Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

De jogador a desenvolvedor

Analisando o potencial de jogos no desenvolvimento de software livre.

Mateus Santos Freire

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Wilson Kazuo Mizutani

São Paulo
2023

Agradecimentos

Agradeço a minha família por me fornecer uma vida em que posso sentir orgulho de quem sou, e a meus amigos por me apoiarem no caminho que decidi seguir. Agradeço, mais do que ninguém, à minha querida mãe, por ser um pilar de resiliência e fortitude, e me ensinar a não desistir, e sempre ser capaz de me levantar e prosseguir seja o que a vida colocar em meu caminho. Por fim, agradeço a meu supervisor por acreditar em minha visão do grande valor que jogos têm em nosso mundo.

Resumo

Mateus Santos Freire. **De jogador a desenvolvedor: *Analisando o potencial de jogos no desenvolvimento de software livre.***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A relevância do software livre e aberto nos dias atuais mostra que o trabalho voluntário e contribuição de comunidades são um ótimo benefício para a evolução de software. Porém, a barreira de entrada da área de programação pode afastar um grande número de possíveis desenvolvedores. Alguns videogames, porém, também tomam uso de contribuições feitas por sua comunidade para crescer, tendo uma curva de aprendizado amenizada por seu objetivo principal ser o lazer. O objetivo deste trabalho foi a análise de jogos que fornecem ao jogador a capacidade de criar seu próprio conteúdo, de modo a averiguar o potencial de tais jogos como porta de entrada para o âmbito de desenvolvimento, e observar as práticas de acessibilidade que aplicam que poderiam ser estabelecidas para reduzir a barreira de entrada do ambiente FOSS. A análise foi feita de forma padronizada em três categorias de jogos: *Makers*, Jogos abertos a *Mods* e Plataformas de Jogos. *Makers* são jogos onde jogadores podem criar níveis. *Mods* oferecem ao jogador a habilidade de alterar e adicionar conteúdo a um jogo pré-existente. Plataformas de Jogos são jogos que fornecem ferramentas para a construção de outros jogos. Para cada categoria, foram analisadas as formas como o jogo oferece a função de desenvolvedor ao jogador, além de ser avaliada a sua efetividade em nutrir o interesse de jogadores em se tornarem desenvolvedores de software. Para efetuar a avaliação, foram definidas 5 métricas: Acessibilidade, Barreira de Conhecimento, Aprendizado na Prática, Interação com Comunidade e Motivação para Desenvolvimento. Analisando a forma como jogadores são atraídos por esses jogos, foi também proposta a possibilidade de extrair tais práticas e inserir as mesmas no ecossistema de software livre a fim de reduzir sua barreira de entrada para potenciais desenvolvedores emergentes. Ao desenvolver da análise, *Makers* se mostraram como a categoria mais acessível para públicos leigos obterem aprendizado de conceitos computacionais, enquanto que *Mods* se mostraram como a mais efetiva porta de entrada ao oferecer a experiência e ambiente mais próximo ao de reais desenvolvedores. Com os conhecimentos obtidos, é proposto o maior uso do elemento de lazer e criatividade presentes em jogos para tornar o aprendizado de programação mais cativante para iniciantes. Devido à falta de estudos referentes à transição de jogadores para desenvolvedores de software, as conclusões extrapoladas por essa pesquisa podem ser utilizadas como premissas para pesquisas, com o objetivo de gerar dados concretos que validam a eficácia de jogos digitais na introdução de pessoas à carreira de desenvolvedores.

Palavras-chave: Jogos digitais. Desenvolvimento de jogos. Software livre. Software de código aberto. Modificações em jogos.

Abstract

Mateus Santos Freire. **From player to developer: *An analysis on the potential of games in open-source software***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

The relevance of open and free software in recent times shows that volunteer work and community contributions are a great benefit for the evolution of software. However, the barrier to entry in the programming field can put off a large number of potential developers. Some video games, however, also make use of contributions made by their community to grow, with a learning curve softened by the fact that fun is their main objective. The aim of this project was to analyze games that provide the player with the ability to create their own content, in order to ascertain the potential of such games as a gateway to the development sphere, and to observe the accessibility practices that they apply that could reduce the barrier to entry to the FOSS environment. We applied a standardized analysis on three categories of games: *Makers*, Games open to *Mods*, and Game Platforms. *Makers* are games where players can create levels. *Mods* offer the player the ability to alter and add content of and to a pre-existing game. Game Platforms are games that provide tools for building other games. For each category, the ways in which the game offers the role of developer to the player were analyzed, as well as their effectiveness in nurturing players' interest in becoming software developers. To carry out the evaluation, 5 metrics were defined: Accessibility, Knowledge Barrier, Learning in Practice, Interaction with Community and Motivation for Development. By analyzing how players are attracted to these games, we also proposed the possibility of extracting these practices and inserting them into the free software ecosystem in order to reduce the barrier to entry for potential emergent developers. During the analysis, *Makers* proved to be the most accessible category for laypeople to learn computing concepts, while *Mods* proved to be the most effective gateway by offering an experience and environment closer to that of real developers. With the knowledge obtained, we propose to make greater use of the element of leisure and creativity present in games to make learning programming more engaging for beginners. Due to the lack of studies on the transition from gamers to software developers, the conclusions extrapolated by this research can be used as premises for surveys, with the aim of generating concrete data that validates the effectiveness of digital games in introducing people to the carreer of developers.

Keywords: Videogames. Game development. Free software. Open source software. Game modding.

Lista de abreviaturas

FOSS	Software de código livre e aberto (*Free and Open-Source Software*)
GPL	Licença Pública Geral GNU (*GNU General Public License*)
FPS	Tiro em primeira pessoa (*First-Person Shooter*)
ESRB	Entertainment Software Rating Board (*Entertainment Software Rating Board*)
WAD	Onde estão todos os dados (*Where's All The Data*)
DIY	Faça-Você-Mesmo (*Do-It-Yourself*)
DMCA	Lei dos Direitos Autorais do Milênio (*Digital Millennium Copyright Act*)
CFAA	Lei Fraude e Abuso de Computador (*Computer Fraud and Abuse Act*)
EULA	Acordo de licença do usuário final (*End-User License Agreement*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Sumário

1 Introdução **1**
1.1 Motivação . . . 1
1.2 Objetivos . . . 2
1.3 Estrutura do Texto . . . 2

2 Conceitos **3**
2.1 Jogos Eletrônicos . . . 3
2.2 Jogos e Software Livre . . . 4
2.3 Game Engines . . . 5
2.4 Doom . . . 6
2.5 Jogo e Ferramenta . . . 7

3 Metodologia **9**
3.1 Estrutura da Análise . . . 9
3.2 Critérios de Escolha de Ferramentas . . . 10
3.3 Métricas de Efetividade . . . 10
3.3.1 Acessibilidade . . . 10
3.3.2 Barreira de Conhecimento . . . 10
3.3.3 Aprendizado na Prática . . . 11
3.3.4 Interação com Comunidade . . . 11
3.3.5 Motivação para Desenvolvimento . . . 11

4 *Makers*: Jogos de construção de níveis **13**
4.1 Caracterização . . . 13
4.1.1 Conceitos . . . 14
4.1.2 Características . . . 15
4.2 Práticas . . . 15
4.2.1 Editor de Níveis . . . 16
4.2.2 Sistema de Compartilhamento de Níveis . . . 18

4.2.3 Visualizador de Níveis . . . 18
4.2.4 Tabela de Classificação . . . 19
4.3 Efetividade . . . 19
4.3.1 Acessibilidade . . . 20
4.3.2 Barreira de conhecimento . . . 21
4.3.3 Aprendizado na Prática . . . 21
4.3.4 Interação com Comunidade . . . 22
4.3.5 Motivação para Desenvolvimento . . . 23

5 *Mods*: Jogos abertos à extensão e customização 25
5.1 Caracterização . . . 25
5.1.1 Tipos de Mods - Mods de Customização . . . 27
5.1.2 Tipos de Mods - Mods de Extensão . . . 28
5.1.3 Características . . . 30
5.2 Práticas . . . 31
5.2.1 Data-Driven Design . . . 31
5.2.2 Design Orientado a Objetos . . . 31
5.2.3 Organização de Arquivos . . . 33
5.2.4 Ferramentas Auxiliares . . . 34
5.2.5 Plataformas de Hospedagem . . . 35
5.3 Efetividade . . . 37
5.3.1 Acessibilidade . . . 37
5.3.2 Barreira de conhecimento . . . 38
5.3.3 Aprendizado na Prática . . . 39
5.3.4 Interação com Comunidade . . . 39
5.3.5 Motivação para Desenvolvimento . . . 41
5.4 Jogos derivados de Mods . . . 42
5.4.1 Counter-Strike . . . 42
5.4.2 The Stanley Parable . . . 42
5.4.3 Dota . . . 43

6 Plataformas para a criação e compartilhamento interno de jogos. 45
6.1 Caracterização . . . 45
6.1.1 Características . . . 46
6.2 Práticas . . . 46
6.2.1 Construtor de Jogos . . . 46
6.2.2 Monetização . . . 50
6.3 Efetividade . . . 52

6.3.1 Acessibilidade 52
6.3.2 Barreira de conhecimento 53
6.3.3 Aprendizado na Prática 54
6.3.4 Interação com Comunidade 54
6.3.5 Motivação para Desenvolvimento 55

7 Conclusão **57**
7.1 Trabalhos Futuros 58

Referências **59**

Capítulo 1

Introdução

Ao oferecer desenvolvedores a chance de diretamente se integrar a um produto, o modelo *Free and Open-Source-Software* (FOSS) oferece um potencial de produzir tecnologias de forma mais rápida, inovadora, flexível, e barata (Sharma *et al.*, 2002). Nos dias atuais, a ascensão do código aberto nos mostrou que o trabalho voluntário e uma comunidade ativa são componentes vitais quando se trata do desenvolvimento de software livre. Porém, devido à natureza intimidadora da programação como um todo, é possível argumentar que o alcance e acessibilidade da área de desenvolvimento é majoritariamente reduzida por sua curva de aprendizado, acentuada por fatores como falta de documentação ou apoio da comunidade.

Em contraponto, certos videogames como **Minecraft (Mojang, 2011)** e **Super Mario Maker (Nintendo, 2015)**, mesmo não sendo explicitamente software livre, conseguiram construir comunidades extensas de produtores de conteúdo ao longo dos anos, devido à sua natureza muito mais acessível.

1.1 Motivação

Devido a certos jogos serem desenvolvidos com a contribuição de seus jogadores em mente, onde a produção de conteúdo realizada pelo público-alvo leigo é parte integral do produto sendo vendido, essa área da mídia teve grande sucesso em atrair pessoas com interesse em desenvolvimento (NintendoOfAmerica, 2016). Abrangendo simples modificações no comportamento de jogos como os inúmeros *mods* de Minecraft, até novos projetos programados em ferramentas de desenvolvimento de jogos especializadas como as *conversões totais* **Counter-Strike (Valve, 2000)** e **DotA (IceFrog, 2003)**, foram criados, em jogos, espaços de desenvolvimento que em grande parte, sugerem oportunidades na busca de maior acessibilidade em software.

Essa porta de entrada para o desenvolvimento é efetiva por atrair pessoas de todas as idades e áreas profissionais, sendo a interatividade e customização uma forma simples e divertida de expressar a criatividade do jogador. Além disso, oferece uma forte motivação para aprender, um dos maiores fatores de desistência entre aqueles que desejam se tornar desenvolvedores (Kinnunen e Malmi, 2006). Sendo assim, analisando as formas como

desenvolvedores de jogos como a Mojang e Nintendo atraem novas pessoas ao desenvolvimento, podemos julgar a efetividade de jogos como uma porta de entrada ao âmbito de software livre, assim como possivelmente encontrar meios de trazer essa alta acessibilidade ao ecossistema FOSS.

1.2 Objetivos

O objetivo deste trabalho será a análise de diferentes práticas que são tomadas por desenvolvedores de jogos eletrônicos que instigam o interesse dos jogadores em desenvolver através da habilidade de adicionar, modificar e criar seu próprio conteúdo para o jogo. Analisando as formas como jogos utilizam sua natureza acessível para construir comunidades de desenvolvedores nascentes, podemos utilizar essas formas como influência para elevar a acessibilidade do código aberto.

Para o propósito desta análise, a acessibilidade e facilidade de customização serão divididos em três principais categorias de jogos, sendo essas: (i) Jogos de construção de níveis, (ii) Jogos com suporte oficial a *Mods*[1], e (iii) Plataformas para a criação e compartilhamento interno de jogos. O principal esforço deste projeto então, será a análise de cada uma destas categorias, trazendo exemplos de jogos onde cada política de desenvolvimento foi escolhida, assim como sua efetividade em engajar seus jogadores a se interessar em desenvolvimento. Com os resultados da análise, poderemos extrair conclusões tanto sobre a efetividade de jogos como possível porta de entrada para o ramo de desenvolvimento de software livre, como também possíveis práticas que poderiam ser aplicadas para elevar a acessibilidade do ambiente FOSS para potenciais desenvolvedores.

1.3 Estrutura do Texto

No Capítulo 1, introduzimos as motivações e objetivos da análise a ser feita. No Capítulo 2, serão explanados conceitos gerais que serão relevantes para o entendimento da análise. No Capítulo 3, será explicada a forma como a análise de cada categoria de jogos será feita. No Capítulo 4, serão analisados jogos de construção de níveis, ou *Makers*. No Capítulo 5, analisaremos os *Mods*, jogos abertos à customização e adições por jogadores. No Capítulo 6, será feita a análise de Plataformas de Jogos, jogos cujo objetivo é a criação de outros jogos. Por fim, no capítulo 7, agruparemos os resultados obtidos ao longo da análise e derivaremos conclusões.

[1] O significado de *Mods* será explicado no Capítulo 5

Capítulo 2

Conceitos

Serão explicados aqui certos conceitos gerais, relevantes para a melhor compreensão do resto da análise.

2.1 Jogos Eletrônicos

Videogames, conhecidos como **jogos eletrônicos**, ou apenas **jogos**, são uma gênero de mídia de software onde um jogador fornece uma entrada e recebe um feedback visual e/ou auditivo em resposta. Essa interação é então prolongada e reiterada de forma que é possível simular digitalmente um jogo ou atividade convencional, como pode ser visto na Figura 2.1

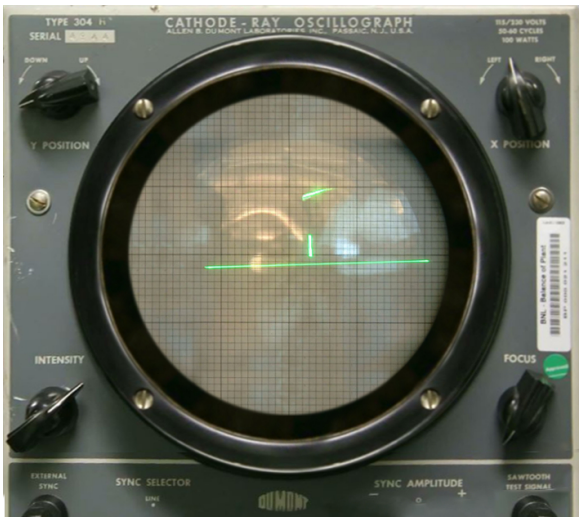

Figura 2.1: *Exibido em um osciloscópio, o jogo Tennis For Two, desenvolvido pelo físico William Higinbotham em 1958, é considerado um dos primeiros videogames já feitos. O videogame simula uma quadra de tênis onde um jogador utiliza um controle para tentar rebater a bola para o outro lado. Fonte: https://en.wikipedia.org/wiki/Tennis_for_Two*

Pode ser dito que a experiência de jogar grande parte dos videogames é similar a de **percorrer um labirinto tradicional**. Dependendo da forma em que um labirinto é construído, é possível que existam múltiplas entradas, caminhos ou saídas, assim como particulares regras ou soluções. Porém, o que define um labirinto em questão é o fato de que as entradas, saídas e caminhos não podem ser alterados após sua construção. Não importa a forma em que o "jogador" possa resolver o labirinto — quais caminhos levou, a velocidade em que o percorreu, a dificuldade que teve em chegar a alguma saída ou até se foi capaz de encontrá-la — pois todas as possíveis formas de percorrer o caminho entre os dois estarão contidas no labirinto em si. Todas as possibilidades são definidas pelo jogo antes mesmo dele começar, tal como uma partida de xadrez. O que podemos dizer com isso é que na maioria dos videogames, o conteúdo do jogo é construído totalmente pelos desenvolvedores, e o papel do jogador é apenas explorar esse conteúdo da forma que desejar, sem poder romper a estrutura já definida.

Nesta análise, trataremos de jogos que oferecem ao jogador a função de desenvolvedor, que diferem desta grande maioria por permitir que o jogador "altere o labirinto" após sua construção. As possibilidades não estarão mais contidas por completo na estrutura do jogo, pois a própria estrutura do jogo pode ser modificada. A forma em que o jogador pode modificar a estrutura será agrupada em categorias, assim como conceitos relevantes a elas.

2.2 Jogos e Software Livre

Videogames tiveram uma inerente conexão ao software livre. Assim como software, jogos feitos com código aberto eram apenas o natural, distribuídos livremente por desenvolvedores sem a presença ou necessidade de licença alguma. Com o aumento do software proprietário e o grande declínio da indústria de jogos no período de popularização dos *home computers* no início dos anos 80, o desenvolvimento de jogos se tornara um campo majoritariamente fechado (Hempson, 2011). Porém, ao desenvolver do movimento de software livre fundando por Richard Stallman (Edward M. Corrado e Mitchell, 2018) e liderado pelo GNU Project, muitos desenvolvedores de jogos aderiram à causa, com diversos desenvolvedores de jogos lançando seus jogos como software livre, licenciados pela nova licença *GNU General Public License (GPL)*. Como tal ação dos desenvolvedores de jogos foi feita em um período de alta popularização de PCs, criou-se um avanço notável na participação do software livre no desenvolvimento de jogos entre desenvolvedores independentes.

Desde então, assim como o código livre estendeu seu alcance e encontra-se em um processo de constante normalização, a utilização de tal em desenvolvimento de jogos segue o mesmo caminho. Nos anos atuais, grande parte da ascendente geração de **jogos independentes** utilizam software livre em seu desenvolvimento (Blake, 2011).

2.3 Game Engines

Game engines, ou **motores de jogos**, são arcabouços de software compostos por uma coleção de bibliotecas, interfaces e ferramentas designadas para a criação de jogos eletrônicos (Sherrod, 2006). Sendo um arcabouço, o objetivo de um motor de jogo é abstrair o código de baixo nível e prover uma forma generalizada de desenvolver um certo software, no caso, um videogame.

Grande parte das grandes empresas de jogos possuem suas próprias *game engines* proprietárias, que usam para desenvolver jogos licenciados, sendo o processo inteiro fechado. Porém, atualmente, *game engines* lançadas como software livre e aberto como a Godot (ilustrada na Figura 2.2) se tornaram mais populares (Dealessandri, 2020).

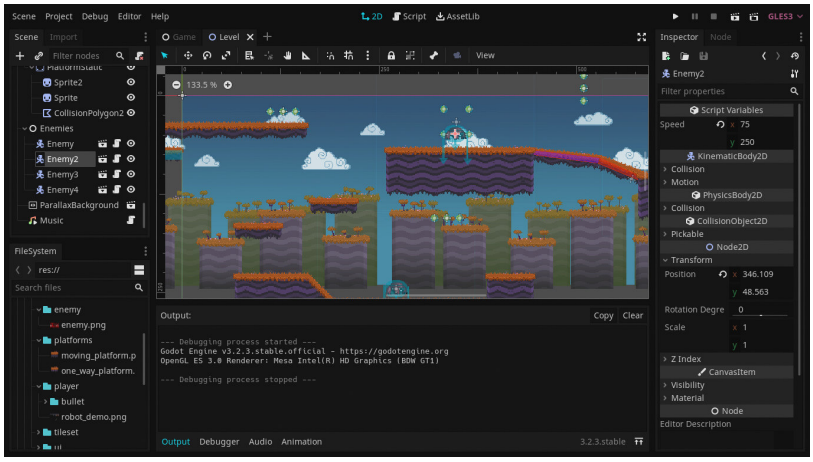

Figura 2.2: *Um jogo de plataforma sendo desenvolvido na game engine Godot. Fonte: https://en.wikipedia.org/wiki/Game_engine*

Nos primórdios do desenvolvimento de jogos, todo videogame era programado puramente utilizando linguagens de programação ou código de máquina do início ao fim, sem espaço para qualquer abstração. Porém, quando foi notado que certos componentes de código poderiam ser reusados múltiplas vezes para a criação de múltiplos jogos, particularmente no fim dos anos 80, houve a criação de ferramentas e interfaces que implementavam tais componentes, elevando o nível de abstração. Esses componentes então se tornaram as *game engines* (Anderson *et al.*, 2008). Ao longo dos anos, esses componentes se tornaram arcabouços, e o processo de desenvolvimento de jogos se tornou mais simples e generalizado, podendo uma única *engine* ser usada para desenvolver jogos de todos os gêneros.

