Resquício: Jogo para Introdução à Programação

Igor Minerva[1], **Artur Mihok**[1], **Leonardo Bravo**[1]

[1]Instituto Federal de Santa Catarina (IFSC)
Rua Heitor Villa Lobos, 255 - 88506-400 - Lages - SC - Brasil

{igor.m12,artur.mc,}@aluno.ifsc.edu.br, leonardo.bravo@ifsc.edu.br

Abstract. *The use of gamification in undergraduate courses contributes to more effective teaching and engages students in the learning process. However, in computer science undergraduate programs, this tool is rarely applied, resulting in a high dropout rate during the early stages. Based on this, the objective of this work is to create a digital RPG-style game that allows students to solve exercises in a playful way. The game can be applied in the classroom to incorporate gamification concepts into introductory programming courses in computer science programs.*

Resumo. *O uso da gamificação nos cursos de graduação contribui para um ensino mais proveitoso e engaja os alunos no processo de aprendizado. Apesar disso, nos cursos de graduação em ciência da computação essa ferramenta é raramente aplicada, resultando em um grande índice de desistências nas fases inicias. Tendo isso como base, o objetivo desse trabalho é criar um jogo digital no estilo RPG que permite que os alunos resolvam exercícios de forma lúdica. O jogo poderá ser aplicado em sala de aula para assim implementar os conceitos de gamificação nas matérias de introdução à programação dos cursos de ciência da computação.*

1. Introdução

No curso de Ciência da Computação, a Introdução à Programação é uma Unidade Curricular fundamental e seus conceitos acompanham o aluno durante todo o percurso da graduação. Segundo Zhao et al. (2022) a programação é uma das matérias mais difíceis do currículo *Science, Technology, Engineering and Mathematics* (STEM). Como a nova geração de alunos é exposta à tecnologia já nos primeiros anos de vida, eles ficam com dificuldades a aprender conceitos de programação na maneira tradicional. Por isso, vários acadêmicos buscam jogos para entender melhor os conceitos ensinados, utilizando assim técnicas de gamificação para auxiliar o aprendizado.

> "A gamificação, tradução do termo em inglês *gamification*, pode ser entendida como a utilização de elementos de jogos em contextos fora de jogos, isto é, da vida real. O uso desses elementos – narrativa, *feedback*, cooperação, pontuações etc. – visa a aumentar a motivação dos indivíduos com relação à atividade da vida real que estão realizando." (Murr e FERRARI, 2020)

Os maiores problemas relacionados à programação no curso, são as dificuldades que os alunos enfrentam com várias habilidades como escrever código, resolução de

problemas e algoritmos complexos (Maryono et al., 2022). Isso pode causar perda de interesse dos alunos, fazendo com que abandonem o curso de Ciência da Computação nas fases iniciais.

É interessante fazer um estudo sobre esses problemas para saber se o uso de jogos pode melhorar o aprendizado dos alunos e determinar se o índice de desistência do curso pode ser reduzido. Assim, pode-se entender também se a utilização da gamificação como forma de aprendizado será mais eficiente em conjunto com a maneira tradicional do que esta última sozinha, no sentido de melhor aprendizado pelos alunos. Porque, sendo o jogo uma atividade integrante do nosso contexto cultural, é comum que haja o uso de jogos em outros contextos, como práticas pedagógicas, proporcionando ao estudante a vivência de experiências de aprendizagem que talvez não fossem tão fáceis de serem alcançadas através do ensino tradicional (Giardinetto e Mariani, 2005).

Diante disso, o objetivo do trabalho será criar um jogo eletrônico que, ao ser aplicado em sala de aula utilizando o conceito de gamificação, ajudará os alunos ao melhor entendimento dos conceitos básicos da programação. O jogo será do gênero *Role Playing Game* (RPG), onde o aluno precisa ajudar os habitantes de uma cidade resolvendo desafios através de uma linguagem de programação. Os conceitos que serão utilizados para resolver os problemas serão aqueles vistos na primeira fase do curso. Os objetivos específicos do trabalho são:

- Pesquisar referências de jogos e trabalhos relacionados a ensino de conceitos técnicos na computação, sobretudo envolvendo linguagem de programação;
- Desenvolver a narrativa, focada em resolução de problemas voltados a programação de computadores;
- Realizar o *gamedesign* e desenvolvimento do jogo digital;

Este trabalho está organizado da seguinte forma: Na Seção 1 encontra-se a introdução; na Seção 2, o detalhamento da metodologia; na Seção 3, o referencial teórico; na Seção 4, o desenvolvimento; na Seção 5, resultados e discussões; e na Seção 6, conclusão.

