Universidade Federal
do Rio de Janeiro
Escola Politécnica

XS-GAME: ENGENHARIA DE JOGOS VOLTADA PARA DESENVOLVEDORES INDIVIDUAIS

Brian Rocha Confessor

Projeto de Graduação apresentado ao Curso de Engenharia de Computação de Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores: Geraldo Xexéo
Eduardo Mangeli

Rio de Janeiro

Março de 2019

XS-GAME: ENGENHARIA DE JOGOS VOLTADA PARA DESENVOLVEDORES INDIVIDUAIS

Brian Rocha Confessor

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO DE INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO.

Autor:

Brian Rocha Confessor

Brian Rocha Confessor

Orientador:

Prof. Geraldo Bonorino Xexéo, D. Sc.

Orientador:

Eduardo Freitas Mangeli de Brito, M. Sc.

Examinador:

Prof. Jano Moreira de Souza, Ph. D.

Examinador:

Marcelo Arêas Rodrigues da Silva, D. Sc.

Rio de Janeiro - RJ, Brasil

Março de 2019

Confessor, Brian
C748x XS-GAME: ENGENHARIA DE JOGOS VOLTADA PARA DESENVOLVEDORES INDIVIDUAIS / Brian Confessor. -- Rio de Janeiro, 2019.
80 f.

Orientador: Geraldo Xexéo.
Coorientador: Eduardo Mangeli.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Escola Politécnica, Bacharel em Engenharia de Computação e Informação, 2019.

1. Engenharia de Software. 2. Engenharia de Jogos. 3. Prototipagem. 4. Jogos Digitais. I. Xexéo, Geraldo, orient. II. Mangeli, Eduardo, coorient. III. Título.

"It matters not how strait the gate,
How charged with punishments the scroll,
I am the master of my fate,
I am the captain of my soul."

- **William Ernest Henley, 'Invictus'**

Dedicatória

Dedico este trabalho à minha avó Cleonice, pois sem ela não teria chegado até aqui.

Agradecimentos

Agradeço primeiramente à Deus no plano celestial, e espero que Ele continue a me proporcionar a energia e dedicação que preciso para alcançar novos patamares. Sem Sua vontade e Sua benevolência não existiria ou estaria onde estou hoje.

Em segundo lugar, agradeço à minha avó Cleonice no plano terrestre, o anjo que Deus pôs na Terra para me criar e guiar da melhor forma possível, sempre estando presente e disponível para me auxiliar, por mais que muitas vezes não fosse digno de tamanho carinho e amor. Sem você não seria metade do ser humano que sou hoje, e nada disso seria possível.

Agradeço ao meu tio Henrique, que desde minha infância me mostrou as maravilhas do campo de estudo em que me encontro hoje, e por me auxiliar sempre que estou com dúvidas e dificuldades. Sem você não estaria no campo e curso que estou hoje.

Agradeço à minha irmã Isabelle, aos meus pais Lester e Andrea, e aos meus avós Zara e Rildo, que apesar das eventuais desavenças, sempre acreditaram no meu potencial e me apoiaram quando estava em maus momentos. Sem seu auxílio não conseguiria chegar tão longe.

Agradeço aos meus amigos, tanto àqueles que ainda se encontram presentes em minha vida quanto àqueles que já se foram por algum motivo, por terem tido paciência comigo por todos esses anos, terem compartilhado as alegrias e as tristezas, comemorações e momentos de mágoas e quase desistências. Todos tiveram uma importância na minha vida, e sem vocês não suportaria tudo o que suportei para chegar aqui.

Agradeço também aos professores dos quais já fui aluno, que contribuíram para minha formação do longo desses anos, tanto profissionalmente quanto pessoalmente. Sem vocês com certeza não teria me tornado quem me tornei.

Por fim, mas não menos importante, agradeço à sociedade brasileira por arcar com os custos da minha faculdade, mesmo que de maneira indireta e, por vezes, indesejada. Espero que possa contribuir tanto profissionalmente quanto pessoalmente para dar um retorno a este investimento.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

XS-GAME: ENGENHARIA DE JOGOS VOLTADA PARA DESENVOLVEDORES INDIVIDUAIS

Brian Rocha Confessor

Março de 2019

Orientadores: Geraldo Xexéo e Eduardo Mangeli
Curso: Engenharia de Computação e Informação

Ao longo das últimas décadas, a indústria de jogos digitais vem sofrendo um crescimento acelerado, sendo composta por equipes de desenvolvimento de diversos tamanhos e com títulos cada vez mais complexos e polidos.

Com o aumento ambicioso da escala dos produtos criados, aumenta-se também os riscos associados a possíveis atrasos ou cancelamentos de projetos. Por esse motivo, é interessante para tais empresas que busquem maneiras de maximizar a probabilidade de sucesso de seus jogos, como a criação de um protótipo, para validar suas mecânicas e conceitos principais antes de se iniciar o desenvolvimento do produto final, preferencialmente utilizando o mínimo de recursos possível.

Este projeto, portanto, propõe um modelo de desenvolvimento de software, adaptado a partir de modelos existentes na literatura, e voltado para desenvolvedores individuais, visando a criação e entrega de um protótipo de jogo digital e artefatos que possam auxiliar terceiros na validação do protótipo criado.

Palavras-Chave: Engenharia de Software, Engenharia de Jogos, prototipagem, jogos digitais.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

XS-GAME: GAME ENGINEERING FOR SOLO DEVELOPERS

Brian Rocha Confessor

March/2019

Advisors: Geraldo Xexéo and Eduardo Mangeli
Course: Computer and Information Engineering

For the past decades, the game industry has been growing at a fast pace, comprised of development teams of different sizes and ever more complex and polished titles.

With this ambitious scale-up of products, comes an equally scaled risk associated with possible delays or cancellations of a project. Therefore, such companies should strive to find ways to maximize their games' probability of success, such as creating a prototype which can validate its mechanics and core concepts prior to the final product's development, preferably using the least amount of resources possible.

With this in mind, the current project proposes a software development process adapted from existing processes, aimed at solo developers, with the goal of creating and delivering a digital game prototype, along with assets that can help third parties validate said prototype.

Keywords: Software Engineering, Games Engineering, Prototyping, digital games

Sumário

Lista de Siglas e Abreviaturas

XP: *Extreme Programming*

GDD: *Game Design Document*

WIP: *Work in Progress*

LSI**:** Limitante Superior Inicial

UI: *User Interface*

Capítulo 1

1. Introdução

1.1. Motivação:

Desde a década de 1950, jogos eletrônicos como *Bertie the Brain* [1] começaram a surgir em nossa sociedade, e o avanço exponencial da tecnologia ao longo dos anos trouxe inúmeras melhorias para esse campo. São inegáveis o crescimento e a importância da indústria de jogos no paradigma atual da sociedade, com mais de US$130 bilhões movimentados pela mesma em 2018 [2]. De pequenos times até empresas com equipes de projeto na casa das centenas ou até milhares de indivíduos [3,4], o aumento exponencial de uma indústria desse porte gera muitos empregos e é uma parte relevante da economia, porém também traz consigo riscos igualmente elevados.

Cancelamentos de projetos na indústria não são incomuns, e dependendo do escopo do projeto podem ocasionar prejuízos exorbitantes, como os milhões perdidos pela Blizzard com o cancelamento de Titan [5]. Dentre os diversos motivos para tal, pode-se apontar problemas de escopo, conceito de jogo ou mecânicas idealizadas como possíveis causadores de adversidades. De maneira similar, problemas nos conceitos fundamentais de um projeto podem ser encontrados somente após a entrega, o que torna o seu custo de manutenção extremamente mais elevado, como apontado por Dawson *et. al.* [6].

Para buscar mitigar ao máximo tais ocorrências, uma possível estratégia seria a criação de um protótipo do jogo, como forma de explorar uma versão mínima da aplicação das mecânicas, conceitos e estilos propostos, e buscar obter melhorias em cima dos mesmos, executar modificações em suas bases, ou mesmo desistir da ideia inicial, caso o resultado não esteja do agrado da equipe de desenvolvimento.

Tal abordagem pode ser utilizada tanto por equipes desenvolvendo um protótipo para suas próprias companhias, quanto por desenvolvedores que desejam oferecer ideias

para empresas maiores. Um exemplo conhecido do segundo caso foi Super Smash Bros., lançado em 1999, cuja equipe de prototipagem da HAL Laboratories, que buscava apresentar um conceito de jogo já trabalhado para a Nintendo, envolveu apenas três desenvolvedores [7].

Ainda assim, não é trivial construir um protótipo dessa natureza, sobretudo por equipes de pequeno porte, que compõem atualmente uma parcela significativa da indústria. Segundo um relatório da Unity realizado em 2018, 59% das 1445 empresas entrevistadas pela empresa [8] possuem até cinco integrantes, enquando um censo nacional realizado em 2018 [9] pelo Ministério da Cultura apontou que 38,6% das 99 empresas não formalizadas e 37,8% das 276 empresas formalizadas possuem o mesmo limite superior. Portanto, a fim de economizar recursos financeiros e humanos, sobretudo por desenvolvedores individuais que possuem uma pequena quantidade de ambos, seria interessante a existência de uma metodologia de pequeno porte, capaz de auxiliar no desenvolvimento rápido e eficaz de um protótipo, permitindo aos desenvolvedores conseguirem em um curto espaço de tempo informações acerca de suas ideias, e adaptá-las ou refazê-las prontamente.

Com base em tais fatos, o presente trabalho visa propor uma adaptação de modelos ágeis existentes destinado ao desenvolvimento de protótipos de jogos digitais, com foco particular em desenvolvedores individuais.

1.2. Objetivos:

1.2.1. Objetivo Geral:

O presente trabalho tem dois objetivos principais. A primeira parte visa realizar o desenvolvimento de um protótipo de jogo digital, adaptando modelos já existentes na literatura apresentados no Capítulo 2 para tal projeto. Na segunda parte, será realizada a formalização de tais adaptações utilizadas na primeira parte, culminando na proposição de um modelo adaptado para a criação de protótipos de jogos digitais por desenvolvedores individuais.

1.2.2. Objetivos Específicos:

- Buscar dentro da literatura de Engenharia de Software e Engenharia de Jogos metodologias adequadas e relevantes para desenvolvimento de jogos digitais, e propor um modelo adaptado com base em tais metodologias com foco particular na criação de protótipos por apenas um desenvolvedor.
- Desenvolver um protótipo de um jogo eletrônico baseado na metodologia proposta anteriormente.
- Realizar uma avaliação da aplicação desenvolvida e um relato da experiência de desenvolvimento.

1.3. Organização do Trabalho:

O presente trabalho se dispõe da seguinte maneira:

Capítulo 1. Introdução: Apresenta motivações e objetivos do trabalho

Capítulo 2. Fundamentação Teórica: Apresenta e descreve os conceitos teóricos relevantes ao trabalho.

Capítulo 3. Desenvolvimento do projeto: Apresenta as ferramentas utilizadas no presente projeto, a tomada de decisões acerca do mesmo, e detalha as etapas do processo de desenvolvimento.

Capítulo 4. XS-Game: Descreve a metodologia XS-Game, proposta do presente trabalho, e detalha cada uma de suas etapas.

Capítulo 5. Considerações Finais: Apresenta a conclusão do processo, limitações encontradas durante seu desenvolvimento, e propostas de trabalhos futuros.

Capítulo 2

2. Fundamentação Teórica

O presente capítulo visa fornecer informações acerca da definição de Engenharia de Software tradicional e Engenharia de Software voltada para jogos, bem como descrever conceitos que serão utilizados durante o desenvolvimento do projeto, como metodologias ágeis. Para tal, separamos o capítulo em duas partes principais. Na seção 1, serão apresentadas definições sobre a Engenharia de Software tradicional, descrevendo métodos clássicos e métodos ágeis relevantes ao projeto, enquanto na Seção 2 será discutida a Engenharia de Software voltada para jogos, além de conceitos fundamentais de jogos em si.

2.1. Engenharia de Software

2.1.1. Definição

Segundo Schach [10], Engenharia de Software pode ser definida como uma disciplina cujo objetivo é a produção de software livre de falhas, que satisfaz as necessidades do usuário, e é entregue dentro do prazo e orçamento estabelecidos. Para alcançar tal objetivo, técnicas apropriadas devem ser usadas ao longo da produção do software durante as fases de especificação, *design*, desenvolvimento e manutenção do sistema após sua entrega. A Engenharia de Software se mostra presente em todas as fases do ciclo de vida do software e pode incorporar diversos campos de estudo, tornando-se assim um campo multidisciplinar.

A divisão de etapas dentro de um modelo de ciclo de vida varia entre os modelos e seus nomes, mas alguns aspectos comuns à maioria deles podem ser observados. Podemos caracterizar sucintamente as principais atividades de um ciclo de vida como:

- Análise Preliminar: Busca descobrir os objetivos do cliente para com o software requisitado; em outras palavras, busca descobrir o que o cliente deseja alcançar

com o uso do software planejado, para assim poder propor soluções alternativas e melhor descrever os requisitos do projeto na etapa seguinte à essa.

- Análise e Levantamento de Requisitos (Planejamento): Define os requisitos funcionais e não-funcionais do sistema a ser desenvolvido. Tais requisitos, que podem ser condições ou capacidades que o produto deve alcançar ou restrições que o mesmo terá, devem possuir certas características, como a não-ambiguidade, completude, corretude e necessidade. Nesta etapa, é gerado o Documento de Requisitos do Sistema, que será usado como referência ao longo do projeto pela equipe de desenvolvimento.
- *Design*: Enquanto os requisitos ditam o que o produto deve fazer, o *design* (ou projeto) dita como o produto deve fazê-lo. O objetivo dessa atividade é refinar os artefatos gerados na etapa de Análise, a fim de moldá-los em um formato que possa ser melhor implementado pelos programadores. Para isso, a equipe de projeto decompõe o produto em módulos, pedaços independentes de código com interfaces bem definidas e documentadas.
- Implementação: Realiza o desenvolvimento do projeto em si, isto é, os programadores se utilizam dos artefatos criados a partir das fases anteriores para elaborar um sistema que esteja em conformidade com as demandas do cliente.
- Testes: Na etapa de testes, a equipe de desenvolvimento cria Casos de Teste, especificações de parâmetros de entrada, condições de execução e passos detalhados e reproduzíveis, que serão executados em testes no projeto para minimizar possíveis defeitos nos módulos implementados, e averiguar se tais módulos estão de fato em conformidade com o Documento de Requisitos. Inicialmente são realizados testes unitários em cada módulo desenvolvido, e após sua validação ocorre a união dos módulos e subsequentes testes de integração. Por fim, após um teste de aceitação por parte do cliente para que o mesmo aprove o que foi implementado, o produto se encontra pronto para o lançamento.
- Lançamento e manutenção: Uma vez criada e lançada a primeira versão do produto, a equipe deve realizar a manutenção do produto, a fim de reparar quaisquer erros que possam ter passado despercebidos durante a fase de testes, além de possivelmente efetuar a implementação de novos requisitos para o sistema, caso necessário.

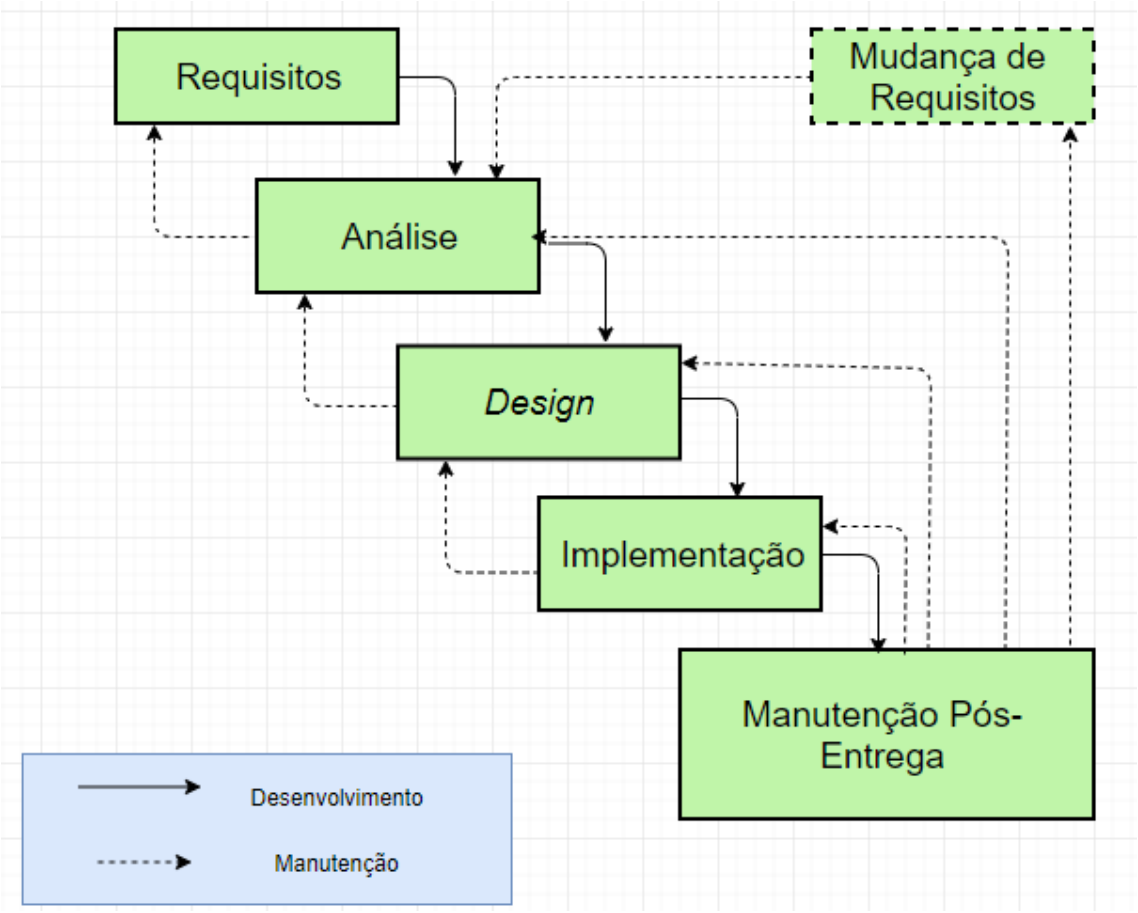

Figura 1 - Modelo de ciclo de vida Cascata.

Fonte: Schach [12].

Existem diversos modelos de ciclo de vida na Engenharia de Software, como o modelo Cascata [11], visto na Figura 1, que consiste na execução sequencial das fases do seu ciclo de vida, com pequeno espaço para mudanças ou retornos à fases anteriores; modelo de prototipagem rápida [12], que tem como característica marcante a criação rápida de um protótipo para interação com o cliente e coleta de *feedback*, para então seguir com o ciclo de vida a partir de tal *feedback*; e modelos ágeis, como *Scrum* e *Extreme Programming*. Os modelos ágeis são explicados com mais detalhe na seção 2.2.