Mesmo através do uso de um arcabouço, a abstração em *game engines* como um todo não se elevou ao ponto de conhecimento programático não ser necessário para seu uso. Logo, pode se dizer que usuários de uma *game engine* estão de fato, desenvolvendo software. Dessa forma, **motores de jogos não serão discutidos em nossa análise.** Como o objetivo da análise é identificar práticas em ferramentas atribuídas a jogos que motivem jogadores a se tornarem desenvolvedores, explorar as *game engines* como uma porta de entrada não seria produtivo, uma vez que o usuário já atua como um desenvolvedor de software ao usá-las.

2.4 Doom

Um jogo excepcional para a história de software livre no âmbito de desenvolvimento de jogos é o *First-Person shooter* (FPS), ou Tiro em Primeira Pessoa, chamado *Doom**(id Software, 1993)**. Doom* é considerado o pai do gênero FPS, e um dos jogos mais importantes da história, sua influência mudando completamente a percepção do que era um videogame na época em que foi lançado e nas muitas décadas a seguir (Therrien, 2015), sendo citado até hoje em inúmeros artigos e trabalhos acadêmicos. Entre os inúmeros feitos do jogo, destacam-se: **(i)** a popularização do termo *"game engine"*(Balasubramanian, 2022) e dos jogos de computador nos anos 90; **(ii)** ser um dos motivos para a fundação da empresa **ESRB** (Entertainment Software Rating Board), responsabilizada pelo maior sistema de classificação etária indicativa utilizado para jogos; **(iii)** ser uma influência na fundação do modelo de negócios de distribuição online (Zachary, 1996); **(iv)** e provocar a ascensão de jogos em rede, tecnologia licenciada entre desenvolvedores, geração de conteúdo designado a jogadores, e modificações em jogos (Donovan, 2010).

Uma das características mais vitais do jogo para esta análise é a *engine* na qual *Doom* foi criado. A denominada *Doom Engine*, programada nas linguagens de programação C e *Assembly*, e considerada a primeira *game engine*, ilustrou os grandes benefícios da reutilização de código na forma em que separava os sistemas abstraídos pelo software, e o conteúdo do jogo em si. O jogo não era desenvolvido dentro da engine, mas apenas **renderizado** dentro dela. Toda a informação do jogo, como níveis, música e gráficos, eram salvos em arquivos *Where's All The Data* (WAD), que então eram lidos pela *engine* e então renderizados para "construir" o jogo a partir deles. Uma representação visual do arquivo pode ser observada na Figura 2.3.

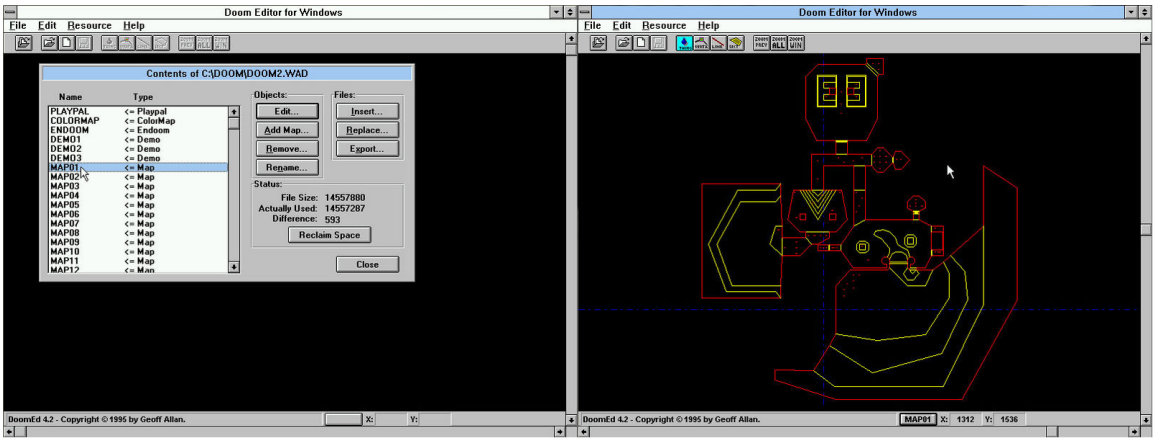

Figura 2.3: *Na esquerda, temos um arquivo WAD, extensão para arquivos que contêm toda a informação pertinente ao jogo Doom, como mapas, gráficos, e música, aberto no editor de níveis* ***DoomEd 4.2****, uma extensão do DoomEd, o editor de níveis original utilizado pela id Software no desenvolvimento do jogo. Na direita, vemos a forma em que um mapa dentro do arquivo WAD é lido e então renderizado no momento em que o jogo inicia. Fonte: https://doomwiki.org/wiki/DoomEd_4.2*

Devido a isso, utilizando ferramentas para alterar esses arquivos WAD, o conteúdo que compunha o jogo poderia ser alterado livremente e então reconhecido pela engine, tornando possível a modificação e criação de novos níveis, estabelecendo uma expansiva e

inovadora comunidade *online*, onde pessoas compartilhavam seus arquivos WAD, contendo modificações no conteúdo do jogo, além de novos níveis e gráficos. Após o lançamento do código fonte da *Doom Engine* de forma aberta em 1999, através da *GNU General Public License*, múltiplas versões alternativas do software foram criadas, resultando em inúmeros produtos, tanto livres quanto comerciais, a serem desenvolvidos utilizando a *engine* e o jogo como base, tornando muitos de seus jogadores em futuros desenvolvedores e *designers* de jogos.

A forma como *Doom* popularizou a criação de níveis customizados e o conceito de modificações de jogos em nível de software e *engine* se mostrará relevante ao longo deste projeto, influenciando todas as três categorias a serem discutidas. Afinal, foi este o jogo que estabeleceu a cultura em jogos de compartilhamento e desenvolvimento aberto que estamos analisando.

2.5 Jogo e Ferramenta

Um conceito importante a introduzir para a leitura dessa análise é a distinção entre **jogo** e **ferramenta**, assim como o título do indivíduo que interage com tal. Nesta análise, um **jogo** é um produto desenvolvido por seus **desenvolvedores**. O jogo, em si, poderá conter **ferramentas** que oferecem ao **jogador** a habilidade de interagir com o conteúdo do jogo de alguma forma. Esse **jogador**, mesmo exercendo funções de desenvolvedor, não deve ser confundido com os desenvolvedores do jogo em si, pois estão apenas tomando uso das **ferramentas** proporcionadas pelo jogo após seu desenvolvimento. Dessa forma, jogos **contém** ferramentas, e jogadores se tornam **usuários** dessas ferramentas para interagir com o conteúdo do jogo.

Capítulo 3

Metodologia

Para entender como desenvolvedores de jogos constroem, em seus produtos, ferramentas que atraem jogadores com potencial interesse em desenvolvimento de software, será feita uma análise dessas ferramentas, sua implementação, e as práticas que instigam este potencial.

Práticas diferem fortemente de acordo com o nível de customização e controle que a ferramenta fornece ao usuário, logo perderíamos informações ao analisar todas as ferramentas nessa escala em conjunto. Iremos então realizar a análise de múltiplas categorias de ferramentas, informadas no Capítulo 1. Entretanto, para poder compará-las, é necessário padronizar o processo de análise, estabelecendo uma estrutura definida para tal.

3.1 Estrutura da Análise

A análise de cada categoria seguirá uma estrutura composta pelas seções **Caracterização, Práticas e Efetividade**. Dentro de tais seções, as seguintes perguntas visarão ser respondidas:

- **Caracterização:** "O que define esta categoria? Por que uma ferramenta estaria nesta categoria em vez de outra? Quais conceitos são pertinentes a esta categoria?"
- **Práticas:** "Que práticas presentes nestas ferramentas oferecem ao usuário a função de desenvolvedor? Como elas funcionam? Como elas podem ser implementadas em futuras ferramentas da mesma categoria?"
- **Efetividade:** "Essas práticas atraem possíveis desenvolvedores? Por que elas são capazes ou não de fazer isso?

  Em conclusão, averiguaremos a forma como cada política surgiu, e como é implementada durante o escopo do desenvolvimento. Em seguida, observaremos exemplos de jogos onde tal política foi escolhida, assim ganhando conhecimento de seu alcance e como tal funciona de forma prática. Após analisar todas as categorias, serão agrupados os resultados obtidos e utilizaremos eles para comparar suas efetividades.

3.2 Critérios de Escolha de Ferramentas

A escolha de ferramentas para cada uma das categorias não é arbitrária. Além dos fatores exclusivos à categoria, explanados na seção 3.1, existem certos critérios universais que toda ferramenta deve respeitar para ser elegível à análise. Tais critérios são:

- O jogo ou ferramenta deve estar, no tempo da escrita deste artigo, abertamente disponível, seja por compra em alguma plataforma de venda ou por oferecimento oficial do desenvolvedor.
- As ferramentas que oferecem ao jogador a habilidade de modificar seu funcionamento devem ser reconhecidas de forma oficial pelos desenvolvedores originais do produto. Logo, modificações de jogos desenvolvidas através da desconstrução não autorizada do código fonte não serão discutidas ou consideradas para análise.

3.3 Métricas de Efetividade

Após compreender *como* a ferramenta induz o desejo de desenvolver em seu usuário, devemos determinar sua **efetividade** em fazer o mesmo, sendo tal efetividade análoga ao potencial da ferramenta em tornar um usuário em um futuro desenvolvedor de software.

Logo, se desejamos encontrar as melhores práticas para aumentar a acessibilidade do ramo de software livre, devemos observar os aspectos em que cada categoria apresenta maior efetividade, e então agrupar e comparar os dados obtidos.

3.3.1 Acessibilidade

Acessibilidade representa a facilidade de obter a ferramenta em questão. Esse aspecto afeta intrinsecamente o alcance da ferramenta. Uma maior facilidade em obter a ferramenta resulta em uma quantidade maior de potenciais usuários, enquanto que uma maior facilidade em utilizar a ferramenta resulta em um nível maior de engajamento. Em conclusão, **um nível maior de acessibilidade indica maior efetividade.** Fatores que influenciam essa métrica são: preço da ferramenta, tempo de lançamento, idade mínima para o uso, acessibilidade e extensividade de documentação.

3.3.2 Barreira de Conhecimento

A Barreira de Conhecimento remete ao nível de conhecimento de desenvolvimento necessário para tomar uso da ferramenta. Uma barreira de conhecimento alta condiz com uma natureza mais intimidadora, porém, estabelecer uma barreira de conhecimento também indica que os usuários das ferramentas possuem o conhecimento de como tal funciona. Em conclusão, **uma barreira de conhecimento promissora seria aquela que não aliena usuários inexperientes, mas também favorece usuários que já possuem experiência com a tecnologia envolvida.** Fatores gerais que influenciam essa métrica são: necessidade de programação, linguagem de programação escolhida, existência de ferramentas auxiliares.

3.3.3 Aprendizado na Prática

A métrica de Aprendizado na Prática remete ao nível de aprendizado programático obtido pelo usuário durante o processo de manusear a ferramenta. Sendo o objetivo da análise encontrar práticas que atraem desenvolvedores, muito valor deve ser dado a práticas que oferecem ao jogador a experiência mas próxima o possível de desenvolver um software de fato. Logo, essa métrica favorece jogos que ensinam, mesmo que indiretamente, conceitos programáticos como funções, padrões de projeto, herança, código limpo, entre outros. Fatores que influenciam essa métrica são: Interação com linguagem de programação e necessidade de resolução de problemas.

3.3.4 Interação com Comunidade

Esta métrica trata da exposição do usuário à comunicação com outros usuários através do uso da ferramenta. Software livre é um ramo baseado em colaboração, onde pessoas ajudam umas as outras, logo se a ferramenta incentiva a interação entre desenvolvedores, isso é um grande fator.

Fatores que influenciam essa métrica são: Existência de ambientes físicos ou online reservados para a discussão ou desenvolvimento relevante à ferramenta, como fóruns, competições, e sites de hospedagem e compartilhamento de conteúdo. Além disso, a interação direta com repositórios, se possível, oferece conhecimento valioso sobre versionamento e interação real com desenvolvedores de software em espaços nativos.

3.3.5 Motivação para Desenvolvimento

Essa métrica remete à efetividade da ferramenta em motivar o usuário a engajar em desenvolvimento de software, mesmo que indiretamente. Em questão da análise de jogos como porta de entrada para potenciais desenvolvedores de software livre, esse fator é de vital importância. Alguns fatores que influenciam essa métrica são: interação com código, incentivos para aprendizado, presença de desenvolvedores na comunidade.

Capítulo 4

Makers: Jogos de construção de níveis

Neste capítulo, analisaremos a categorias dos *Makers*.

4.1 Caracterização

Jogos de construção de níveis, como o nome vem a indicar, são jogos onde sua principal função não é oferecer conteúdo já anteriormente produzido e definido durante seu desenvolvimento. Ao invés disso, oferecem ferramentas de desenvolvimento ao jogador, com o objetivo de lhes dar a capacidade de fazer seu próprio conteúdo por meio de ferramentas pré-definidas. Logo, como o gênero é conhecido pela habilidade do jogador "fazer"o próprio jogo, daqui em diante irei me referir a esta categoria de videogames como ***Makers***.

A emersão e consequente sucesso de jogos *Maker* podem ser associados à contemporânea **cultura Maker** (Contini *et al.*, 2020), uma subcultura do movimento DYI[1]. Essa cultura encoraja um processo de desenvolvimento por aprendizado ativo, onde criações são motivadas primariamente pela diversão e autorrealização (Sharples *et al.*, 2013). Tais criações são resultantes de um ambiente de colaboração e compartilhamento de práticas, informações, e tecnologias em espaços informais chamados de *Makerspaces*. Essa filosofia de desenvolvimento em um espaço convidativo, motivada pela diversão e colaboração, se mostra explícita no principal Maker que será utilizado como exemplo ao longo deste capítulo, o jogo ***Super Mario Maker (Nintendo, 2015)***, segundo seu diretor Yosuke Oshino(Oshino, 2015).

Porém, a cultura Maker é também fortemente relacionada ao movimento de software livre. Hackerspaces[2], uma variação de Makerspaces, são espaços informais onde indivíduos com interesses em diversas tecnologias se reúnem, sua grande maioria participando no uso e desenvolvimento de tecnologias *open source*, sejam tais software ou hardware.

[1] Do-It-Yourself, ou Faça-Você-Mesmo

[2] https://www.npr.org/2010/11/12/131268511/diy-hackers-tinker-everyday-things-into-treasurel

Assim, como a cultura de desenvolvimento de Makers ase alinha com a de software livre, é relevante analisar a forma como Makers funcionam e habilitam seus jogadores a criar, e encontrar formas de reaproveitar as práticas de acessibilidade que tais exercem como jogos para introduzir pretendentes desenvolvedores ao ramo de software livre.

4.1.1 Conceitos

De modo a tornar a análise dos Makers mais compreensível e intuitiva, irei introduzir, primeiramente, alguns conceitos relevantes:

Níveis

Jogos, principalmente aqueles de gênero de ação ou aventura, são majoritariamente divididos em três partes: a interface, o protagonista e o ambiente. O protagonista é considerado a ***base*** da jogabilidade, representando o jogador nas ações que podem ser tomadas em um ambiente. O ambiente de um jogo pode tomar várias formas, porém podemos considerar que todos os jogos são separados em *níveis*, espaços onde o jogador age para atingir algum objetivo(Bates, 2004).

Níveis, sendo o ambiente em que a jogabilidade acontece, devem ser diferenciados do protagonista e da interface do jogo. A interface do jogo, composta por elementos como *menus, inventários, telas iniciais, créditos*, mesmo tendo um vínculo direto com a jogabilidade, interagem com o **jogador**, não o protagonista. Em contraponto, o protagonista será sempre limitado por sua programação, e sua interação com o ambiente se estende apenas até o que foi implementado por um desenvolvedor que tenha a capacidade de alterar suas propriedades.

Em conclusão, enquanto outras categorias de ferramentas que iremos analisar eventualmente fornecerão ao usuário o poder de customizar tanto a aparência quanto a funcionalidade da interface e protagonista do jogo a ser criado, deve ser ressaltado que **apenas níveis podem ser modificados** em Makers.

Objetos

Níveis, por si só, são apenas ambientes. O que compõe esses ambientes chamaremos de **objetos**. Objetos podem tomar várias formas dependendo do jogo, mas abrangem tudo desde plataformas, obstáculos e recompensas.

Todo nível existente em qualquer jogo é composto por sua própria seleção de **objetos**. Em jogos de plataforma, o principal gênero de jogo a se tornarem Makers, objetos são inseridos em uma **grade**, e quando agrupados de forma deliberada, formam um **nível**. A Figura 4.1 contém um exemplo de um nível feito de objetos. A manipulação de *objetos* é a forma pela qual níveis são construídos em Makers.

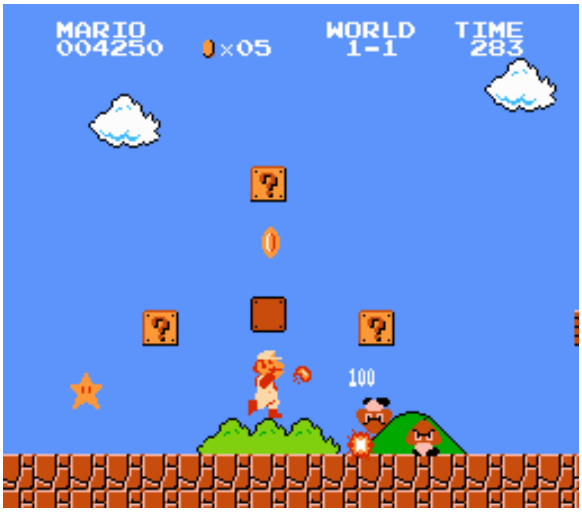

Figura 4.1: *Visualizando o primeiro nível do jogo de plataforma Super Mario Bros. (Nintendo, 1985), vemos que os blocos, inimigos, moedas, estrelas, entre outros, são objetos. Mesmo objetos pequenos como blocos singulares, podem ser agrupados aos montes para se tornar o chão em que o Mario pisa. Fonte: https://pt.wikipedia.org/wiki/Super_Mario_Bros.*

4.1.2 Características

É importante definir o que difere *Makers* de outras categorias de jogos em que o jogador exerce função de desenvolvedor.

- **Níveis são construídos, não criados:** Apesar de seu nome, todo o conteúdo de *Makers* já foi feito pelos desenvolvedores. O jogador apenas utiliza as ferramentas oferecidas pelos desenvolvedores e compõe algo a partir do que já está presente no jogo. O jogo oferece uma variedade de peças, e o jogador constrói com elas, sem poder modificá-las ou adicionar mais - uma política de desenvolvimento que chamarei de blocos de construção.

- **Conteúdo desenvolvido pelo jogador sempre será parte do jogo:** Não é possível levar nada do que foi construído dentro do jogo para fora dele. Assim, o conteúdo feito e compartilhado pelo usuário permanece interno mesmo após a fase de desenvolvimento, sendo impossível jogar o nível montado sem possuir o jogo original.

- **A criação e compartilhamento de conteúdo é o propósito do jogo:** Mesmo se o jogo oferece uma seleção de níveis feitos pelos desenvolvedores, a construção e compartilhamento de níveis por seus jogadores deve permanecer o objetivo do jogo.

4.2 Práticas

Para determinar como *Makers* incentivam seus usuários a construir, analisaremos algumas das práticas que aplicam.

4.2.1 Editor de Níveis

O **editor de níveis**, também chamado de **construtor de níveis**, é a principal ferramenta oferecida pelos Makers. Em uma interface acessível e simples, o usuário, tendo a liberdade de selecionar entre uma quantidade pré-definida de objetos, insere eles em um ambiente para formar seus níveis. Originalmente, editores de níveis foram criados por desenvolvedores durante a fase de produção de jogos, sendo sua função a padronização e facilitação de criação de níveis. Desde então, ocasionalmente, tais editores de níveis foram deixados nos jogos para serem usados pelos jogadores após seu lançamento, ou incluídos como conteúdo adicional. Assim, jogos que fornecem ao jogador ferramentas necessárias para desenvolver seus próprios níveis existem desde os anos 80, como os jogos ***Excite Bike (Nintendo, 1984)*** e ***Wrecking Crew (Nintendo, 1984)*** (demonstrado na Figura 4.2), até os dias de hoje, como o jogo de quebra-cabeça ***Baba is You (Hempuli Oy, 2019)***, demonstrado na Figura 4.3.