2. Metodologia

Do ponto de vista da natureza, esse trabalho é uma pesquisa aplicada. Quanto à abordagem do problema, trata-se de uma pesquisa qualitativa. Referente aos objetivos, classifica-se como uma pesquisa exploratória. Considerando os procedimentos técnicos, a pesquisa é bibliográfica. Conforme ilustrado na Figura 1, o trabalho foi desenvolvido em duas grandes etapas:

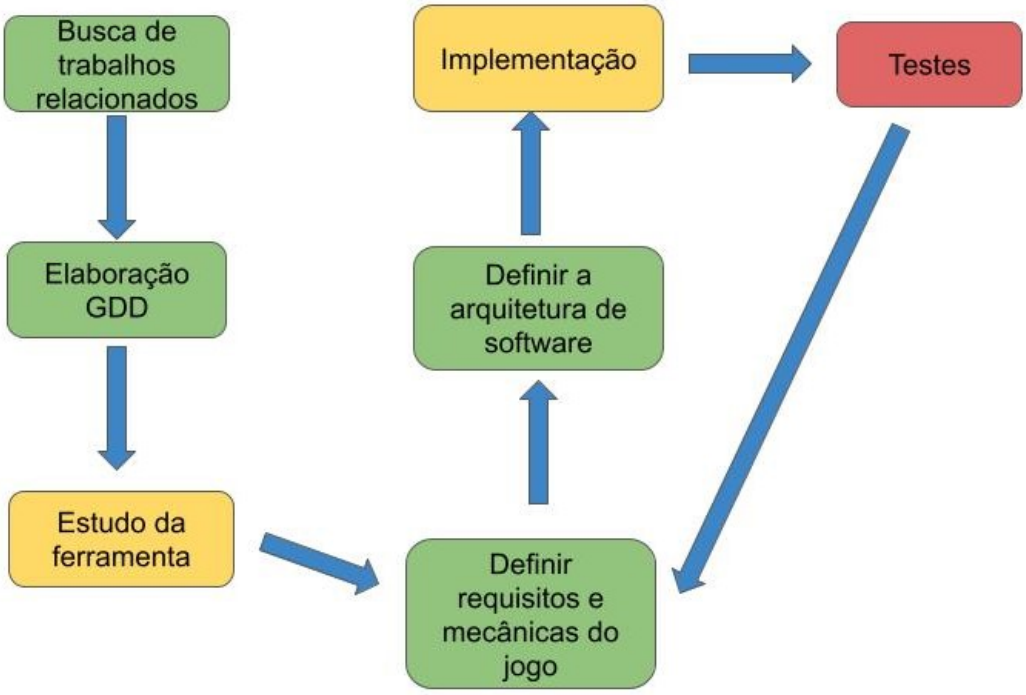

Figura 1. Fluxograma da metodologia utilizada. Em verde estão os processos teóricos e em amarelo e vermelho os práticos.

A primeira etapa para o desenvolvimento do trabalho consistiu em pesquisar através de plataformas de trabalhos acadêmicos jogos que ensinem, o uso deles na educação, jogos do tipo RPG e outras ferramentas que se alinham com a proposta do trabalho. Após isso, foi elaborado um *Game Design Document* (GDD), documento teórico com a história do jogo, diálogos entre os personagens, escolhas de design, elementos que compõe o cenário do jogo, entre outros detalhes.

Na segunda etapa, foi realizado o estudo das ferramentas utilizadas, sobretudo do *Godot Engine*, uma *game engine* baseada principalmente nas linguagens *C#* e *GDScript*. Assim, foram definidos conceitos no GDD relacionados aos requisitos da parte mecânica do jogo, como movimento do personagem, interação com objetos, câmera, mecanicas de escrita da linguagem de programação e o uso de interpretador dentro do jogo, entre outros. Ainda na segunda etapa, foi definida a arquitetura do *software* e, posteriormente, a implementação de fato. Durante a etapa de implementação foram revistos conceitos, realizados testes manuais, novos requisitos foram criados de acordo com o andamento do jogo e a arquitetura foi refinada; assim, o desenvolvimento não foi linear e seguiu um ciclo de revisão e aprimoramento.

3. Referencial Teórico

Nesta seção será abordado alguns conceitos teóricos utilizados na criação do jogo, como a criação da história, conceitos da modelagem 3D, as ferramentas utilizadas e os trabalhos relacionados ao tema deste projeto.

3.1. Introdução à Programação

A disciplina de Introdução à Programação é ministrada nas fases iniciais dos cursos de ciência da computação e tem como objetivo apresentar os conceitos fundamentais da programação de computadores, proporcionando aos alunos a capacidade de analisar e interpretar problemas. Essa base teórica e prática é essencial para que os estudantes sejam capacitados a traduzir algoritmos em programas utilizando uma linguagem de programação de alto nível. O conteúdo inicial abrange os princípios de algoritmos e a relação com a programação, introduzindo aspectos essenciais como variáveis e tipos de dados.

3.2. Jornada do Herói

Segundo Campbell (2008), a Jornada do Herói delineia uma estrutura narrativa recorrente encontrada em mitos e narrativas ao redor do mundo. Este modelo analítico identifica uma sequência de estágios através dos quais o protagonista de uma história passa, desde o seu estado inicial no "Mundo Comum" até sua transformação e retorno, geralmente trazendo consigo um elixir ou conhecimento valioso. Os estágios principais incluem o "Chamado à Aventura", a "Recusa ao Chamado", o "Encontro com o Mentor", o "Cruzamento do Limiar" e a "Ressurreição do Herói", entre outros. Embora nem todas as narrativas sigam esta estrutura de forma linear ou incluam todos os estágios, a Jornada do Herói oferece uma lente poderosa para compreender a jornada de autodescoberta, desafios e triunfos enfrentados pelos protagonistas, refletindo aspectos universais da experiência humana através do poder da narrativa. Isso é muito importante para a criação de diálogos e para dar um sentido de história ao jogo.

