim, foi preciso criar um efeito que permitisse transicionar entre a capa bloqueada e a capa real da fase. Para tal, foi desenvolvido um *shader* que pudesse ler duas texturas e que alternasse entre elas gradativamente dado um valor arbitrário. Normalmente efeitos como este utilizam uma textura extra em escala de cinza que guia qual pixel apagar de acordo com a intensidade de cor do pixel desta textura. Entretanto, para evitar o uso de uma textura extra apenas para essa função, foi utilizado o canal de alfa da textura principal como guia. Como as capas dos discos são sólidas, foi possível descartar a função de transparência deste canal para servir à outro propósito. Em um programa de edição de imagens foi aplicado um gradiente de transparência à textura principal, no sentido desejado ao efeito de transição. Já no código, o *shader* é instruído a mostrar a textura principal sempre que o valor arbitrário de descarte for maior que o valor na camada de alfa desta. Assim sendo, ao incrementar o valor de descarte gradativamente quando uma fase nova precisa ser desbloqueada na interface, é possível gerar um resultado de transição animada, exibido na Figura 17.



Figura 17: Efeito de desbloqueio em processo. À direita uma fase ainda bloqueada. Fonte: Autor.

5.3.2 UI e objetos animados

Com uma interface que está frequentemente em movimento, seria no mínimo destoante que certos elementos visuais, que normalmente são animados *in-game*, como o avatar do jogador por exemplo, fossem apresentados de forma estática na interface. Ainda, como as *skins* fazem parte do modelo de monetização do jogo, representá-las da forma mais fiel possível é essencial, tanto para chamar mais a atenção de possíveis compradores como para não correr o risco de confundir o jogador. Para resolver este problema, fez-se o uso de *Render Textures*. Na Unity, *render textures* são texturas especiais que podem receber renderização direta de uma câmera. Em *Electro Jam*, foi utilizada uma câmera secundária que renderizava o que estava dentro de seu campo de visão em uma destas texturas. No caso da tela de seleção de *skins*, onde diversos itens são visualizados simultaneamente, foram posicionadas todas as *skins* disponíveis dentro do campo de visão da câmera, e então utilizada essa textura em um quadrilátero que se sobrepõe à interface. O conteúdo das *skins* que ficam para fora de suas molduras individuais não são renderizados por conta do uso do *stencil buffer* que verifica se existe uma moldura por trás do quadrilátero para só então renderizar o pixel da textura. Já na tela da loja, como a visualização dos itens ocorre apenas um por vez, os botões da lista de itens disponíveis são responsáveis por desabilitar a renderização do item atual e habilitar seu item correspondente.

5.4 Sound Design

Por conta da grande influência musical no jogo, certos cuidados foram necessários na hora da implementação do som e suas interações, fazendo com que certas soluções precisassem ser desenvolvidas. A primeira delas é ter a capacidade de sincronizar certos aspectos estéticos, como animações e certos eventos, ao ritmo da música. A segunda implementação surgiu da necessidade de vincular a música ao *level design*, porém a velocidade variável do personagem do jogo impede que este design seja feito diretamente em cima de uma trilha musical, pedindo assim uma solução mais dinâmica.

5.4.1 Sincronização de BPM

Permite sincronizar animações e eventos ao ritmo da música, para isso é necessário informar o BPM (batidas por minuto) da mesma, o que não é problema já que cada música é vinculada à uma fase específica. Utiliza de uma implementação modificada do método sugerido por Christian Floisand[8]. Nele se faz uso de um *listener* que verifica se uma batida ocorreu comparando a posição atual do áudio com a posição esperada da próxima batida. A próxima posição esperada é calculada utilizando a frequência de BPM mais o tempo da última batida. Sendo assim, este algoritmo não suporta músicas cujo BPM se altere ao longo da mesma. Este *listener* por sua vez informa à seus objetos vinculados quando ocorre uma batida, cabendo à eles tratarem desta informação individualmente.