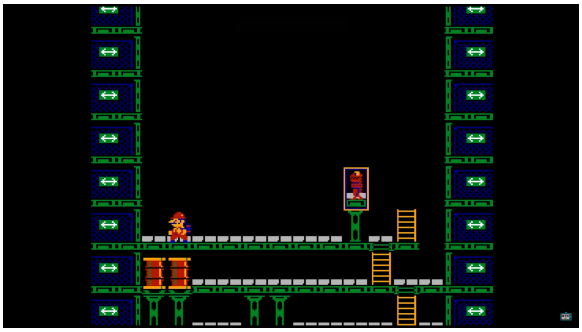

Figura 4.2: *O editor de níveis em Wrecking Crew (Nintendo, 1984). Sem nenhuma interface de seleção de objetos, o jogador é dado apenas um cursor e precisa pressionar um botão para iterar por uma lista de objetos até encontrar o que deseja inserir no nível.* ***Fonte:*** *https://www.youtube.com/watch?v=Es6xVeuU6Ig*

Figura 4.3: *O editor de níveis em Baba is You (Hempuli Oy, 2019). O editor oferece uma interface moderna e intuitiva, onde o jogador possui um menu para selecionar objetos e inserir tais em um ambiente para criar seus níveis. Opções convenientes também são visíveis, como ferramentas de seleção, um botão de desfazer ações e múltiplas camadas para inserir objetos.* ***Fonte:*** *https://www.youtube.com/watch?v=U4W_9rfjSus*

É necessário ressaltar que mesmo oferecendo um construtor de níveis, tais jogos não qualificam para a definição de Makers. Para estes jogos, um editor de níveis é apenas uma

opção secundária, sendo o conteúdo criado pelos desenvolvedores o real ponto focal da jogabilidade, enquanto que para um Maker, o conteúdo gerado pelo próprio jogador é o foco. Porém, um jogo originalmente focado em conteúdo criado por desenvolvedores, dado tempo e suporte suficiente para funcionalidades direcionadas à criação de conteúdo pelos usuários, pode se tornar um Maker, sendo o caso do jogo **Geometry Dash (RobTop Games, 2013)**.

Figura 4.4: *O editor de níveis em Super Mario Maker 2 (Nintendo, 2019), vemos todas as opções oferecidas ao jogador em uma única tela, incluindo o tema do nível, tempo limite, assim como múltiplas categorias de objetos que o jogador pode inserir em seu nível.* ***Fonte:*** *https://www.youtube.com/watch?v=AjJWzJC8Kfk*

Quando se trata da implementação de um editor de níveis, a interface do editor de níveis deve colocar foco no nível em si, reservando as bordas da tela para a interface de seleção de objetos e escolha de elementos como **tempo limite, música, tela de fundo, etc**, demonstrado na Figura 4.4. Devido ao número elevado de objetos, tais são organizados em categorias, como inimigos, terreno, itens, entre outros. Isso pode ser observado na Figura 4.5.

Figura 4.5: *No editor de níveis em Super Mario Maker 2 (Nintendo, 2019), os objetos que o jogador pode inserir nos níveis são divididos em múltiplas categorias.* ***Fonte:*** *https://www.youtube.com/watch?v=AjJWzJC8Kfk*

O jogador então navega o ambiente primeiramente vazio, seleciona objetos dentre estas categorias, e insere tais objetos para construir um nível. A combinação de objetos pode trazer estruturas únicas, e a ferramenta como um todo sempre coloca ênfase na facilidade de testar o nível rapidamente a qualquer momento na fase de criação, através de um botão de teste. Essa facilidade é crucial pois constantes pequenas alterações são comuns na fase de criação, assim como um código que compila rapidamente é mais simples de alterar e consertar. Em implementação, isso significa que a ferramenta deve priorizar a rápida renderização do nível, de forma que o jogador pode começar a jogar o nível em qualquer ponto de sua criação, e após a mudança ser testada, retornar instantaneamente para o editor. Para ter sucesso nisso, ferramentas muitas vezes compilam os níveis enquanto são criados, salvando cada mudança feita em tempo real.

4.2.2 Sistema de Compartilhamento de Níveis

Após a criação de um nível, é possível compartilhar ele pela internet, integrando-o a uma enorme seleção de níveis feitos pela comunidade. Esse sistema de compartilhamento de níveis é a fonte principal de engajamento do jogo, sendo o aspecto de ter seus níveis jogados por outras pessoas a principal motivação para o desenvolvimento de níveis.

O compartilhamento de um nível se inicia antes mesmo da publicação, no processo de verificação. Neste processo, um usuário deve ser capaz de terminar seus próprios níveis antes de compartilhar eles com outras pessoas. Isso garante que níveis impossíveis não sejam acidentalmente oferecidos a jogadores, além de impor um certo padrão de qualidade, motivando desenvolvedores a produzir *designs* acessíveis para um número maior de pessoas.

O nível, após passar pelo processo de verificação, é registrado em nome do jogador e dado um código de identificação antes de ser publicado. Um usuário que deseja jogar o nível publicado pode então encontrar tal nível de diversas maneiras: pelo código de identificação, pelo nome do usuário, ou por sua aparência no **visualizador de níveis**. A implementação se assemelha a um grande banco de dados, onde cada nível é um dado que é inserido após sua publicação, e, então, jogadores podem selecioná-lo para jogar através de uma interface de busca.

Por esse motivo, Makers tendem a aplicar formas de limitar a quantidade de níveis que um usuário pode publicar, afim de reduzir a carga de armazenamento de seus servidores.

4.2.3 Visualizador de Níveis

O Visualizador de Níveis é a interface que ilustra os níveis que são compartilhados pelos jogadores. A forma como um visualizador de níveis é organizado difere com o Maker em que está implementado, porém, a forma mais comum é um menu onde níveis são categorizados por fatores como popularidade, ou recência. Em Super Mario Maker 2 (Nintendo, 2019), demonstrado pela Figura 4.6, níveis são categorizados em: *Popular Courses*, para os níveis de melhor classificação; *New Courses*, para níveis publicados recentemente; e *Hot Courses*,

para níveis recentes que tem alto potencial de se tornarem populares.[1]

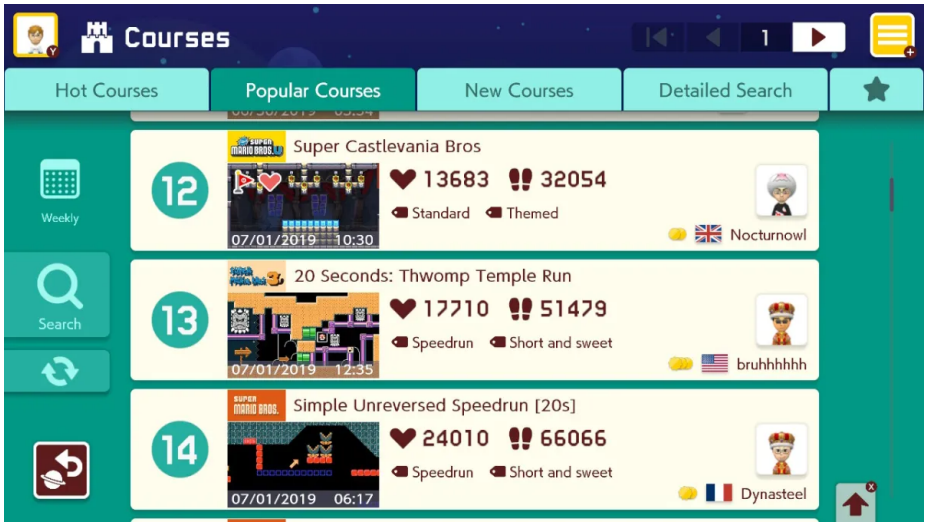

Figura 4.6: *O visualizador de níveis em Super Mario Maker 2 (Nintendo, 2019), um menu onde o jogador pode observar, buscar, e então jogar todos os níveis criados por outros jogadores.* ***Fonte:*** *https://techcrunch.com/2019/07/03/with-super-mario-maker-2-nintendo-both-unleashes-and-leashes-creators/*

Como dito antes, porém, um visualizador de níveis deve fornecer uma ferramenta de busca, permitindo jogadores a encontrar níveis específicos através de seu código de identificação ou autor. Um nível, quando selecionado no visualizador de níveis, demonstrará informações sobre tal, como o usuário que o criou, e seu código de identificação registrado anteriormente na fase de publicação. Informações adicionais como dificuldade, *Likes* e comentários também podem ser visualizadas se o Maker em questão oferece essa funcionalidade. A Figura 4.7 mostra um exemplo da variedade de informações de um nível que pode ser observada pelo jogador.

4.2.4 Tabela de Classificação

Dentro do visualizador de níveis existe a **Tabela de Classificação**. A Tabela de Classificação é onde níveis que recebem um número maior de impressões positivas ganham visibilidade. A existência de níveis de popularidade mais elevada estabelecem um importante exemplo que outros usuários devem seguir. Como tais níveis tendem a ter um índice de qualidade maior, cria-se uma associação entre qualidade e popularidade, onde jogadores procuram melhorar a aparência e funcionalidade de seus níveis para alcançar renome na comunidade.

4.3 Efetividade

Tendo em consideração as práticas aplicadas por *Makers*, averiguaremos sua efetividade segundo as métricas definidas anteriormente.

[1] https://supermariomaker.nintendo.com/news/course-world-tips/

Figura 4.7: *Informações de um nível publicado por um usuário em Geometry Dash (Robtop Games, 2013). Entre as informações, temos o nome do nível, usuário que o criou, a dificuldade e duração do nível, e o número de likes e downloads.* ***Fonte:*** *https://geometry-dash.fandom.com/wiki/User_Levels*

4.3.1 Acessibilidade

Ao contrário de ferramentas de outras categorias, Makers são estritamente jogos. Sendo apenas jogos, não existe a pressão de conhecimentos prévios como em outras ferramentas de desenvolvimento, sendo a diversão o principal elemento em mente quando é utilizada.

A popularidade de jogos da Nintendo com públicos de todas as idades como o **Super Mario Maker (Nintendo, 2015)**, assim como a natureza gratuita de muitos Makers no mercado *mobile* como o extremamente popular **Geometry Dash (RobTop Games, 2013)**, sugere que Makers sejam uma das categorias mais acessíveis ao público geral.

As ferramentas dentro dos Makers se beneficiam por serem simples, deliberadamente sacrificando funcionalidade afim de tornar a interface e desenvolvimento menos intimidadora. A alteração de propriedades do protagonista, adição de mais configurações e objetos, entre outras ferramentas mais avançadas, mesmo tendo a capacidade de tornar a experiência mais complexa e oferecer mais poder de customização ao jogador, também aumentariam a barreira de conhecimento. (Oshino, 2015)

Em termos de engajamento, uma vez que um jogo Maker possui uma comunidade de tamanho favorável, a constante adição de novos níveis compartilhados por jogadores todos os dias pode oferecer uma quantidade quase ilimitada de conteúdo, mantendo a atenção de jogadores por um período de tempo superior ao de um jogo comum, cujo conteúdo é limitado ao ter sido desenvolvido por completo antes do lançamento. O jogo **New Super Mario Bros. U Deluxe (Nintendo, 2019)**, considerado o jogo de plataforma mais recente da franquia **Super Mario Bros.** no tempo de escrita, oferece um total de 164 níveis. Enquanto isso, o jogo **Super Mario Maker (Nintendo, 2015)**, um ano após seu lançamento, teve um número de níveis desenvolvidos por usuários que chegou a ultrapassar mais de 7 milhões. (NintendoOfAmerica, 2016).

Em conclusão, Makers, devido a seu favorável preço quanto simplicidade de uso, são altamente acessíveis. A interface interativa e simples torna tanto o processo de criação

quanto o aprendizado intuitivo para todas as idades, maximizando o potencial de introduzir um interesse em desenvolvimento de jogos ou formas de criação colaborativa em comunidade.

4.3.2 Barreira de conhecimento

As ferramentas de Makers são construídas de modo a ser o mais simples e intuitivas possíveis. A interface não toma uso de nenhum conceito programático, e o usuário é livre para usufruir das ferramentas livremente sem a necessidade de compreender como elas funcionam, ou como foram implementadas. A documentação em Makers é geralmente interna, tomando a forma de guias interativos e conteúdo pré-desenvolvido, exercendo uma função de exemplos a serem seguidos. Se o usuário desejar encontrar informações adicionais sobre o funcionamento ou combinações de determinados objetos, terá que buscar guias criados por outros usuários. Logo, em Makers, tipicamente **não há notável Barreira de Conhecimento**, sendo possível atingir o máximo potencial da ferramenta sem nenhum conhecimento de programação.

4.3.3 Aprendizado na Prática

A política de desenvolvimento de *blocos de construção* seguida por Makers garante que o usuário nunca consiga mudar o comportamento das ferramentas ou adicionar novas funcionalidades ou objetos que já não foram implementadas pelos desenvolvedores originais do jogo. Porém, a ausência desses elementos induz aos jogadores a deliberar formas criativas de utilizar o que já está implementado para construir seus próprios mecanismos para exercer essas funções.

Um exemplo claro disso é presente em **Super Mario Maker (Nintendo, 2015)**. O jogo, em seu lançamento, não oferecia um objeto que oferecia a habilidade do jogador a delimitar tempo. Logo, afim de implementar uma forma de atribuir eventos a um tempo limite, foram desenvolvidos *Timers* a partir da combinação de certos objetos, ou seja, cronômetros que contam uma certa quantidade determinada de tempo. Essas combinações de objetos separados para construir novas funcionalidades são comumente chamadas de ***contraptions***, traduzidos como engenhocas, ou dispositivos, exemplificado na Figura 4.8.

Devido ao sistema de classificação, usuários menos experientes são incentivados a inovar através desses dispositivos mais avançados para tornar seus níveis mais populares, tornando o aprendizado uma progressão natural da experiência do jogo.

Podemos ver que em Makers, não temos uma interação direta com código ou programação, mas ao longo do processo de criação, é possível experienciar e conhecer vários **conceitos computacionais**. Como exemplo, as ***contraptions***, mencionadas anteriormente, exibem uma forte associação com ***funções*** no ambiente de programação, sendo ambas estruturas formadas por pedaços separados que, quando combinados, exercem um certo propósito, podendo ser reutilizados múltiplas vezes. Essa noção é refletida por Fornós (2020), em um estudo cujo objetivo era ilustrar o potencial do jogo **Super Mario Maker 2 (Nintendo, 2019)** como uma ferramenta de co-design, onde estudantes de engenharia química utilizaram o editor de níveis para criar níveis que representam processos químicos.

Figura 4.8: *Um Shellmet Timer, um dos dispositivos mais simples do Super Mario Maker (Nintendo, 2015). O jogador, como input, acerta o bloco ? que então impulsiona um casco utilizando uma mola, o casco, destruindo blocos em seu caminho, se movimenta por um percurso por uma quantidade determinada de tempo, antes de acionar o bloco ? do outro lado, representando a saída e sinalizando o fim do Timer. Os blocos, molas e cascos, objetos separados com suas próprias interações, são combinados para criar objetos únicos.* ***Fonte:*** *https://www.youtube.com/watch?v=unyU0OecjBE*

Tal estudo demonstrou evidências que o uso de um editor de níveis pode ser uma ferramenta eficaz de aprendizado, utilizando mecânicas de jogo para simular um conteúdo educativo e torná-lo mais divertido.

Em conclusão, em Makers, temos uma forte ausência de interação alguma com linguagem ou interface programática. Porém, como ferramenta educacional para a introdução de conceitos computacionais, **Makers possuem um enorme potencial para o aprendizado de novos desenvolvedores.**

4.3.4 Interação com Comunidade

A interação com comunidade é o fator mais essencial em Makers. Comentários, downloads e likes, além de oferecerem uma recompensa fácil de ser percebida, estimulam o usuário a criar mais níveis, através do desejo de obter uma reputação na comunidade. Dessa forma, constrói-se um ciclo de *feedback* positivo, onde o jogador próprio se motiva a desenvolver mais pelo desejo de continuamente melhorar seus *designs* e ser recompensado por tal. O desejo de obter renome é considerado, por *Bitzer et al.*, como uma das três principais motivações para um desenvolvedor iniciar um projeto de software livre (Bitzer *et al.*, 2007), logo a introdução dessa motivação pelos Makers ajuda jogadores a identificar um possível interesse em participar de comunidades através de contribuições, como no ecossistema de software livre.

Assim como a introdução de opiniões externas é crucial para a evolução da qualidade de um software(Carreño e Winbladh, 2013), o usuário de um Maker adquire experiência

não apenas em como extrair retorno relevante, como também em aplicar um processo de iteração constante em sua política de desenvolvimento, entrando em um ciclo de desenvolver, avaliar resultados, e então aplicar esses resultados no próximo design. Novamente, tal processo de iteração é recorrente em desenvolvimento de software(Wynn e Eckert, 2017), ilustrando que a criação de níveis induz o uso de boas práticas de desenvolvimento de software de uma forma espontânea.

Existe um forte encorajamento para um criador de níveis não apenas compartilhar seu próprio conteúdo, mas também jogar os níveis de outras pessoas. Ao analisar o conteúdo de desenvolvedores conhecidos, o usuário pode tomar como exemplo certas práticas ou técnicas de desenvolvimento que encontrou, aplicando-as em seus próprios níveis de modo a elevar a qualidade deles, e então, seus próprios níveis serão inspiração para outros desenvolvedores. Em resumo, o ambiente em um Maker se assemelha muito a um *Hackerspace*, onde conhecimento é constantemente compartilhado por todos, sendo as criações de cada desenvolvedor um fruto da colaboração da comunidade como um todo.

Em conclusão, nesta métrica, observa-se que Makers podem introduzir **um ambiente de comunidade similar ao de desenvolvimento de software.** O catálogo de níveis pode ser considerado como uma simulação de um repositório, onde todos são livres para publicar suas criações, auxiliar o desenvolvimento de outros e usar aprendizados compartilhados para novos projetos.

4.3.5 Motivação para Desenvolvimento

Makers, em essência, são jogos que encorajam seu jogador a criar, mas quando se trata de desenvolvimento de software, não podemos derivar uma relação direta entre o lazer de construir um nível e o desejo de desenvolver um software. Em um *Maker*, é possível aplicar um aprendizado de conceitos computacionais de forma divertida e acessível, porém os conceitos em questão são limitados a aqueles que podem ser executados pelas ferramentas oferecidas pelo jogo. Um jogador pode intuitivamente aprender a como construir uma *contraption* que exerce uma função similar a um *if statement*, mas a aplicação desse conhecimento será restrita ao ambiente do *Maker*. Logo, *Makers* são efetivas potenciais ferramentas de aprendizado, como uma ótima porta de entrada para introduzir um público leigo à conceitos valiosos para um desenvolvedor, mas não oferecem a habilidade de aplicar esse conhecimento de mesma forma que um desenvolvedor faria.

Em conclusão, como o jogador não possui interação com código, e sim com ferramentas que deliberadamente reduzem sua complexidade e flexibilidade afim de tornar a experiência mais simples e acessível, **não podemos concluir que *Makers* introduzem uma motivação forte para o desenvolvimento.** Afinal, existe uma enorme e proposital discrepância de complexidade entre a construção de um nível e o desenvolvimento de um software, criando uma obstáculo que em grande parte, será grande demais para um jogador desejar ultrapassar.

Makers são jogos, e é isso que desejam ser. A simplicidade das ferramentas e desnecessidade de conhecimento de desenvolvimento para o uso delas são uma vantagem

para o propósito do produto, pois deve existir lazer no processo de criação. *Makers* induzem o desejo de criar, mas não podemos concluir que induzem também o desejo de desenvolver.

Capítulo 5

Mods: Jogos abertos à extensão e customização

Neste capítulo, falaremos sobre Jogos que são abertos à extensão e customização, na forma de *Mods*.

5.1 Caracterização

Mods, uma abreviação de *modifications*, são alterações feitas a um produto, seja ele software ou hardware, por seus usuários. No contexto de videogames, o que chamaremos de *modding*[1], é na verdade *game modding*, uma subseção do conceito geral de modificações, reservado para a modificação de um jogo por seus jogadores, com o intuito de alterar aspectos do jogo original. Aqueles que praticam o desenvolvimento de *mods* para um jogo não são seus desenvolvedores, mas sim os jogadores, chamados então de *modders*[2]. A forma com que um *mod* altera um jogo varia muito, podendo significar o conserto de bugs, a modificação de conteúdo para aprimorar a experiência de jogatina, ou a adição de novo conteúdo não presente no jogo quando tal foi lançado (Poor, 2014).

O conceito de *modding* em jogos e, consequentemente, de uma "comunidade de *modding*", um grupo de jogadores que compartilham suas extensões e alterações pelo uso de plataformas *online* comunicativas, nasceu com *Doom* ***(id Software, 1993)*** (Gregory, 2018). A forma como *Doom* ofereceu aos jogadores a habilidade de modificar e adicionar conteúdo ao jogo de forma facilmente compartilhável se tornou um exemplo de como jogos poderiam ser mais que produtos, e sim uma plataforma para os jogadores exibirem sua criatividade. Desde então, *mods* apenas continuaram a crescer em relevância, com mais e mais desenvolvedores fornecendo aos jogadores a habilidade de interagir com o conteúdo de seu jogo diretamente.

A existência de *mods* para um jogo oferece, como um todo, diversos benefícios, sendo o maior deles um benefício similar ao dos Makers: o de longevidade. Mesmo após seu

[1] Traduzindo diretamente do inglês, significa modificar, ou fazer modificações.

[2] Gíria que pode ser traduzida como "modificadores".

lançamento, um jogo que possui suporte para *mods* e uma comunidade assídua poderá continuar a receber conteúdo novo desenvolvido por seus jogadores, resultando em um constante influxo de interesse. Como um exemplo, o jogo **Skyrim (Bethesda, 2011)**, foi lançado em 2011, e teve sua última expansão oficial em fevereiro de 2013 (Karmali, 2013). Porém, mesmo uma década após seu lançamento, o jogo continua relevante, sendo relançado mais de **8 vezes diferentes** em **10 plataformas** (Szpytek, 2021), e também um dos 10 jogos mais vendidos de todos os tempos (Yaden, 2023). Uma das principais razões para esse contínuo interesse no jogo anos após seu lançamento é sua comunidade de *modding*, com mais de **70 mil** *mods* para o jogo espalhados em suas múltiplas plataformas, muitos deles adicionando dezenas de horas de conteúdo e formas completamente novas de jogar (Alexander, 2022).