3.3. Modelagem 3D

Nesta subseção serão apresentados vários conceitos de modelagem 3D que foram utilizados no desenvolvimento do trabalho para sua parte visual. Em ordem, será explicado o que é um modelo 3D, o que é o formato *GL Transmission Format (glTF)* e o porque da sua utilização, o que são *textures* e a suas diferentes tipologias, o conceito de material e o que é *shading* e seus diferentes tipos.

3.3.1. Modelo 3D

Para Akenine-Möller et al. (2018), um modelo 3D é uma representação matemática de qualquer objeto tridimensional, criado através de técnicas computacionais. Na base da formação de um modelo 3D estão os vértices, que são pontos no espaço tridimensional. Esses vértices são conectados por arestas para formar polígonos, geralmente triângulos ou quadriláteros, que juntos compõem a superfície do objeto. A estrutura básica de um modelo 3D, chamada de malha, pode ser extremamente simples ou muito complexa, dependendo do nível de detalhe desejado. Na Figura 2 está ilustrado um exemplo das arestas de um modelo de vara de pesca.

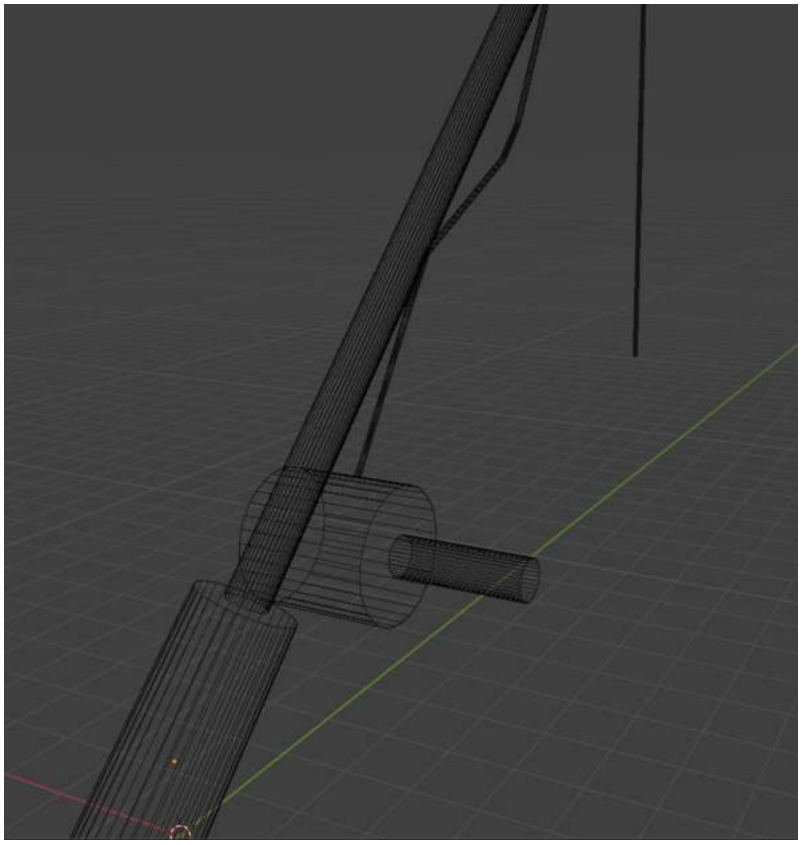

Figura 2. Exemplo das arestas de um modelo de uma vara de pesca.

A construção de um modelo 3D começa com a modelagem, onde os artistas ou engenheiros criam a forma básica do objeto. Existem várias técnicas de modelagem, como a modelagem de polígonos, que é a mais comum, a modelagem de superfícies subdivididas, que suaviza as formas, e a modelagem *Non-Uniform Rational Basis Spline (NURBS)*, que é usada para criar superfícies curvas suaves com alta precisão. A topologia da malha é um aspecto crucial da modelagem, influenciando diretamente a qualidade e a eficiência da renderização e da animação.

3.3.2. Formato *glTF*

Segundo Group (2021), o formato *glTF* é um padrão para a transmissão e o carregamento de modelos 3D que foi desenvolvido pelo *Khronos Group*. Este formato é otimizado para a transmissão eficiente e a renderização de cenas 3D em aplicações *web* e outras plataformas. O *glTF* pode armazenar informações essenciais de um modelo 3D, como a geometria (vértices, normais, e texturas), materiais, animações, e hierarquias de cena em um único arquivo ou em arquivos separados para maior modularidade.

O formato é baseado em *JavaScript Object Notation (JSON)* para a estrutura de dados descritiva, com a geometria e outros dados binários armazenados em *buffers* externos ou embutidos. Essa combinação permite que o *glTF* seja facilmente lido, transmitido e renderizado por motores gráficos em tempo real. Além disso, o *glTF* suporta uma ampla gama de recursos avançados, incluindo texturas *Physically-Based Rendering (PBR)*,

que permitem a criação de materiais realistas ao definir como a superfície de um objeto interage com a luz de maneira física e precisa. O formato também suporta compactação de texturas, redução de malhas, e animações complexas, garantindo que mesmo modelos altamente detalhados possam ser carregados e renderizados com rapidez. Este modelo foi utilizado no projeto para o formato de todos os objetos 3D que foram modelados.