5.4.2 Troca dinâmica de *loops* musicais

Utiliza um grupo de *loops* já determinados linearmente e realiza a troca de um para outro quando necessário, através de um *trigger* em algum evento predefinido, como surgimento de algum inimigo especial, troca de seções da fase ou ao final do *loop* anterior. Pode cortar *loops* musicais e encaixá-lo no próximo. Para isso, após ser ativada a troca de um *loop* para outro, utiliza da sincronização de BPM para aguardar a próxima *downbeat*. Isso permite com que sejam vinculadas segmentos específicos do *level design* à partes das músicas, tornando a transição imperceptível. Além disso também permite com que as músicas tenham segmentos de introdução que fica de fora do *loop* principal da trilha. É necessário ressaltar, porém, que boa parte da qualidade da troca de *loops* de uma música não depende apenas desta implementação, mas sim da forma como a música é produzida, cabendo ao músico encarregado criar segmentos de música que possam encaixar uns nos outros independentemente da *downbeat* em que são cortados.

6 TESTES

Diferentemente dos testes realizados durante o desenvolvimento, que tinham o objetivo de avaliar e identificar mudanças necessárias ao projeto, os testes desta seção visam validar tudo que foi criado e adaptado ao longo deste processo, servindo para atestar o potencial comercial do jogo em questão. Para tal, foram realizados testes com possíveis jogadores, sendo cedido à estes uma *build* do jogo a ser testado e um formulário³ para ser preenchido após o término do teste, tudo de forma remota, e ocorrendo no período de 30 de Setembro até 22 de Novembro. O formulário contém diversas perguntas que abrangem vários aspectos do jogo, com respostas em escala linear, permitindo ao jogador classificar elementos do jogo em uma escala de 1 à 10, de acordo com sua percepção em relação à estes. Ainda, segundo Davis, Steury, e Pagulayan[9], enquanto questões de escala linear permitem identificar áreas no jogo que apresentam problemas, as questões dissertativas colaboram para identificar o motivo associado às notas dadas, contribuindo para que os desenvolvedores possam compreender melhor o feedback recebido. Por conseguinte, após cada questão em escala linear é disponibilizado um campo dissertativo opcional que permite com que o participante discorra um pouco mais sobre sua experiência com o elemento recentemente avaliado. Por fim, algumas questões dissertativas extras permitem avaliar de forma ainda mais profunda a compreensão e experiência que cada participante teve com o jogo.

6.1 Resultados

Com a participação de 24 jogadores, foi possível recolher resultados e *feedbacks* variados, proveniente de indivíduos com diferentes antecedentes em jogos. Nos resultados obtidos, pôde ser visto que os jogadores atribuíram a dificuldade média de 4.25 para compreender a forma de controle do jogo. Após entender como controlar o personagem, estes participantes classificaram com uma média de 5.29 a execução desta forma de controle utilizada pelo jogo. No geral foi indicado clareza da interface gráfica apresentada, com uma média de 8.12 atribuída à facilidade de uso e compreensão desta. O item da estética visual foi o que recebeu as notas mais satisfatórias, com todos participantes marcando valores de 8 ou mais para este item, resultando em uma média de 9.12. A trilha sonora apresentada também recebeu notas altas, com os jogadores atribuindo à trilha uma média de 8.41. Já as mecânicas de jogo e o *level design* apresentados durante os testes foram avaliados com médias de 8.45 e 8.37 respectivamente. Ainda, os 24 jogadores atribuíram uma dificuldade geral de 7.4 ao jogo testado. Finalmente, os participantes demonstraram em sua maioria (91.1%) interesse em adquirir o mesmo em estado completo, com 41.4% indicando certeza de que adquiriria o jogo quando lançado.