Como resultado, muitas empresas produtoras de jogos lançam seus jogos com suporte para *mods* em mente, afim de prolongar sua longevidade e estabelecer sua relevância em longo prazo (Au, 2002). Como desenvolvedores de jogos muitas vezes teriam que sacrificar tempo e carga de trabalho para oferecer suporte e continuidade para um jogo antigo enquanto produzem um novo jogo, a existência de *modders* pode aliviar essa pressão ao oferecer novo conteúdo para tais jogos antigos (Lee *et al.*, 2020). Além disso, ao retirar o controle sobre o design do jogo de seus desenvolvedores originais e oferecê-lo a mentes externas ao processo de desenvolvimento, é possível atingir um nível maior de inovação (Scacchi, 2011) e criatividade, que em alguns casos, pode até servir como inspiração para as empresas responsáveis pelos jogos a serem modificados.

As práticas aplicadas na criação de *mods* de jogos, assim como a importância de fatores como o voluntarismo e posse compartilhada, estabelecem um paralelo com o ramo de desenvolvimento de *Open-Source Software (OSS)* (Au, 2002), afinal, um videogame é um software. A modificação, extensão e customização de software e hardware é algo muito comum entre desenvolvedores, sendo *case modding* e *forking* subculturas de *modding* prevalentes. Assim como desenvolvedores são usuários das ferramentas que desenvolvem, *modders* são jogadores dos jogos que reconfiguram (Scacchi, 2011). Um *modder* que aprende como um jogo é implementado para então adicionar algo a ele, exerce um papel quase equivalente ao de um desenvolvedor que desenvolve uma extensão para um software aberto que usa. Dessa forma, vemos que através de *modding*, um indivíduo aprende valiosas habilidades em programação, arte e design, seja dentro (Scacchi, 2003) ou fora de um ambiente acadêmico (El-Nasr e Smith, 2006), atributos de grande valor para desenvolvedores de jogos comerciais (Wallace, 2014) e software como um todo.

Atualmente, *modding* se mostra como a principal forma com que jogadores diretamente experienciam jogos como um projeto de software autoral e interagem com o processo de desenvolvimento, chegando ao ponto onde alguns *modders* tornam seus *mods* em jogos, ou são recrutados por empresas produtoras de jogos (algo que veremos com mais detalhes na Seção 5.4). Em ambos os casos, vemos exemplos claros e concretos de uma possível transição de jogador para desenvolvedor, logo é vital realizarmos uma análise de quais elementos desses jogos alimentam esse potencial de desenvolvimento em seus usuários. Para isso, precisamos primeiramente entender as diferentes maneiras com que jogos implementam *mods*.

5.1.1 Tipos de Mods - Mods de Customização

Mods de customização são modificações cujo principal objetivo é a **alteração** do conteúdo original do jogo. *Mods* dessa categoria tipicamente alteram fatores já existentes no jogo, como elementos visuais ou configurações. Entre os *mods* de customização, alguns exemplos são:

Troca de Texturas

Mods de troca de texturas, como indicado pelo nome, são *mods* que alteram os gráficos do jogo de alguma forma. A troca de gráficos pode ser feita em pequena escala, como a mudança da roupa de algum personagem, mas pode também ser estendida em escopo para cobrir o jogo inteiro, para mudar completamente sua estética, ou aumentar a qualidade visual de sua interface.

O jogo **Minecraft (Mojang, 2011)**, um jogo de mundo aberto com foco em exploração e construção, possui muitos *mods* que alteram a textura de todos os objetos que compõem o jogo, incluindo seus itens, inimigos, e os blocos que compõem todos os seus diversos biomas. Um exemplo desse tipo de *mod* é ilustrado na Figura 5.1.

Figura 5.1: *Uma ilustração de um texture pack do jogo* ***Minecraft (Mojang, 2011)****. Texture packs são mods que trocam a textura do jogo inteiro. Na esquerda, podemos ver os gráficos e texturas padrão, do jogo; enquanto que na direita, podemos ver as texturas modificadas pelo mod. Fonte: https://texture-packs.com/resourcepack/john-smith-legacy/*

Patch Inoficial

No contexto de um videogame, um *patch* é um tipo de atualização oferecida pelos desenvolvedores cujo principal propósito não é expandir o conteúdo do jogo, mas sim reestruturar o conteúdo já existente para melhorar a experiência de jogo. *Patches* habitualmente tomam a forma de conserto de *bugs*, rebalanceamento de dificuldade, ou cortes de conteúdo.

Jogos que não recebem mais suporte de seus desenvolvedores, geralmente por serem antigos, muitas vezes enfrentam uma ausência de *patches* de seus desenvolvedores. *Patches inoficiais* são meios para jogadores continuarem a melhorar um jogo após seu lançamento

mesmo depois do jogo não receber mais atualizações de seus desenvolvedores, da mesma forma que um desenvolvedor de software livre pode criar um *fork* de uma ferramenta descontinuada para consertar possíveis *bugs* e melhorar sua funcionalidade.

Porém, não são apenas jogos que não recebem mais *patches* oficiais que possuem mods desse tipo. Muitas vezes, *patches inoficiais* são feitos para melhorar a experiência dos jogadores de formas que não serão feitas pelos desenvolvedores. Por exemplo, o jogo **Elden Ring (FromSoftware, 2022)**, foi um dos maiores jogos a ser lançado em seu ano, renomado por seu expansivo mundo aberto, combate intenso e elevada quantidade de conteúdo. Porém, o jogo também recebe sua popularidade por ser um jogo de altíssima dificuldade que pune qualquer erro do jogador de forma severa. Devido ao design impiedoso do jogo ser um elemento fundamental de seu design, os desenvolvedores nunca lançariam, em capacidade oficial, uma forma de diretamente diminuir o nível de dificuldade. Esse fator pode alienar uma grande porção de jogadores que, por alguma razão ou outra, não são capazes de alcançar o nível de precisão e destreza para jogar o jogo, mas ainda gostariam de ter a experiência de jogá-lo. Porém, **Elden Ring (FromSoftware, 2022)**, assim como quase todos os outros jogos desenvolvidos pela **FromSoftware**, possuem *mods* que tornam o jogo mais fácil, aumentando sua acessibilidade. Assim, mesmo sendo um jogo que ainda recebe *patches* oficiais atualmente, *modders* podem incluir suas próprias mudanças ao jogo para melhorar a experiência de jogadores.

5.1.2 Tipos de Mods - Mods de Extensão

Mods de extensão são modificações cujo principal objetivo é a **extensão** do conteúdo original do jogo. *Mods* de extensão geralmente adicionam elementos em um jogo que não estão presentes no jogo original. Tipicamente, um *mod* de extensão requer interação direta com o código e estrutura do jogo, sendo um conhecimento programático necessário para realizar seu desenvolvimento mesmo quando o escopo da modificação é baixo. Outro fato importante é que *mods* de extensão também podem incluir alterações em elementos do jogo, assim como *mods de customização*, mesmo que a alteração não seja seu principal propósito. Alguns exemplo de *mods* de extensão são:

Add-on

Add-ons são adições de elementos ou conteúdo a um jogo. Tipicamente um *add-on* toma a forma de algum tipo de conteúdo já presente em um jogo, como uma arma em um FPS ou um veículo em um jogo de corrida, porém totalmente implementado pelo jogador, sem a interação dos desenvolvedores originais.

No jogo **Minecraft (Mojang, 2011)**, é possível encontrar diversos **minérios** em cavernas. Ao coletá-los, é possível transformar esses minérios em barras, e tais barras em ferramentas e armadura. Por exemplo, ao encontrar um minério de ferro, o jogador pode minerá-lo com sua picareta de pedra, fundir ele com sua fornalha, e usar as barras que recebeu para construir uma picareta de ferro, que é usada para quebrar minérios de nível maior.

Na atualização 1.17 do jogo em 2021, foi adicionado um novo tipo de minério, o **minério de cobre**, porém inicialmente, houve um desapontamento entre os fãs pois o minério

não tinha muita utilidade, e não podia ser usado para construir ferramentas ou armadura (Wood, 2023). Por causa disso, foram criados inúmeros *mods* que adicionaram ferramentas e armadura de cobre ao jogo. A lógica por trás de implementação de itens como ferramentas e armadura já estava presente no jogo, porém o que os jogadores fizeram foi utilizar essa lógica pré-existente para adicionar um item novo ao jogo que melhorava sua experiência de jogatina. Um exemplo de um desses mods pode ser visto na Figura 5.2.

Figura 5.2: *Uma ilustração de um mod add-on do jogo* ***Minecraft (Mojang, 2011)****. A função do mod é adicionar equipamento de cobre ao jogo, algo que não estava presente no jogo quando o minério de cobre foi adicionado pelos desenvolvedores. Fonte: https://www.curseforge.com/minecraft/mc-mods/copper-equipment*

Total Conversion

Total conversions, traduzido como conversões totais ou completas, são *mods* que completamente reconstroem um jogo, tanto em termos gráficos quanto da jogatina. Conversões totais são desenvolvidos com o intuito de alterar completamente o jogo, podendo tomar a forma de uma nova campanha e história, adição de modos completamente novos de jogar, ou até uma completa mudança do gênero do jogo. Uma conversão total pode ser considerada o tipo de *mod* mais ambicioso, e frequentemente requer que o usuário possua um nível de conhecimento similar ao de um real desenvolvedor de software.

O jogo **Half-Life (Valve, 1998)** foi um FPS que recebeu um grande número dessas conversões totais, devido ao suporte e encorajamento de **Valve**, sua produtora. O *mod* ***Black Mesa*** foi uma conversão total que completamente recriou o jogo, recriando os gráficos, adicionando conteúdo e até modificando elementos da jogatina e história. Um

exemplo de como o *mod* veio a atualizar e recriar o jogo original é ilustrado na Figura 5.3.

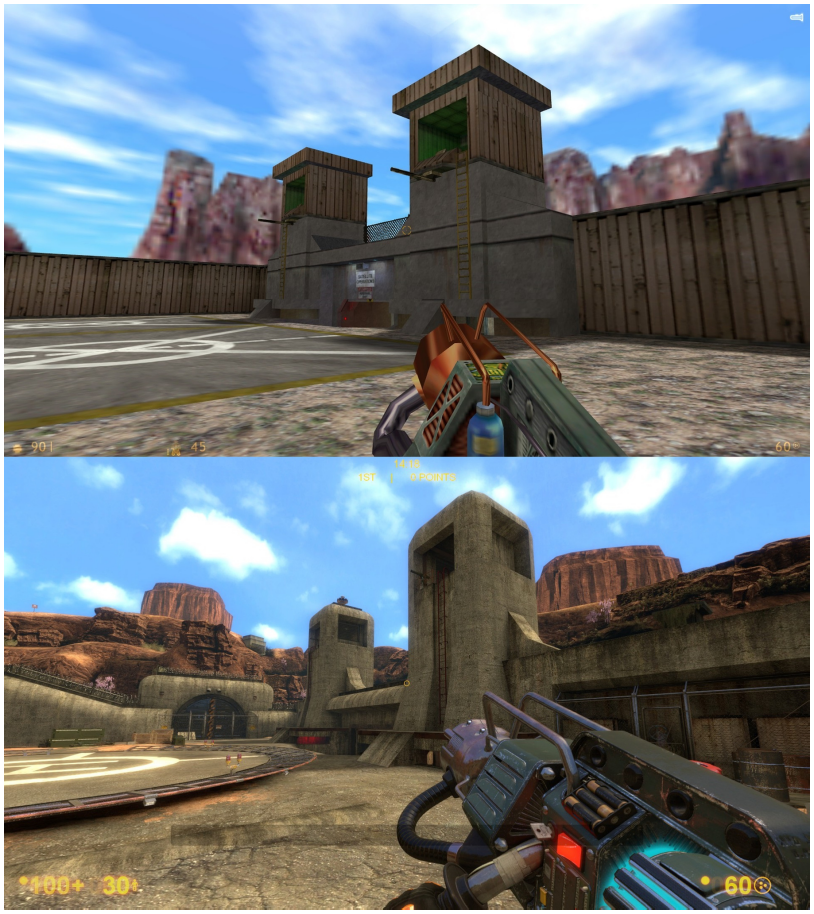

Figura 5.3: *Uma comparação dos gráficos do jogo* ***Half-Life (Valve, 1998)*** *e sua full conversion Black Mesa, sendo o original presente na parte de cima, e o mod na parte de baixo. Podemos ver que mesmo apenas modificando o jogo pré-existente, temos uma grande diferença tanto na estética como na interface. Fonte: https://steamcommunity.com/sharedfiles/filedetails/?id=835948326*

Algo relevante a se notar é que **muitas conversões totais eventualmente se tornam jogos próprios**, como é o caso do jogo **Black Mesa (Crowbar Collective, 2020)** e outros jogos que serão ilustrados na Seção 5.4. *Mods* que eventualmente se tornaram jogos são elementos cruciais para essa análise pois são um exemplo concreto da transição de um jogador para um desenvolvedor de software.

5.1.3 Características

É importante definir o que difere *Mods* de outras categorias de jogos em que o jogador exerce função de desenvolvedor.

- **É preciso ter um jogo para ter um *mod*:** *Mods* não são sistemas *stand-alone*, pois requerem que o jogador possuem uma cópia original ou licenciada do jogo sem *mods* (Scacchi, 2011). *Modders* precisam do jogo para modificá-lo, e mesmo jogadores que apenas desejam utilizar *mods* de outros usuários precisam do jogo para aplicá-los.

- **O desenvolvimento de *mods* é feito fora do jogo:** Ao contrário de Makers, o desenvolvimento de *mods* é feito de forma externa. Como é necessário interagir com os arquivos que compõem o jogo, seja para recolocação, alteração ou adição de conteúdo, não é possível desenvolver um *mod* enquanto dentro de um jogo.
- ***Mods* podem ser compartilhados:** Um *mod* é, na maior parte dos casos, uma combinação de arquivos que, ao ser integrados ao jogo que modifica, aplicam as mudanças em si definidas. Dessa forma, um *mod* pode ser compartilhado da mesma forma que um arquivo qualquer. Porém, é impossível acessar um *mod* desenvolvido por outra pessoa sem alguma via de compartilhamento.

5.2 Práticas

Existem certas práticas aplicadas por jogos abertos a *mods* que ampliam seu sucesso em oferecer a função de desenvolvedor a seus usuários. Discutiremos elas nesta Seção.

5.2.1 Data-Driven Design

Para induzir o desejo de um jogador construir um mod, e então possivelmente o motivar a desenvolver, é necessário oferecer ao jogador um produto facilmente customizável e **extensível**. Uma das formas mais efetivas de realizar isso é utilizar **data-driven design**, ou seja, design orientado a dados.

Jogos são construídos por dois componentes, lógica e dados. A lógica define as regras e algoritmos do funcionamento de um jogo, enquanto que os dados definem o conteúdo e seu comportamento. Ao criar um sistema que separa esses dois componentes claramente, cada um deles pode evoluir sozinho. O objetivo do **data-driven design** no contexto de *modding* é abstrair ao máximo a lógica do jogo para que seja possível modificar e adicionar novos dados livremente (Rabin, 2000). Nessa política de design flexível, o jogo, através de sua lógica, constrói a si mesmo utilizando seus dados. Um exemplo dessa política de design previamente visto nessa análise foi *Doom*, onde todos os fatores do jogo eram salvos em arquivos WAD, e tudo o que a *engine* do jogo fazia era ler os dados presentes nesse arquivo e aplicar a lógica do jogo a eles.

5.2.2 Design Orientado a Objetos

Outra política de *design* relevante é a de *design* orientado a objetos, onde dados do jogo são abstraídos para se tornarem instâncias genéricas que são também facilmente modificáveis e extensíveis. Para ilustrar como *design* orientado a objetos funciona em um jogo atual que possui grande suporte para mods, usemos de exemplo o popular jogo **Slay the Spire (MegaCrit, 2019)**. No jogo **Slay the Spire (MegaCrit, 2019)**, o jogador teve tomar uso de cartas e relíquias que encontra ao longo de sua jornada para derrotar inimigos, encontrar tesouro, e então vencer o jogo. As cartas são o principal componente do combate de **Slay the Spire (MegaCrit, 2019)**, ilustrado na Figura 5.4. O jogador, durante seu turno, pode pagar um **custo** para utilizar uma carta de seu baralho em um inimigo, causando algum dano e/ou efeito. Cada uma das mais de 250 cartas do jogo tem seu próprio custo, dano e efeito.

Figura 5.4: *Uma ilustração de um combate do jogo* ***Slay the Spire (MegaCrit, 2019)****. O personagem controlado pelo jogador utiliza cartas que encontra ao longo do jogo para derrotar inimigos. Fonte: https://store.steampowered.com/app/646570/Slay_the_Spire/*

A forma como a lógica do jogo funciona é um sistema que lê as cartas como instâncias de uma classe ***Card***, ou **Carta**. As cartas, nesse contexto, são os dados, e todos os fatores que definem uma carta, como seu custo, efeito, e a quantidade de dano que inflige, são determinados na instância da classe Carta. Quando o jogo inicia, ele então carrega todas as instâncias da classe Carta e insere todas elas no jogo utilizando sua lógica interna.

Como cada carta é um dado, tudo que seria necessário para modificar o custo ou efeito de uma carta do jogo seria alterar as variáveis definidas no código da instância da carta em questão. O jogo então irá ler o dado de tal carta e a implementará no jogo com as alterações em efeito. Porém, esse sistema também facilita **a adição de novas cartas**.

```
1    public class Flare
2    extends CustomCard {
3       public static final String ID = "myModID:Flare";
4       private static CardStrings cardStrings = CardCrawlGame.languagePack.
            getCardStrings(ID);
5       // Get object containing the strings that are displayed in the game.
6       public static final String NAME = cardStrings.NAME;
7       public static final String DESCRIPTION = cardStrings.DESCRIPTION;
8       public static final String IMG_PATH = "img/my_card_img.png";
9       private static final int COST = 0;
10      private static final int ATTACK_DMG = 3;
11      private static final int UPGRADE_PLUS_DMG = 3;
12      private static final int VULNERABLE_AMT = 1;
13      private static final int UPGRADE_PLUS_VULNERABLE = 1;
14
15      public Flare() {
16         super(ID, NAME, IMG_PATH, COST, DESCRIPTION,
17            AbstractCard.CardType.ATTACK, AbstractCard.CardColor.RED,
18            AbstractCard.CardRarity.UNCOMMON, AbstractCard.CardTarget.ENEMY);
19         this.magicNumber = this.baseMagicNumber = VULNERABLE_AMT;
20         this.damage=this.baseDamage = ATTACK_DMG;
21
22         this.setBackgroundTexture("img/custom_background_small.png", "img/
               custom_background_large.png");
```

```
23
24          this.setOrbTexture("img/custom_orb_small.png", "img/custom_orb_large.
                png");
25
26          this.setBannerTexture("img/custom_banner_large.png", "img/
                custom_banner_large.png");
27      }
28
29      @Override
30      public void use(AbstractPlayer p, AbstractMonster m) {
31        AbstractDungeon.actionManager.addToBottom(new com.megacrit.cardcrawl.
              actions.common.DamageAction(m,
32          new DamageInfo(p, this.damage, this.damageTypeForTurn),
33          AbstractGameAction.AttackEffect.SLASH_DIAGONAL));
34        AbstractDungeon.actionManager.addToBottom(new ApplyPowerAction(m, p, new
              VulnerablePower(m, this.magicNumber, false), this.magicNumber, true,
              AbstractGameAction.AttackEffect.NONE));
35      }
36
37      @Override
38      public AbstractCard makeCopy() {
39          return new Flare();
40      }
41  }
```

Programa 5.1: *Trecho do código que representa a classe de uma carta customizada no jogo* ***Slay the Spire(MegaCrit, 2019)****. Fonte: https://github.com/daviscook477/BaseMod/wiki/Custom-Cards*

Podemos observar no trecho do Programa 5.1, a classe de uma carta customizada que pode ser adicionada ao jogo. É possível ver que todos os fatores da carta, como custo, efeito, dano, e até elementos visuais atrelados a ela, são definidos nessa instância. Logo, para adicionar uma nova carta ao jogo, é preciso criar uma nova instância da classe Carta, e modificar suas variáveis de acordo com o desejado. Como nenhuma carta do jogo é *hard-coded*, o jogo irá carregar a carta nova da mesma forma que carregaria uma carta nativa ao jogo.

5.2.3 Organização de Arquivos

Levando em consideração a política favorável de *data-driven design*, uma das principais práticas tomadas por jogos abertos a *mods* é a forma em que organizam seus dados, como texturas, modelos 3d, UI, e código. Um jogo aberto a *mods* tipicamente organiza seus arquivos de forma que mesmo um jogador sem experiência com programação consiga identificar a localização dos dados que deseja modificar e adicionar. Organizar os arquivos de forma que modificações podem ser inseridas e então reconhecidas pelo jogo com facilidade traz um grande incentivo para jogadores "brincarem" com os arquivos e criarem seus primeiros *mods*.