3.3.3. Texturas

Por Akenine-Möller et al. (2018), as texturas em modelos 3D são imagens aplicadas às superfícies dos objetos para adicionar detalhes visuais que vão além da geometria básica. Elas desempenham um papel crucial em definir a aparência final do modelo ao simular diferentes características de materiais. Entre os tipos mais comuns de texturas estão: *Base Color, Normal, Ambient Occlusion, Metallic, Roughness* e *Emissive*.

A *Base Color* é a textura que define a cor principal do objeto, sem efeitos de iluminação ou sombreamento, representando as cores vistas em condições neutras, como na Figura 3.

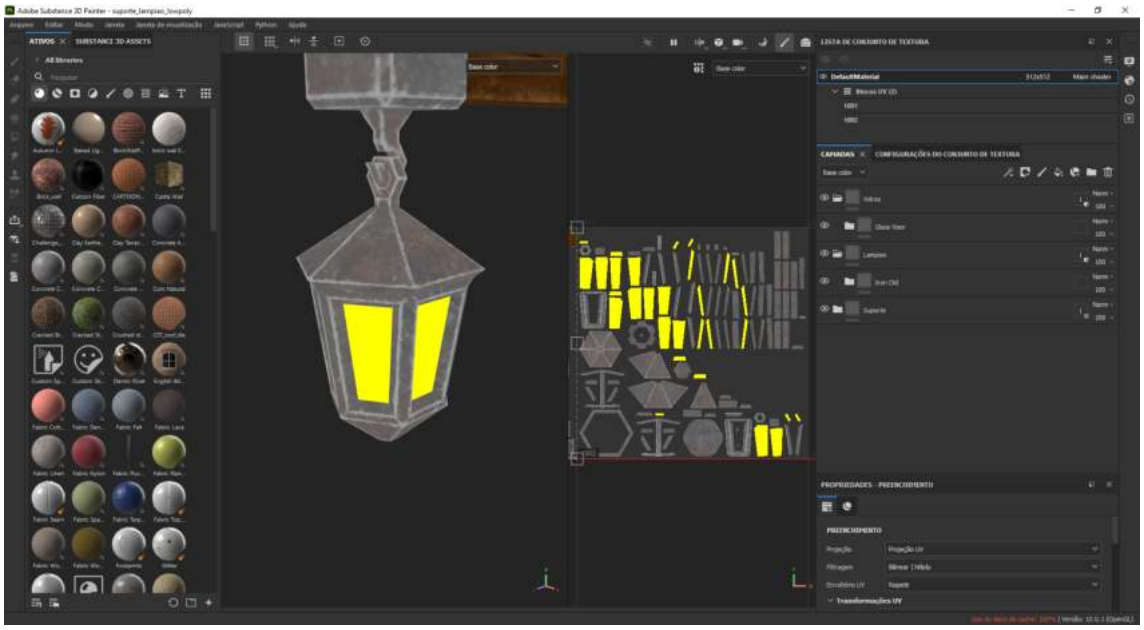

Figura 3. Exemplo de textura *Base Color* em um lampião 3D

Já a textura Normal é usada para simular pequenos detalhes e irregularidades na superfície, como ranhuras ou relevos, sem a necessidade de adicionar mais polígonos ao modelo. Isso é feito ao alterar a forma como a luz interage com a superfície, criando a ilusão de complexidade geométrica, exemplo a Figura 4.

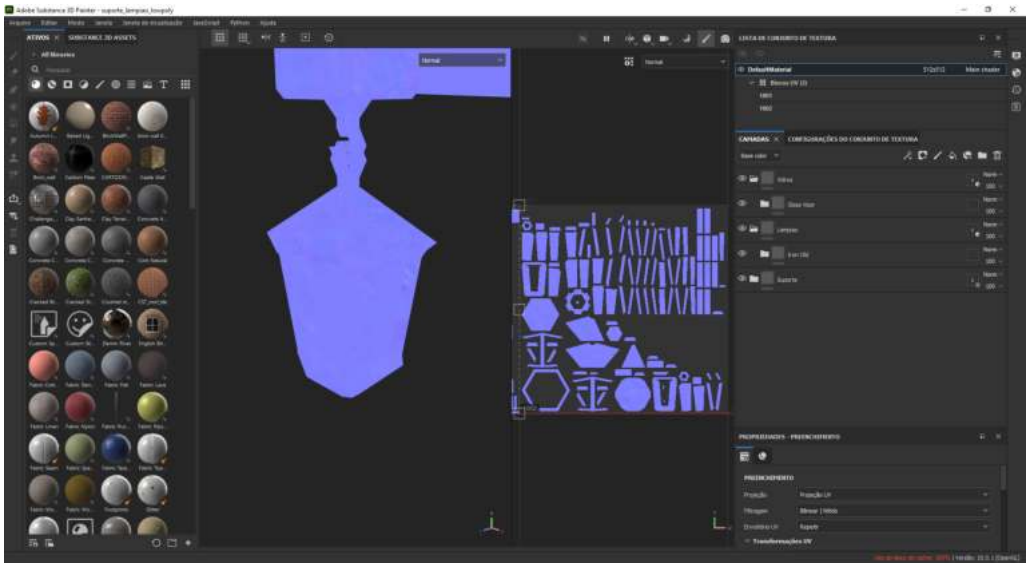

Figura 4. Exemplo de textura *Normal* em um lampião 3D

A *Ambient Occlusion* é uma textura que simula sombras suaves em áreas onde a luz natural tem dificuldade de alcançar, como cantos e reentrâncias, adicionando uma sensação de profundidade e realismo, como na Figura 5.