2.2. Métodos Ágeis

2.2.1. Definição

Desenvolvimento Ágil de Software é um conjunto de métodos e práticas baseados nos valores e princípios do Manifesto Ágil [13]. Tal tipo de desenvolvimento

começou a tomar forma no final dos anos 90, quando diversas metodologias que enfatizavam a colaboração entre a equipe de desenvolvimento e os *stakeholders* começaram a surgir. Outros princípios de tais metodologias, como entrega frequente de pequenos incrementos do produto (ao contrário de metodologias de desenvolvimento mais tradicionais como o desenvolvimento cascata, que só entregavam o produto pronto em sua totalidade ao final do ciclo de vida), maior importância dada às relações interpessoais dentro da equipe de desenvolvimento, e uma aceitação maior da mudança de requisitos em estágios mais avançados do desenvolvimento também foram incorporados aos princípios desse manifesto [14].

Dentre os diversos tipos de desenvolvimento Ágil existentes na atualidade, dois são relevantes para o presente trabalho, que apresenta características em comum com os tipos de projetos para os quais tais métodos costumam ser propostos. Tais métodos são o *Scrum* e *Extreme Programming* ("Programação Extrema", em português, também conhecido como XP), que são apresentados em mais detalhe adiante.

2.2.2. *Scrum*

Ken Schwaber, co-criador do *Scrum*, diz que 'se você tem uma visão, e ela faz parte de um campo complexo ou até caótico, e você quer descobrir se consegue criar algo útil a partir desta visão, sem desperdiçar dinheiro buscando algo que não vai funcionar, *Scrum* é uma solução' [15].

De acordo com seu site oficial [16], *Scrum* é um *framework* co-criado por Ken Schwaber e Jeff Sutherland, utilizado para resolução de problemas complexos e adaptativos, centrado na colaboração de equipes, empirismo, e em uma coleção de componentes interligados. Originalmente modelado a partir do artigo de Hirotaka Takeuchi e Ikujiro Nonaka [17], seu uso é vasto, podendo ser utilizado para pesquisar e identificar possíveis mercados, desenvolver produtos lançamentos de novas funcionalidades com alta frequência, e realizar manutenções de produtos já existentes.

De acordo com o guia do *Scrum* [18], podemos ver alguns aspectos primordiais de sua teoria, como:

- Transparência: aspectos relevantes do processo devem ser visíveis para aqueles responsáveis pela busca do objetivo

- Inspeção: Usuários devem frequentemente inspecionar artefatos e progresso para detectar variações indesejáveis.
- Adaptação: Se a inspeção revelar aspectos indesejáveis e fora dos limites permitidos, o processo ou material processado deve ser ajustado.

Além disso, o guia descreve em detalhes a Equipe *Scrum*, com cada um de seus componentes-chave. Os componentes principais são:

- *Product Owner*: É o indivíduo responsável por descrever elaboradamente os itens do *Product Backlog*, que é uma lista de todas as coisas que precisam ser implementadas dentro do projeto. Além disso, o *product owner* é responsável por ordenar esses itens de modo a melhor alcançar os objetivos.
- Equipe de desenvolvimento: Grupo de profissionais cujo dever é entregar um incremento 'concluído' do projeto ao final de cada *Sprint*. A equipe é auto-organizada (isto é, não há ordens externas ditando o papel de cada integrante durante o desenvolvimento), não cria títulos para os seus membros nem cria sub-times para domínios específicos. O tamanho da equipe deve ser pequeno o suficiente para manter sua flexibilidade, mas grande o suficiente para conseguir implementar os itens dispostos em cada *Sprint*.
- *Scrum Master*: Responsável por promover e fiscalizar o uso dos princípios do *Scrum* como descritos no Guia do *Scrum*, auxiliando os demais integrantes da equipe a entender a teoria, regras, valores e práticas do *framework*.

O desenvolvimento sob o sistema do *Scrum* funciona primariamente como uma sequência de eventos de tamanho fixo, chamados *sprints*, representados na Figura 2. Durante uma *sprint*, que tem duração máxima de um mês, a equipe de desenvolvimento deve implementar um incremento 'concluído' e utilizável para o produto. *Sprints* possuem uma fase de Planejamento de *sprint*, *Scrum* Diário, a etapa de desenvolvimento em si, a Revisão da *Sprint* e a Retrospectiva da *Sprint*.

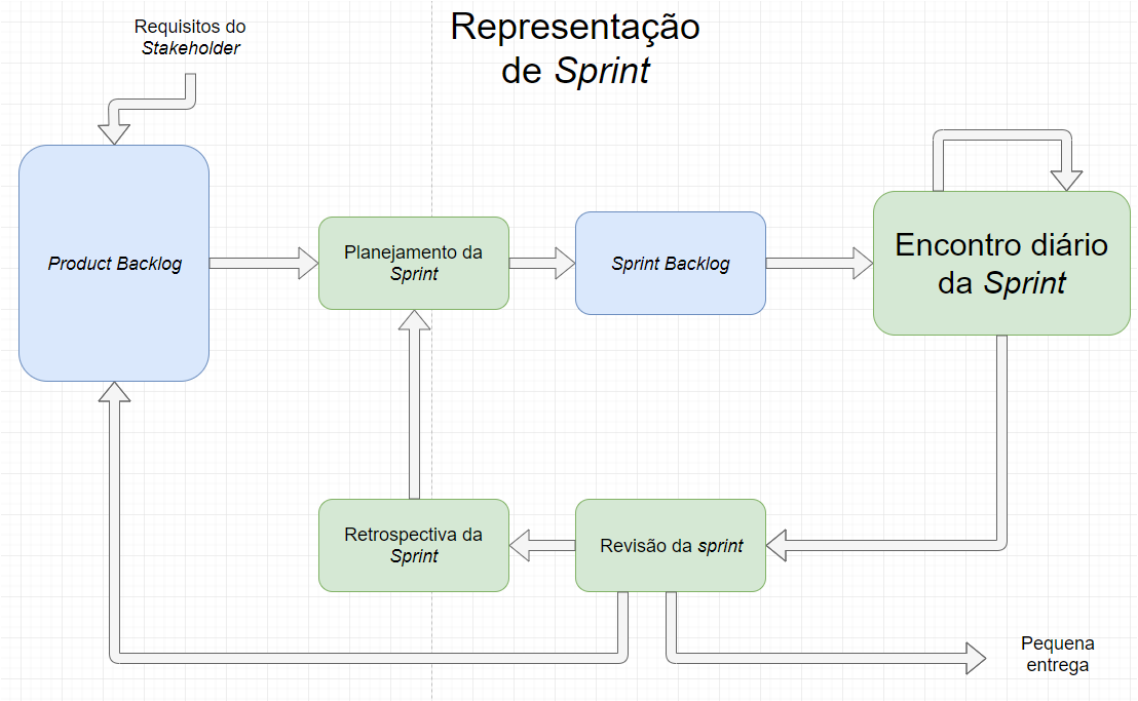

Figura 2 – Representação de uma *Sprint*.

Fonte: Site do *Scrum* [16].

Na fase de Planejamento, a equipe se reúne para definir o que pode ser entregue no incremento resultante da *sprint* planejada, e como o trabalho será dividido. Cria-se nessa etapa um *sprint backlog*, que contém as Histórias a serem desenvolvidas durante a *sprint* vigente, e a ordem de implementação de tais Histórias escolhidas pode ser modificada em função da sua prioridade pelo *Product Owner*.

A quantidade de Histórias de uma *Sprint* pode ser limitada pela Velocidade [19] da *sprint*. A Velocidade é uma métrica calculada a partir da soma dos pontos das Histórias completamente implementadas durante a *sprint* anterior, e serve para auxiliar a equipe de desenvolvimento a evitar um preenchimento excessivo ou insuficiente de cada *sprint backlog*. Além disso, também deve ser criado um objetivo para a *sprint*, isto é, o que a equipe de desenvolvimento pretende alcançar com aquela *sprint*. Um detalhe importante é que, uma vez escolhidos os itens que constituirão o atual *sprint backlog*, não se pode adicionar ou retirar itens do mesmo.

Após o planejamento, a fase de desenvolvimento da *sprint* costuma ter um *Scrum* Diário, que é uma reunião com a equipe de desenvolvimento que dura no máximo 15 minutos. Seu objetivo é planejar as próximas 24 horas de trabalho, além de possivelmente discutir o desenvolvimento do dia anterior, e se há algo que impede a conclusão da *sprint* com o objetivo alcançado. Um possível artefato gerado como parte

dessa etapa é um gráfico de *burndown* [20], que detalha graficamente a quantidade de itens do *sprint backlog* ainda não finalizados ao longo do tempo. Tal artefato é útil para se ter uma noção de qual o prazo estimado de término de cada item remanescente.

Ao final da fase de desenvolvimento da *sprint*, temos sua revisão. Na Revisão da *Sprint*, que ocorre no final de uma *sprint*, o time e os *stakeholders* inspecionam e discutem informalmente sobre o incremento produzido durante a *sprint* vigente. O resultado desta revisão é um *product backlog* revisado que definirá os possíveis itens presentes no próximo *sprint backlog*.

Por fim, a Retrospectiva da *Sprint* é um evento que ocorre após uma Revisão, mas antes do próximo Planejamento de *Sprint*, e visa a auto-inspeção por parte da equipe de desenvolvimento, buscando melhorias em seus relacionamentos, maneiras de trabalho, entre outros aspectos importantes para a otimização do desenvolvimento do produto.

2.2.3. Extreme Programming

Criada por Kent Beck, engenheiro e um dos criadores do Manifesto Agile, o *Extreme Programming (*"Programação Extrema", em português), também conhecido por XP, é uma metodologia voltada para equipes de pequeno a médio porte, para uso em projetos que possuem como uma de suas características uma constante mudança de requisitos ao longo do projeto.

Em seu livro sobre a metodologia [21], Beck explica que a parte "extrema" presente no nome da metodologia deve-se ao fato da mesma levar ao extremo boas práticas da Engenharia de Software. Exemplos disso incluem a prática constante de revisão de código (prática chamada *pair programming*, ou "programação em pares"), integração e testes constantes (integração contínua), e constantes e curtas iterações de desenvolvimento (conhecidas como *The Planning Game*, ou "O Jogo do Planejamento", e pode ser visto na Figura 3).

Vemos em [22,23] que, assim como *Scrum,* XP também aceita que requisitos irão mudar com o passar do tempo, e aceita tal fato ao invés de lutar para impedi-lo. Outras regras relevantes da metodologia são o uso de Histórias [24] e Épicos [25], similares a casos de uso e possíveis substitutos de um longo documento de requisitos; pequenos lançamentos iterativos com frequência; refatoração de código quando

possível; criação de funcionalidades apenas quando necessário, e não antes; entre outras [1].

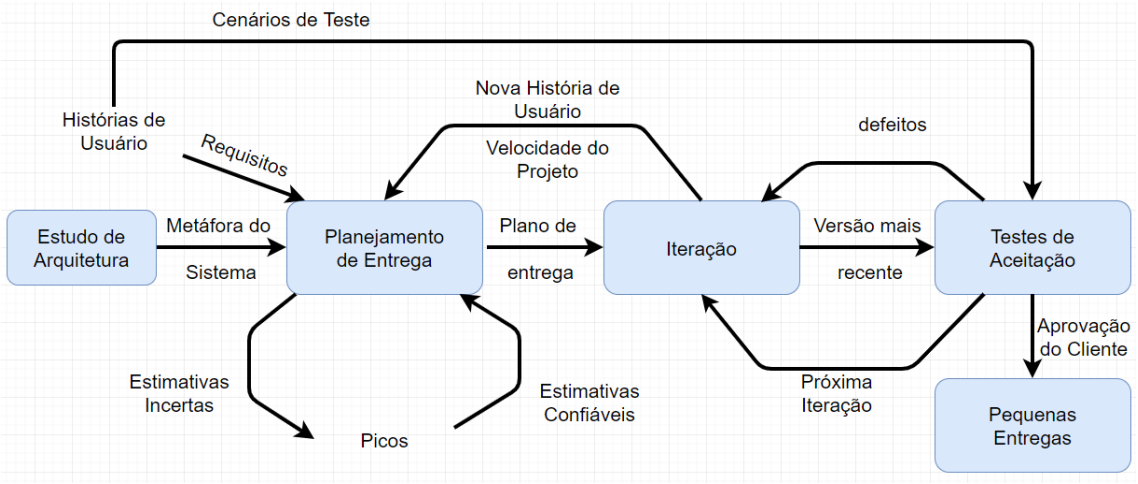

Figura 3 - Fluxograma de um processo de Extreme Programming.

Fonte: Site do Extreme Programming [26].

A metodologia XP é baseada em 5 valores primordiais [27]:

- Simplicidade: Implementar o requisitado, e não mais. Assim, se maximiza o valor do produto criado.
- Comunicação: É necessária a comunicação entre os membros da equipe, a fim de otimizar a solução de problemas e o desenvolvimento como um todo.
- *Feedback*: Deve-se discutir o projeto e o resultado das iterações já entregues, a fim de aprimorar o processo de desenvolvimento.
- Respeito: Deve-se manter o respeito entre os membros da equipe e suas opiniões.
- Coragem: É necessária honestidade sobre o progresso e estimativas do projeto, a fim de poder entregar o produto final com mais agilidade e qualidade.

2.2.4. Kanban

Kanban (do japonês 看板, literalmente “letreiro” ou “quadro de avisos”) é um *framework* de gerenciamento de criação de produtos criado pela Toyota no final dos

[1] A lista completa de regras da metodologia pode ser encontrada com mais detalhes em http://www.extremeprogramming.org/rules.html

anos 40. De natureza altamente visual e com ênfase em entregas contínuas, permitia que equipes de desenvolvimento trocassem informações com mais facilidade sobre o que era necessário de se implementar, de tal forma que não coloque trabalho em demasia de uma vez para a equipe de desenvolvimento, além de reduzir gastos e maximizar o valor do produto. Itens em desenvolvimento são visualizados num *Kanban Board* (“Quadro de Kanban”, em português, visto na Figura 4), o que permite que a equipe de desenvolvimento tenha noção do progresso do projeto a qualquer momento [28]. O *framework* também visa limitar a quantidade de trabalho sendo feito ao mesmo tempo, ao criar valores numéricos, ou “pesos”, para cada item de trabalho, e implantar limites para cada uma das 3 áreas do Quadro (“A fazer”, “fazendo”, “feito”). Assim, nenhuma área pode ter um “peso total” (dado como a soma dos pesos dos itens presentes naquela área) maior do que o seu limite. Isso auxilia também na busca de gargalos no processo de desenvolvimento [29].

Atualmente, uma ferramenta utilizada por equipes de desenvolvimento que utilizam Kanban é o Trello [30], que permite a rápida criação e compartilhamento de quadros de avisos entre diversos usuários.

Figura 4 - Exemplo de um Quadro de Kanban

Fonte: Produzido pelo autor

O Kanban é baseado em três princípios básicos [31]:

- Visualizar o que pode ser feito: Também conhecido como *workflow*, permite a visualização e comparação dos itens a serem feitos, a fim de ter um entendimento maior do estado atual do projeto.
- Limitar a quantidade de trabalho em progresso: O trabalho em progresso (conhecido em inglês pelo termo "*work in progress*", ou WIP) deve ser controlado, a fim de manter a abordagem fluida do Kanban, impedindo que a equipe de desenvolvimento se comprometa a entregar muita coisa de uma vez.
- Melhorar o fluxo: Quanto um pedaço de trabalho é finalizado, o próximo item de maior prioridade presente no *backlog* é trazido para o quadro.

Tanto Kanban quanto *Scrum* focam em entregar *software* de forma rápida e frequente, porém existem algumas diferenças entre ambos, como as evidenciadas abaixo:

Tabela 1 - Comparação entre *Scrum* e Kanban.

Scrum	**Kanban**
Sprints com janela de tempo fixa	Entrega contínua
Trabalho coletado em lotes (*backlog* da *sprint*)	Trabalho coletado em itens individuais
Mudanças dentro de uma *sprint* não são permitidas	Mudanças podem ocorrer em qualquer momento
Indicado para situações onde trabalho pode ser separado em lotes imutáveis	Indicado para situações com um alto nível de variabilidade nas prioridades de cada item

Como pode ser visto, o sistema do Kanban é mais indicado para situações onde a prioridade de cada requisito pode sofrer mudanças ao longo do projeto, além de permitir uma entrega mais rápida de partes do projeto ao diminuir a duração dos ciclos de desenvolvimento e auxiliar visualmente a equipe com o Quadro de Kanban.

2.2.5. Comparação entre XP e *Scrum*

Gill e Henderson-Sellers [32] detalham em seu artigo uma comparação entre os dois métodos, ressaltando pontos de concordância e divergência entre os mesmos em aspectos como escopo, grau de agilidade, valores ágeis e processo de software. Embora *Scrum* e XP, por terem origens similares, possuam várias similaridades, também possuem alguns aspectos básicos que os tornam distintos. Com base em Gill e Henderson-Sellers [33], além de pontos já mencionados nas seções 2.2.2 e 2.2.3., podemos destacar algumas distinções de cada metodologia que são relevantes para o presente trabalho, descritos na tabela abaixo:

Tabela 2 - Comparação entre XP Programming e *Scrum*.

Aspecto	**XP**	***Scrum***
Tempo de iteração	Uma a duas semanas.	Duas semanas a um mês.
Flexibilidade de mudança em iterações	Permite mudanças dentro de iterações (desde que item retirado não tenha tido sua implementação iniciada).	Não permite mudanças dentro de suas iterações (*sprints*) uma vez iniciadas.
Prioridade de tarefas	Definido pelo cliente, seguida rigorosamente pela equipe.	Definida pelo *Product Owner*, mas a equipe pode decidir a ordem de implementação.
Práticas de engenharia	Estabelece práticas de engenharia nos projetos, como o uso de programação em pares, refatoramento de código, e utilização de Histórias para levantar os requisitos do projeto.	Não especifica nenhuma prática de engenharia a ser seguida.

Para o presente trabalho, foram escolhidas características de cada metodologia que mais se encaixavam na natureza do projeto como um desenvolvimento de jogo digital, além de levar em consideração também à configuração da equipe de desenvolvimento.

2.3. Engenharia de Jogos

Além da importância do entendimento da Engenharia de Software clássica para o desenvolvimento do trabalho, também há vital importância na análise da engenharia específica à tal campo, a fim de entender similaridades e diferenças entre os processos de desenvolvimento de cada uma.

A Engenharia de Jogos aplica princípios e técnicas da Engenharia de Software clássica ao desenvolvimento de jogos eletrônicos, a fim de agilizar o processo de desenvolvimento e diminuir a chance de falhas estarem presentes na versão final do produto. Porém, não é trivial se utilizar de Engenharia de Software na criação de um jogo, pois diversos aspectos primordiais que os caracterizam diferem de software genéricos, criando a necessidade de adaptações e modificações no processo de desenvolvimento do produto.

2.3.1. Diferenças entre Engenharia de Software e Engenharia de Jogos

Devido a aspectos únicos presentes em jogos que não se encontram em softwares de modo geral, como artefatos artísticos, musicais ou literários, é de se esperar que alguma adaptação do processo tradicional de desenvolvimento seja necessária, de modo a englobar tais áreas e otimizar o processo como um todo.

Wurzer e Lichtl [34] apresentam algumas diferenças fundamentais entre projetos de jogos e projetos de software em geral. Uma dessas diferenças é a métrica de sucesso para o projeto: enquanto um software genérico pode ter seu sucesso medido a partir da análise de conformidade de todos os seus requisitos, para um jogo, apenas estar implementado não é o suficiente; há diversos outros aspectos a se considerar na medida de sucesso de um jogo, como seu conteúdo, fator de diversão e fluxo. Portanto, vemos que um jogo é muito mais que um software genérico, englobando diversas áreas de conhecimento.