Algo importante a se considerar é também a forma como o jogo lê seus dados quando é executado. Um jogo naturalmente irá ler os dados presentes em seu diretório de instalação, assim como grande parte dos programas. Assim, se um jogo teve seus dados modificados por um jogador, essa mudança será aplicada ao jogo quando for executado.

Porém, jogos com suporte para *mods* muitas vezes definem também um diretório de dados externo, cuja função é abrigar os dados modificados pelo jogador. Esse diretório para *mods*, chamado comumente de um *mods folder*, é considerado um clone do diretório de dados, e o jogo, quando executado, verifica ambos os diretórios. Se o diretório de *mods* conter dados que compartilham o mesmo nome do original, **os dados modificados serão lidos em vez dos dados originais**, enquanto que se o diretório de *mods* conter dados novos, **tais serão lidos como uma extensão dos dados originais**. Uma ilustração desse conceito pode ser vista no diagrama da Figura 5.5.

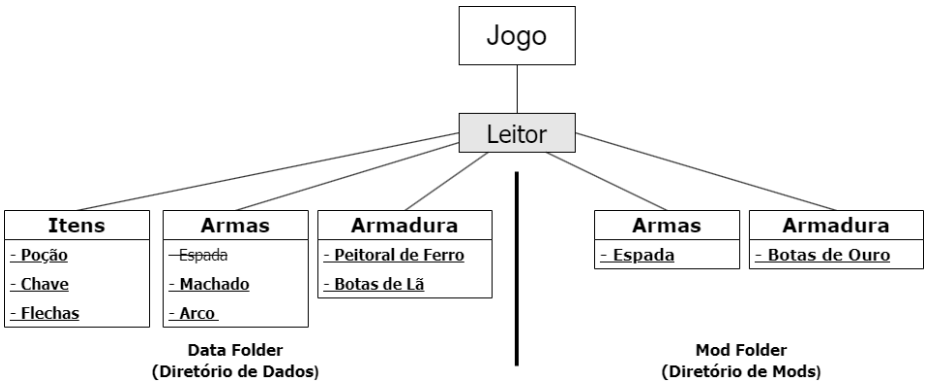

Figura 5.5: *Um simples diagrama que demonstra a forma como um jogo que implementa um diretório para mods pode carregar seus dados. Ao iniciar o jogo, são verificados ambos os diretórios de dados e mods, para que seus conteúdos sejam lidos e aplicados no jogo. No diagrama, os dados sublinhados são aqueles que serão lidos pelo jogo. Como é possível ver, as botas de ouro, um novo dado em Armaduras, será lido como uma extensão do original. Enquanto isso, como ambos os diretórios Armas possuem um dado para a Espada, o jogo lerá o dado modificado, desconsiderando seu dado original para a Espada.*

Ao implementar um diretório de mods, *mods* se tornam **auto-contidos**, retirando a necessidade de alterar diretamente os dados de jogo e facilitando a aplicação de mods.

5.2.4 Ferramentas Auxiliares

Ferramentas auxiliares à criação de *mods* são ferramentas cujo propósito é facilitar o processo de desenvolvimento de modificações de alguma forma. A função dessas ferramentas variam de forma e uso dependendo do jogo, podendo exercer funções como a abstração do código fonte do jogo para viabilizar uma interação intuitiva, a extração e compactação de dados compactados normalmente inacessíveis, entre outros. No início da popularização de jogos de computador e modificações em jogos nos anos 90, ambos originados em grande parte pelo jogo *Doom*, videogames de PC começaram a fornecer consigo ferramentas para modificá-los (Burger-Helmchen e Cohendet, 2011), resultando em um número extenso de *mods* para jogos da época como **Half-Life (Valve, 1998)**, **The Sims (Maxis, 2000)**, **Quake (id Software, 1996)**, entre outros. As ferramentas oferecidas podiam tomar várias formas, desde um construtor de mapas em jogos como **Warcraft III (Blizzard, 2002)** (ilustrado na Figura 5.6), até um *Software Development Kit* (SDK), um conjunto de ferramentas explicitamente usado pelos desenvolvedores para desenvolver o jogo original. O oferecimento de SDKs era raro, porém empresas desenvolvedoras como a **Valve** comumente ofereciam ferramentas desse calibre de forma livre e aberta.

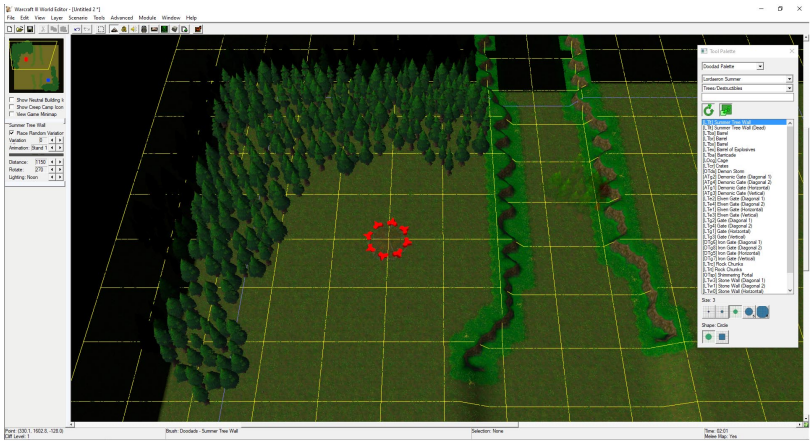

Figura 5.6: *Uma ilustração do criador de mapas do jogo* ***Warcraft III (Blizzard, 2002)****. Utilizando a ferramenta, o jogador é capaz de montar um mapa com o terreno da forma que desejar, inserir objetos, e até introduzir novos scripts, áudio e recursos visuais, não incluídos no jogo original. Após a criação de um mapa, o jogador era capaz de jogar tal mapa com seus amigos pela internet. Fonte: https://news.blizzard.com/pt-br/warcraft3/23395649/revisitando-o-editor-de-warcraft-iii*

Nos dias atuais, na maior parte dos casos, ferramentas auxiliares não são mais oferecidas com o jogo após sua compra, mas sim integradas à **Plataformas de Hospedagem** em que o jogo se apresenta, como a **Oficina Steam**. O fornecimento de ferramentas auxiliares pode ser visto como a melhor forma para um desenvolvedor demonstrar a seu público que o jogo é aberto a modificações, e sua existência é vista como uma vital motivação para a criação de conteúdo por parte dos jogadores (Poretski e Arazy, 2017). Afinal, um jogador sentirá muito mais dificuldades em desenvolver um *mod* para um jogo se tudo o que tiver são seus arquivos. Porém, para jogos abertos a *mods* que não possuem ferramentas auxiliares, podem ser introduzidas ferramentas abertas desenvolvidas por outros jogadores (Scacchi, 2011). O uso e criação dessas ferramentas não apenas facilita a criação de *mods*, como também eleva o nível de interação entre membros da comunidade, além de diretamente indicar a presença de software livre no ecossistema de *game modding*.

5.2.5 Plataformas de Hospedagem

A interação de um *modder* com a comunidade do jogo que modifica é um grande fator motivacional. Um *modder*, na grande maioria das vezes, não constrói um *mod* apenas para si mesmo, mas para compartilhar com outros jogadores do mesmo jogo (Poor, 2014). Logo, facilitar o processo de compartilhamento e visualização de *mods* em meio a essa comunidade consequentemente estimula a criação de mods, através do desejo de um *modder* de observar seu *mod* ser jogado pela comunidade.

A forma pela qual é facilitada a demonstração de *mods* desenvolvidos pela comunidade é o uso de **Plataformas de Hospedagem**. No início da popularização de mods, as vias pela qual comunidades compartilhavam seus *mods* e interagiam com *mods* de outras pessoas eram **fórums especializados, centrados no jogo a ser modificado**. Porém, atualmente, plataformas de hospedagem se tornaram mais generalizadas e intuitivas, ilustrando centenas de milhares de *mods* de múltiplos jogos diferentes, em sites como

Nexus Mods[1](Ilustrado na Figura 5.7) e ***Curseforge***[2].

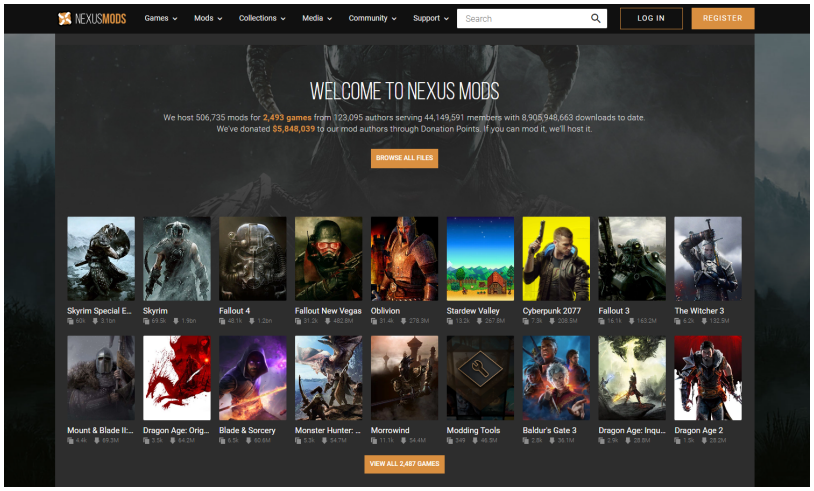

Figura 5.7: *A página principal do site NexusMods, uma das maiores plataformas de hospedagem de mods para jogos. Fonte: https://www.nexusmods.com/*

Atualmente, uma dos principais plataformas de hospedagem para *mods* é a **Oficina Steam**, um serviço atrelado à **Steam**, a maior plataforma de distribuição digital para jogos de computador, desenvolvida pela Valve e lançada em 2003. A Oficina Steam é um sistema de armazenamento em backend e páginas web públicas que facilitam o armazenamento, organização, ordenação, avaliação e download de conteúdo para um jogo ou aplicativo. Usuários do jogo usam ferramentas fornecidas pelo desenvolvedor para criar modificações, enviando o conteúdo à Oficina por meio de um formulário integrado à ferramenta. Outros clientes então podem descobrir, ordenar, avaliar e se inscrever em itens que desejam adicionar ao jogo pelo site da Oficina Steam, na Comunidade Steam[3]. Desenvolvedores de jogos são incentivados então a integrar seus jogos à Oficina Steam, oferecendo à comunidade de seus jogos ambas as ferramentas para construir o conteúdo que desejar, e o meio de compartilhar esse conteúdo com todos.

Jogos com um número excepcional de *mods* podem possuir plataformas de hospedagem exclusivas para si. Minecraft, como exemplo, possui múltiplos sites para conter seus mais de 100 mil *mods*, como o site **Modrinth** (ilustrado na Figura 5.8).

Estabelecendo um vínculo com Makers, pode se dizer que uma plataforma de hospedagem exerce uma função equivalente ao de um **Visualizador de Níveis** presente em Makers. Uma plataforma de hospedagem tem como objetivo a exibição de conteúdo construído e compartilhado por fãs e jogadores, para promover o descobrimento de novas criações e interação entre a comunidade ao redor delas.

[1] https://www.nexusmods.com/

[2] https://www.curseforge.com/

[3] https://partner.steamgames.com/doc/features/workshop/implementation

Figura 5.8: *A página principal do catálogo de mods do site Modrinth, uma plataforma que hospeda exclusivamente mods do jogo* ***Minecraft (Mojang, 2011)****. Fonte: https://modrinth.com/mods*

5.3 Efetividade

Levando em consideração as práticas aplicadas, avaliaremos agora sua efetividade.

5.3.1 Acessibilidade

Mods são construídos a partir de jogos, logo a acessibilidade depende de fatores como o preço, a idade mínima e a disponibilidade do jogo que o jogador deseja modificar. A existência de um número alto de jogos que oferecem suporte para *mods* nos dias atuais (Au, 2002), assim como a extrema popularidade de jogos que possuem um número elevado de *mods*, torna *mods* uma das formas mais acessíveis possíveis de um jogador ter a experiência de desenvolver.

O número relativamente baixo de Makers no mercado de jogos restringe a possibilidade de seu uso como porta de entrada, enquanto que um jogador qualquer tem uma alta probabilidade de jogar múltiplos jogos com suporte para *mods* ao longo de sua vida, mesmo sem ser com o intuito de aprender a desenvolver. Como exemplo, o jogo **Minecraft (Mojang, 2011)** é atualmente o jogo mais vendido do mundo com **300 milhões de cópias vendidas** (Parrish, 2023), sendo popular com públicos de todas as idades desde crianças até idosos. Mas além disso, também é um dos jogos mais modificados do mundo.

A acessibilidade das ferramentas para a criação de *mods* varia com o jogo. Jogos que possuem uma comunidade ativa, ou ferramentas oficiais fornecidas pelos desenvolvedores, são inerentemente mais acessíveis. Porém, com a popularização da internet como uma via de aprendizado, jogadores que desejam construir um *mod* terão, na grande maioria das vezes, formas alternativas de aprender o necessário para desenvolver seu próprio *mod* mesmo sem a ajuda de orientações fornecidas pelo jogo em questão.

Em termos de engajamento, *mods* têm a grande vantagem de serem diretamente atrelados ao elemento natural de lazer em jogos. Um jogador que gosta muito de um jogo não só terá um interesse maior em explorar o conteúdo adicional proporcionado por *mods*, como também terá uma motivação intrínseca para desenvolver seu próprio *mod* para ele – sua afinidade pelo jogo (Poor, 2014). Além disso, como anteriormente dito, jogos com suporte para *mods* tendem a possuir um nível maior de longevidade, aumentando assim o alcance do jogo por se manter relevante para possíveis novos membros de sua comunidade.

Porém, quando se trata de acessibilidade, *mods* também sofrem uma desvantagem ao serem uma **experiência inteiramente opcional**. Um jogador de um Maker sempre usufruirá da experiência de tanto jogar o jogo quanto interagir com o conteúdo criado por outras pessoas. Enquanto isso, é possível que um jogador experiencie um jogo que possui grande suporte para *mods* sem **nunca interagir com esses aspectos do jogo**. Afinal, o conteúdo principal do jogo já está presente, tornando *mods* apenas um fator que pode ser totalmente ignorado por alguém que só deseja jogar o jogo tal como foi lançado, usufruindo apenas do conteúdo original do videogame sem modificação alguma.

5.3.2 Barreira de conhecimento

Em Mods, a barreira de conhecimento é dependente do jogo a ser modificado. Fatores como a existência de ferramentas auxiliares, organização de arquivos, disponibilidade de documentação e nível de atividade da comunidade afetam a facilidade que um jogador tem em criar um *mod*, e logo favorece o interesse desse mesmo jogador em desenvolvimento.

A barreira de conhecimento, além de ser dependente do jogo, também é dependente do tipo de *mod* que o jogador deseja desenvolver. Como exemplo, se um jogador deseja apenas trocar a textura de um objeto, construindo então um *mod* de customização, sua interação com o código ou estrutura de software do jogo será relativamente baixa, **reduzindo a barreira de conhecimento**. Em contraponto, se um jogador deseja construir um *mod* de extensão, é possível que o jogador tenha que não só interagir diretamente com os dados do jogo, como também **entender a forma como o jogo foi desenvolvido**. A capacidade de compreender o código de outra pessoa, assim como entender as decisões, pensamentos e base lógica por trás de seu design, é uma habilidade extremamente valiosa para todo tipo de desenvolvedor. Porém, considerando que exercer essa capacidade é um desafio até para **desenvolvedores profissionais** (Burgos *et al.*, 2007), exigir esse nível de conhecimento de um jogador inexperiente em desenvolvimento eleva bruscamente a barreira de conhecimento.

Em conclusão, quando se trata de *mods*, a barreira de conhecimento é proporcional à quantidade de interação necessária entre o usuário e a estrutura do jogo como software. De forma mais simples, **a barreira de conhecimento é proporcional à ambição do *mod*.** Mesmo que, devido à natureza inerentemente programática do desenvolvimento de *mods*, temos um nível de conhecimento programático mínimo necessário, a variabilidade da barreira de conhecimento também significa que um usuário inexperiente pode "escolher" sua barreira.

Um usuário inexperiente pode escolher desenvolver um *mod* mais simples, enquanto que um jogador que já possui um conhecimento e interesse para com desenvolvimento

pode escolher desenvolver um *mod* mais ambicioso. Dessa forma, o impacto da barreira de conhecimento é relacionado à zona de conforto do desenvolvedor.

5.3.3 Aprendizado na Prática

Em comparação com Makers, onde a política de desenvolvimento usada não permitia interação alguma entre o jogador e o código do jogo, *mods* não só permitem que o jogador visualize o código do jogo, como também fornecem a habilidade de o alterar e estender. Jogadores que aprendem a desenvolver *mods*, de certa forma, precisam aprender a estrutura do jogo que querem modificar, e como o jogo foi desenvolvido (Monterrat *et al.*, 2012).

Dessa forma, *mods* não apenas induzem o aprendizado de desenvolvimento na forma de interação direta com programação, como também estimulam o usuário a pensar em como se pode modificar ou estender um software existente, algo valioso na formação de um desenvolvedor de software livre.

Assim como no fator Barreira de Conhecimento, devido à grande variedade de tipos de *mods*, um jogador pode escolher o nível de interação com programação e design que deseja. Mesmo que isso torne impossível medir de forma generalizada a eficácia do aprendizado recebido, pois isso dependerá de fatores exclusivos para cada jogo e tipo de *mods*, essa liberdade também pode ser vista como um grande benefício. Um jogador que apenas deseja implementar um novo item a seu jogo favorito pode vir a gostar do processo e lentamente elevar o nível de ambição em seus futuros *mods*, criando uma natural progressão onde o jogador interage com desenvolvimento conforme seu conforto e preferência, até eventualmente adquirir um real aprendizado de programação e estrutura de software.

5.3.4 Interação com Comunidade

Modding, além de ser um método eficaz de aprender como se tornar um desenvolvedor, é também um aprendizado de como trabalhar com outras pessoas, especialmente em grandes projetos (Scacchi, 2011). A interação com comunidade é altamente prevalente na criação de *mods*, e podemos nos referir a uma pesquisa feita em 2014 por Nathaniel Poor para ilustrar isso. Na pesquisa, foram entrevistados 111 *modders*, e entre eles, foi confirmado que cerca de 86% dos *game modders* tomam parte da comunidade referente ao jogo que modificam (Poor, 2014).

Porém, nem todos *modders* participam da mesma forma. De acordo com os resultados da pesquisa de Nathaniel Poor, ilustrados na Figura 5.9, podemos observar que a grande maioria de *modders* oferecem seu feedback em *mods* de outras pessoas, com cerca de 75% de *modders* contribuindo diretamente algo para eles. Como foi mencionado nessa mesma seção em *Makers*, um ambiente onde tanto criadores quanto usuários podem oferecer *feedback* na criação de outros é favorável na evolução da qualidade de um software, e induz um política de iteração constante no processo de desenvolvimento. Quando um jogador sabe que seu *mod* será jogado por outras pessoas, é natural que o mesmo seja motivado e construir um produto de qualidade (Monterrat *et al.*, 2012). Além disso, um fator que eleva ainda mais a comunidade de *mods* como porta de entrada é o fator de contribuição. Um *modder* pode diretamente contribuir seu próprio conteúdo a um *mod*

pré-existente, seja para o melhorar, estender, ou até construir seu próprio *mod* baseado nele, resultando em um ambiente altamente similar ao de desenvolvimento de software livre, que é enraizado no conceito de colaboração.

	No	Yes
I have told a modder I liked their mod or thanked them for making it.	7.2 (8)	92.8 (103)
I have made comments in order to help someone improve their mod.	12.6 (14)	87.4 (97)
I have contributed code, scripting, voice, visual elements, or other content to someone else's mod.	23.4 (26)	76.6 (85)
I have co-authored a mod with others.	43.2 (48)	56.8 (63)
I have taken ownership of a mod someone else stopped working on.	82.9 (92)	17.1 (19)

Note: Results shown are percentage and, in parentheses, the *N*. *N* = 111.

Figura 5.9: *Uma tabela pertencente à pesquisa de Nathaniel Poor referente ao nível de interação que um modder tem com sua comunidade. Fonte:* Poor, 2014

Na mesma pesquisa, vemos um resultado favorável no aspecto de ensino e aprendizado, ou seja, na atitude que *modders* possuem na questão de ensinar outros jogadores a criarem seus próprios *mods*, assim como sua experiência aprendendo a criar *mods* através da ajuda de outros membros da comunidade. Segundo os resultados desse quesito, ilustrados na Figura 5.10, cerca de 82% dos *modders* aprenderam muito sobre *modding* com outros, e cerca de 76% dos *modders* já ajudaram outros a aprender como criar seus próprios *mods*. Além disso, em média, cerca de 64% concordam que se divertem em ajudar aos outros tanto para aprender como criar *mods* como também ensiná-los a como melhorar eles. Isso indica que em *modding*, temos um índice favorável de membros da comunidade que viriam a possivelmente ensinar jogadores inexperientes em desenvolvimento a desenvolverem seus próprios *mods*, e quando tal jogador se tornar mais experiente em desenvolver, o mesmo vir a ensinar outros membros da comunidade.