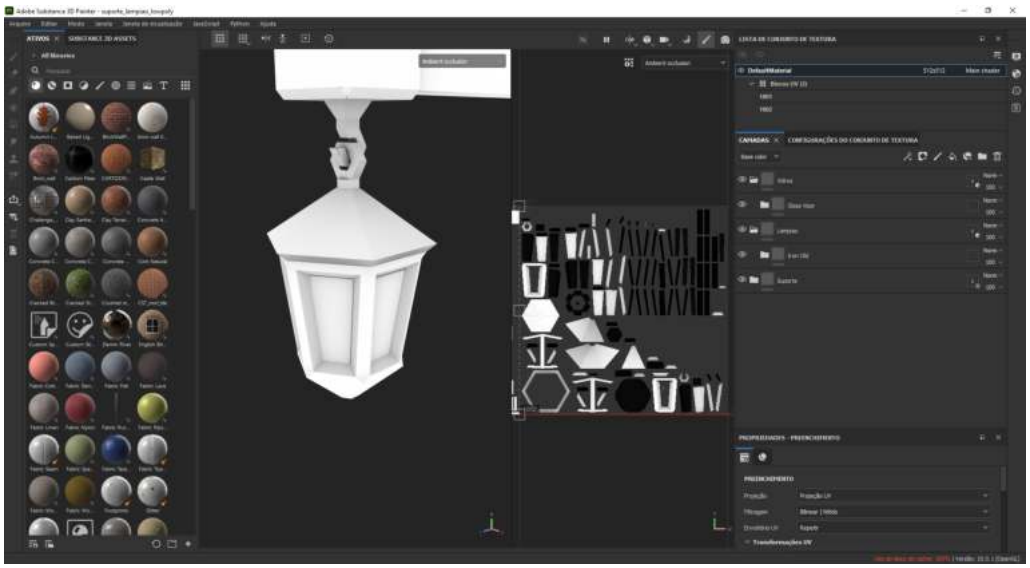

Figura 5. Exemplo de textura *Ambient Occlusion* em um lampião 3D

A textura *Metallic* indica quais partes da superfície devem se comportar como um metal, refletindo a luz de maneira especular, como por exemplo na Figura 6.

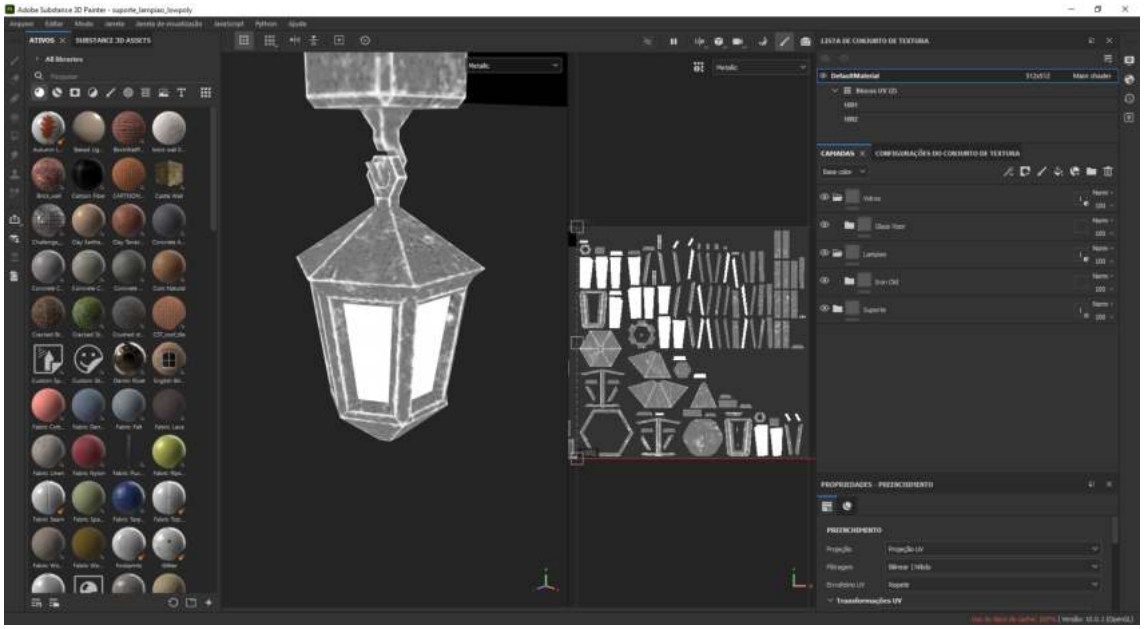

Figura 6. Exemplo de textura *Metallic* em um lampião 3D

Em contraste, a textura *Roughness* define o quão suave ou áspera é a superfície, afetando a difusão da luz e, consequentemente, a aparência do brilho; superfícies mais ásperas espalham a luz de maneira mais difusa, como na Figura 7.