Outra diferença crucial são as fases de desenvolvimento. Enquanto softwares comuns têm seu ciclo de vida iniciado normalmente na fase de análise(especificação), jogos possuem uma fase anterior à esta, que é a fase de conceito de jogo.

Dentro desta fase, a equipe deve procurar ideias do que criar, cenários presentes no jogo, rascunhos de arte e mecânicas presentes no jogo, entre outras tarefas de natureza mais criativa. Apenas após ter encontrado esse conceito inicial, mesmo que em linhas gerais, pode-se dar prosseguimento ao ciclo de vida do projeto. Por ser um trabalho de natureza extremamente subjetiva e criativa, é difícil estimar um prazo para a completude de tal tarefa. Empresas costumam dar um tempo para seus funcionários da área criativa descansarem e assim retornarem ao trabalho com ideias frescas para o próximo ciclo de desenvolvimento. Wurzer e Lichtl citam como exemplo disso a companhia Origin Software, que após o término do desenvolvimento de um jogo dá férias à seus artistas e outros funcionários de áreas criativas, para que retornem no próximo ciclo de desenvolvimento com mentes descansadas e ideias novas.

Nesta fase também podem ser implementados pequenos protótipos, normalmente na forma de protótipos de papel [35], para testar a estética e mecânica idealizadas pela equipe e checar se as mesmas são funcionais e condizentes com o conceito criado até o momento. A vantagem do uso dessa técnica, bem comum no desenvolvimento de jogos, é que são rápidos e fáceis de implementar e modificar, não requerem conhecimentos técnicos, e fornecem um teste rápido das ideias iniciais de mecânica e dinâmica de jogo do projeto, podendo assim apontar falhas e melhorias no conceito inicial.

Um dos artefatos mais importantes que resultam da fase de conceito de jogo é o Documento de Design Geral (“Overall Design Document”, mais conhecido como “*Game Design Document*”, ou GDD). No GDD, que costuma ser escrito por várias pessoas de diferentes áreas da equipe de desenvolvimento, são descritos elementos do jogo como descrições breves e detalhadas do conceito do jogo, arte, música, mecânicas centrais, narrativa (se alguma), entre outros aspectos do mesmo. Dependendo do escopo do jogo e do tamanho da equipe, um GDD pode ter de apenas uma até dezenas de páginas, podendo também ser subdividido em áreas (como um documento específico para a parte de narrativa).

Embora haja argumentos questionando a necessidade de tal artefato nos dias atuais [36,37], ter um GDD bem detalhado pode ser extremamente útil para servir de referência para a equipe.

As fases seguintes do ciclo de vida são similares às de ciclos de vida mais tradicionais, com alguns aspectos específicos de desenvolvimento de jogos relevantes [34]. Por exemplo, na fase de análise, os requisitos que costumam surgir a partir de algum cliente têm sua origem no conceito do jogo, ditados principalmente pelo GDD. Além de fazer com que requisitos sejam mais específicos e focados em performance, interface, e experiência de usuário do que um projeto de software comum, tal fato não causa muita mudança no decorrer do projeto.

Na fase de *design* em desenvolvimento de jogos, tal fase pode ser subdividida em criação da arquitetura técnica (reduzir o trabalho a módulos, e buscar soluções de reutilização de software e *assets* externos, como música, arte, entre outros) e design de módulos (e criação dos Documentos de Design de Módulo, que ditam de forma técnica como cada módulo funciona).

A fase de testes, comum tanto em projetos de software gerais quanto em projetos de jogos, apresenta algumas diferenças quando se trata dessa segunda categoria. Além de testes unitários e de integração para garantir que o sistema funcione individualmente e como um todo, também é necessário realizar um controle de qualidade para garantir que a parte de estética do jogo (atmosfera, telas de menus, aparência de personagens, etc) esteja consistente e em conformidade com o estilo do jogo descrito no GDD.

Um estágio muito importante da fase de testes e exclusivo ao desenvolvimento de jogos é o *playtesting* (“teste do jogo” ou “teste jogável”, em tradução livre), que é o ato de jogar o jogo desenvolvido para ter a experiência de como funciona. Essa etapa visa responder perguntas difíceis de quantificar durante as fases anteriores do processo, como:

- “Como a mecânica e a dinâmica funcionam na prática?”
- “O tutorial e manuais são bons?”
- “Como é a curva de aprendizado e o fluxo do jogo?”
- “Há algum problema na jogabilidade?”

Figura 5 - *Glitch* do *MissingNo*, visto em *Pokémon Red* e *Pokémon Blue*
Fonte: Site da Kotaku [39].

Jogadores dessa etapa, chamados *playtesters*, tentam explorar todo o conteúdo do jogo, desvendando todos os seus quebra-cabeças e percorrendo todo o mapa do jogo, a fim de buscar erros, enganar o sistema, ou até fazê-lo cair a fim de expor problemas. A falta de um *playtest* completo e minucioso pode levar à comercialização de jogos com defeitos que podem prejudicar ou beneficiar injustamente e incorretamente o jogador, como a famosa falha do *MissingNo* da primeira geração de jogos de *Pokémon*, visto na Figura 5 e explicada com maiores detalhes em [38]. Em tal *bug*, ao manipular o jogo com ações bem específicas, é possível forçá-lo a um estado onde ele não sabe qual a informação correta a se passar para o jogador, enviando-o informação indevida em seu lugar e causando consequências com repercussões inesperadas. Esse é um grande exemplo da importância de *playtesting* em jogos, pois as consequências de uma falha encontrada há mais de duas décadas atrás ainda causam problemas em jogos da atualidade, provocando problemas em versões mais recentes da franquia [39].

2.3.2. Métodos Ágeis em Desenvolvimento de Jogos

O campo do desenvolvimento de jogos em seu paradigma atual, por ser relativamente jovem e extremamente multidisciplinar, não possui uma literatura tão extensa quanto vertentes mais clássicas da Engenharia de Software, tendo portanto um maior uso de técnicas e metodologias adaptadas de processos mais clássicos.

Clinton Keith, treinador Agile e Treinador *Scrum* Certificado, relata em seu livro [40] e em uma entrevista à autora Heather Maxwell Chandler [41] que encontrou muitos projetos seguindo modelos tradicionais de desenvolvimento, como o modelo cascata, no início de sua carreira com jogos, e constatou que a grande maioria de tais projetos acabavam apresentando atrasos por diversas razões, como começar cedo demais a etapa de produção sem ter o conceito bem definido, ou subestimar os desafios técnicos do projeto. Em seus projetos iniciais na indústria, buscou formas alternativas de desenvolvimento, optando por *Scrum* por diversos motivos.

Keith descreve que, por conta da natureza iterativa do *framework*, os aspectos mais importantes do jogo são implementados ainda no início da fase de produção, permitindo assim que a equipe foque no polimento e implementação de funcionalidades com menor prioridade, mas que podem aumentar o valor final do projeto. Assim, ele enfatiza que é melhor criar um protótipo funcional o mais rápido possível e ir iterando melhorias em cima deste, ao invés de gastar tempo na construção de um documento de *design* enorme e demasiadamente detalhado.

Outro ponto positivo que surge com o uso de *Scrum* no desenvolvimento é permitir que os *stakeholders* possam ver a evolução do projeto de forma mais clara, a partir da avaliação das *sprints* e gráficos de *burndown*, o que os dá mais segurança na hora de investir no projeto.

No entanto, Keith aponta um problema no uso atual do *framework*, apontando que ainda não se sabe ao certo como incluir de forma eficaz arte e *design* no processo, possivelmente por se tratarem de campos mais subjetivos, o que demonstra uma possível limitação ainda não superada do *framework* no processo de desenvolvimento.

Além das contribuições apresentadas por Keith em respeito a adaptação de metodologias clássicas ao paradigma do desenvolvimento, outra adaptação relevante ao presente trabalho é o *Game-Scrum*, proposto por André Godoy e Ellen Barbosa [42]. Em sua proposta, os autores caracterizam sua metodologia como um híbrido entre

Scrum e XP, e dividem o processo de desenvolvimento em Pré-Produção, Produção e Pós-Produção.

Durante a fase de Pré-Produção, os autores apontam que a equipe de desenvolvimento deve priorizar a busca do elemento de diversão e entretenimento do projeto, a partir de iterações com protótipos simples, seguido da devida documentação em um GDD de tais elementos. Quanto mais completo o GDD estiver ao final desta etapa, menores as chances de atrasos por falta de detalhamento das mecânicas e conceitos básicos do jogo. Um GDD bem regido também pode diminuir a probabilidade de *feature creep* (caracterizado pela inserção excessiva de funcionalidades em um projeto, muitas vezes ocasionando atrasos no projeto como um todo). Um detalhe pertinente é a possível opcionalidade do GDD para o caso de equipes de desenvolvimento pequenas.

Na fase de Produção, os autores destacam que o escopo do projeto deve estar bem definido, para então começar a tradução do GDD para um *backlog* do projeto (caso o GDD tenha sido criado). Tal *backlog* deve então ser dividido em partes menores, para serem executadas em cada iteração durante o desenvolvimento.

Por fim, a fase de Pós-Produção engloba o *playtesting* para controle de qualidade, além de um *feedback* do processo como um todo, culminando na geração de um *postmortem*, artefato onde se destacam os pontos positivos, negativos e muito negativos manifestados durante a execução do processo.

2.3.3. Elementos de um jogo

Como mencionado anteriormente, uma métrica básica de sucesso em um projeto de software genérico (isto é, “se o software funciona e sua implementação está em conformidade com seus requisitos”) não pode necessariamente ser aplicada de maneira direta no projeto de um jogo. Nestes, assim como a implementação de requisitos é importante, a forma como tais requisitos são implementados e apresentados para o usuário final tem tanta importância quanto.

Para entender melhor tais nuances e subjetividades presentes na medição de sucesso de um projeto de jogo eletrônico, primeiramente precisamos entender quais são os elementos que compõem um jogo.

Mark Rosewater, designer chefe do jogo *Magic: The Gathering* [43] desde 2003, apresenta alguns elementos que, segundo ele, são importantes para um jogo [44]:

1. **Objetivo:** Deve haver um motivo para o jogo ser jogado. O que os jogadores estão buscando?
2. **Regras:** Deve existir uma lista de ações que os jogadores podem e não podem tomar. Em termos de Engenharia de Software, isso é análogo à noção de regras de negócio.
3. **Interação:** Deve existir algum aspecto que encoraje os jogadores à interagir entre si.
4. **Característica de recuperação:** Deve haver uma maneira que permita que jogadores que estejam indo mal no jogo tenham chance de se recuperar. A falta de uma mecânica dessa natureza pode tornar o jogo frustrante, além de apontar uma falta de fluxo do mesmo [45].
5. **Inércia:** Deve haver algo no jogo que o faça mover naturalmente em direção à sua conclusão.
6. **Surpresa:** Devem haver elementos no jogo imprevisíveis aos jogadores.
7. **Estratégia:** Devem haver elementos no jogo que permitam que jogadores aprendam e se tornem melhores ao longo do tempo.
8. **Diversão:** Como entretenimento é a principal razão pela qual pessoas jogam jogos, deve-se ter algo no jogo que permita que os jogadores se divirtam.
9. **Temática:** A parte de estética é no geral tão importante para um jogo quanto a de mecânica. Deve haver no jogo um tema bem desenvolvido que seja condizente com sua história contada e mecânicas implementadas.
10. **Um gancho:** Deve haver algo que atraia os jogadores e os faça querer jogar o jogo, seja por causa de sua história, mecânicas de jogo, música, ou outros possíveis elementos.

Ao analisarmos os elementos propostos acima, podemos ter uma ideia de métricas que podem ser utilizadas durante a fase de *playtesting* do presente trabalho, a fim de auxiliar na validação do projeto como um jogo bem-sucedido.

2.4. Considerações finais

Com este capítulo é possível entender um pouco mais sobre as metodologias ágeis utilizadas no desenvolvimento de software contemporâneo, além de entender pontos positivos e negativos e situações mais indicadas de uso de cada uma. Além

disso, é possível ver como a Engenharia de Software afeta o desenvolvimento de jogos eletrônicos, seu papel e limitações dentro de um campo multidisciplinar, e outros elementos além do âmbito de computação que também compõem um jogo.

Capítulo 3

3. Desenvolvimento Do Projeto

O presente projeto foi implementado com o auxílio de algumas ferramentas comuns na indústria de jogos atual, a fim de agilizar seu desenvolvimento. Além disso, seu desenvolvimento foi regido por uma versão híbrida do *framework* do *Scrum* e XP, adaptados para este projeto, denominada XS-Game. A descrição da implementação da prova de conceito utilizando o XS-Game pode ser encontrada na seção 3.2, e a descrição mais detalhada de tal adaptação pode ser encontrada no capítulo 4.

3.1. Ferramentas utilizadas

3.1.1. Unity 3D

Criada em 2005 pela Unity Technologies, Unity [47] é um "motor de jogos" (do inglês "*game engine*") e uma IDE ("*Integrated Development Environment*", ou "Ambiente de Desenvolvimento Integrado", em português). Originalmente desenvolvido exclusivamente para OS X, hoje Unity pode ser encontrado também em plataformas Linux e Windows. A ferramenta pode ser utilizada para criar jogos 2D e 3D para diversas plataformas, como Nintendo Switch, Playstation 4, Xbox One, Nintendo 3DS, entre outras. A mesma também é capaz de criar simuladores para diversos fins, incluindo Realidade Virtual e Aumentada.

Com uma versatilidade enorme, Unity pode ser utilizado tanto por grandes empresas quanto por pequenos desenvolvedores, sendo uma das principais escolhas de motor de jogos por desenvolvedores independentes [48]. Podemos ver na Figura 6 o ambiente principal de trabalho da ferramenta.

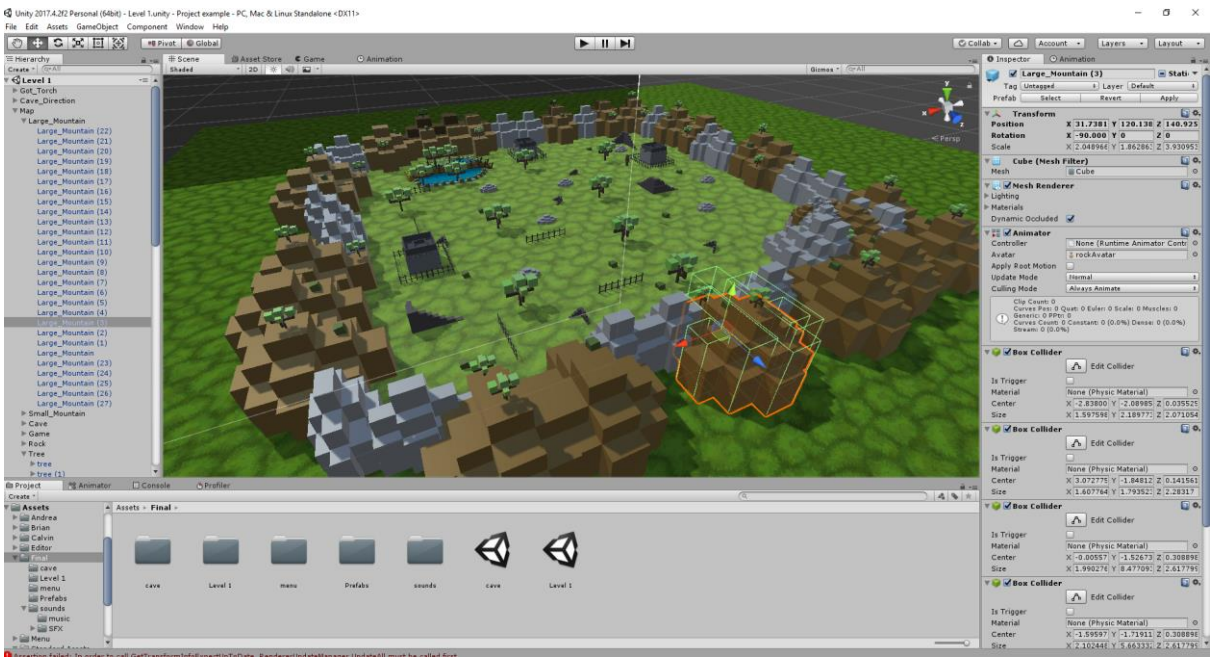

Figura 6 - Interface da ferramenta Unity 3D.

Fonte: Produzido pelo autor.

Além do desenvolvimento com uma funcionalidade *"drag and drop"* ("arraste e solte" em português), o desenvolvimento no Unity também é feito com linguagens de programação. Anteriormente compatível com as linguagens *Boo* e *Unityscript* (uma versão de *Javascript* própria da ferramenta), hoje a linguagem única da *API* do motor é o C#.

Para os fins deste trabalho, foi utilizada a versão 2018.2.14f1 do Unity.

3.1.2. Linguagem de Programação C#

A linguagem C# é uma linguagem de programação desenvolvida pela Microsoft. Buscando unir o poder computacional de Java e C++ com a elegância do Visual Basic, a linguagem é robusta e escalável, permitindo a construção de diversas aplicações com o Framework .NET, também da Microsoft.

Como o código escrito em C# é compilado para uma Linguagem Intermediária por uma máquina virtual, e após isso é convertido em instruções de máquina nativas ao Sistema Operacional onde foi executado, a linguagem é caracterizada como multiplataforma [49].

3.1.3. Maya

Autodesk Maya [50], mais conhecido como apenas Maya, é uma aplicação de computação gráfica multiplataforma, utilizada para criação de aplicações 3D interativas, como jogos digitais, animações, filmes e efeitos visuais.

Para o escopo do presente trabalho, a aplicação foi utilizada para leves alterações de animações e modelos importados da loja virtual da Unity.

3.1.4. Audacity

Audacity [51], ilustrado na Figura 7, é um software multiplataforma e de código aberto, cujo propósito é edição, pós-processamento e gravação de áudio digital. Lançado em 2000, sua capacidade de execução de múltiplas faixas sonoras paralelamente permite a junção, remoção e sobreposição entre várias amostras de áudio. Seu código aberto, interface simples e robustez de funcionalidades o tornaram em uma das mais baixadas aplicações de edição de áudio.