	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
I have learned a lot about modding from others.	0.9 (1)	6.3 (7)	9.9 (11)	39.6 (44)	43.2 (48)
I have helped others learn about modding.	4.5 (5)	9.0 (10)	9.9 (11)	45.0 (50)	31.5 (35)
I like to help others learn how to make mods.	0.9 (1)	2.7 (3)	26.1 (29)	39.6 (44)	30.6 (34)
I like to help other people improve their mods.	0.0 (0)	4.5 (5)	21.6 (24)	44.1 (49)	29.7 (33)

Note: Results shown are percentage and, in parentheses, the *N*. *N* = 111.

Figura 5.10: *Uma tabela pertencente à pesquisa de Nathaniel Poor referente à opinião de modders sobre o ensino e aprendizagem de modding em sua comunidade. Como é possível observar, os resultados indicam que a maioria dos modders tanto aprendem com outros membros da comunidade quanto também os ensinam. Fonte:* Poor, 2014

5.3.5 Motivação para Desenvolvimento

As motivações de ambos desenvolvedores de *mods* e de software livre são tanto intrínsecas e extrínsecas, sendo fatores como satisfação pessoal, ganho de reputação, e exposição a possíveis oportunidades profissionais grandes motivadores compartilhados entre os dois (Thiel e Lyle, 2019). Jogadores que se divertem quando desenvolvem modificações para jogos podem se interessar em desenvolver seu próprio jogo, principalmente aqueles que desenvolvem *mods* que requerem alta interação com o código e estrutura do jogo. Esse fator de divertimento e política de "aprender por fazer" traz os mesmos benefícios discutidos em *Makers*, onde a *gamification* (que pode ser traduzido para gamificação) do aprendizado o torna mais efetivo em muitos aspectos (Landers *et al.*, 2017).

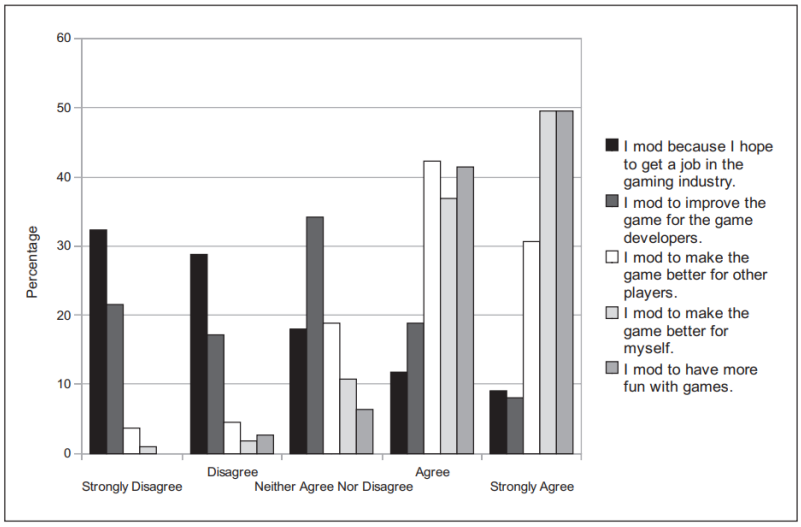

Figura 5.11: *Uma tabela pertencente à pesquisa de Nathaniel Poor referente à motivação de modders em desenvolver seus mods. Como é possível observar, a grande maioria das motivações por parte dos modders se alinham com o desejo de tornar a experiência de jogo mais divertida para tanto o criador do mod como para outros jogadores do jogo. Fonte: Poor, 2014*

Revisitando a pesquisa de Nathaniel Poor, ilustrado na Figura 5.11, cerca de 20% de *modders* criam *mods* pelo desejo de se tornar um desenvolvedor de jogos profissional. Considerando o alto número de *mods* sendo desenvolvidos nos tempos atuais, com cerca de 78.000 *mods* sendo lançados na plataforma NexusMods apenas no ano de 2022, uma estimativa de 20% é altamente favorável, pois *modders* que desenvolvem *mods* pelo desejo de se tornarem desenvolvedores ilustram a capacidade de *mods* como motivação para um jogador tomar interesse no âmbito de software. Para demonstrar ainda mais como *mods* podem ser uma ferramenta na incorporação de jogadores no ecossistema de desenvolvimento de software, serão ilustrados na Seção 5.4 alguns exemplos de jogos que originalmente eram *mods* desenvolvidos por jogadores.

5.4 Jogos derivados de Mods

Serão ilustrados nessa Seção jogos que originalmente eram *mods* de jogos pré-existentes.

5.4.1 Counter-Strike

O jogo **Counter-Strike (Valve, 2000)** (ilustrado na Figura 5.12), um FPS onde dois times de jogadores conectados pela internet devem se confrontar utilizando armas e facas, começou como uma conversão total de **Half-Life (Valve, 1998)** desenvolvida pela dupla Minh Lee e Jess Cliffe. Minh Lee, desejando realizar algo no campo de desenvolvimento de jogos para conseguir um emprego como desenvolvedor, começou a criar o *mod*, utilizando um Software Development Kit (SDK) fornecido pela Valve, produtora do jogo original. Através das ferramentas auxiliares, a dupla começou a lançar versões *beta* do jogo, e com a aplicação do feedback recebido e a contínua evolução do jogo, o interesse público no projeto cresceu exponencialmente(McLean-Foreman, 2001). Por volta do ano 2000, a dupla foi contatada pela Valve, que ofereceu comprar a propriedade intelectual do jogo e contratar os dois como desenvolvedores profissionais para desenvolver oficialmente o jogo como um produto comercial separado de **Half-Life (Valve, 1998)**. Desde então, o jogo se tornou uma franquia extremamente bem-sucedida, com seu jogo mais recente, **Counter-Strike: Global Offensive (Valve, 2012)** sendo um dos jogos de PC mais jogados de todos os tempos (Bankhurst, 2020).

Figura 5.12: *Uma imagem do jogo **Counter-Strike (Valve, 2000)**, originalmente um mod do jogo **Half-Life (Valve, 1998)**. Na imagem, podemos ver um jogador no time dos terroristas, armado com uma AK-47. Fonte: https://en.wikipedia.org/wiki/Counter-Strike_(video_game)*

5.4.2 The Stanley Parable

O jogo **The Stanley Parable (Galactic Cafe, 2013)**, uma experiencia *singleplayer* onde o propósito do jogo é ir contra a narrativa definida pela história, foi originalmente um *mod* do jogo **Half-Life II (Valve, 2004)** desenvolvido pelo *modder* Davey Wreden (ilustrado na Figura 5.13). Davey desejava criar algo que não havia experienciado antes em um jogo: uma narrativa não convencional onde o jogador tinha a habilidade de ir contra o próprio

narrador (Finch, 2011). Davey não possuía nenhuma experiência prévia com programação, logo teve de aprender a programar através da documentação e fóruns relacionados às ferramentas de *modding* que utilizava (Matias, 2011). Após receber *feedback* de colegas próximos, Davey publicou seu *mod* em 2011 no site **ModDB**, semanas antes de se formar. Inicialmente, Davey estava preparado para desistir do ramo de desenvolvimento, sendo a criação de uma conversão total sem experiência prévia de programação um processo árduo que sufocou sua ambição. Porém, após o sucesso do *mod*, Davey mudou seus planos e desejou desenvolver o jogo como um produto comercial *stand-alone* de forma independente, recusando ofertas de emprego de diversas produtoras de jogos (Matias, 2011). A nova versão do jogo, desenvolvida utilizando a mesma *game engine* que **Half-Life II (Valve, 2004)**, foi então lançada em 2013 na plataforma *Steam*, vendendo mais de 100.000 cópias em seus três primeiros dias de lançamento.

Figura 5.13: *Uma ilustração do mod original que eventualmente se tornaria o jogo* ***The Stanley Parable (Galactic Cafe, 2013)****. Fonte: https://en.wikipedia.org/wiki/The_Stanley_Parable*

5.4.3 Dota

O *mod* **Defense of the Ancients (IceFrog, 2003)**, conhecido como **DotA**, foi um *mod* para o jogo de estratégia em tempo real **Warcraft III (Blizzard, 2002)**. O *mod* constitui um mapa composto por duas bases em lados opostos do mapa, conectados por três caminhos, em que dois grupos de heróis controlados por múltiplos jogadores se enfrentavam através da internet (uma imagem do design do mapa pode ser visto na Figura 5.14). Cada grupo inicia em sua base, e seu objetivo é alcançar e destruir a base do outro grupo de jogadores no lado oposto do mapa. O *modder* inicial do jogo, Kyle Sommer, desenvolveu o *mod* utilizando as ferramentas auxiliares oferecidas para o jogo, que forneciam a habilidade de construir mapas e cenários novos para o jogo (Dean, 2014). DotA teve como inspiração um outro *mod*, chamado **Aeon of Strife**, que modificava o jogo **Starcraft (Blizzard, 1998)**. A fórmula e design de DotA teve tanto sucesso que foi inspiração de diversos outros *mods* similares (chamados comumente de "clones"), e o desenvolvedor principal do jogo, Ice Frog, foi contratado pela Valve como desenvolvedor para um jogo que seria uma sequência do *mod*, chamado DotA 2. Nos dias atuais, DotA é considerado o principal originador do gênero de jogos Multiplayer Online Battle Arena (MOBA), um dos gêneros de jogos mais populares de todos (Minotti, 2014).

Figura 5.14: *O mapa do jogo* ***Dota 2 (Valve, 2013)****, sequência do mod* ***Defense of the Ancients (IceFrog, 2003)****. Embora os gráficos sejam diferentes e de maior qualidade comparados ao mapa de Dota 1, o design permanece o mesmo, com duas bases em lados contrários do mapa, conectados por três principais caminhos: um inferior, um superior, e um central. Este design de mapa se tornou um marco no gênero MOBA, um gênero de jogo considerado criado pelo mod. Fonte: https://dota2.fandom.com/wiki/Map*

Capítulo 6

Plataformas para a criação e compartilhamento interno de jogos.

Neste capítulo, falaremos sobre plataformas para a criação e compartilhamento de jogos, assim como sua possível influência em introduzir novas pessoas ao desenvolvimento de software.

6.1 Caracterização

Plataformas para a criação e compartilhamento interno de jogos, que daqui em diante serão abreviadas para plataformas de jogos ou apenas plataformas, são uma categoria relativamente nova de jogos. Sua presença é notável, porém, devido à sua alta relevância e potencial, especialmente quando se trata de seu principal representante: o jogo **Roblox (Roblox Corporation, 2006)**. Outro jogo pertencente a essa categoria que iremos analisar é o jogo **Dreams (Media Molecule, 2020)**.

Podemos considerar que plataformas de jogos são como uma combinação de ***Makers*** com ***game engines***. Elas exercem uma função similar a de uma *game engine* em abstrair o desenvolvimento de jogos afim de diminuir a interação direta com código e realizar grande parte da implementação de módulos reutilizáveis. O que a torna similar a *Makers* é o fato de ser um jogo onde a construção de conteúdo pelos seus jogadores é seu principal propósito, e tal conteúdo ser **construído**, e não **criado**. De certa forma, podemos criar uma distinção entre as três categorias ao dizer que em *Makers*, o usuário **constrói níveis**; em *Mods*, o usuário **modifica jogos**; e em Plataformas de Jogos, o usuário **constrói jogos**.

As ferramentas utilizadas na construção de jogos em Plataformas de Jogos se assemelham a *Makers* ao tentar ser acessíveis a iniciantes, porém demonstram um nível de complexidade similar ao de uma *game engine*, com alguns jogos requerendo o uso de programação para utilizar as ferramentas ao seu dispor em seu maior potencial. Em contraste com as duas outras categorias previamente analisadas, Plataformas de Jogos colocam ênfase na liberdade de criação de jogos como um todo. Em *Makers*, o usuário era limitado

à criação de níveis, sem poder introduzir mecânicas novas ou modificar o comportamento do protagonista. Em *Mods*, a criação de um jogo novo através da modificação de um jogo pré-existente tipicamente era feita apenas em *mods* de alto escopo e complexidade como conversões totais. O objetivo dessas plataformas de jogos é tanto dissipar limitações e facilitar o processo de criação de jogos dentro de um jogo, sugerindo um potencial forte para a atração de pessoas ao âmbito de desenvolvimento.

6.1.1 Características

Assim como as duas categorias anteriores, ilustramos aqui características que definem Plataformas de Jogos.

- **Jogos são construídos, mas elementos podem ser criados:** De forma similar a *Makers*, as ferramentas de plataformas de jogos tendem a seguir a política de desenvolvimento de blocos de construção. Isto é, a ferramenta oferece peças e o usuário constrói jogos a partir delas, habilitando a criação de elementos complexos através da junção de elementos mais simples (algo que chamamos de *contraptions* na análise de *Makers*). Porém, em certas plataformas de jogos, é possível criar novos objetos e manipular seu comportamento através de ferramentas externas ou uso de código.
- **A construção de jogos pode requerer desenvolvimento interno e externo:** Em *Makers*, a construção de conteúdo era contida nas ferramentas internas ao jogo, e em *Mods*, a criação ou modificação havia de ser externa. Em plataformas de jogo, temos que a construção de jogos pode possuir elementos com os quais se interagem tanto de forma interna — no uso de ferramentas de construção como em **Dreams (Media Molecule, 2020)** — ou de forma externa na programação e manuseio de scripts, como em **Roblox (Roblox Corporation, 2006)**.

6.2 Práticas

Assim como em *Makers* e *Mods*, plataformas de jogos aplicam práticas a analisar.

6.2.1 Construtor de Jogos

O construtor de jogos pode ser definido como o conjunto de ferramentas oferecidas por uma plataforma de jogos. Para simplificar, podemos considerar o construtor de jogos como um **editor de níveis** de alta escala. Um fator divergente dos editores de níveis, porém, é o escopo das ferramentas. Em um *Maker*, o jogo implementava todo o comportamento do protagonista e interface, assim como oferecia um número pré-estabelecido de elementos visuais e auditivos, sendo níveis o limite do escopo de construção e edição das ferramentas. Porém, plataformas de jogos devem oferecer ao jogar a habilidade de construir jogos inteiros, e isso deve incluir funcionalidades para a criação ou implementação de recursos visuais, como arte, modelos, texto e interface; recursos auditivos, como música e efeitos sonoros; e recursos referentes a experiência de jogatina, como controles, mecânicas e comportamento de protagonistas e inimigos.

A construção de um jogo funciona de forma relativamente similar ao de um editor de níveis: temos um ambiente vazio, e para construir um jogo, populamos esse ambiente com objetos. Porém, tais objetos não são limitados a apenas uma lista pré-determinada de elementos físicos com comportamentos já definidos pelos desenvolvedores. Os objetos em um construtor de jogos podem ter sua aparência e comportamento modificados, tomando a forma dos recursos do jogo que o jogador-desenvolvedor tem em mente. Um exemplo de ambiente de construção do jogo e da modificação do comportamento dos objetos pode ser observado na Figura 6.1.

Figura 6.1: *Uma ilustração dos comportamentos referentes a um protagonista de um jogo feito na plataforma* ***Dreams (Media Molecule, 2020)****. Elementos como a velocidade de movimento, altura de seu pulo, entre outros, podem ser livremente alterados, o que não era possível nas ferramentas de construção em Makers. Fonte:* Pogomix, 2021

Porém, a maior parte da implementação de recursos nessas ferramentas não é feita pelo usuário em si, mas sim através de um sistema de **importação**. Além de fornecer uma variedade de objetos base, plataformas fornecem ao usuário a capacidade de importar recursos criados por outros usuários. Dessa forma, um usuário da ferramenta não precisa criar todos os recursos de seu jogo do zero, tendo a opção de utilizar recursos criados por outras pessoas (funcionalidade ilustrada na Figura 6.2), abrangendo recursos simples como um prédio ou carro capaz de ser dirigido, até módulos e sistemas de combate. Além disso, a plataforma pode oferecer a capacidade de importar recursos de programas externos à plataforma em si. A plataforma **Roblox (Roblox Corporation, 2006)**, para não ter de implementar um programa de criação de música em si mesmo, simplesmente oferece ao usuário a capacidade de adicionar um arquivo de música ao jogo.

Dito isso, uma plataforma de jogos pode permitir ao usuário a criar todos os recursos necessários para a construção de um jogo dentro da plataforma em si. Além de oferecer a funcionalidade de importar recursos publicados por outras pessoas, **Dreams (Media Molecule, 2020)** possui em si uma extensa coleção de ferramentas que habilitam a modelagem de ambientes e personagens, a criação de animações para jogadores e objetos, e até composição e mixagem de músicas e efeitos sonoros. A robusta ferramenta de composição de música é ilustrada na Figura 6.3.

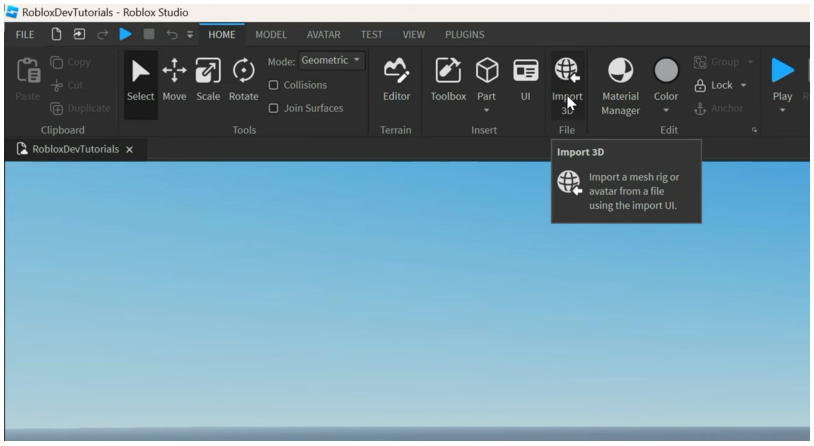

Figura 6.2: *Recorte da interface da ferramenta Roblox Studio, utilizada para construir jogos para a plataforma* ***Roblox (Roblox Corporation, 2006)****. O foco da imagem é a função de importar um modelo 3D para o jogo, seja ele feito por outra pessoa, ou em um programa externo. Fonte:* RobloxDevTutorials, 2023

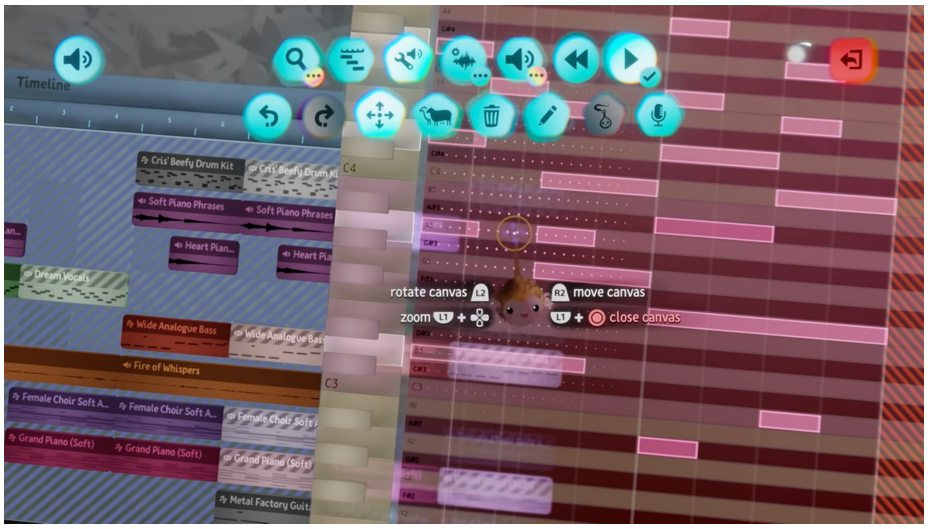

Figura 6.3: *Uma ilustração do* ***Music Mode****, a ferramenta de composição e mixagem de música da plataforma* ***Dreams (Media Molecule, 2020)****. Na esquerda, temos a timeline, onde o jogador insere porções de notas e arranjos usando instrumentos pré-implementados. Na direita, temos a composição de uma dessas porções, onde o usuário insere notas ao longo de um período para criar melodias. Fonte:* Pogomix, 2021

Em **Roblox (Roblox Corporation, 2006)**, é possível utilizar *scripts* escritos na linguagem **Luau**[1], um dialeto da linguagem de programação **Lua**[2], durante a construção de um jogo para invocar certas funcionalidades que não poderiam ser obtidas apenas com o uso das ferramentas não-programáticas oferecidas. O uso de tais *scripts* é opcional, porém encorajado em jogos de maior escopo e ambição, trazendo um incentivo para seu aprendizado. Um exemplo de código que pode ser aplicado em um jogo construído em **Roblox (Roblox Corporation, 2006)** pode ser observado no Programa 6.1.