³Disponível em: <https://goo.gl/forms/p7UvhUlOUXIOfnhT2>

7 CONSIDERAÇÕES FINAIS

Fazer o uso de uma metodologia e ferramentas que colaborassem com a organização do projeto ao longo do seu processo de desenvolvimento foi essencial. Além disso, certas facilidades inerentes ao motor de jogo escolhido também contribuíram para o projeto chegar ao estado atual. É preciso salientar que o foco no desenvolvimento deste jogo é o de lançamento de um produto ao mercado, o que elevou o grau de atenção dado à diversos aspectos deste, como mostrado anteriormente.

Foi imprescindível adotar um dispositivo padrão para os testes feitos ao longo do desenvolvimento, em especial para precisar o impacto no desempenho das implementações criadas durante o projeto. Neste sentido, foram vitais os cuidados descritos neste artigo com a otimização destas. Estas etapas tomadas para aperfeiçoar as soluções criadas foram o que permitiu com que o dispositivo com requerimentos mínimos escolhido pudesse executar o jogo de forma fiel à experiência planejada, inclusive com uma taxa de quadros acima do esperado, significando que dispositivos um pouco inferiores ao utilizado também poderão ter acesso ao jogo. Mesmo assim, ainda existem outras otimizações planejadas que não foram documentadas neste artigo, com objetivo de melhorar ainda mais o desempenho deste jogo, pois à medida com que as implementações atuais são aperfeiçoadas, mais espaço é aberto para novo conteúdo.

Com os testes de validação, o formato de formulário adotado permitiu receber *feedbacks* que abrangeram diversas particularidades do jogo, como também apresentou a possibilidade de obter comentários mais elaborados dos jogadores. Foi percebido, entretanto, que disponibilizar todas as questões dissertativas como opcionais causou com que alguns participantes não prenchessem nenhum destes campos, não compartilhando possíveis observações que poderiam vir a ser úteis no desenvolvimento. Por conseguinte, viu-se a necessidade de elencar os campos dissertativos mais importantes e tornar sua resposta obrigatória, a fim de recolher um feedback mais rico dos participantes, sempre atentando para não tornar o preenchimento destes uma tarefa muito árdua para possíveis voluntários.

Já analisando os resultado obtidos dos testes, foi possível verificar que a dificuldade do jogo percebida pelos jogadores está relativamente alta, se encaixando em parte com o pretendido para o projeto. Entretanto, a curva de dificuldade precisa ser trabalhada, em especial na primeira fase, onde o jogador ainda está aprendendo a executar as mecânicas principais. Neste sentido, apesar do tutorial passar facilmente ao jogador como este deve controlar o jogo, termina de forma abrupta, sem permitir aos jogadores experimentar mais destas mecânicas. Por conta de apresentar controles não convencionais, é sentido que este momento deve garantir que o jogador sinta-se seguro antes de continuar com a progressão do jogo. Sendo assim, planeja-se modificar o tutorial de forma que seja permitido à quem jogue continuar testando a forma de controle após seu término, dando seguimento ao jogo no momento que se sentir preparado. Além disto, esta solução também pretende atenuar a parcela de jogadores que tiveram uma dificuldade mais elevada em executar o controle do personagem após compreender a forma de realizá-lo.

As notas positivas dadas de forma unânime aos elementos visuais do jogo verificam o sucesso das diversas implementações descritas anteriormente, em criar uma experiência distinta neste sentido. Além disso, a alta taxa de conversão entre visualizações e downloads (mais de 55% no presente momento) da versão enviada ao portal GameJolt, consideravelmente menos polida que a versão atual, também serve para indicar o potencial atrativo do jogo. Similarmente, a facilidade de compreensão e utilização da interface gráfica, também apontada nos testes, e o fato destes terem sido realizados já nas plataformas-alvo, indicam que a interface gráfica está em conformidade com o estipulado para o desenvolvimento e com o esperado para um jogo *mobile*. Mesmo assim, e embora muitas

das soluções apresentadas já tenham este objetivo, são planejados maiores polimentos, buscando melhorar a experiência do usuário e poder acomodar melhor mais conteúdo que será adicionado daqui para frente.