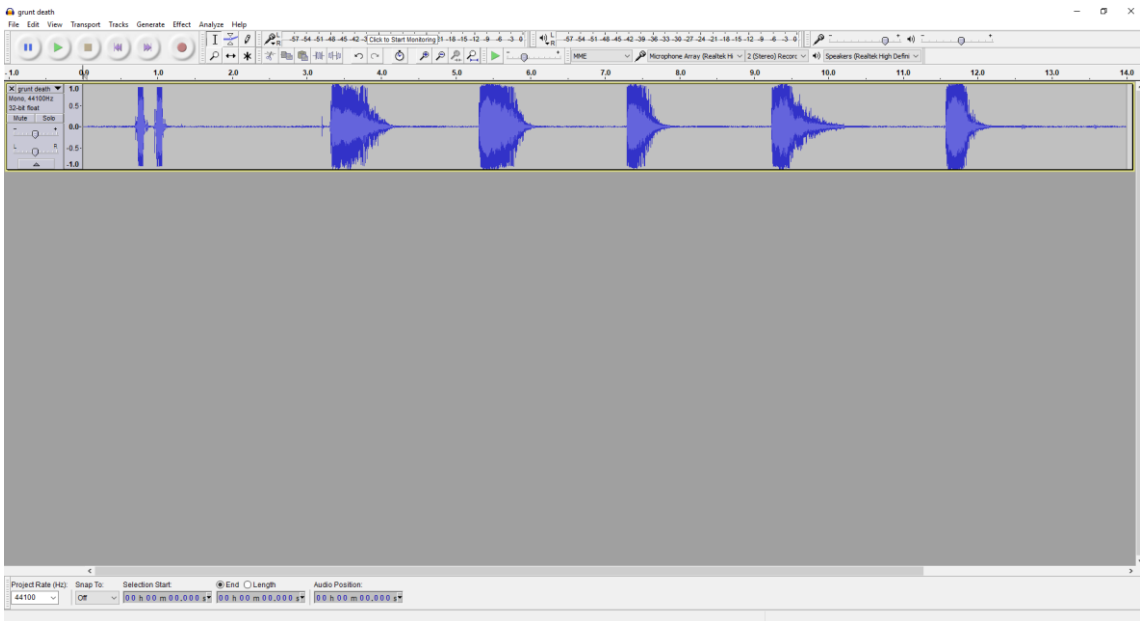

Figura 7 - Interface do Audacity.

Fonte: Produzido pelo autor.

Para o escopo do presente trabalho, a aplicação foi utilizada para edições, diminuições e *loops* de músicas e efeitos sonoros que estarão presentes na parte prática do trabalho.

3.2. Aplicação dos modelos e início do desenvolvimento

O desenvolvimento da prova de conceito do presente trabalho segue as fases do modelo proposto (discutido em mais detalhes no capítulo 4), que por sua vez é influenciado pela divisão de fases apontada por Wurzer e Lichtl [34], explicitadas a seguir:

- Pré-Produção (ou Conceito de Jogo)
- Análise e Especificação
- *Design* (ou Projeto)
- Implementação (ou Desenvolvimento)
- *Playtesting* e Balanceamento
- Entrega

Uma vez definidas as ferramentas e o modelo de desenvolvimento a serem utilizados no projeto, pôde-se começar a fase inicial do processo.

3.2.1. Pré-produção

Na etapa de pré-produção (ou fase de conceito de jogo), o autor dedicou um período de tempo de duas semanas à idealização de um jogo, buscando diversas possibilidades de conceitos como a temática do jogo, estilo da arte e música presentes, condições de derrota e vitória do jogador e mecânicas para alcançar tais condições. Após o final do prazo estipulado, criou-se um GDD a partir de um *template* público [52], onde foram documentadas as decisões realizadas acerca dos conceitos previamente discutidos. Tal artefato seria a principal fonte de informação na fase de análise e especificação, pois a partir deste que seriam levantados vários requisitos do projeto. A versão final do GDD pode ser encontrada no Apêndice A.

3.2.2. Análise e Especificação

A partir do GDD criado na etapa de conceito de jogo, ocorreu a conversão de suas informações para requisitos do projeto. Quanto ao seu formato, foi decidido que o documento de requisitos tradicional seria substituído por um documento de Histórias [24, 46], mais facilmente descrito por conta de sua linguagem mais informal, visando minimizar o tempo consumido na documentação do mesmo. Tal decisão também visava

fazer melhor uso do Quadro do Kanban, a fim de rastrear de forma rápida e visual quais funcionalidades estão sendo desenvolvidas a qualquer momento do projeto.

Sendo assim, foram geradas Histórias contendo as funcionalidades primárias e essenciais do projeto, além de outras de menor importância que não fazem parte do Mínimo Produto Viável, mas adicionariam valor ao protótipo caso houvesse tempo viável de implementá-las. Estas Histórias foram inseridas em um documento de Histórias, artefato interno utilizado apenas pelo desenvolvedor para o preenchimento do *backlog* no início de cada *sprint*.

3.2.3. *Design*

Com base no GDD criado na etapa de conceito de jogo e as Histórias criadas na etapa de análise, foi criado um protótipo físico do mapa principal do jogo, a fim de realizar uma simulação rápida e de fácil alteração de uma partida do jogo. A partir de tal simulação do protótipo físico criado, visível na Figura 8, foram encontradas diversas possíveis melhorias e problemas com o conceito inicial do jogo, como escopo e mecânicas. A cada possível melhoria idealizada, realizava-se uma nova simulação com a inclusão de tal melhoria, a fim de tentar verificar se sua inclusão tornaria a jogabilidade melhor ou se traria um maior balanceamento ao jogo. Após diversas iterações, a simulação no protótipo físico foi considerada satisfatória, e as melhorias encontradas e aceitas como novas funcionalidades foram devidamente documentadas no GDD e documento de Histórias.

Figura 8 - Protótipo físico do jogo.

Fonte: Produzido pelo autor.

Foi também ao longo dessa fase que se deu atenção ao tipo de objetos, como prédios e ruas, que deveriam estar presentes no cenário. Uma vez listados os objetos necessários no mapa do jogo, foi decidido que os mesmos seriam compostos em sua totalidade de objetos gratuitos presentes na loja de *assets* da Unity ou de outras fontes, dada a inexperiência do autor com o campo de modelagem 3D.

Após algumas iterações de testes no protótipo e eventuais revisões dos artefatos, deu-se início a fase de implementação.

3.2.4. Implementação

Na etapa de implementação, ocorreram *sprints* com duração variável entre 6 e 11 dias, visando iterações longas o suficiente para a execução de várias Histórias em cada uma, mas curtas o suficiente para gerar uma sensação de urgência e finitude de tempo disponível para realizá-las. Em cada *sprint*, o *backlog* da *sprint* foi populado com quantidades variáveis de Histórias; para controlar a quantidade de trabalho em cada

sprint, cada História contava com um valor entre 1 e 5 (variando entre ‘muito simples’, ‘simples’, ‘moderado’, ‘difícil’, ‘muito difícil’), que aumentava proporcionalmente ao aumento do nível de dificuldade e complexidade de implementação daquele item, levando também em consideração a familiaridade do desenvolvedor com o campo de cada História.

A soma dos valores dos itens no *backlog* da *sprint* (em outras palavras, a Velocidade da *sprint*), inicialmente, não poderia ultrapassar 15 pontos; ou seja, em uma dada *sprint*, para evitar uma possível sobrecarga e subsequente dívida técnica, fora permitida a inserção de 5 itens ‘médios’ no *backlog* da *sprint*, ou três itens ‘muito difíceis’, ou outra combinação equivalente. Tal Limitante Superior Inicial poderia sofrer adaptações na fase de Retrospectiva de *Sprint*, caso sua alteração fosse julgada necessária. Por exemplo, se em uma dada *sprint*, a quantidade inicial de Histórias escolhidas fosse implementada com uma maior rapidez do que o esperado, o limitante superior inicial poderia ser acrescido para a *sprint* seguinte. Durante a implementação do protótipo em questão, houveram mais momentos em que o limitante superior foi acrescido do que decrescido, o que poderia sugerir um entendimento adequado do autor acerca de suas limitações e capacidades, e um impedimento ativo de adicionar Histórias em demasia em um *sprint backlog*.

Além disso, a soma dos valores das Histórias em estado de implementação não poderia exceder 5 pontos; por conta disso, em qualquer momento da *sprint*, não poderia haver mais do que cinco Histórias ‘muito simples’ sendo implementadas, ou uma ‘muito difícil’, ou outra combinação equivalente. Tal mecanismo tinha como objetivo impedir que o autor tentasse realizar muitas tarefas paralelamente, focando ao invés disso em implementar tarefas de maneira mais sequencial, a fim de minimizar o número de Histórias parcialmente implementadas ao final de uma dada *sprint*.

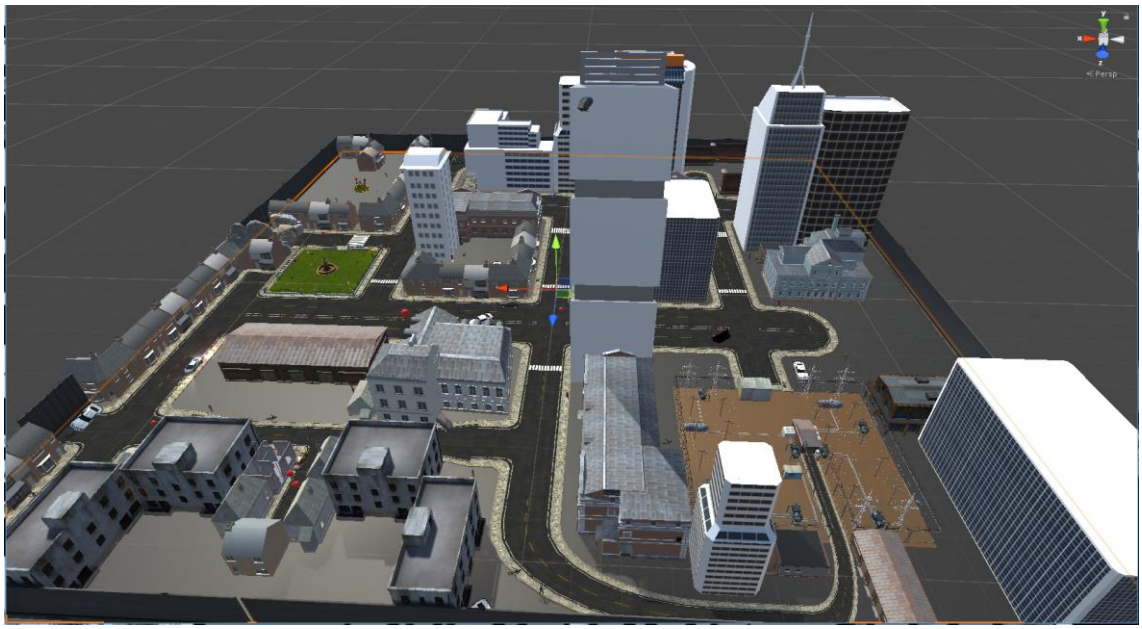

Figura 9 - Mapa do jogo criado.

Fonte: Produzido pelo autor.

As Histórias foram descritas de forma a evidenciar os profissionais envolvidos em cada uma; tal tática foi utilizada pelo autor para, com o auxílio de personas para cada profissional, enxergar os possíveis problemas e desafios associados a cada História e campo de desenvolvimento. Visando tornar mais homogênea a divisão de trabalho feito por cada persona, foi decidido variar a quantidade de Histórias por *sprint* de cada profissional, tomando como base para tal divisão o conhecimento do autor acerca de cada um dos campos do desenvolvimento. Por tal motivo, houveram mais Histórias voltadas para a área de programação do que demais áreas em uma grande quantidade de *sprints*, dada a maior familiaridade do autor em tal campo.

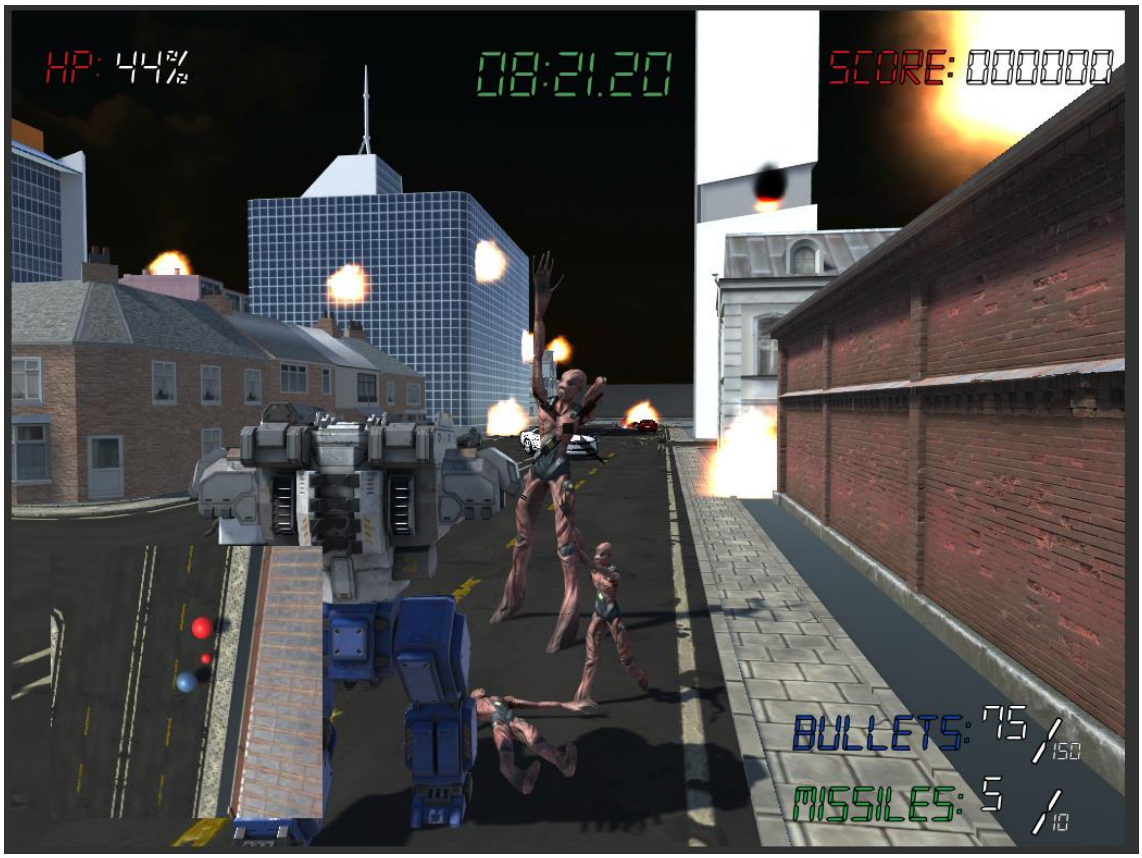

Figura 10 - Execução do jogo criado.

Fonte: Produzido pelo autor.

Devido ao curto prazo de entrega e ao fato de que o presente projeto não produziria um produto final e pronto para comercialização, mas sim apenas um protótipo, foi decidido que não havia necessidade de execução de testes formais de funcionalidades, e em seu lugar seriam executadas apenas verificações locais do código durante a implementação de cada História, a fim de buscar algum tipo de validação acerca da de sua corretude, ainda que incompleto. Pelo mesmo motivo, foi decidido que tais verificações não seriam documentadas formalmente, uma vez que isso também consumiria tempo que poderia ser melhor utilizado no desenvolvimento de mais Histórias. Portanto, tais verificações acabaram sendo atreladas à implementação das funcionalidades em si; isto é, qualquer História implementada deveria obrigatoriamente passar por uma checagem simples antes de poder ser considerada como devidamente implementada.

O desenvolvimento da prova de conceito, que pode ser vista nas Figuras 9 e 10, seguiu por nove *sprints*. Inicialmente, houve um foco na parte de *design* e criação do mapa principal, utilizando modelos gratuitos externos para populá-lo. Após isso,

começou-se a etapa de programação das mecânicas principais, em particular o funcionamento das três rotas de fuga descritas no GDD e detalhadas no documento de Histórias, e todas as etapas que antecedem cada uma.

Em seguida, houve foco nos personagens (jogador e inimigos), particularmente em suas máquinas de estados de animações, programação de controles e câmera do jogador, inteligência artificial dos inimigos, *scripts* de aquisição e armazenamento de itens, e sistema de criação esporádica de inimigos.

Por fim, foi dada prioridade à Histórias envolvendo a interface de usuário e efeitos sonoros, incluindo a implementação de um mini-mapa, indicadores de pontos de vida e munição, contador regressivo de tempo e pontuação de jogo, tela de fim de jogo, efeitos sonoros de explosões e tiros, grunhidos de inimigos e música de fundo.

Após o término da sétima *sprint* (período onde já se havia concluído a maioria das Histórias idealizadas) foi iniciada a fase de *playtesting*.

3.2.5. *Playtesting* e Balanceamento

Na fase atual, foram realizados *playtests*, seguido do preenchimento de um pequeno questionário por parte dos voluntários e eventual balanceamento de algumas variáveis, caso houvesse necessidade.

Durante a etapa de *playtesting*, voluntários munidos de uma folha de instruções do jogo jogaram algumas partidas, buscando a vitória e incentivados a explorar o mapa o máximo possível, relatando posteriormente a experiência através de um pequeno questionário, encontrado no Apêndice C. Os questionários anônimos utilizam uma escala Likert para buscar informações sobre o nível de entretenimento e dificuldade percebidos por cada voluntário, além de contar com perguntas mais abertas, que oferecem ao voluntário a possibilidade de relatar qualquer sensação ou experiência que não fora abordada nas demais perguntas. As respostas presentes nos questionários foram utilizadas para realizar melhorias em funcionalidades já implementadas, assim como balanceamento de variáveis (como pontos de vida de inimigos ou quantidade de munição disponível ao jogador). Opiniões sobre aspectos dos conceitos e mecânicas do protótipo que poderiam ter relevância para uma futura equipe de desenvolvimento mas que seriam complexos demais para uma implementação no presente desenvolvimento foram documentados em um Relatório de Resultados, artefato final que seria entregue junto ao protótipo desenvolvido na fase de Entrega.

Um aspecto relevante a se ressaltar é que, por vezes, houve uma intercalação entre as fases atual e a de Implementação. Tal ocorrência deu-se pela necessidade de implementação de Histórias vitais à conclusão do projeto, ao mesmo tempo em que se mostrava necessário o avanço da etapa de *playtesting* e balanceamento, a fim de refinar o que já havia sido desenvolvido e coletar informações que posteriormente seriam úteis à criação do Relatório de Resultados.

3.2.6. Entrega

Na fase de entrega do protótipo, além da entrega do projeto digital em si e do GDD em sua versão mais atual, houve também a criação de um relatório, denominado Relatório de Resultados, contendo impressões do projeto realizado. Tais impressões foram decorrência de uma análise feita pelo autor em cima dos Questionários de *Playtesting* da etapa de Testes, sumarizados e categorizados a fim de encontrar pontos positivos, negativos e muito negativos observados pelos voluntários durante a execução do jogo. O Relatório de Resultados criado ao final do presente protótipo desenvolvido pode ser encontrado no Apêndice B.

3.3. Considerações finais

O presente capítulo tratou de descrever os procedimentos realizados durante as etapas da criação de um protótipo de jogo digital. No capítulo seguinte, serão generalizadas tais decisões de desenvolvimento, de modo que o modelo adaptado proposto possa ser mais facilmente entendido pelo leitor, possibilitando assim seu reuso.

Capítulo 4

4. XS-Game

O presente capítulo descreve em mais detalhes as decisões e adaptações feitas no *framework* do Scrum e XP para adaptá-los para o presente projeto, culminando na proposição do XS-Game.

4.1. Análise e adaptação de modelos existentes

Com as fundamentações teóricas e a comparação entre modelos de desenvolvimento de software apresentadas no Capítulo 2, foi possível instanciar uma união adaptada das metodologias apresentadas, buscando características de cada uma que fossem mais benéficas para a natureza do projeto e configuração da equipe-alvo de desenvolvimento.

Como Keith reafirma [40,41], *Scrum* é um bom método para se utilizar no desenvolvimento de um projeto com um alto nível de incerteza, algo inerente à natureza de um protótipo. Portanto, utilizar tal método como a base do método proposto de desenvolvimento seria uma boa escolha.