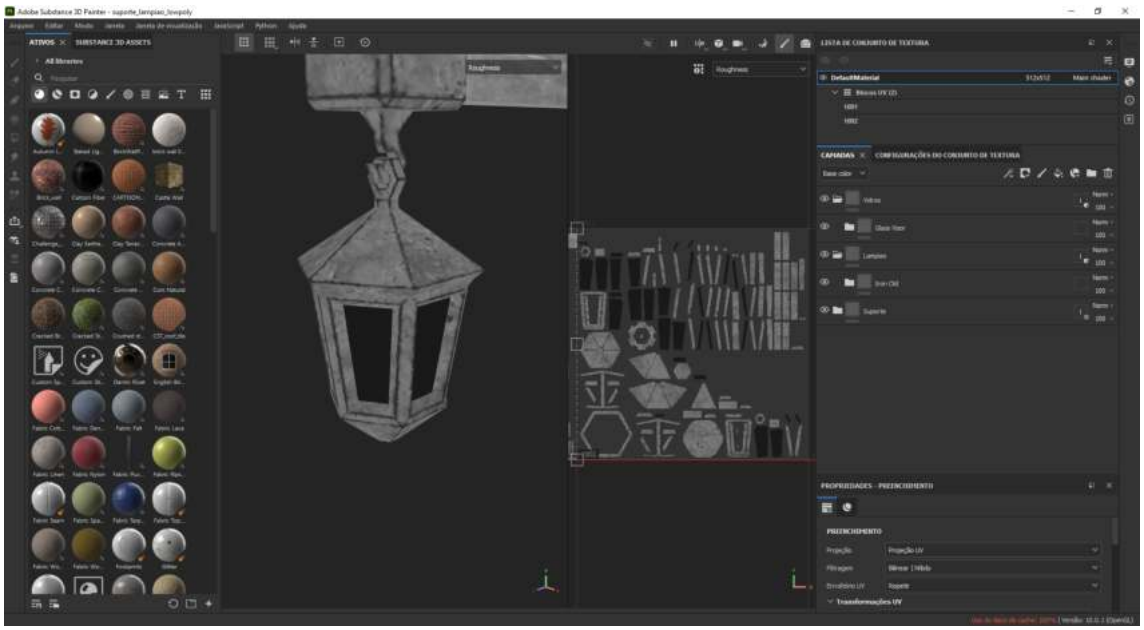

Figura 7. Exemplo de textura *Roughness* em um lampião 3D

Por fim, a textura *Emissive* é responsável por determinar quais partes do modelo emitem luz própria, criando o efeito de que certas áreas brilham independentemente da iluminação externa, como no exemplo da Figura 8.

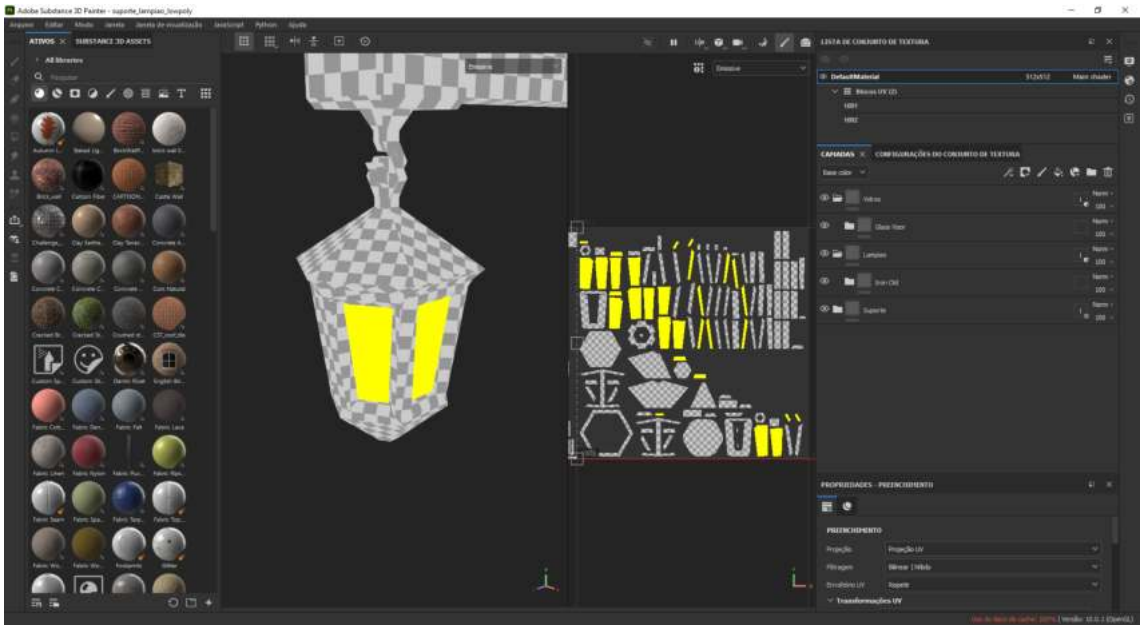

Figura 8. Exemplo de textura *Emissive* em um lampião 3D

Cada uma dessas texturas é essencial para criar materiais fisicamente realistas em renderizações 3D modernas, especialmente em motores que utilizam técnicas de PBR, onde a interação da luz com os materiais é simulada de forma precisa para gerar resultados visuais mais próximos da realidade. A junção das texturas pode ser observada na Figura 9.