[1] https://luau-lang.org/

[2] https://www.lua.org/home.html

```
local lava = script.Parent

local function kill(otherPart)
   local partParent = otherPart.Parent
   local humanoid = partParent:FindFirstChild("Humanoid")
   if humanoid then
      humanoid.Health = 0
   end
end

lava.Touched:Connect(kill)
```

Programa 6.1: *Trecho de código escrito em Luau aplicado a um objeto Lava em um jogo construído em* ***Roblox (Roblox Corporation, 2006),*** *sendo o propósito do código executar a função kill no objeto que encostar na lava. Se o objeto que encostou na lava é um humanóide, a função reduz a vida do objeto para zero. Fonte: Documentação oficial do Roblox* [3]

Quando o jogo é construído, é possível compartilhar ele pela internet publicamente. O jogo então pode ser encontrado no **Visualizador de Jogos** correspondente ao jogo. Esse **Visualizador de Jogos** pode ser visto como um equivalente do **Visualizador de Níveis** em *Makers*. Através de um sistema de busca, é possível separar jogos novos por quesitos como popularidade e relevância. Quando um nível é selecionado, são disponíveis informações pertinentes ao jogo como nome, criador e número de *downloads*, assim como funcionalidades para oferecer *feedback* ao criador, na forma de *likes* e comentários. Uma ilustração dessa funcionalidade de comentários pode ser vista na Figura 6.4.

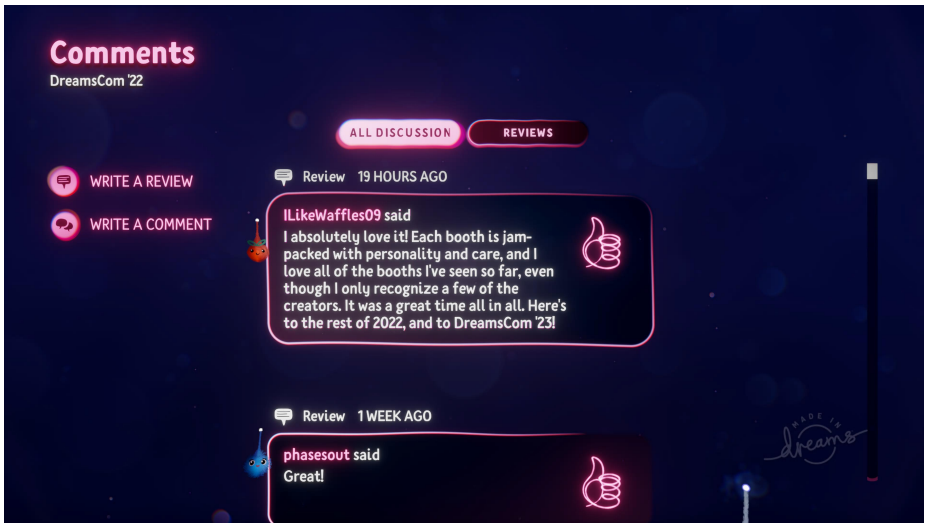

Figura 6.4: *Uma ilustração de um comentário positivo referente a um jogo construído na plataforma* ***Dreams (Media Molecule, 2020)****. Um jogador pode acessar um jogo feito por alguém e deixar seu feedback, comunicando ao criador sua opinião sobre a experiência. Fonte: Documentação oficial da plataforma Dreams.*[4]

[3] https://create.roblox.com/docs/pt-br/tutorials/scripting/basic-scripting/deadly-lava

[4] https://docs.indreams.me/en/create/resources/collaboration

6.2.2 Monetização

Um dos fatores mais controversos dessa emergente categoria de ferramenta é a habilidade de um usuário **monetizar suas criações**. Em jogos construídos em plataformas de jogos, o usuário que toma papel de desenvolvedor pode vir a implementar sistemas em seus jogos onde é feito um pagamento por parte do jogador a fim de desbloquear alguma funcionalidade, receber alguma recompensa dentro do jogo — no caso de **Roblox (Roblox Corporation, 2006)** (exemplificado na Figura 6.5) — ou para a venda de algum conteúdo feito pelo criador, como arte, animação, ou música — no caso de **Dreams (Media Molecule, 2020)**.

Figura 6.5: *Uma ilustração da tela de morte do jogo* ***Doors****, criado na plataforma* ***Roblox (Roblox Corporation, 2006)****. No jogo* ***Doors****, o jogador normalmente tem apenas uma vida para chegar da primeira até a última fase. Porém, se o jogador quiser continuar do ponto em que morreu, existe a possibilidade de gastar 30* ***Robux****, a moeda virtual da plataforma, para comprar uma vida extra, através do botão* ***Revive.*** *Fonte: https://doors-game.fandom.com/wiki/Death_Screen*

A monetização de conteúdo atrelado a um jogo sempre foi visto de forma problemática, devido à nebulosidade legal de conteúdo criado utilizando como base um outro jogo, sendo o dilema em questão os **direitos autorais** desse conteúdo. O que devemos relembrar é que jogos são produtos, e como mídia, são protegidos legalmente por direitos autorais (Kow e Nardi, 2010). Em *Makers* e *Mods*, não é possível realizar a monetização do conteúdo feito por jogadores pois na grande maioria dos casos, construir ou modificar conteúdo de um produto já pré-existente e vender esse conteúdo para outras pessoas pode entrar em direta violação com diversos contratos relacionados a direitos autorais. Alguns desses contratos incluem o ***Digital Millennium Copyright Act*** **(DMCA)**, o ***Computer Fraud and Abuse Act of 1986*** **(CFAA)**; ou também o ***End-User License Agreement*** **(EULA)** (Kretzschmar e Stanfill, 2019), um contrato de licença comumente atrelado a jogos, que em grande maioria dos casos, explicitamente proíbe jogadores de vender seus *mods* (Joseph, 2018).

Em 2015, a maior plataforma de venda de jogos para PC, a *Steam*, testou uma funcionalidade que autorizava *modders* de certos jogos específicos a vender seus *mods* (Wawro, 2015). Porém, esse recurso foi altamente criticado por tanto jogadores e *modders* logo

após seu lançamento, devido ao surgimento de *mods* caros e *mods* que vendiam conteúdo originalmente gratuito de outros *mods* (Joseph, 2018). Além disso, havia também o sentimento de que conseguir *modificar* o jogo e jogar *mods* de outros jogadores era a razão pela qual jogadores compravam o jogo original. Colocar uma barreira de pagamento nesse conteúdo tipicamente enraizado pelo voluntarismo e colaboração teria efeitos adversos na comunidade de *modding* como um todo (Joseph, 2018). A funcionalidade foi então removida da *Steam* dias após sua introdução, e todos os jogadores que compraram *mods* foram reembolsados pelo valor que gastaram com eles (Prescott, 2015).

A monetização é um elemento crucial no ecossistema de **Roblox (Roblox Corporation, 2006)**. A moeda virtual do jogo, **Robux**, foi introduzida em 2007, e é utilizada para fazer compras dentro do jogo, sendo essas compras tipicamente elementos cosméticos como roupas e acessórios que podem ser vestidos pelo avatar do jogador dentro do *videogame*. Usuários podem então introduzir essas compras dentro dos jogos que criam. Quando jogadores usam seus Robux para realizar compras dentro de um jogo criado por um usuário, 29% desses Robux são dados diretamente ao criador de jogo. O criador do jogo pode então converter os Robux que recebe de seu jogo para uma moeda real. Uma ilustração oficial do processo de monetização de jogos criados em Roblox pode ser observada na Figura 6.6.

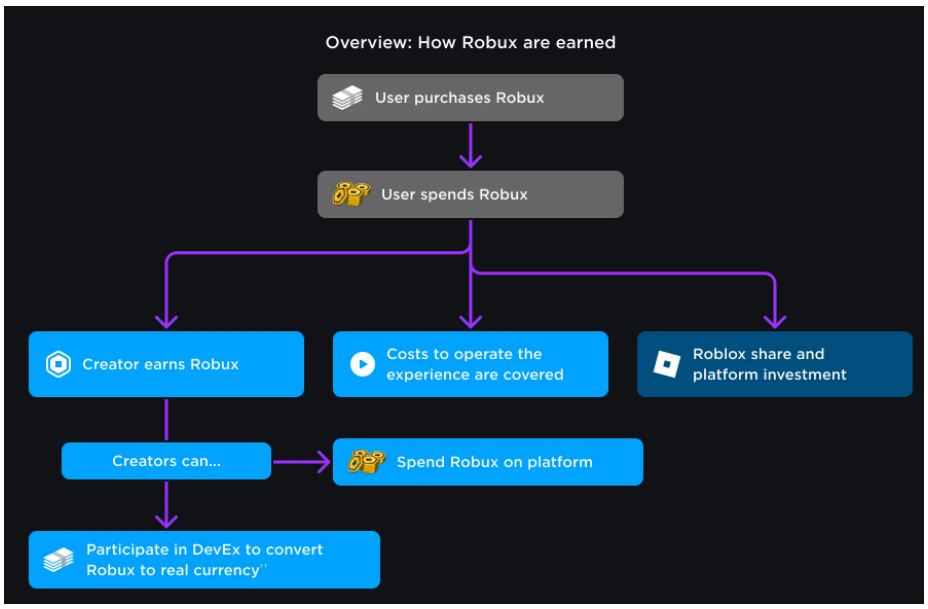

Figura 6.6: *Uma ilustração oficial do sistema de monetização do jogo* ***Roblox (Roblox Corporation, 2006)****. Um criador insere formas para o jogador gastar Robux em seu jogo, e em troca, recebe parte desses Robux, que podem ser convertidos em dinheiro. Nota-se que o "usuário" no diagrama se refere ao jogador do jogo, não o criador. Fonte: Documentação oficial do Roblox* [5]

Outro fator perigoso da monetização em **Roblox (Roblox Corporation, 2006)** são **golpes**, sendo o mais problemático deles a prática de *phishing*, onde golpistas criam sites maliciosos que oferecem **Robux**, itens e acessórios gratuitos a visitantes, com o intuito de roubar informações confidenciais (Chalk, 2022). Esse fator é ainda mais agravado quando

[5] https://create.roblox.com/docs/pt-br/production/earning-on-roblox

é levada em consideração a idade dos jogadores que sofrem esses golpes. Segundo fontes oficiais da plataforma, mais da metade dos jogadores de **Roblox (Roblox Corporation, 2006)** em 2020 eram menores de 13 anos, tornando crianças e pré-adolescentes, que tipicamente não possuem o conhecimento necessário para reconhecer e se proteger desses golpes, os principais alvos.

Além disso, a expectativa de ganho monetário que é imposta a esses jovens usuários os incentiva a impor muito de seu tempo e esforço para construir um alto número de jogos, causando uma forte saturação, onde a excessiva quantidade de jogos resulta em apenas os jogos mais populares receberem a chance de serem vistos por outros usuários. Usuários recebem uma porção muito pequena de seus reais ganhos, onde Roblox oferece apenas 29% do que seu jogo fatura, enquanto que a loja **Steam** oferece cerca de 70% (Statt, 2018). Ademais, usuários recebem em uma moeda virtual cuja conversão para dinheiro é regulada diretamente pela empresa. Logo, a alta acessibilidade da ferramenta e do sistema de monetização omite a dificuldade de receber dinheiro na plataforma (PeopleMakeGames, 2021).

Enquanto isso, a monetização em **Dreams (Media Molecule, 2020)** é feita de uma forma mais implícita, onde ao invés de implementar um sistema de compras dentro de jogos criados com a ferramenta, tudo o que ela faz é autorizar a venda de conteúdo criado na plataforma, como arte, modelos, música, animações, entre outros (Vieira, 2020). Devido à capacidade de criar álbuns e filmes completos com as ferramentas robustas da plataforma, usuários podem utilizar as ferramentas e vender seus produtos da mesma forma que um artista poderia vender um curta metragem que produziu em um software de animação qualquer.

6.3 Efetividade

Averiguaremos agora a efetividade de plataformas de jogos nas métricas definidas em comparação com as duas outras categorias.

6.3.1 Acessibilidade

Plataformas de jogos são similares a *Makers* no sentido de priorizarem a acessibilidade e diversão em suas ferramentas, afim de maximizar seu alcance para o maior número de jogadores possível. Porém, quando se trata de outros fatores como preço e disponibilidade, tais diferem de jogo para jogo. O jogo **Dreams (Media Molecule, 2020)**, lançado apenas para PS4 e PS5 por $20, vendeu cerca de 2 milhões de cópias até 2022, sofrendo um forte e constante declínio de usuários ativos ao longo dos anos (Kow e Nardi, 2022), culminando no fim de suporte oficial do jogo pela Media Molecule em setembro de 2023 (Welsh, 2023). Enquanto isso, o jogo **Roblox (Roblox Corporation, 2006)**, sendo um jogo grátis para todas as idades, disponível para computador, dispositivos móveis e vários consoles, se mostrou um dos jogos mais acessíveis de todos, alcançando cerca de 70 milhões de usuários ativos diariamente em novembro de 2023 (Kaser, 2023).

Em termos da acessibilidade das ferramentas em si, o alto escopo de funcionalidades quando comparado com um editor de níveis sugere uma maior barreira de entrada,

removendo grande parte das restrições deliberadamente impostas sobre *Makers* pelos desenvolvedores a fim de maximizar sua simplicidade. Porém, a abstração fornecida por plataformas de jogos ainda as tornam acessíveis para usuários sem experiência com desenvolvimento em comparação com *Mods*. Além disso, os jogos apresentam formas de aprender através de guias interativos e *templates* de jogos completos dentro do jogo em si. Quando se trata de engajamento, novamente, o jogo **Roblox (Roblox Corporation, 2006)** se mostra altamente relevante. Como mencionado antes, o jogo arrecada dezenas de milhões de jogadores diariamente, e isso é refletido na quantidade de conteúdo sendo criado para o jogo, com 40 milhões de jogos tendo sido criados na plataforma desde seu lançamento (Garrett, 2023), sendo que 70 desses jogos foram jogados mais de 1 bilhão de vezes (Takahashi, 2023).

Em conclusão, o nível de acessibilidade do jogo é alto devido a suas práticas compartilhadas com *Makers*, mas a diferença entre a complexidade de construir apenas um nível e construir um jogo inteiro torna o processo mais intimidador como um todo, reduzindo seu possível alcance para jogadores sem experiência com desenvolvimento. Porém, no caso do jogo **Roblox (Roblox Corporation, 2006)**, a vasta e ativa comunidade resulta em uma quantidade constante de conteúdo mesmo com poucos de seus jogadores tomando uso de suas ferramentas de construção.

6.3.2 Barreira de conhecimento

A barreira de conhecimento de plataformas de jogos é difícil de definir pois se assemelha a *mods* no sentido de que as ferramentas são muito mais flexíveis, logo a ambição do jogo em questão define o nível de interação com as funcionalidades mais complexas da ferramenta. Porém, mesmo tendo uma possível interação programática, as ferramentas em plataformas de jogos se apresentam de forma intuitiva, onde não é necessário aprender a forma como as ferramentas funcionam para as utilizar. Como ilustrado na Figura 6.1, é possível utilizar a ferramenta para modificar variáveis internas como posição e velocidade sem compreender como tal ferramenta interage com o código do objeto sendo modificado.

Entretanto, construir um jogo por completo, na maior parte das vezes, será naturalmente mais difícil que construir um único nível ou *mod*. Em um *Maker*, a quantidade de restrições e simplicidade das ferramentas garante que um nível sempre tenha seu escopo limitado até certo ponto. Enquanto isso, quando se desenvolve um *mod*, o usuário tem o grande benefício de trabalhar com um produto já desenvolvido, tornando o processo de adição ou modificação de algo estabelecido tipicamente mais simples do que construir algo inteiramente do zero. Plataformas de jogos oferecem poucas restrições ao jogador, e mesmo apresentando suas funcionalidades da forma mais acessível possível, a habilidade de construir jogos inteiros significa que conhecimentos em áreas adicionais são necessários para utilizar a ferramenta em seu maior potencial, como modelagem, música e *game balancing*.

A presença de opções de importação de jogos ou recursos amenizam essa dificuldade, e considerando o número alto de jogos sendo construídos com essas ferramentas, a **barreira de conhecimento não se mostra como um fator altamente restringente**, provavelmente devido ao alto número de recursos educativos atrelados ao jogo, como documentação interna/externa, tutoriais educativos, e até cursos especializados em sites como **Udemy** e

Codakid[1] no caso de **Roblox (Roblox Corporation, 2006)**.

6.3.3 Aprendizado na Prática

Em termos de aprendizados obtidos pelo uso das ferramentas em plataformas de jogos, temos uma situação similar a de *Makers*, onde devido à maior parte do processo de criação ser o uso de ferramentas que não exigem uso de uma interface programática, não é obtido um real aprendizado direto em como desenvolver um software. Porém, a maior flexibilidade das ferramentas fortalece a habilidade de aprender conceitos através do que é oferecido ao jogador. Enquanto que *Makers* haviam de usar objetos para representar elementos como variáveis, atributos, e funções, as ferramentas em plataformas diretamente implementam esses elementos nos componentes manuseados pelo usuário. Como exemplo, no capítulo de *Makers*, vimos que é possível simular uma função de *timer* utilizando uma *contraption* composta por objetos primitivos da seleção do editor de níveis, mas em **Dreams (Media Molecule, 2020)**, a ferramenta oferece ao usuário uma função de *timer* nativa. Isso oferece uma experiência muito mais próxima ao de um desenvolvedor mesmo com a ausência de código.

Devido ao alto número de crianças em **Roblox (Roblox Corporation, 2006)**, o uso da linguagem Lua para um *script* de Roblox pode se tornar a primeira interação com programação que o usuário tenha em sua vida. Além disso, vários estudos avaliam a plataforma como uma possível eficaz ferramenta de aprendizado para o ensino de programação a crianças (Han *et al.*, 2023) e até como um possível ambiente de aprendizado a ser integrado a ensino superior (Alhasan *et al.*, 2023).

Em conclusão, plataformas de jogos, assim como *Makers*, podem ser uma efetiva ferramenta educacional para a introdução de conceitos computacionais, fornecendo até uma possível interação com código. Porém, plataformas se encontram na posição difícil de estar no meio ponto entre *Makers* e *Mods*, onde a abstração não é alta o suficiente para ser altamente acessível como *Makers*, mas também não há a forte interação com código que é vista em *Mods*.

6.3.4 Interação com Comunidade

A interação com a comunidade em construção de jogos possui grande semelhança com a construção de níveis em *Makers*. O sistema de *feedback* resulta em um senso de comunidade e um desejo de reiterar e melhorar o conteúdo criado baseado na recepção de outros jogadores, trazendo uma motivação intrínseca para o desenvolvimento. Esse desejo de obter renome com seu conteúdo pode ser magnificado pela presença de monetização. Um criador de jogos é motivado a melhorar seu conteúdo não só para expressar sua criatividade e expandir seu renome, como também receber maior compensação monetária pelas suas criações, resultante do número maior de jogadores experienciando seus jogos.

Fora do **Visualizador de Jogos**, a interação de comunidade tipicamente deve ser feita externamente através de vias de comunicação como fóruns e redes sociais dedicadas à plataforma em questão. Algo que prejudica plataformas de jogos nesse quesito, porém,

[1] Site especializado em ensinar código e game design para crianças.

é o escopo de tais vias de comunicação. Em *Makers*, como níveis possuem um escopo pequeno e são desenvolvidos apenas por uma pessoa, a existência de um fórum ou rede social única que abrange o *Maker* inteiro não é um empecilho. Porém, quando se trata de jogos, onde pode haver um grupo de usuários e muitos recursos diferentes a serem usados e discutidos para um único projeto, é possível chegar a um ponto onde um fórum ou rede social dedicada inteiramente ao projeto deve ser feita, algo que não é oferecido de forma oficial pelas plataformas. Em *Mods*, onde esse mesmo problema poderia surgir, o risco é amenizado pelo fato de comunidades de *mods* serem enraizadas em comunidades pré-existentes de jogos modificados, oferecendo a *modders* um espaço já estabelecido para discussão. Logo, em plataformas de jogos, pode ser necessária a constante criação de comunidades externas, em serviços como o **Discord**[6], para conter comunicações referentes a cada jogo sendo produzido.

Algo a se reiterar é o fato da comunidade de **Roblox (Roblox Corporation, 2006)** ser composta em grande parte por crianças e pré-adolescentes, tornando a interação entre jogadores um possível risco. Sendo uma plataforma social, onde jogadores usam um sistema de *chat* para conversar uns com os outros, crianças ficam expostas constantemente a possíveis comunicações com adultos sem a supervisão de parentes. Para mitigar esse risco, **Roblox (Roblox Corporation, 2006)** impõe inúmeros sistemas de moderação e proíbe diretamente a postagem de imagens ou números em qualquer conversa feita dentro do jogo (Brown, 2023). Porém, esse sistema de moderação se estende apenas a conversas feitas na plataforma em si. Com a criação dessas comunidades externas mencionadas anteriormente, a comunidade composta majoritariamente por menores de idade pode se inserir em um ambiente que não possui moderação apropriada para a proteger de interações perigosas com possíveis criminosos.

Em conclusão, a comunicação com comunidade é equivalente em eficácia e importância com *Makers*. Porém, a discrepância de escopo entre níveis e jogos pode introduzir novos problemas se a plataforma não for construída para acomodar a constante discussão de conteúdo. Além disso, o maior representante de plataforma de jogos sofre várias controvérsias relacionadas a seu sistema de comunicação.