Considerando a existência de apenas um desenvolvedor, uma aderência rigorosa às regras de imutabilidade das *sprints* do modelo *Scrum* não seria benéfico, pois adicionaria uma camada desnecessária de burocracia e consumiria tempo precioso de desenvolvimento. Além disso, tal fato também permitiria uma maior flexibilidade acerca de quais requisitos serão implementados em cada *sprint*, uma vez que todas as decisões seriam tomadas por um indivíduo, que teria tanto o papel de cliente quanto de desenvolvedor. Portanto, foi definido para o modelo proposto que o ciclo de *sprints* tradicional do *Scrum* seria adaptado com elementos da metodologia XP; especificamente, o período reduzido de duração de *sprints* (durando no máximo uma semana), e a possibilidade de alteração do *backlog* da *sprint* durante sua execução, tornando o desenvolvimento mais ágil e flexível frente à eventuais mudanças que pudessem ocorrer no decorrer do projeto.

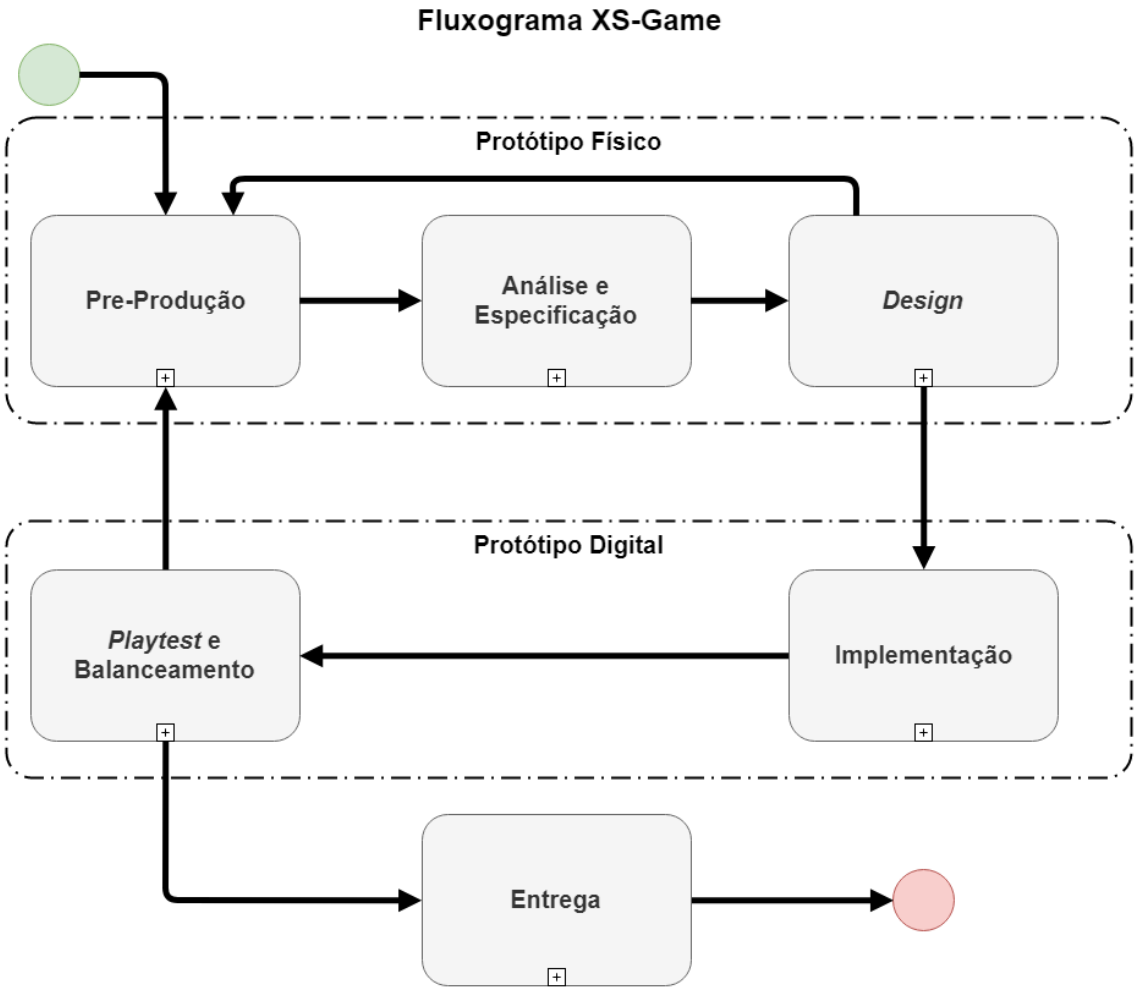

Figura 11 - Fluxograma XS-Game (simplificado).

Fonte: Produzido pelo autor.

O XS-Game, portanto, foi concebido seguindo a divisão de fases ilustrada na Figura 11, influenciado pela divisão apresentada por Wurzer e Lichtl:

- Pré-Produção (ou Conceito de Jogo)
- Análise e Especificação
- *Design* (ou Projeto)
- Implementação (ou Desenvolvimento)
- Testes
- Entrega

4.2. Pré-Produção

Na fase de Pré-Produção, vista na Figura 12, o desenvolvedor deve focar em atividades e técnicas criativas, a fim de buscar ideias de conceitos de jogo sobre as quais possa expandir; tais ideias incluiriam o gênero do jogo, número de jogadores, público-alvo, mecânicas principais, estilos artísticos e sonoros, aspectos gerais da história (caso seja um jogo que possua uma), entre outros detalhes. Deve-se alocar de antemão um prazo máximo para a finalização dessa etapa, para poder assim dar continuidade ao desenvolvimento.

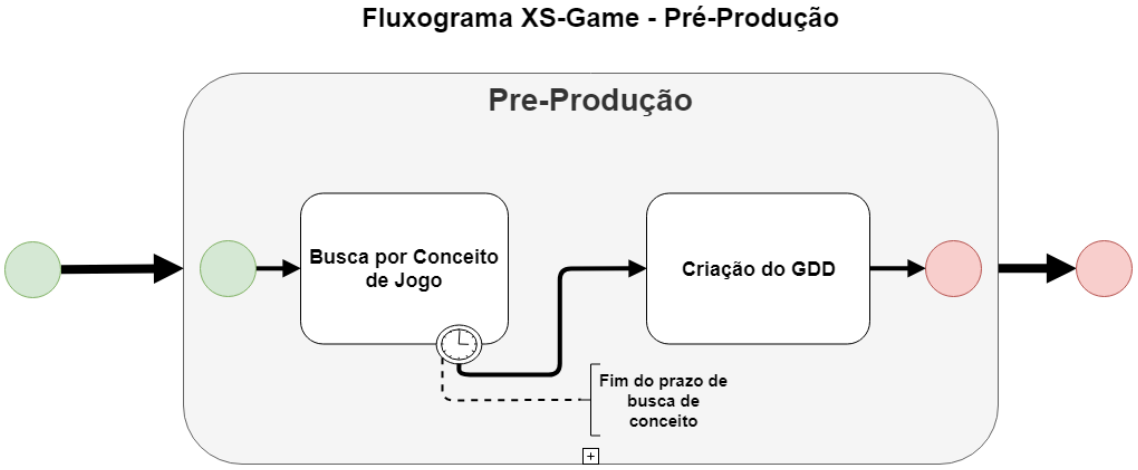

Figura 12 - Fase de Pré-Produção do XS-Game.

Fonte: Produzido pelo autor.

O sucesso de tal etapa (e possivelmente do projeto inteiro) depende em grande parte da criatividade do desenvolvedor. Por conta disso, é importante que o mesmo se utilize de diversas técnicas criativas, como *brainstorming*, que ainda que seja utilizado majoritariamente por equipes compostas por várias pessoas, também pode ser executado por indivíduos, tendo resultados aceitáveis na produção de ideias. [53]

Após algumas iterações de tais atividades, o desenvolvedor pode então documentar as ideias encontradas no *Game Design Document*, que será um artefato vital para o desenvolvimento do documento de requisitos, criado na próxima fase.

4.3. Análise e Especificação

Na fase de Análise e Especificação, abordada na Figura 13, deve-se começar a traduzir o GDD criado na fase anterior em um documento de requisitos. Idealmente, o documento deve ser categorizado em áreas profissionais (arte, música, *design*, etc) para facilitar a organização. Dessa forma, a criação posterior de *backlogs* para as *sprints* torna-se mais ágil, poupando tempo de produção. Uma possível alternativa é escrever os requisitos na forma de Histórias, pois sua maior informalidade torna estes mais simples de serem descritos, sobretudo por profissionais com menos experiência em Engenharia de Software e levantamento formal de requisitos.

Figura 13 - Fase de Análise e Especificação do XS-Game.
Fonte: Produzido pelo autor.

Um detalhe a se atentar é a subjetividade inerente à alguns requisitos, sobretudo aqueles envolvendo arte ou música. A fim de maximizar a objetividade e não-ambiguidade do documento de requisitos, sugere-se descrever apenas a parte objetiva de um requisito em tal artefato, deixando sua parte subjetiva (se houver alguma) para ser detalhada no GDD. Dessa forma, o documento de requisitos permanece como uma fonte mais objetiva e clara de requisitos e necessidades do projeto, enquanto o GDD se molda em um documento mais voltado para conceitos abstratos, para ser utilizado em referências mais aprofundadas e complexas, particularmente para áreas mais artísticas como modelagem e música.

4.4. Design

Na fase de *design*, o desenvolvedor, munido dos artefatos criados nas fases anteriores, deve começar a dividir o projeto em módulos, a fim de conseguir enxergar o projeto de forma mais técnica e fragmentada. É também nesta fase que se deve definir possíveis reutilizações de *assets* e código externo. Por fim, a fase de *design* é o momento onde pode-se criar um protótipo físico da idealização de jogo.

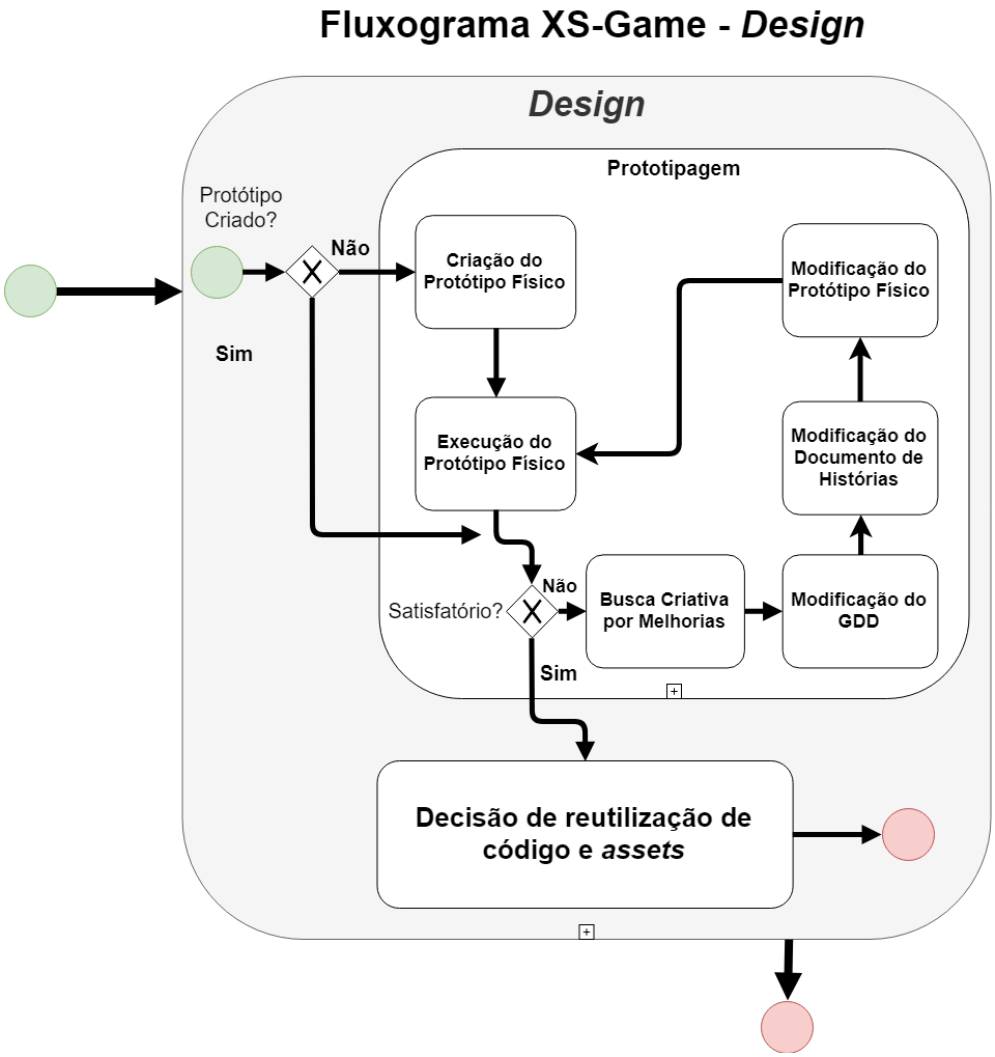

Figura 14 - Fase de *Design* do XS-Game.

Fonte: Produzido pelo autor.

A criação de um protótipo físico ainda no início do projeto, além de possuir rápida implementação e manutenção, pode auxiliar a na busca de melhorias do conceito atual de jogo, além de apontar possíveis problemas em mecânicas. Portanto, é

extremamente recomendável que se crie um protótipo físico, pois o mesmo pode apontar cedo no desenvolvimento problemas fundamentais de escopo ou no conceito de "diversão" do jogo, e a alternativa de encontrar tais problemas apenas na fase de *playtesting*, no final do ciclo de desenvolvimento, pode ter consequências catastróficas para o projeto, podendo inclusive culminar em seu cancelamento.

Deve-se atentar ao fato de que a etapa de prototipagem física é iterativa, como pode ser visto na Figura 14, que descreve o seu ciclo; em outras palavras, problemas trazidos à tona pelas iterações em cima do protótipo físico culminam em mudanças e correções no GDD e no documento de requisitos, enquanto que modificações sofridas por tais artefatos são refletidas na próxima iteração do protótipo físico.

Após algumas iterações em cima do protótipo físico e as devidas correções nos artefatos criados até o momento, o desenvolvedor pode dar prosseguimento ao desenvolvimento, seguindo para a fase de Implementação.

4.5. Implementação

Na fase de Implementação, é possível perceber elementos mais claros de *Scrum* e *Extreme Programming*. O desenvolvimento ocorre com pequenas iterações (também denominadas *sprints*, por simplicidade), com duração de no máximo duas semanas. O objetivo da fixação de um prazo é principalmente um dispositivo psicológico para criar a sensação de finitude e completude de objetivos ao finalizar cada História. Apesar disso, o desenvolvimento em si segue uma visão ideológica mais próxima do XP do que do *Scrum*, visando o desenvolvimento contínuo, especialmente pela subjetividade inerente à algumas áreas, que pode causar atrasos inesperados.

Antes de toda *sprint*, um pequeno planejamento é realizado, no qual o desenvolvedor decide quais Histórias irão compor o *backlog* da presente *sprint*, baseando-se nas Histórias ainda não implementadas, na dívida técnica de *sprints* anteriores (caso haja alguma), ou quaisquer outros fatores que o desenvolvedor considere relevante. A quantidade e tipos de Histórias inicialmente inseridas em qualquer *sprint* é controlada por um Limitante Superior Inicial (ou LSI), onde a soma das complexidades atribuídas à cada História presente na *sprint* não deve ultrapassar esse valor. Para a primeira iteração, o LSI pode ser definido arbitrariamente pelo desenvolvedor; após isso, seu valor pode ser alterado na etapa de Retrospectiva de *sprint*, que ocorre após todas as execuções de *sprint*. Uma vez decidido o *backlog* da

próxima *sprint* (documentado no artefato de Histórico de *Sprints*), inicia-se a etapa de execução da *sprint*.

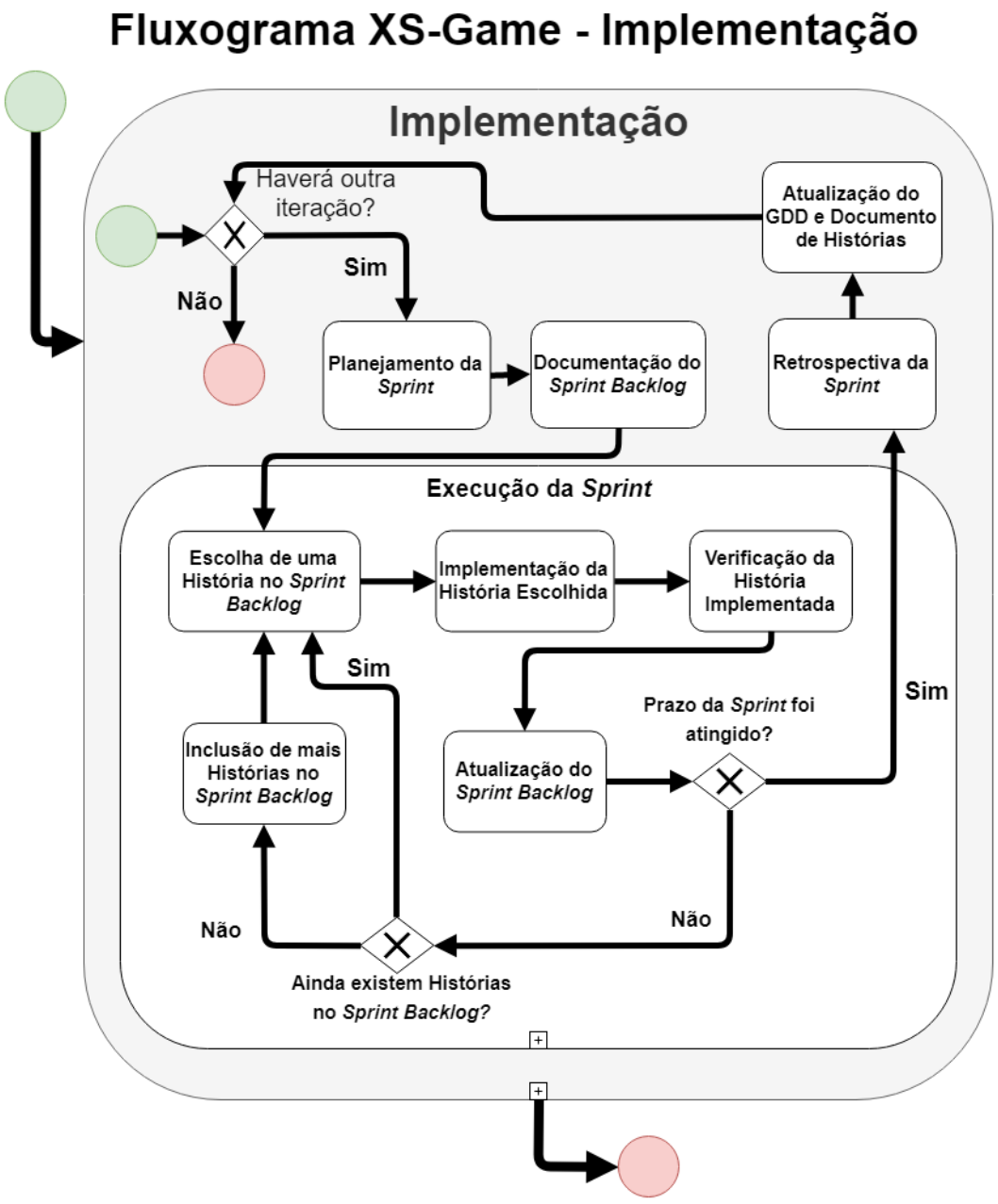

Figura 15 - Fase de Implementação do XS-Game.
Fonte: Produzido pelo autor.