A trilha sonora criada até o momento também foi bem recebida, entretanto notou-se através dos testes que estas podem ficar repetitivas por conta do jogador perder diversas vezes ao longo de uma sessão de jogo, reiniciando a música constantemente. Apesar desta situação ser de certa forma inevitável pela natureza do jogo, este problema pode ser amenizado com uso de algumas das soluções desenvolvidas para gerar um *sound design* mais elaborado, reiterando a importância de implementações propostas anteriormente para esta área, que tiveram sua utilização deixada em segundo plano para esta entrega.

Foi observado que as mecânicas apresentadas durante os testes foram bem-sucedidas em divertir, surpreender e propor o desafio pretendido à quem joga. Aliadas à um *level design* bem estruturado, foi possível engajar o jogador e criar um ambiente que desafia o mesmo à superar os obstáculos propostos. Junto à isto, as regras adotadas para escolha das cores que compõem as fases obtiveram êxito em transmitir ao jogador o que é pretendido. Levando em consideração que ainda existem muitas mecânicas e elementos de *gameplay* planejados para o jogo que não foram apresentados, e que estes foram pensados de forma com que possam interagir entre si para criar situações inusitadas ao jogador, fica claro o potencial do jogo em apresentar uma experiência singular.

Considerando a boa recepção de todos elementos do jogo durante os testes, não é de se surpreender que a grande maioria dos participantes tenham demonstrado interesse em adquirir o jogo testado quando em estado final, com alguns inclusive demonstrando interesse em adquiri-lo no estado atual, o que apenas corrobora com o potencial comercial do mesmo.

Dando continuidade ao desenvolvimento, se planeja gerar mais conteúdo para o jogo, finalizando as fases restantes, utilizando-se tanto de mecânicas já mostradas como também das não utilizadas. Serão também acrescentados mais *skins* e itens. Ainda, sempre há a preocupação de aumentar o polimento no geral. Incrementar o *sound design* também ajudará a elevar o nível da experiência como um todo. Com mais *skins* disponíveis, fases, e uma interface bem resolvida, é possível buscar o apoio de uma *publisher* para a divulgação e publicação do projeto no mercado.

REFERÊNCIAS

- [1] SuperData Research. (2016, Dez) 2016 Year in review. [Online]. Available: <https://www.docdroid.net/RPwDoyG/superdata-year-in-review-2016.pdf>
- [2] D. Alexander. (2016, Set) A brief history of endless runners. [Online]. Available: <http://retrogamingmagazine.com/2016/09/10/brief-history-endless-runners/>
- [3] M. Porter. (2017, Abr) Super mario run downloaded 10 times more than fire emblem heroes, makes less money. [Online]. Available: <http://www.ign.com/articles/2017/04/27/super-mario-run-downloaded-10-times-more-than-fire-emblem-heroes-makes-less-money>
- [4] M. Kearn. (2016, Jul) 30 essential stats on in-app purchases to know. [Online]. Available: <https://www.appboy.com/blog/in-app-purchase-stats/>
- [5] I. Schreiber. (2009, Jul) Game design concepts. [Online]. Available: <https://gamedesignconcepts.wordpress.com/2009/07/02/level-2-game-design-iteration-and-rapid-prototyping/>
- [6] Unity Technologies. (2017, Mar) Mobile hardware stats. [Online]. Available: <https://hwstats.unity3d.com/mobile/display.html>
- [7] Encyclopædia Britannica. (2017, Set) Hooke's law. [Online]. Available: <https://www.britannica.com/science/Hookes-law>
- [8] C. Floisand. (2014, Jan) Beat synchronizer in unity. [Online]. Available: <https://github.com/cflosand/beat-synchronizer-unity>
- [9] J. P. Davis, K. Steury, and R. Pagulayan, "A survey method for assessing perceptions of a game: The consumer playtest in game design," *Game Studies*, vol. 5, no. 1, 2005.