6.3.5 Motivação para Desenvolvimento

Podemos julgar que plataformas de jogos apresentariam um nível de motivação alto pois um usuário que possui experiência nas ferramentas oferecidas por estes jogos poderá utilizar o mesmo aprendizado em ferramentas similares, como *game engines*, e eventualmente desenvolver seu próprio jogo. Devido à direta interação não só com ferramentas que se aproximam em complexidade à ferramentas de desenvolvimento, assim como o alto número de usuários da ferramenta, plataformas de jogos ensinam aprendizados valiosos a um alto número de pessoas, especialmente se tal aprendizado nutrir um desejo de criação latente do jogador.

Porém, um fator importante a destacar quando se trata de motivação é a **monetização**. Devido ao fato de ser possível ganhar dinheiro construindo jogos utilizando essas ferramentas, isso pode se tornar a principal motivação de seus desenvolvedores. Mesmo que

[6] https://discord.com/

recompensa monetária seja considerada uma forte fonte de motivação, capaz de induzir mesmo jogadores totalmente inexperientes a tentar usar as ferramentas pelo desejo de ganhar dinheiro com isso, ter ganho monetário como a principal razão de desenvolver pode discordar com a mentalidade do software livre (Krishnamurthy *et al.*, 2014). Um jogador tornado usuário cujo maior objetivo é ganhar dinheiro construindo jogos utilizando plataformas de jogos possivelmente terá maior dificuldade em fazer uma transição a desenvolvimento com fins não estritamente lucrativos, algo comum em ecossistemas FOSS (Krishnamurthy *et al.*, 2014).

Em conclusão, plataformas de jogos possuem **alto potencial para motivação**, mas não é possível concretamente dizer que essa motivação é majoritariamente direcionada ao desenvolvimento de software livre. Enquanto que o elemento de voluntarismo presente em *Makers* e *Mods* indicava um potencial alto de transição para código aberto, a presença de um senso maior de autoria e elementos de monetização em Plataformas de Jogos torna a efetividade da transição nebulosa.

Capítulo 7

Conclusão

Levando em consideração a análise de todas as categorias, podemos extrair alguns resultados relevantes. ***Makers*** se mostraram como efetivos no quesito de acessibilidade e barreira de conhecimento, oferecendo ferramentas que quase qualquer um pode utilizar com facilidade, deliberadamente limitadas ao ponto de incentivar a criatividade de seus usuários, se tornando uma efetiva ferramenta de aprendizado. Como o escopo da criação de níveis é baixo e a interação de comunidade é encorajadora e simplista, não há a introdução de elementos ou ambientes externos desnecessários, encorajando um puro ciclo de constante iteração de criar níveis, publicar níveis, e aprender como criar níveis melhores. ***Mods*** trazem um destaque próprio por serem a categoria com exemplos concretos de jogadores que se tornaram desenvolvedores, e grande parte disso vem da forma como intuitivamente inserem o usuário no papel de um desenvolvedor. Se um *modder* deseja desenvolver um *mod* de alta escala, deve aprender sobre estrutura de código, além de programar o seu próprio código com base no software já existente. **Plataformas de jogos** são peculiares em possuir tanto inúmeras qualidades quanto inúmeros defeitos. A acessibilidade e barreira de conhecimento é favorável, porém a maior dificuldade em construir jogos em vez de níveis torna a barreira de entrada pior do que em *Makers*. Como ferramentas em plataformas se assemelham a *game engines*, há muito potencial para o aprendizado, mas tais ferramentas abstraem ao máximo qualquer interação com código, tornando *Mods* a melhor opção em engajamento com software de fato. A interação entre usuários é favorável como em *Makers*, porém o maior escopo de jogos comparado a níveis torna o gerenciamento de comunidades complexo, sem mencionar as controvérsias relacionadas ao fator social do maior representante de plataformas de jogos: **Roblox (Roblox Corporation, 2006)**.

No quesito de porta de entrada, a categoria mais efetiva há de ser *Mods*. Mesmo apresentando uma barreira de aprendizado maior devido à interação com código, *Mods* possuem uma forte combinação de possuir um alto alcance e oferecer uma experiência quase equivalente a de um desenvolvedor profissional. Como informado na Seção 5.1, jogos atuais são continuamente incentivados a oferecer suporte para *mods* devido aos benefícios que tais trazem para os desenvolvedores. Logo, o número e alcance de jogos modificáveis apenas aumenta com o tempo, aumentando a chance de um jogador encontrar um jogo que possa vir a desejar modificar, tipicamente devido à sua afinidade com o jogo. Ao começar o processo de criação de *mods*, o usuário terá de entender a estrutura interna do jogo e

interagir diretamente com ela por meio da programação. Um *modder* visualiza o jogo como um software, e ao modificar ou criar novo conteúdo em cima desse software, tal *modder* está exercendo função quase equivalente a um desenvolvedor. Dessa forma, é intuitivo deduzir que um jogador que obtêm prazer ao desenvolver *mods* terá a maior possibilidade de estender tal interesse para o desenvolvimento e modificação de software de forma mais geral, algo concretamente demonstrado pela existência de *mods* que se tornaram jogos próprios.

Quando se trata da proposta de aumento de acessibilidade no ambiente FOSS, a prática mais efetiva, exercida por todas as categorias, é a gamificação do aprendizado. Um jogo é uma experiência divertida e interativa, onde tarefas são realizadas simplesmente pelo prazer de as realizar. No caso em questão, a introdução de elementos de diversão e recompensa no aprendizado de conceitos computacionais pode tornar o processo de programação muito mais palatável. Um exemplo de como isso pode ser feito é o uso de guias interativos para o aprendizado de novas linguagens de programação, cuja etapa final é o desenvolvimento de algum projeto. Jogos motivam seus jogadores com objetivos, e motivam seus jogadores a atingir esses objetivos por meio da interatividade. Guias que estabelecem um projeto final a ser trabalhado, e tornam a jornada de desenvolvimento divertida através da interatividade, podem então transferir essa inerente atratividade que jogos possuem para a introdução de um maior número de pessoas ao âmbito de desenvolvimento de software.

Como um todo, podemos concluir que jogos, mesmo sendo apenas meios de lazer e distração, podem fornecer conhecimentos valiosos a seus jogadores. Através desses jogos, é possível ver que o desejo de desenvolver está presente em muitas pessoas, tudo o que é preciso fazer é oferecer um ambiente em que elas se sentem confortáveis em criar o que desejam criar. Logo, se é possível trazer esses potenciais novos membros para a comunidade de desenvolvimento, e tornar o ecossistema FOSS nesse ambiente de conforto e criatividade presente em jogos, a integração entre os dois pode resultar em diversos benefícios para o futuro da tecnologia e da inovação.

7.1 Trabalhos Futuros

Porém, existem várias formas de estender os tópicos discutidos nessa análise para futuros projetos. Devido ao baixo número de estudos relacionados ao potencial de jogos como porta de entrada para o desenvolvimento, fatores como aprendizado através da prática e motivação ao desenvolvimento tinham de ser extrapolados puramente através de informações públicas e generalizadas. Como extensão desse projeto, podem ser realizadas pesquisas que visam agregar estatísticas mais concretas sobre o nível de aprendizado que jogadores acumulam ao jogar esses jogos, além da proporção de jogadores que de fato constroem conteúdo porque possuem um interesse em desenvolvimento, e quantos deles de fato se tornam desenvolvedores. Além disso, os resultados obtidos nessa pesquisa podem ser uma premissa para a criação de ferramentas que visam tornar o processo de aprendizado de desenvolvimento em algo similar a um desses jogos, construindo então uma experiência que aplica tanto a efetividade do ensino e acessibilidade de *Makers* quanto a experiência prática presente em *Mods*.

Referências

[Alexander 2022] Harry Alexander. "Why skyrim is still so popular in 2022". *CBR* (2022) (citado na pg. 26).

[Alhasan *et al.* 2023] Khaled Alhasan, Khawla Alhasan e Sama' Al Hashimi. "Roblox in higher education: opportunities, challenges, and future directions for multimedia learning". *http://i-jet. org/* 18.19 (2023) (citado na pg. 54).

[Anderson *et al.* 2008] Eike Falk Anderson, Steffen Engel, Peter Comninos e Leigh McLoughlin. "The case for research in game engine architecture". In: *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*. 2008, pp. 228–231 (citado na pg. 5).

[Au 2002] Wagner J Au. "Triumph of the mod: player-created additions to computer games aren 'ta hobby anymore—they 're the lifeblood of the industry". *Salon. com* 16 (2002), pp. 399–415 (citado nas pgs. 26, 37).

[Balasubramanian 2022] Karthik Balasubramanian. "Game engines: all you need to know" (2022) (citado na pg. 6).

[Bankhurst 2020] Adam Bankhurst. "Steam reveals the top-selling and most-played games of 2020". In: IGN, 2020 (citado na pg. 42).

[Bates 2004] Bob Bates. *Game Design*. 2ª ed. Thomson Course Technology, 2004 (citado na pg. 14).

[Bitzer *et al.* 2007] Jürgen Bitzer, Wolfram Schrettl e Philipp JH Schröder. "Intrinsic motivation in open source software development". *Journal of comparative economics* 35.1 (2007), pp. 160–169 (citado na pg. 22).

[Blake 2011] Michael Blake. "Pc gaming: doomed? or zdoomed?" (2011). URL: https://web.archive.org/web/20140221222302/http://www.ign.com/articles/2011/06/23/pc-gaming-doomed-or-zdoomed (citado na pg. 4).

[Brown 2023] Maressa Brown. "Is roblox safe for kids? here's what the experts have to say". *Parents* (2023) (citado na pg. 55).

[Burger-Helmchen e Cohendet 2011] Thierry Burger-Helmchen e Patrick Cohendet. "User communities and social software in the video game industry". *Long Range Planning* 44.5-6 (2011), pp. 317–343 (citado na pg. 34).

[Burgos *et al.* 2007] Carlos L Burgos, Julie JCH Ryan e Edward Lile Murphree. "Through a mirror darkly: how programmers understand legacy code". *Information Knowledge Systems Management* 6.3 (2007), pp. 215–234 (citado na pg. 38).

[Carreño e Winbladh 2013] Laura V. Galvis Carreño e Kristina Winbladh. "Analysis of user comments: an approach for software requirements evolution". In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 582–591. doi: 10.1109/ICSE.2013.6606604 (citado na pg. 22).

[Chalk 2022] Andy Chalk. "A new report on roblox reveals how hackers and scammers are continuing to rip off kids". *PC Gamer* (2022) (citado na pg. 51).

[Contini *et al.* 2020] Guilherme Cardoso Contini, Gabrielly Del Carlo Richene e Dorival Campos Rossi. "Análise sobre a trajetória do design maker e o diy nos games contemporâneos" (2020) (citado na pg. 13).

[Dealessandri 2020] Marie Dealessandri. "What is the best game engine: is godot right for you?" (2020). url: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-godot-right-for-you (citado na pg. 5).

[Dean 2014] Paul Dean. "The story of dota; how a bastard mod became its own genre". In: Eurogamer, 2014 (citado na pg. 43).

[Donovan 2010] Tristan Donovan. *Replay: The History of Video Games*. Yellow Ant, abr. de 2010, p. 501. isbn: 978-0-9565072-0-4 (citado na pg. 6).

[Edward M. Corrado e Mitchell 2018] Heather Moualison Sandy Edward M. Corrado e Erik T. Mitchell. "Nullis in verba: the free software movement as a model for openness and transparency". *Technical Services Quarterly* 35.3 (2018), pp. 269–279. doi: 10.1080/07317131.2018.1456849. eprint: https://doi.org/10.1080/07317131.2018.1456849. url: https://doi.org/10.1080/07317131.2018.1456849 (citado na pg. 4).

[Finch 2011] Greg Finch. "The man behind the deconstructionist mod the stanley parable". In: Vice, 2011 (citado na pg. 43).

[Fornós 2020] Silvia Fornós. "Super mario maker 2 as a tool for educational game design". In: *14th European Conference on Game Based Learning Proceedings*. Academic Conferences e Publishing International Limited. 2020, pp. 801–804 (citado na pg. 21).

[Garrett 2023] Ural Garrett. "What is roblox? here's everything you need to know". *CNN* (2023) (citado na pg. 53).

[Gregory 2018] Jason Gregory. *Game engine architecture*. crc Press, 2018 (citado na pg. 25).

[Han *et al.* 2023] Jining Han, Geping Liu e Yuxin Gao. "Learners in the metaverse: a systematic review on the use of roblox in learning". *Education Sciences* 13.3 (2023), p. 296 (citado na pg. 54).

[Hempson 2011] Lewis Hempson. "Striking a balance: is nintendo digging its grave with shovelware?" (2011). url: https://www.nintendojo.com/features/editorials/striking-a-balance (citado na pg. 4).

[Joseph 2018] Daniel James Joseph. "The discourse of digital dispossession: paid modifications and community crisis on steam". *Games and Culture* 13.7 (2018), pp. 690–707 (citado nas pgs. 50, 51).

[Karmali 2013] Luke Karmali. "No further skyrim dlc planned, team "moving on"". *IGN* (2013) (citado na pg. 26).

[Kaser 2023] Rachel Kaser. "Roblox reports 38% revenue spike in q3, 70m daily active users". *VentureBeat* (2023) (citado na pg. 52).

[Kinnunen e Malmi 2006] Päivi Kinnunen e Lauri Malmi. "Why students drop out cs1 course?" In: *Proceedings of the second international workshop on Computing education research*. 2006, pp. 97–108 (citado na pg. 1).

[Kow e Nardi 2010] Yong Ming Kow e Bonnie Nardi. "Who owns the mods?" *First Monday* (2010) (citado na pg. 50).

[Kow e Nardi 2022] Yong Ming Kow e Bonnie Nardi. "Dreams has excellent numbers, but there are fewer and fewer players". *Pledge Times* (2022) (citado na pg. 52).

[Kretzschmar e Stanfill 2019] Mark Kretzschmar e Mel Stanfill. "Mods as lightning rods: a typology of video game mods, intellectual property, and social benefit/harm". *Social & legal studies* 28.4 (2019), pp. 517–536 (citado na pg. 50).

[Krishnamurthy *et al.* 2014] Sandeep Krishnamurthy, Shaosong Ou e Arvind K Tripathi. "Acceptance of monetary rewards in open source software development". *Research Policy* 43.4 (2014), pp. 632–644 (citado na pg. 56).

[Landers *et al.* 2017] Richard N Landers, Michael B Armstrong e Andrew B Collmus. "How to use game elements to enhance learning: applications of the theory of gamified learning". *Serious Games and Edutainment Applications: Volume II* (2017), pp. 457–483 (citado na pg. 41).

[Lee *et al.* 2020] Daniel Lee, Dayi Lin, Cor-Paul Bezemer e Ahmed E Hassan. "Building the perfect game–an empirical study of game modifications". *Empirical Software Engineering* 25 (2020), pp. 2485–2518 (citado na pg. 26).

[Matias 2011] Jeff Matias. "Interview: the stanley parable developer davey wreden". In: Shacknews, 2011 (citado na pg. 43).

[McLean-Foreman 2001] John McLean-Foreman. "Interview with minh le". In: Gamasutra, 2001 (citado na pg. 42).

[Minotti 2014] Mike Minotti. "The history of mobas: from mod to sensation". In: VentureBeat, 2014 (citado na pg. 43).

[Monterrat *et al.* 2012] Baptiste Monterrat, Elise Lavoué e Sébastien George. "Learning game 2.0: support for game modding as a learning activity". In: *Proceedings of the 6th European Conference on Games Based Learning*. 2012, pp. 340–347 (citado na pg. 39).

[El-Nasr e Smith 2006] Magy Seif El-Nasr e Brian K Smith. "Learning through game modding". *Computers in Entertainment (CIE)* 4.1 (2006), 7–es (citado na pg. 26).

[NintendoOfAmerica 2016] NintendoOfAmerica. *If you played every level in SuperMarioMaker for 1 minute each, it would take you nearly 14 years to play them all!* 2016. URL: https://twitter.com/NintendoAmerica/status/732624228428750848 (citado nas pgs. 1, 20).

[Oshino 2015] Yosuke Oshino. *Interview: Yosuke Oshino On The Origins And Evolution Of Super Mario Maker*. Set. de 2015. URL: https://www.nintendolife.com/news/2015/09/interview_yosuke_oshino_on_the_origins_and_evolution_of_super_mario_maker (citado nas pgs. 13, 20).

[Parrish 2023] Ash Parrish. *Minecraft has sold over 300 million copies*. The Verge, 2023. URL: https://theverge.com/2023/10/15/23916349/minecraft-mojang-sold-300-million-copies-live-2023 (citado na pg. 37).

[PeopleMakeGames 2021] PeopleMakeGames. *Investigation: How Roblox Is Exploiting Young Game Developers*. Youtube. 2021. URL: https://www.youtube.com/watch?v=_gXlauRB1EQ (citado na pg. 52).

[Pogomix 2021] Pogomix. *Dreams is so powerful it's absurd*. Youtube. 2021. URL: https://www.youtube.com/watch?v=QE996r92obE (citado nas pgs. 47, 48).

[Poor 2014] Nathaniel Poor. "Computer game modders' motivations and sense of community: a mixed-methods approach". *New media & society* 16.8 (2014), pp. 1249–1267. DOI: https://doi.org/10.1177/1461444813504266 (citado nas pgs. 25, 35, 38–41).

[Poretski e Arazy 2017] Lev Poretski e Ofer Arazy. "Placing value on community co-creations: a study of a video game'modding'community". In: *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*. 2017, pp. 480–491 (citado na pg. 35).

[Prescott 2015] Shaun Prescott. “Valve has removed paid mods functionality from steam workshop”. In: PC Gamer, 2015 (citado na pg. 51).

[Rabin 2000] Steve Rabin. “The magic of data-driven design”. *Game Programming Gems, Charles River Media* (2000), pp. 3–7 (citado na pg. 31).

[RobloxDevTutorials 2023] RobloxDevTutorials. *How to IMPORT a BLENDER Mesh into Roblox Studio!* Youtube. 2023. url: https://www.youtube.com/watch?v=EO4CBJYzWlQ (citado na pg. 48).

[Scacchi 2003] Walt Scacchi. “Free/open source software development practices in the computer game community”. *Institute for Software Research, University of California, Technical Report* (2003) (citado na pg. 26).

[Scacchi 2011] Walt Scacchi. “Modding as an open source approach to extending computer game systems”. In: vol. 3. Jul. de 2011, pp. 62–74. isbn: 978-3-642-24417-9. doi: 10.4018/jossp.2011070103 (citado nas pgs. 26, 30, 35, 39).

[Sharma *et al.* 2002] Srinarayan Sharma, Sugumaran Vijayan e Rajagopalan Balaji. “A framework for creating hybrid-open source software communities”. *Information Systems Journal* 12 (2002), pp. 7–25 (citado na pg. 1).

[Sharples *et al.* 2013] Mike Sharples *et al. Innovating Pedagogy 2013: Exploring new forms of teaching, learning and assessment, to guide educators and policy makers.* Set. de 2013. isbn: 978-1-78007-937-0 (citado na pg. 13).

[Sherrod 2006] Allen Sherrod. *Ultimate 3D game engine design & architecture.* Charles River Media, Inc., 2006 (citado na pg. 5).

[Statt 2018] Nick Statt. “Valve’s new steam revenue agreement gives more money to game developers”. *The Verge* (2018) (citado na pg. 52).

[Szpytek 2021] Peter Hunt Szpytek. “Skyrim: every version of the game that’s been released so far”. *GameRant* (2021) (citado na pg. 26).

[Takahashi 2023] Dean Takahashi. “Roblox says 70 user-created games have crossed a billion plays”. *VentureBeat* (2023) (citado na pg. 53).

[Therrien 2015] Carl Therrien. “Inspecting video game historiography through critical lens: etymology of the first-person shooter genre”. *Game Studies* 15.2 (2015) (citado na pg. 6).

[Thiel e Lyle 2019] Sarah-Kristin Thiel e Peter Lyle. “Malleable games-a literature review on communities of game modders”. In: *Proceedings of the 9th International Conference on Communities & Technologies-Transforming Communities.* 2019, pp. 198–209 (citado na pg. 41).

[Vieira 2020] Douglas Vieira. “Dreams testa sistema de monetização de conteúdo original”. *Tecmundo* (2020) (citado na pg. 52).

[Wallace 2014] Ryan Wallace. “Modding: amateur authorship and how the video game industry is actually getting it right”. *BYu L. REv.* (2014), p. 219 (citado na pg. 26).

[Wawro 2015] Alex Wawro. “Game mods can now be sold on the steam workshop for real money”. In: Game Developer, 2015 (citado na pg. 50).

[Welsh 2023] Oli Welsh. “Sony ending support for dreams in september”. *Polygon* (2023) (citado na pg. 52).

[Wood 2023] Martin Wood. “Minecraft’s 1.21 update could be copper’s time to shine”. *GameRant* (2023). url: https://gamerant.com/minecraft-121-update-copper-trial-chambers/ (citado na pg. 29).

[Wynn e Eckert 2017] David C Wynn e Claudia M Eckert. “Perspectives on iteration in design and development”. *Research in Engineering Design* 28 (2017), pp. 153–184 (citado na pg. 23).

[Yaden 2023] Joseph Yaden. “Skyrim is now within the top 10 best-selling games of all time”. *GameSpot* (2023) (citado na pg. 26).

[Zachary 1996] George Zachary. “Generator: how a little game called doom may have changes the business world forever”. *Next Generation* 21 (1996), p. 20 (citado na pg. 6).