Na etapa de execução, ilustrada na Figura 15, o desenvolvedor deve primeiramente escolher uma História do *backlog* da *sprint*. Tal escolha é arbitrária, e o desenvolvedor possui plena liberdade para decidir a ordem de implementação das Histórias do atual *backlog*, de maneira similar ao observado no *Scrum*. Além disso, caso

o desenvolvedor julgue possível, o mesmo pode implementar mais de uma História em paralelo. Caso tal caminho seja escolhido, deve-se criar um limite máximo de Histórias executadas paralelamente, tomando como possível limitante superior a soma de suas complexidades, a fim de evitar atrasos e dívida técnica. O uso do Quadro de Kanban para a visualização de Histórias pendentes e Histórias executadas paralelamente, como visto no exemplo da Figura 4, se mostra útil no controle e manutenção de tais limites.

Após a implementação da História escolhida, ocorre uma verificação local da História, a fim de buscar alguma garantia de corretude para a História implementada. Uma vez constatado o sucesso de tal validação, a História é dada como concluída, e o *backlog* da *sprint* deve ser atualizado para refletir tal fato.

Caso o prazo da *sprint* ainda não tenha sido alcançado, deve-se continuar implementando itens do *backlog* da *sprint*, reiniciando o ciclo de execução e incluindo mais Histórias no *backlog* da *sprint* caso este tenha sido totalmente concluído. Por fim, uma vez alcançado o prazo de término da presente *sprint*, deve-se dar prosseguimento à etapa de Retrospectiva.

Durante a Retrospectiva da *Sprint*, o desenvolvedor avalia a *sprint* realizada, destacando os pontos positivos e negativos que observou durante sua execução e tomando nota da dívida técnica gerada ao final da mesma, a fim de refinar e otimizar o processo para a próxima iteração, deixando-as cada vez mais fluidas e produtivas a cada execução do ciclo. É também na etapa em questão que o desenvolvedor pode alterar o valor do LSI, permitindo a inclusão inicial de mais Histórias em uma *sprint* (caso a *sprint* anterior tenha apresentado uma produtividade maior do que o esperado), ou diminuir tal capacidade inicial (caso o desenvolvedor tenha se sentido sobrecarregado com a quantidade de Histórias alocadas para implementação, e decida diminuir sua quantidade para diminuir a possibilidade de uma dívida técnica). A modificação do LSI após cada iteração se justifica pela constante necessidade de auto-avaliação por parte do desenvolvedor, para que o mesmo não se sobrecarregue de Histórias irresponsavelmente, mas ainda assim consiga implementar uma quantidade adequada das mesmas, fomentando um sentimento de completude que o incentivará ao longo do projeto.

A última etapa da fase de Implementação é a atualização do GDD e do documento de Histórias, adicionando à ambos quaisquer novos elementos ou requisitos que possam ter surgido durante a execução do ciclo de implementação. Ainda que um dos princípios do Manifesto Ágil seja “software funcional acima de documentação

abrangente", é importante manter um certo nível de atualização nos artefatos criados, sobretudo no GDD, uma vez que este será um artefato de saída e poderá ser importante para uma equipe de desenvolvimento futura que possa vir a dar continuidade ao projeto.

4.6. *Playtesting* e Balanceamento

O presente modelo proposto possui como uma de suas características distintas a ausência de testes tradicionais, como descritos na Engenharia de Software clássica. Isso se justifica pela natureza de protótipo do produto gerado; uma vez que o mesmo não é um produto concluído que será utilizado por um cliente final, mas sim uma implementação imperfeita e incompleta de uma ideia para fins de validação de seus conceitos e jogabilidade, é possível que se haja uma maior flexibilidade quanto a necessidade de corretude em todos os seus aspectos.

Além disso, como o modelo é destinado à um desenvolvedor individual, e visa maximizar a implementação de funcionalidades em um curto espaço de tempo, a criação de mais artefatos como um documento de Casos de Teste iria no contrafluxo de implementações velozes, criando mais camadas de burocracia e diminuindo tempo hábil de desenvolvimento em si.

Tendo tais justificativas em mente, a presente fase do modelo, apresentada na Figura 16, foca na execução de *playtests*, e no balanceamento de variáveis quando tal ação se mostrar necessária.

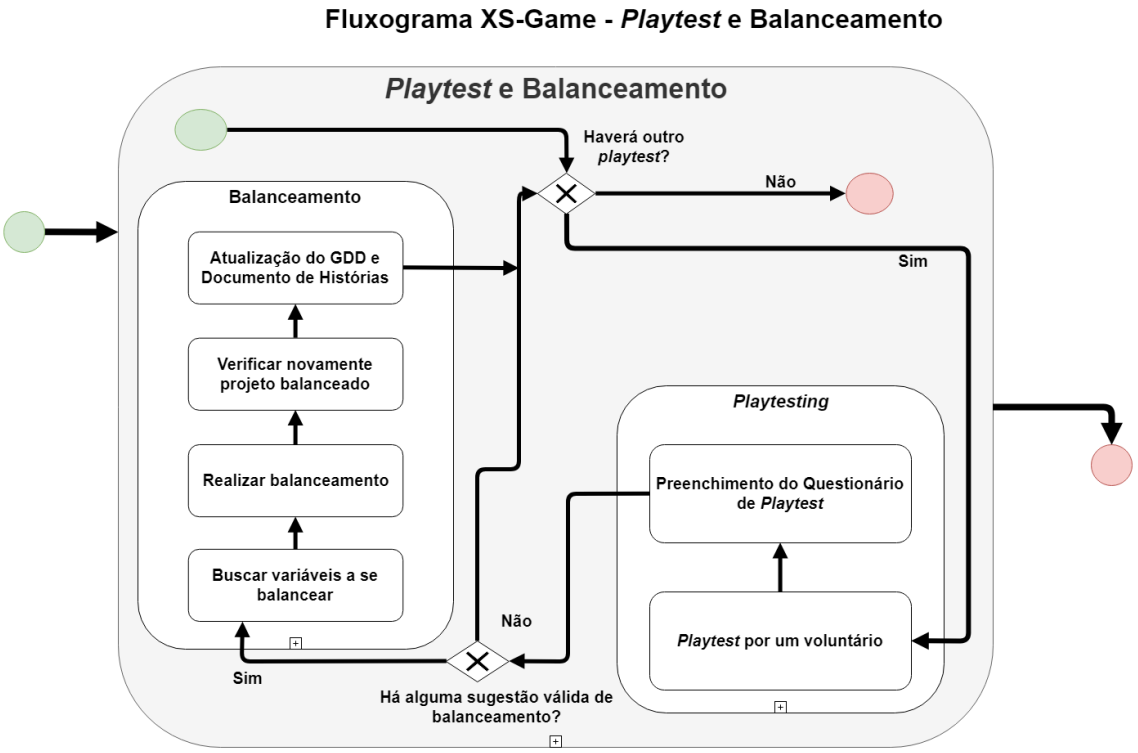

Figura 16 - Fase de *Playtesting* e Balanceamento do XS-Game.

Fonte: Produzido pelo autor.

Outro detalhe digno de nota é que, embora a execução da presente fase ocorra na maioria das vezes relativamente próxima do prazo de entrega do projeto, ainda é possível retornar à fases anteriores, caso se mostre necessário. Isso significa que, caso um *playtest* aponte um defeito que necessite de correções urgentes, ou ainda existam Histórias essenciais ao protótipo final, é possível retornar à fase de Implementação após uma iteração de *playtest* para tratar de novas Histórias.

Na etapa de *playtest*, o desenvolvedor deve buscar pessoas, denominadas *playtesters*, que possam jogar o protótipo digital implementado até o momento, a fim de testar mecânicas e outros elementos como o fator de "entretenimento" do produto. Após cada execução do protótipo, os *playtesters* devem preencher um Questionário de *Playtest*, formulado pelo desenvolvedor, que pode conter perguntas sobre diversos aspectos do protótipo jogado. Analisando cada Questionário preenchido, o desenvolvedor pode então retirar analisar os mesmos para buscar sugestões válidas de melhorias para o protótipo.

Define-se uma sugestão como 'válida' se houver algum motivo que caracterize sua possível inclusão como uma 'melhoria' ao projeto. Como a decisão final parte do desenvolvedor, e o conceito de 'melhor' é subjetivo, nem sempre uma sugestão que

possa ser vista por alguns como válida de fato é tratada como tal. Por exemplo, se uma sugestão for aumentar o poder de ataque do jogador, com o intuito de tornar o jogo mais balanceado, e um dos objetivos do desenvolvedor é buscar uma jogabilidade mais justa e balanceada, tal sugestão provavelmente seria considerada como válida.

Uma vez definida a validez de uma sugestão, deve-se analisar se a mesma é simples ou complexa. Uma sugestão simples é aquela que pode ser implementada com um pequeno ajuste de variáveis, ou uma pequena correção de código, sem grandes alterações nos artefatos criados e sem grandes esforços por parte do desenvolvedor. Uma vez que uma sugestão gere a necessidade de criação de novas Histórias, planejamento de novos módulos ou uma quantidade razoável de codificação adicional, a mesma pode ser considerada como uma sugestão complexa.

Como a fase de testes costuma ocorrer mais próxima ao final do ciclo de desenvolvimento, sugestões complexas normalmente não são implementadas, pois acarretariam em mais *sprints* e mais tempo de desenvolvimento aplicado. Porém, dependendo de quão cedo o desenvolvedor iniciou sua fase de testes e quanto tempo ainda lhe resta antes do prazo de entrega, o mesmo pode optar por adicionar tais sugestões aos artefatos existentes, e buscar implementá-las em uma futura *sprint*.

Por fim, podem haver diversas iterações pela fase de testes enquanto houver tempo hábil e necessidade de sua realização. Uma vez decidido por seu fim, segue-se para a fase final de entrega do projeto.

4.7. Entrega

Na fase de entrega do protótipo, presente na Figura 17, além da entrega do projeto em si, ocorre também a entrega de um Relatório de Resultados, que traz em si pontos positivos, negativos e bem negativos apontados pelos *playtesters* durante a fase de Testes.

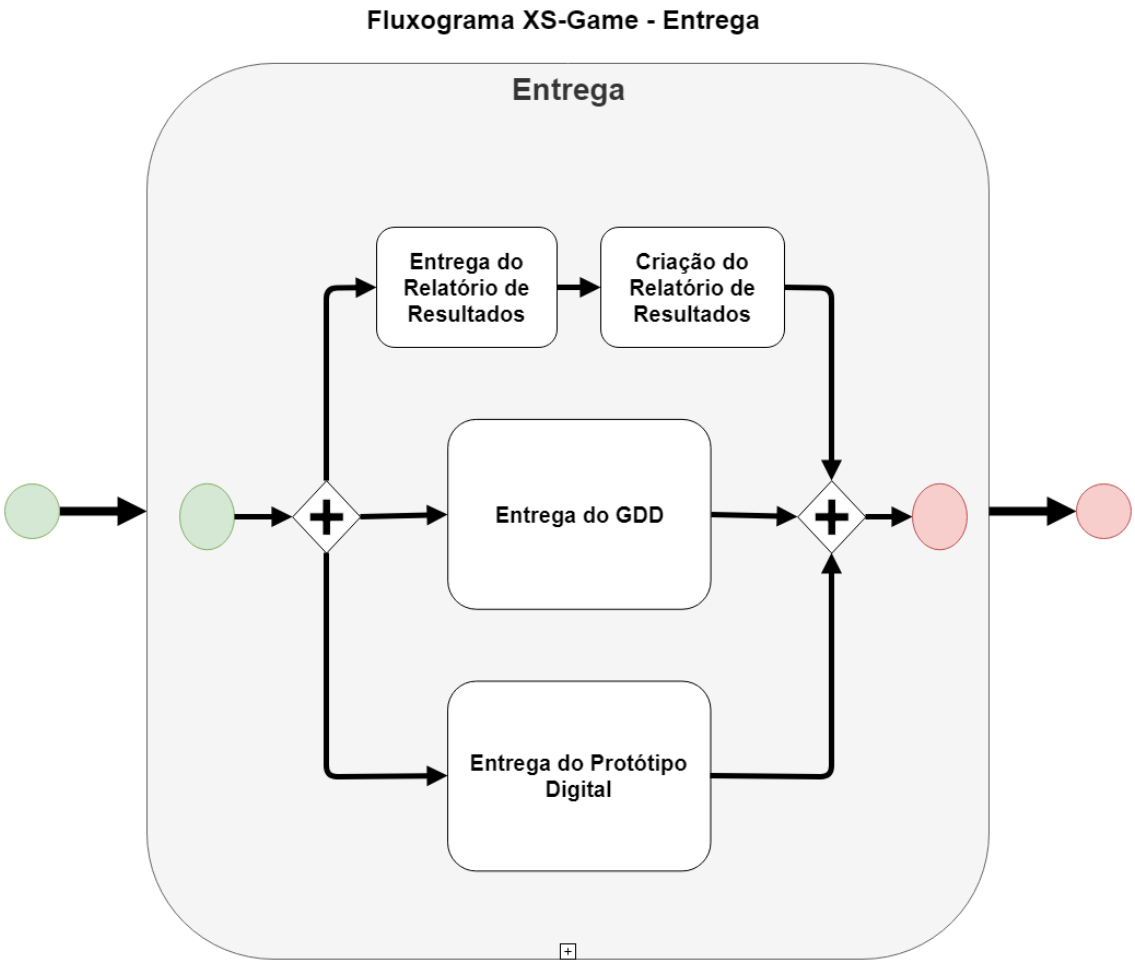

Figura 17 - Fase de Entrega do XS-Game.

Fonte: Produzido pelo autor.

O objetivo da análise dos questionários e criação de tal relatório é fornecer informações relevantes e observações acerca do protótipo produzido para uma equipe de desenvolvimento que possa vir a dar continuidade ao projeto (e que pode inclusive conter o desenvolvedor do protótipo inicial, ou até mesmo ser composta unicamente pelo mesmo desenvolvedor). Dessa forma, pontos positivos podem ser enfatizados ou expandidos prontamente, e pontos negativos podem ser corrigidos com mais facilidade e agilidade, capacitando a futura equipe de desenvolvimento a realizar com mais rapidez o desenvolvimento do produto final, pois já possuiria um sólido ponto de partida.

De maneira similar, caso o resultado do protótipo não tenha se mostrado muito promissor pelos *playtesters*, tal fato pode ser refletido no Relatório de Resultados, e com base nessa informação a equipe futura de desenvolvimento pode optar por modificar elementos básicos do contexto do jogo, ou mesmo recriá-lo do início ou cancelar seu desenvolvimento por completo.

4.8. Considerações finais

Com o presente capítulo, é possível entender de maneira mais generalizada o modelo proposto do presente trabalho, cada uma de suas fases e etapas internas das mesmas, além de compreender o motivo das escolhas tomadas durante a idealização do modelo e de sua instanciação na criação da prova de conceito. No capítulo final, vemos os resultados da aplicação desse conceito, pontos positivos e negativos da sua aplicação conforme fora formulado, e sugestões de possíveis expansões do mesmo em futuros trabalhos.

Capítulo 5

5. Conclusão

O presente trabalho propôs uma adaptação de metodologias clássicas de desenvolvimento ágil de software adaptadas à criação de protótipos de jogos digitais. Embora não traga novos elementos para o campo de Engenharia de Jogos, o modelo proposto utiliza características de metodologias já consolidadas com foco em uma maior otimização para desenvolvedores individuais, buscando maximizar o tempo hábil de desenvolvimento, ao passo em que minimiza a burocracia atrelada a documentação formal e mitiga uma possível perda de performance.

No que se diz respeito ao desenvolvimento em si, foi percebido um gargalo maior na parte de documentação de artefatos, exacerbado pelo fato de haver apenas um desenvolvedor, o que causou um certo atraso ao longo do desenvolvimento, sobretudo nas etapas finais do processo. Uma possível solução para o problema seria tornar a documentação interna, isto é, aquela que é utilizada apenas pelo desenvolvedor, mais informal, e consequentemente de menor complexidade de manutenção. Portanto, poderia haver um foco menor na formalidade de artefatos como o Documento de Requisitos ou *Backlogs* de *Sprints*, mas ainda mantendo um nível aceitável de qualidade no GDD e o Relatório de Resultados, artefatos que farão parte da entrega do projeto.

Um elemento interessante que surgiu durante o desenvolvimento foi a percepção, até então oculta, da separação e coexistência entre o documento de requisitos (nesse caso, de Histórias) e o GDD. Ao se deparar com a necessidade de inserção de requisitos subjetivos, como inclusão de músicas no jogo que evocassem certos sentimentos, foi notada a necessidade de separar os aspectos objetivos de tais requisitos (inserção de uma música em uma fase do jogo) dos aspectos subjetivos (tipo de sensações que deveriam ser passadas através da música). Tais aspectos subjetivos devem ser mantidos fora de um documento de requisitos tradicional, utilizado primariamente por programadores, sendo por sua vez delegados ao Documento de Design, artefato de maior uso por parte de designers e artistas. Além do mais, há sentido em manter tais aspectos documentados no GDD, uma vez que os mesmos são parte dos

conceitos que definem o jogo, e portanto devem ser entregues ao final do processo, para que uma futura equipe possa dar continuidade ao projeto após o término de sua prototipagem.

Ao comparar o XS-Game com outras metodologias focadas no desenvolvimento de jogos como o Game-Scrum, pode-se notar características comuns a ambos os métodos, o que pode ser tomado como indício de validade das propostas apresentadas. Ainda que o modelo proposto do presente trabalho seja voltado para prototipagem, ao contrário do Game-Scrum, ambos apresentam argumentos similares, como o foco na idealização do conceito do jogo ainda em fases iniciais, a fim de evitar que a fase de implementação perca tempo valioso de implementação com experimentações de conceitos. Os dois modelos também defendem o bom uso do GDD, sob o argumento de que, quando bem mantido pelos desenvolvedores, se torna uma fonte insubstituível de referências para o projeto e pode diminuir a possibilidade de *feature creep*, consolidando a importância de tal artefato para o processo criativo.

5.2. Limitações e problemas encontrados

Petrillo [54] argumenta que, dentre os problemas encontrados no desenvolvimento de jogos, destacam-se como mais frequentes os de escopo, planejamento e *crunch time* (aumento extremo da carga de trabalho, normalmente ocorrendo nas semanas antes do prazo final de entrega do projeto). Por conta de diversos fatores internos e externos ao desenvolvimento, o presente trabalho também sofreu com tais problemas, em escalas variadas, detalhados adiante.

A maior limitação do trabalho foi o escopo do mesmo, dado o prazo de entrega pré-estabelecido, e a dificuldade de adaptar os processos e artefatos existentes nos modelos apresentados no Capítulo 2 à realidade de um desenvolvedor individual. Além disso, a dificuldade da manutenção de um cronograma eficiente durante todo o desenvolvimento do projeto também se mostrou um gargalo durante sua execução, uma vez que o desenvolvedor possuía outras responsabilidades profissionais alheias ao projeto. Por conta de tais atrasos no planejamento, algumas Histórias de menor importância inicialmente planejadas para o protótipo não conseguiram ser implementadas. Outra consequência de tal fato foi a impossibilidade de idealização e implementação de novas Histórias a partir das respostas dos questionários de *playtesting*.