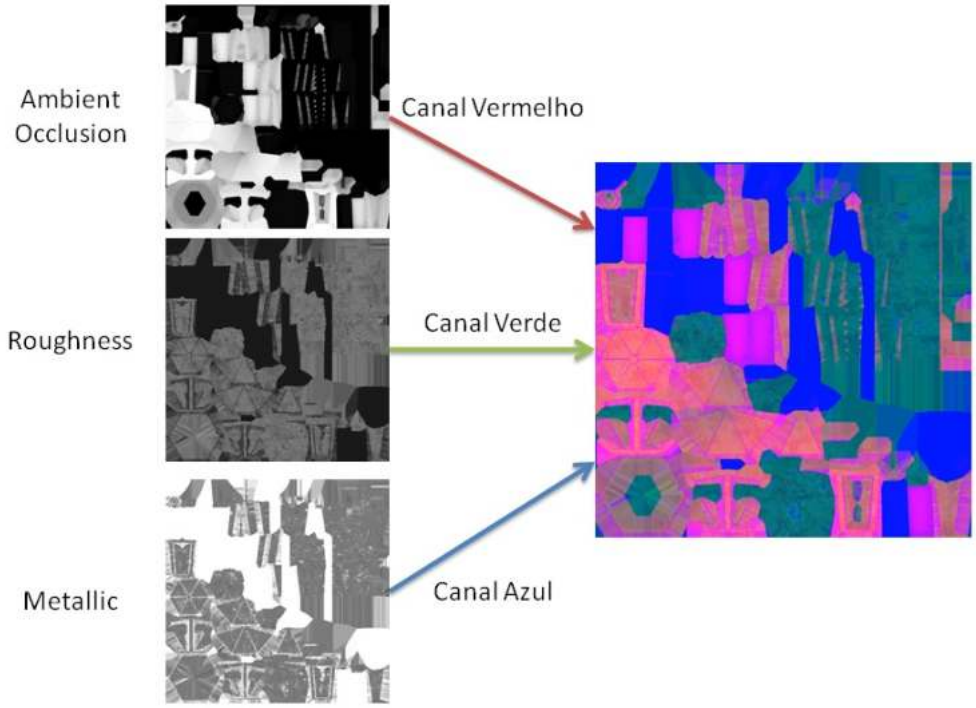

Figura 9. Exemplo da junção de diferentes texturas

3.3.4. Material

Segundo Pharr et al. (2023), na modelagem 3D, o conceito de material é fundamental para determinar como a superfície de um objeto interage com a luz e, consequentemente, como ele é visualizado em um ambiente virtual. Materiais não apenas definem a cor de um objeto, mas também suas propriedades físicas e ópticas, como brilho, rugosidade, transparência, refração, reflexão e *subsurface scattering*, que é o fenômeno de luz penetrando a superfície e sendo dispersa internamente antes de sair. Essas propriedades são cruciais para simular texturas realistas, como o brilho metálico de uma moeda, a opacidade de uma peça de madeira ou a translucidez da pele humana.

A criação de materiais em *software* de modelagem 3D é muitas vezes realizada através de um sistema de nós, onde diferentes parâmetros e texturas são combinados para criar o efeito desejado. Por exemplo, um material pode utilizar uma textura para definir sua cor base, outra para descrever sua rugosidade e uma terceira para os detalhes normais da superfície.

A especificação de materiais permite a criação de uma ampla gama de efeitos visuais que são essenciais para gerar imagens realistas, assim a manipulação precisa dessas propriedades pode afetar drasticamente o realismo de uma cena renderizada. Além disso, o uso de *shaders*, pequenos programas que determinam como cada pixel deve ser renderizado com base na luz e na posição da câmera, permite um controle ainda mais granular sobre o comportamento do material. Segue a Figura 10 com um exemplo de colocar um material em um modelo 3D seguindo o padrão glTF.