Em termos de documentação, uma dificuldade encontrada foi o grande tempo gasto criando-a, visto que havia apenas um desenvolvedor para realizar todo o projeto. Ao longo do processo de desenvolvimento, o autor buscou um equilíbrio entre artefatos bem documentados, a fim de auxiliar eventuais referências. Tal decisão acabou por afetar em certo nível o tempo hábil para desenvolvimento, uma vez que uma parte relevante do tempo era utilizado na manutenção de tais artefatos.

Uma dificuldade em particular encontrada durante o desenvolvimento foi a constante necessidade de controlar o tamanho das Histórias, a fim de impedir que crescessem demais a ponto de virarem Épicos. Tal risco era frequente, visto que diversas funcionalidades possuíam mais de uma etapa para serem totalmente implementadas, normalmente em campos de atuação distintos. Por exemplo, considerando a História "acionar uma animação de explosão quando o Jogador realizasse alguma ação", esta deveria ser dividida em uma 'sub-História' para o *designer* (que deveria escolher e posicionar o objeto de explosão no mundo do jogo), outra para o artista de som (que deveria buscar um efeito sonoro de explosão adequado para ser executado no momento da explosão), e outra para o programador (que deveria realizar a checagem da realização da ação em questão e ativar a explosão e seu respectivo efeito sonoro quando a checagem retornasse um resultado verdadeiro).

Visto que o presente trabalho possui apenas um desenvolvedor, a necessidade de interdisciplinaridade e atuação em várias áreas é levada ao seu extremo, e portanto por vezes mostrou-se difícil separar as Histórias em Histórias menores e mais centradas em uma área de conhecimento como programação ou arte sonora. Tal fato mostrou-se um aspecto limitante e desafiador durante o desenvolvimento, uma vez que a alternância de papéis por parte do autor mostrou-se constantemente necessária, alternando entre programador, designer, artista gráfico ou artista sonoro para alcançar a conclusão de diversos requisitos dentro das *sprints*.

Além disso, como mencionado na Seção 5.1, houveram algumas dúvidas sobre como tratar e devidamente documentar requisitos mais subjetivos (que normalmente envolviam outras áreas fora do campo primário do autor, como arte e música) em um documento de Histórias, que só deveria aceitar requisitos com um mínimo de objetividade. Eventualmente foi decidido que tais requisitos seriam divididos em componentes subjetivos e objetivos, sendo estes elaborados no documento de Histórias, e aqueles descritos no GDD.

Por fim, dadas as limitações do prazo de entrega e da existência de apenas um desenvolvedor na equipe, não foi possível se utilizar de algumas práticas de engenharia possivelmente benéficas ao desenvolvimento, como a Programação em Pares e a Propriedade Coletiva do Código.

5.3. Sugestões de Trabalhos Futuros

Como só houve uma iteração do modelo e o seu produto final tinha uma escala relativamente pequena, alguns aspectos que poderiam ter uma aplicação relevante no processo não foram introduzidos em seu fluxo, como a utilização de um Documento de Módulos para dividir os diferentes módulos do projeto. Um trabalho futuro de maior escala poderia explorar a validade e necessidade de uso de tais aspectos, a fim de refinar o modelo proposto e assim influenciar positivamente na qualidade dos produtos finais entregues.

Em contrapartida, parte dos atrasos sofridos durante o desenvolvimento foram em decorrência da manutenção de toda a documentação com um nível de qualidade aceitável, inclusive aquela que não seria parte da entrega final, como o Histórico de Sprints. Uma investigação futura sobre a validade da ideia de diminuição da formalidade de artefatos internos que não serão entregues à uma futura equipe desenvolvimento pode trazer vantagens aos usuários dos modelos, uma vez que tal diminuição reduziria o tempo gasto mantendo artefatos atualizados, e em troca poderia permitir que mais tempo fosse dedicado à outras tarefas mais importantes.

Além disso, como o foco do modelo proposto recai em um desenvolvedor individual, uma possível evolução do modelo seria em sua adaptação para utilização por mais de um indivíduo, a fim de avaliar o desempenho do modelo quando aplicado por uma equipe de desenvolvimento, e analisar a validade do uso de práticas de desenvolvimento grupal aplicadas no modelo inicial proposto.

Referências Bibliográficas

[1] SIMMONS, M. **Bertie the Brain programmer heads science council.** Jornal Ottawa Citizen (9 out. 1975, p. 17). Disponível em:

<https://news.google.com/newspapers?id=rKYyAAAAIBAJ&sjid=pe0FAAAAIBAJ&pg=916,3790974&dq=josef-kates&hl=en>. Acesso em 4 out. 2018.

[2] BATCHELOR, J. **Global games market value rising to $134.9bn in 2018.** Disponível em: <https://www.gamesindustry.biz/articles/2018-12-18-global-games-market-value-rose-to-usd134-9bn-in-2018>. Acesso em 17 out. 2018.

[3] FRENCH, M. **Inside Rockstar North - Part 2: The Studio**. Disponível em: <https://www.mcvuk.com/development/inside-rockstar-north-part-2-the-studio>. Acesso em 4 out. 2018.

[4] MAKUCH, E. **This is How Much The Witcher 3 Cost to Make**. Disponível em: <https://www.gamespot.com/articles/this-is-how-much-the-witcher-3-cost-to-make/1100-6430409/> Acesso em 4 out. 2018.

[5] BRIGHTMAN, J. **Titan cancellation cost Blizzard $50m or more, say analysts**. Disponível em: <https://www.gamesindustry.biz/articles/2014-09-23-titan-cancellation-cost-blizzard-usd50m-or-more-say-analysts>. Acesso em 13 fev. 2019.

[6] DAWSON, M., BURRELL, D. N., RAHIM, E., BREWSTER, S. **Integrating Software Assurance into the Software Development Life Cycle (SDLC)**. Disponível em:
<https://www.researchgate.net/publication/255965523_Integrating_Software_Assurance_into_the_Software_Development_Life_Cycle_SDLC>. Acesso em 13 fev. 2019.

[7] SOURCE GAMING. **Sakurai on Smash 64: A Phenomenon Is Born**. Disponível em: <https://www.sourcegaming.info/2015/08/06/sakurai64/>. Acesso em 17 fev. 2019.

[8] UNITY. **Game Studio Report 2018: The way small independent studios create**. Disponível em: <https://unity3d.com/game-studio-report-2018>. Acesso em 4 out. 2018.

[9] SECRETARIA ESPECIAL DA CULTURA. **2º Censo da Indústria Brasileira de Jogos Digitais.** Disponível em: <http://cultura.gov.br/105476-revision-v1/>. Acesso em 19 mar. 2019.

[10] SCHACH, S. R. “The Scope of Software Engineering”. In: Schach, S. R. (ed.), **Object-Oriented and Classical Software Engineering**, 8 ed., chapter 1, New York, USA, McGraw-Hill, 2010.

[11] ROYCE, W. W. **Managing the Development of Large Software Systems.** Disponível em: <http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf>. Acesso em: 24 out. 2018.

[12] SCHACH, S. R. “Software Life-Cycle Models”. In: Schach, S. R. (ed.), **Object-Oriented and Classical Software Engineering**, 8 ed., chapter 2, New York, USA, McGraw-Hill, 2010.

[13] AGILE MANIFESTO. **Manifesto for Agile Software Development.** Disponível em: <http://agilemanifesto.org/>. Acesso em 15 out. 2018.

[14] AGILE ALLIANCE. **12 Principles Behind the Agile Manifesto.** Disponível em: <https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>. Acesso em 15 out. 2018.

[15] YOUTUBE. **2017 Scrum Guide Update with Ken Schwaber and Jeff Sutherland.** Disponível em: <https://www.youtube.com/watch?v=WVSQkU5VaC8>. Acesso em 31 out. 2018.

[16] *SCRUM*. **What is *Scrum*?** Disponível em: <https://www.scrum.org/resources/what-is-scrum>. Acesso em 29 out. 2018.

[17] TAKEUCHI, H., IKUJIRO, N. "The new new product development game", *Harvard Business Review*, pp 137-146, Jan. 1986.

[18] *SCRUM* GUIDES. **THE SCRUM GUIDE ™.** Disponível em: <https://www.scrumguides.org/scrum-guide.html>. Acesso em 30 out. 2018.

[19] SCRUM INC. **Velocity.** Disponível em: <https://www.scruminc.com/velocity/>. Acesso em 19 mar. 2019.

[20] AGILE ALLIANCE. **Burndown Chart**. Disponível em: <https://www.agilealliance.org/glossary/burndown-chart/>. Acesso em 6 nov. 2018.

[21] BECK, K, ANDRES, C., **Extreme Programming Explained: Embrace Change**, 2 ed., Massachusetts, USA, Addison-Wesley Professional, 2004.

[22] EXTREME PROGRAMMING. **Extreme Programming: A Gentle Introduction**. Disponível em: <http://www.extremeprogramming.org/>. Acesso em 2 nov. 2018.

[23] EXTREME PROGRAMMING. **What we have learned about Extreme Programming**. Disponível em: <http://www.extremeprogramming.org/lessons.html>. Acesso em 2 nov. 2018.

[24] EXTREME PROGRAMMING. **User Stories**. Disponível em: <http://www.extremeprogramming.org/rules/userstories.html>. Acesso em 18 nov. 2018.

[25] REHKOPF, MAX. **Epics, Stories, Themes and Initiatives.** Disponível em: <https://www.atlassian.com/agile/project-management/epics-stories-themes>. Acesso em 19 mar. 2019.

[26] EXTREME PROGRAMMING. **Extreme Programming Project**. Disponível em: <http://www.extremeprogramming.org/map/project.html>. Acesso em 18 nov. 2018.

[27] EXTREME PROGRAMMING. **The values of Extreme Programming**. Disponível em: <http://www.extremeprogramming.org/values.html>. Acesso em 2 nov. 2018.

[28] PLAINVIEW LEANKIT. **What is Kanban?** Disponível em: <https://leankit.com/learn/kanban/what-is-kanban/>. Acesso em 16 nov. 2018.

[29] KANBANIZE. **Kanban Explained for Beginners.** Disponível em: <https://kanbanize.com/kanban-resources/getting-started/what-is-kanban/>. Acesso em 18 nov. 2018.

[30] TRELLO. Disponível em: <https://trello.com/>. Acesso em 18 nov. 2018.

[31] COLLABNET. **What is Kanban? An Introduction to Kanban Methodology**. Disponível em: <https://resources.collab.net/agile-101/what-is-kanban>. Acesso em 15 nov. 2018.

[32] GILL, A. Q., HENDERSON-SELLERS. "Comparative evaluation of XP and *scrum* using the 4d analytical tool (4-DAT)", **European and Mediterranean Conference on Information Systems (EMCIS)**, Costa Blanca, Alicante, Spain, Jul. 2006**.**

[33] COHN, M. **Differences Between *Scrum* and Extreme Programming**. Disponível em: <www.mountaingoatsoftware.com/blog/differences-between-scrum-and-extreme-programming>. Acesso em 18 nov. 2018.

[34] WURZER, G., LICHTL, B, **Software Engineering in Games**. Disponível em:

<https://www.researchgate.net/publication/267774468_Software_Engineering_in_Games>. Acesso em: 25 out. 2018.

[35] MIGNANO, M. **Use Paper Prototyping to design your games**. Disponível em: <https://www.gamasutra.com/blogs/MarcoMignano/20160725/277766/Use_Paper_Prototyping_to_design_your_games.php>. Acesso em 25 out. 2018.

[36] SWEATMAN, J. **Death of the game design document**. Disponível em: <https://www.mcvuk.com/development/death-of-the-game-design-document>. Acesso em 29 jan. 2019.

[37] GAME DESIGNING. **How to Create a Game Design Document**. Disponível em: <https://www.gamedesigning.org/learn/game-design-document/>. Acesso em 15 nov. 2018.

[38] HERNANDEZ, P. **Pokémon's Famous Missingno Glitch, Explained**. Disponível em: <https://kotaku.com/pokemons-famous-missingno-glitch-explained-1653929141>. Acesso em 20 nov. 2018.

[39] KAMEN, M. **Missingno's revenge: trading the infamous glitch Pokémon to Sun and Moon can break the game**. Disponível em: <https://www.wired.co.uk/article/pokemon-bank-sun-moon-missingno-glitch>. Acesso em 20 nov. 2018.

[40] KEITH, C. "The Crisis Facing Game Development". In: Keith, C. (ed.), **Agile Game Development with *Scrum***, 1 ed., chapter 1, Massachusetts, USA, Addison-Wesley Professional, 2010.

[41] CHANDLER, H. M. “Project Management Methods”. In: Chandler, H. M. (ed.), **The Game Production Handbook**, 3 ed., chapter 3, Massachusetts, USA, Jones & Bartlett Learning, 2014.

[42] GODOY, A., BARBOSA, E. F. “Game-Scrum: An Approach to Agile Game Development”**. IX Simpósio Brasileiro de Jogos e Entretenimento Digital**. Florianópolis, 2010.

[43] MAGIC: THE GATHERING. Disponível em: <https://magic.wizards.com/en>. Acesso em 25 out. 2018.

[44] ROSEWATER, M. **Ten Things Every Game Needs, Part 1 & Part 2**. Disponível em: <https://magic.wizards.com/en/articles/archive/making-magic/ten-things-every-game-needs-part-1-part-2-2011-12-19>. Acesso em 25 out. 2018.

[45] BARON, S. **Cognitive Flow: The Psychology of Great Game Design**. Disponível em: <http://www.gamasutra.com/view/feature/166972/cognitive_flow_the_psychology_of_.php>. Acesso em 25 out. 2018.

[46] MOUNTAIN GOAT SOFTWARE. **User Stories**. Disponível em: <https://www.mountaingoatsoftware.com/agile/user-stories>. Acesso em 18 nov. 2018.

[47] UNITY. Disponível em: <https://unity3d.com/>. Acesso em 6 jan. 2019.

[48] MORAN, D. **5 Leading Game Engines for indie game developers.** Disponível em: <https://www.gamasutra.com/blogs/DylanMoran/20160729/278145/5_Leading_Game_Engines_for_indie_game_developers.php>. Acesso em 15 out. 2018.

[49] MICROSOFT. **Introduction to the C# Language and the .NET Framework.** Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>. Acesso em 16 out. 2018.

[50] AUTODESK MAYA**.** Disponível em: <https://www.autodesk.com/products/maya/overview>. Acesso em 7 jan. 2019.

[51] AUDACITY**.** Disponível em: <https://www.audacityteam.org/>. Acesso em 24 jan 2019.

[52] STANLEY, B., MARKARIAN, A. **Game Design Document**. Disponível em: <https://docs.google.com/document/d/1-I08qX76DgSFyN1ByIGtPuqXh7bVKraHcNIA25tpAzE/edit?usp=sharing>. Acesso em 20 nov. 2018.

[53] LAMM, H., TROMMSDORFF, G. **Group versus individual performance on tasks requiring ideational proficiency (brainstorming): A review**. Disponível em: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/ejsp.2420030402>. Acesso em 23 fev. 2019.

[54] PETRILLO, F., PIMENTA, M., TRINDADE, F., DIETRICH, C. “Houston, we have a problem...: A Survey of Actual Problems in Computer Games Development”. **ACM Symposium on Applied Computing (SAC)**, Ceará, 2008.

Apêndice A

Game Design Document

Elevator Pitch

Um *shooter* para PC com ritmo acelerado e tensão constante, repleto de zumbis, quebra-cabeças e trocadilhos ocasionais. Feito com Unity em C#.

Visão Global

Tele-Command Chassis é um jogo de tiro em terceira pessoa disponível para PC. Situado em uma cidade em quarentena pelo exército, o jogador deve controlar seu personagem, que utiliza um exoesqueleto para combater ondas de mutantes vorazes antes que seu tempo acabe e a cidade seja dizimada, enquanto busca respostas sobre o que aconteceu na sua cidade e uma rota de fuga de seu confinamento forçado.

Tema/Gênero

O jogo será um jogo de tiro em terceira pessoa, com elementos de estratégia e partidas curtas, com duração máxima de 10 minutos.

Público-alvo

O jogo terá um público alvo de jovens a partir de 13 anos, devido ao estilo de jogo e elementos de violência.

Objetivos de design

Os objetivos de design incluem criar um mundo compacto mas verossímil, composto de casas, prédios e ruas. Além disso, o Avatar do jogador, na forma de um exoesqueleto, possuirá componentes removíveis(como membros do corpo), e conforme o jogador é danificado, perderá partes e funcionalidades do seu exoesqueleto para demonstrar isso, aumentando o nível de imersão e comprometimento com detalhes visuais e estéticos.

Além disso, busca-se um constante clima de tensão e dificuldades, portanto o jogador terá um tempo limite para completar a partida, caso contrário será derrotado.

Descrição do Projeto

O jogador terá várias possíveis rotas de fuga, mas cada uma delas terá uma maneira de ser desbloqueada, tornando o jogo mais complexo e difícil de ser completado. Haverá também partes do jogo geradas aleatoriamente, tornando cada nova partida única.

A cada partida (para essa versão, o jogo contará com apenas uma fase), o jogador começará em um ponto específico do mapa. A partir daí, o jogador deve buscar uma saída do mapa a partir de um dos vários pontos de fuga existentes (pontos de fuga descritos em mais detalhes na parte de "Mecânicas Centrais"). Cada partida contará com um limite de tempo, e ao final de cada uma, caso seja vitorioso, será contabilizado a pontuação final do jogador naquela fase.

Mecânicas centrais

A. Conteúdo gerado aleatoriamente:

A cada partida, os itens chave (itens necessários para a abertura de uma rota de fuga) serão criados em uma de várias localidades pré-definidas em tempo de execução. Isso aumentará o fator *replay* do jogo, permitindo que cada nova partida seja um pouco diferente das anteriores.

B. Múltiplos finais:

Além da mecânica básica de derrota (ter sua vida diminuída à 0 ou ser incapaz de escapar no tempo limite), o jogo ainda contará com duas mecânicas de final vitorioso, um 'normal' (onde o jogador somente escapa da cidade, mas sem descobrir o motivo da epidemia e quarentena) e outro 'verdadeiro' (onde o jogador além de escapar, encontra itens-chave que explicam o motivo de tudo que está acontecendo no local, e ao conseguir fugir relata ao mundo o que de fato ocorreu em sua cidade).

Isso aumenta o fator de rejogabilidade, pois os itens chave responsáveis pelo final 'verdadeiro' também são posicionados em mais de uma posição no mapa.

C. Pontos de fuga:

As fases incluirão um ou mais pontos de fuga, e cruzar os mesmos é a condição de vitória para o jogador. Para cada ponto de fuga, o jogador deverá cumprir um ou mais pré-requisitos, que podem ser a destruição de certos objetos ou a aquisição de certos itens espalhados no mapa. Uma vez alcançado tal pré-requisito, o jogador pode escapar pela rota de fuga desbloqueada.

D. Sistema de pontuação:

Para aumentar o interesse do jogador em jogar mais partidas, o jogo contará com um pequeno sistema de pontuação, que pode incentivar o jogador a derrotar mais inimigos em cada partida.

E. Perda de componentes do exoesqueleto:

Conforme o jogador acumula dano em seu exoesqueleto, partes do mesmo vão sendo destruídas, e com a perda de tais partes também ocorre perda de funcionalidade. Um exemplo disso seria o jogador, ao acumular dano suficiente, perder um braço de seu exoesqueleto, e portanto ser incapaz de continuar utilizando a arma que estava presente neste braço.

F. Itens de recuperação:

Haverão três itens no jogo: itens de recuperação de balas, recuperação de mísseis, e recuperação de vida do Jogador. Tais itens podem ser coletados pelo Jogador, e têm chance de serem deixados por inimigos quando estes são derrotados.

Efeitos sonoros e música

- Efeitos sonoros de grunhidos de morte para inimigos, além de sons de ataque/tiro dos mesmos.
- Música de fundo que evoca sensações de tensão, pressa e urgência.

Assets Necessários

Jogador:

Será usado um *asset* de robô/Gundam/Mecha, a fim de tornar o personagem principal mais versátil e fácil de evidenciar danos e avarias.

Inimigos:

Existirão quatro tipos de inimigos no jogo:

- Soldados
- Zumbis
- Mutantes
- Mutantes Gigantes

Cada um terá *scripts* com valores de pontos de vida, pontos de ataque e *drop rate* variáveis.

Pontos-Chave:

Os Pontos-Chave e Rotas de Fuga serão representados por anéis de energia, que possuem uma cor relativa à sua própria rota de fuga (vermelho para a rota militar, roxo para a rota do deslizamento de terra, e verde para a rota do portão eletrônico).

Itens:

Cada um dos três itens existentes tem sua própria imagem, para permitir que o jogador diferencie os mesmos.

Cenário:

Sendo parte de um bairro, serão necessários elementos comuns à uma cidade, como casas, carros, aspectos de natureza (árvores, arbustos, etc), e ruas. Possivelmente elementos que evidenciam destruição e caos também estarão presentes, como fogo, prédios e carros destruídos e corpos sem vida no chão.

Som:

Todos os efeitos sonoros (explosões, tiros, ataques de inimigos e música de fundo) precisam ser buscados em fontes externas, dada a limitação do projeto em termos de profissionais musicais.

Cronograma

Título	Descrição	Data de término
Início do Projeto	Projeto de desenvolvimento do protótipo iniciado em 5 de novembro de 2018.	05/11/2018
Pré-Planejamento	Decisão do conceito inicial do jogo, aspectos a serem utilizados de cada metodologia relevante durante o processo; criação do *Game Design Document.*	20/11/2018
Protótipo (pre-Alpha)	Criação de um protótipo simples, possivelmente em papel, para testar o funcionamento dos controles e ideias relativas ao conceito e mecânicas apresentadas no GDD.	01/12/2018
Protótipo (Alpha)	Buscar *assets* necessários para o jogo, como música, efeitos sonoros, fontes e arte; Popular cenário principal com *assets*, jogador com controles funcionais (andar, atirar); inimigos com IA rudimentar (busca simples pelo jogador, atacar quando próximo) e deixando itens de recarga de munição quando derrotados; criar localidades pré-definidas de criação de itens chave; implementar funcionamento dos pontos de fuga (coleta de itens chave como pré-requisito para suas ativações);	25/12/2018
Protótipo (Beta)	Melhorar IA inimiga; criar destruição modular do jogador; criar mini-mapa para auxílio do jogador; criar outros elementos da UI; melhorar estética do sistema de quebra-cabeça dos pontos de fuga (possivelmente com *cut-scenes*); criar tela de início de jogo e tela de transição/introdução;	25/01/2018
Playtesting	Realização de sessões de jogo do protótipo por voluntários, para receber sugestões e críticas sobre seu funcionamento e balanceamento, além de auxiliarem na busca de defeitos.	10/02/2018
Entrega do projeto final	Entrega da versão final do protótipo, *Game Design Document*, e Relatório de Resultados.	20/02/2018

Apêndice B

Relatório de Resultados

1) **Ambientação:**

Não houve muitos comentários sobre o mapa ou inimigos, então pode se supor que não hajam grandes problemas em tal aspecto. Houveram comentários sobre a falta de ambientação da história do jogo; isto é, o que ocorreu no universo do mesmo até o momento em que o usuário começa a jogar, suas motivações para sair da cidade, etc. Uma possível solução para isso seria a inserção de uma cena anterior à fase principal, onde haveria a explicação dos acontecimentos e motivações que pré-datam o início do jogo.

2) **Controles:**

Uma sugestão apresentada foi a possível inclusão de movimentos laterais no personagem principal. Outro comentário recorrente foi a grande sensibilidade do *mouse*, o que dificultou o controle da mira jogador para alguns *playtesters*. Tal fato provavelmente ocorreu pelo fato da sensibilidade do *mouse* no computador do desenvolvedor (onde foram executados a maioria dos *playtests*) ser mais elevada por escolha pessoal do mesmo. Ainda assim, uma possível correção para esse problema em tempo de execução do jogo seria um controle de sensibilidade interno ao jogo, possivelmente no menu de pausa.

3) **Instruções:**

Um comentário recorrente foi o da falta de explicações mais claras sobre as cores características das três rotas de fuga, além do fato de toda a explicação do jogo ser realizada ainda em seu início, passando muita informação ao Jogador num curto espaço de tempo (ainda que o menu de pausa pudesse ser acessado novamente a qualquer momento). Uma possível melhoria para isso seria a passagem de instruções ao jogador de forma mais diluída, possivelmente com áudio e legendas na tela para um maior entendimento, e explicando cada novo passo no jogo a medida que passos anteriores fossem concluídos.

Outra sugestão apresentada foi a inserção de uma “lista de afazeres”, onde o jogador poderia ver quais itens já coletou e o que ainda precisa fazer, para cada rota de fuga. Tal lista poderia estar presente no menu de pausa, e ter itens riscados a medida que fossem sendo concluídos.

4) **Interface de usuários:**

Possivelmente a parte do jogo que mais apresentou falhas, sua falta de originalidade e contraste aceitável com o resto do mapa causou algumas dificuldades e reclamações, sobretudo no mini-mapa, que não auxiliava muito o jogador na busca por pontos-chave no jogo. Além disso, seu posicionamento e bordas não atraiam muito o olhar do jogador, e a falta de uma bússola e de marcadores dos Prédios-Chave no mapa tornaram seu uso quase inexistente. Uma solução seria a recriação total da Interface de Usuário, buscando contrastar melhor seus elementos com o resto dos elementos na tela, além da criação de um sistema melhor de navegação que indicasse onde se encontram os Prédios-Chave com mais facilidade.

5) **Dificuldade e Balanceamento:**

No geral, não foram observados muitos problemas na dificuldade do jogo, levando em consideração quantidade de itens, força dos ataques do Jogador e pontos de vida dos inimigos. Após cada iteração, quando se julgou necessário, tais valores sofreram leves modificações, e nas últimas rodada de *playtest* não foram apontados mais problemas de balanceamento por parte dos *playtesters*. Tal fato sugere que o jogo se encontra aceitavelmente balanceado para sua atual escala.

Uma sugestão existente foi a de inserção de mais inimigos em pontos relevantes do mapa, como na frente de uma rota de fuga assim que o Jogador liberar seu acesso, para aumentar sua dificuldade. Além disso, foi sugerido que inimigos apareçam em mais pontos do mapa, para que não acumulem em pontos específicos. Tal observação poderia ser implementada criando mais locais de criação de inimigos, além de possivelmente dar preferência para locais mais próximos do jogador ao se criar um novo inimigo.

6) **Conclusões:**

Os maiores problemas apresentados são relativos a UI e a forma de passagem de instruções. Melhorias imediatas poderiam ser feitas nesses dois aspectos, refazendo a

interface de usuário e adicionando elementos como uma bússola e marcadores de pontos-chave no mesmo, e diluindo as instruções ao longo do jogo, possivelmente por meio de áudio e texto na tela de jogo.

Além disso, outras melhorias que poderiam aumentar o fator de *replay* do jogo poderiam ser implementadas, como a criação de mais etapas para cada rota de fuga, inserção de segredos no jogo que incentivassem o jogador a explorar mais o mesmo, e criação de mais mapas de jogo além do atual.

Apêndice C

Questionário de *Playtest*

1) Achou o jogo divertido?

   R:

2) Jogaria de novo?

   R:

3) Compraria o jogo numa versão mais completa?

   R:

4) De 1 à 5, sendo 1 “muito curto” e 5 “muito longo”, indique a duração percebida do jogo. Se cabível, justifique.

   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

   R:

5) De 1 à 5, sendo 1 “poucos inimigos” e 5 “inimigos demais”, indique a quantidade percebida do número de inimigos presentes no jogo. Se cabível, justifique.

   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

   R:

6) De 1 à 5, sendo 1 “pouca munição” e 5 “munição demasiada”, indique a quantidade percebida do número de *power-ups* de munição presentes no jogo. Se cabível, justifique.

   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

   R:

7) De 1 à 5, sendo 1 “poucos itens de vida” e 5 “itens de vida demasiados”, indique a quantidade percebida do número de *power-ups* de recuperação de vida presentes no jogo. Se cabível, justifique.

   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

   R:

8) De 1 à 5, sendo 1 “pouco explicativas” e 5 “muito explicativas”, indique o nível percebido de claridade na explicação das regras do jogo. Se cabível, justifique.

   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

R:

9) De 1 à 5, sendo 1 “pouco intuitivos” e 5 “muito intuitivos”, indique sua opinião acerca dos controles do personagem principal. Se cabível, justifique.

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

R:

10) De 1 à 5, sendo 1 “poucos passos realizados” e 5 “muitos passos realizados”, indique sua percepção acerca da quantidade de passos necessários para a conclusão do jogo. Se cabível, justifique.

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

R:

11) De 1 à 5, sendo 1 “muito difíceis de se encontrar” e 5 “muito fáceis de se encontrar”, indique sua percepção acerca da dificuldade de encontrar os pontos-chave da fase para prosseguir com o jogo. Se cabível, justifique.

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

R:

12) De 1 à 5, sendo 1 “muito chato” e 5 “muito divertido”, indique o que achou do jogo. Se cabível, justifique.

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

R:

13) De 1 a 5, indique o nível percebido de dificuldade geral do jogo. Se cabível, comente sobre outros pontos/problemas/sugestões não mencionados nas nas perguntas acima.

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

R:

Apêndice D

Lista de *assets* utilizados no desenvolvimento

1. MSGDI. **Medium Mech Striker**. Disponível em: <https://assetstore.unity.com/packages/3d/characters/robots/medium-mech-striker-124342>. Acesso em 01 nov. 2018.

2. DYNAMIC ART. **Low Poly Street Pack**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/urban/low-poly-street-pack-67475>. Acesso em 21 nov. 2018.

3. POLYGON BLACKSMITH. **Low Poly Soldiers Demo**. Disponível em: <https://assetstore.unity.com/packages/3d/characters/low-poly-soldiers-demo-73611>. Acesso em 21 nov. 2018.

4. NOBIAX / YUGHUES. **Yughues Free Concrete Barriers**. Disponível em: <https://assetstore.unity.com/packages/3d/props/exterior/yughues-free-concrete-barriers-13248>. Acesso em 21 nov. 2018.

5. REINFORCEDCC. **Free Industrial Building/Garage**. Disponível em <https://assetstore.unity.com/packages/3d/environments/industrial/free-industrial-building-garage-123146>. Acesso em 25 nov. 2018.

6. ANUPOM. **Electricity Tower**. Disponível em: <https://free3d.com/3d-model/electricity-tower-25295.html>. Acesso em 22 nov. 2018.

7. KOBRA GAME STUDIOS. **Chainlink Fences**. Disponível em: <https://assetstore.unity.com/packages/3d/chainlink-fences-73107>. Acesso em 25 nov. 2018.

8. KOBRA GAME STUDIOS. **Storage Building**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/industrial/storage-building-50430>. Acesso em 28 dez. 2018.

9. CGY (YEMELYAN K.). **ATM**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/sci-fi/atm-95057>. Acesso em 25 nov. 2018.

10. PROFI DEVELOPERS. **Building Shed**. Disponível em: <https://assetstore.unity.com/packages/3d/building-shed-1042>. Acesso em 25 nov. 2018.

11. COLTAO. **Gerador2**. Disponível em: <https://www.turbosquid.com/FullPreview/Index.cfm/ID/440245>. Acesso em 27 nov. 2018.

12. ALIYEREDON. **White City**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/urban/white-city-76766>. Acesso em 27 nov. 2018.

13. DEVOTID. **Vehicle Parking Lot Garage Gate PBR.** Disponível em: <https://assetstore.unity.com/packages/3d/environments/roadways/vehicle-parking-lot-garage-gate-pbr-111423>. Acesso em 27 nov. 2018.

14. SKJ. **Particle Collection SKJ 2016_Free samples**. Disponível em: <https://assetstore.unity.com/packages/vfx/particles/particle-collection-skj-2016-free-samples-72399>. Acesso em 28 nov. 2018.

15. RIK4000. **UK Terraced Houses Pack FREE**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/urban/uk-terraced-houses-pack-free-63481>. Acesso em 28 nov. 2018.

16. ANDREJ MIKUS. **Sport Car - 3D model**. Disponível em: <https://assetstore.unity.com/packages/3d/characters/sport-car-3d-model-88076>. Acesso em 3 dez. 2018.

17. CHERMANDIRKUN. **Town Houses Pack.** Disponível em: <https://assetstore.unity.com/packages/3d/environments/urban/town-houses-pack-42717>. Acesso em 3 dez. 2018.

18. DR. BEAN. **Open Building(worn)**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/open-building-worn-112907>. Acesso em 3 dez. 2018.

19. REINFORCEDCC. **FREE Industrial building/Garage**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/industrial/free-industrial-building-garage-123146>. Acesso em 4 dez. 2018.

20. CORE GAMES STUDIO. **Fire Explosion VFX**. Disponível em: <https://assetstore.unity.com/packages/vfx/particles/fire-explosions/fire-explosion-vfx-48795>. Acesso em 4 dez. 2018.

21. KAJAMAN. **Background Car - Free.** Disponível em: <https://assetstore.unity.com/packages/3d/vehicles/land/background-car-free-87053>. Acesso em 4 dez. 2018.

22. ALEKSEY KOZHEMYAKIN. **Abandoned buildings**. Disponível em: <https://assetstore.unity.com/packages/3d/environments/abandoned-buildings-62875>. Acesso em 7 dez. 2018

23. VIS GAMES. **Body Splatter Parts**. Disponível em: <https://assetstore.unity.com/packages/3d/characters/body-splatter-parts-17408> Acesso em 8 dez. 2018.

24. UNITY TECHNOLOGIES. **Raw Mocap Data for Mecanim.** Disponível em: <https://assetstore.unity.com/packages/3d/animations/raw-mocap-data-for-mecanim-5330>. Acesso em 13 dez. 2018.

25. JEAN MORENO. **War FX**. Disponível em: <https://assetstore.unity.com/packages/vfx/particles/war-fx-5669>. Acesso em 18 dez. 2018.

26. CODISCITE. **Cursors and Crosshairs Ultimate Pack.** Disponível em: <https://assetstore.unity.com/packages/tools/gui/cursors-and-crosshairs-ultimate-pack-109530>. Acesso em 30 dez. 2018.

27. LEMMOLAB. **Urban Building**. Disponível em: <https://assetstore.unity.com/packages/3d/props/exterior/urban-building-130318>. Acesso em 29 dez. 2018.

28. THUNDERENT'S ASSETS. **Fountain Prop**. Disponível em: <https://assetstore.unity.com/packages/3d/fountain-prop-75912>. Acesso em 29 dez. 2018.

29. VLAD OCS. **4 Door Sports Car - Mobile.** Disponível em: <https://assetstore.unity.com/packages/3d/characters/4-door-sport-car-mobile-104177>. Acesso em 29 dez. 2018.

30. BEN DROSTE. **Stone Fence.** Disponível em: <https://assetstore.unity.com/packages/3d/props/exterior/stone-fence-2437>. Acesso em 29 dez. 2018.

31. 000734. **Ruined Car.** Disponível em: <https://assetstore.unity.com/packages/3d/vehicles/ruined-car-5909>. Acesso em 29 dez. 2018.

32. CERBERUS. **Modern Sports Car.** Disponível em: <https://assetstore.unity.com/packages/3d/vehicles/land/modern-sports-car-2196>. Acesso em 29 dez. 2018.

33. CERBERUS. **Modern Sports Car 5.** Disponível em: <https://assetstore.unity.com/packages/3d/vehicles/land/modern-sports-car-5-2203>. Acesso em 29 dez. 2018.

34. MOJO-STRUCTURE. **T95 Super Heavy Tank.** Disponível em: <https://assetstore.unity.com/packages/3d/vehicles/land/t95-super-heavy-tank-101164>. Acesso em 29 dez. 2018.

35. IOAN STAN. **Arcade Machine Free.** Disponível em: <https://assetstore.unity.com/packages/3d/arcade-machine-free-92191>. Acesso em 30 dez. 2018.

36. WILL B. **RAZ-Alien.** Disponível em: <https://assetstore.unity.com/packages/3d/characters/humanoids/raz-alien-15043>. Acesso em 5 jan. 2019.

37. CATASTIC. **Zombie Sound Pack - Free Version.** Disponível em: <https://assetstore.unity.com/packages/audio/sound-fx/zombie-sound-pack-free-version-124430>. Acesso em 5 jan. 2019.

38. 3D-BROTHERS. **Basic Motion**. Disponível em: <https://assetstore.unity.com/packages/3d/animations/basic-motion-25834>. Acesso em 5 jan. 2019.

39. PXLTIGER. **Zombie**. Disponível em: <https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232>. Acesso em 6 jan. 2019.

40. MGSVEVO. **Classic Skybox**. Disponível em: <https://assetstore.unity.com/packages/2d/textures-materials/sky/classic-skybox-24923>. Acesso em 18 jan. 2019.

41. DL Sounds. **Organic Beat**. Disponível em: <https://www.dl-sounds.com/royalty-free/tag/angry/>. Acesso em 23 jan. 2019.

42. ANTHOUSAI. **Bullet Shells – linearly 02**. Disponível em: <https://freesound.org/people/Anthousai/sounds/337234/>. Acesso em 23 jan. 2019.

43. FRIDOBECK. **MACHINEGUN_SHOOTING_004**. Disponível em: <https://freesound.org/people/fridobeck/sounds/234615/>. Adaptado. Acesso em 23 jan. 2019.

44. SMCAMERON. **missile_launch_2.wav**. Disponível em: <https://freesound.org/people/smcameron/sounds/51468/>. Acesso em 28 jan. 2019.

45. JUANCAMILOORJUELA. **Pick-up Health.** Disponível em: <https://freesound.org/people/juancamiloorjuela/sounds/204318/>. Acesso em 3 fev. 2019.

46. NIOCZKUS. **1911 Reload.** Disponível em: <https://freesound.org/people/nioczkus/sounds/396331/>. Acesso em 3 fev. 2019.

47. JUSKIDDINK. **Computer startup.wav.** Disponível em: <https://freesound.org/people/juskiddink/sounds/122683/>. Adaptado. Acesso em 3 fev. 2019.

48. LEWIS BAUER. **Beef'd Font**. Disponível em: <https://www.1001freefonts.com/beef-d.font>. Acesso em 10 fev. 2019.