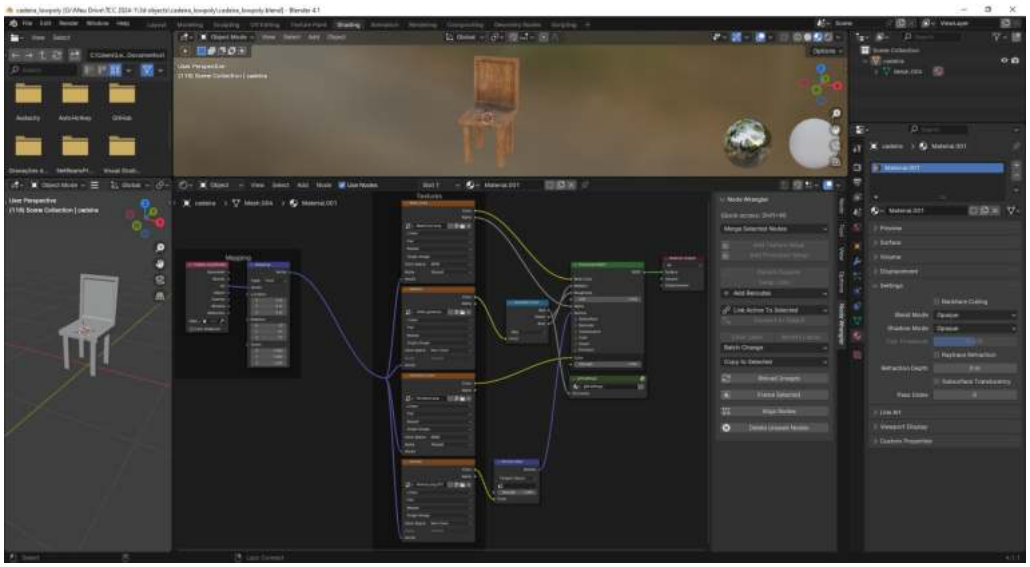

Figura 10. Exemplo de preparação de modelo 3D com material em *Blender*

3.3.5. *Shading*

Para Pharr et al. (2023), *Shading* é um processo crucial na computação gráfica que envolve a determinação da cor e da luminosidade de cada pixel de um objeto 3D, de acordo com a interação da luz com a superfície desse objeto. Este processo é realizado por meio de *shaders*, que são pequenos programas executados na Unidade de Processamento Gráfico (GPU) responsáveis por calcular como a luz é refletida ou absorvida, criando diferentes efeitos visuais.

Existem diferentes tipos de *shaders*, como os *vertex shaders* e *fragment shaders*, cada um atuando em etapas distintas do pipeline gráfico. *Vertex shaders* manipulam vértices, determinando sua posição final na tela e possibilitando transformações como escala e rotação. Já os *fragment shaders* determinam a cor final dos *pixels*, levando em consideração variáveis como a iluminação ambiente, texturas aplicadas e efeitos especiais, como sombras e reflexos.

Através de técnicas de *shading*, é possível reproduzir com alta precisão uma vasta gama de materiais, como superfícies metálicas, plásticos, vidros e até mesmo a pele humana, cada um exigindo um tratamento único para capturar suas propriedades ópticas específicas. Assim, o *shading* é essencial para dar vida às cenas 3D, proporcionando profundidade, detalhe e realismo às renderizações digitais.

3.4. Ferramentas Utilizadas

Nesta subsecção serão apresentadas as ferramentas que foram utilizadas para o desenvolvimento do jogo proposto nesse trabalho.

3.4.1. *Godot Engine*

Godot é um *game engine* de jogos 2D e 3D de uso geral projetado para dar suporte a qualquer tipo de projeto. Pode ser utilizado para criar jogos ou aplicativos que podem ser

lançados em computadores, celulares e também na internet. Para programar um jogo, podem ser utilizados duas linguagens de programação. *GDScript*, uma linguagem específica do *Godot*, ou *C#*, que é popular na indústria de jogos. Essas são as duas principais linguagens de *scripts* suportadas (Godot, 2024). Nesse trabalho, o *Godot* junto com *GDScript* foi utilizado para criar e programar o jogo.

3.4.2. *Blender*

De acordo com Foundation (2024), o *Blender* é um conjunto de *software* de criação 3D gratuito e de código aberto. Ele suporta todo o 3D *pipeline*, ou seja, modelagem, *rigging*, animação, simulação, renderização, composição e rastreamento de movimento, até mesmo edição de vídeo e criação de jogos. O *Blender* foi utilizado para criar os modelos 3D deste projeto.

3.4.3. Linguagem de programação *Lua*

A linguagem utilizada para ser utilizada dentro do jogo pelo jogador é a linguagem *Lua*. Para Lua (2024), *Lua* é uma linguagem de programação poderosa, eficiente e leve, projetada para estender aplicações. Ela permite programação procedural, programação orientada a objetos, programação funcional, programação orientada a dados e descrição de dados.

3.5. Trabalhos Relacionados

Nas subseções abaixo estão listados alguns trabalhos relacionados a este, que desenvolvem o ensino da programação por meio de jogos digitais.

3.5.1. *The Farmer Was Replaced*

The Farmer Was Replaced é um jogo, como reportado por Herzog (2023), onde o jogador precisa programar um drone para recolher recursos como grão, madeira e outros. A mecânica principal é de escrever o código em *python* e executá-lo para que o drone faça o que for pedido. O jogador é assistido por um manual onde ele pode ver o que cada função, bloco ou palavra chave da linguagem *python* significa e faz, como se vê na Figura 11, assim ajudando de maneira divertida quem é iniciante. Conforme o jogo avança, porém, o nível de complexidade evolui, virando um desafio também para os mais experientes.

Figura 11. Imagem do jogo *The Farmer Was Replaced* (Herzog, 2023)

3.5.2. *Bitburner*

Como reportado por Fulcrum-Games (2021), *Bitburner* é um jogo incremental que coloca o jogador no lugar de um *cowboy* terminal na *world wide web*, hackeando, roubando e criando *scripts* para chegar ao topo do mundo. O jogo Desafia o jogador a aprender como seus sistemas funcionam e automatizá-los para alcançar o objetivo. O *Bitburner* permite que o jogador use *JavaScript* dentro do próprio jogo, como pode-se ver na Figura 12. Um conhecimento básico de programação certamente ajuda você a começar, mas não é necessário. Conforme você avança, o jogo apresenta quebra-cabeças e perguntas de programação para ajudá-lo a aprender novas habilidades, ganhar recompensas e ficar mais poderoso. Explorando o mundo distópico do futuro, você conhece novas pessoas e encontra novas redes para se infiltrar e comprometer — ou facções para se aliar. Você também pode pegar novos aprimoramentos *cyberpunk*, impulsionando seu corpo e sua rede para níveis cada vez mais altos, bem como desenvolver as estatísticas do seu personagem. Há também minijogos legais, como negociação de ações, para descobrir e dominar. Este jogo ajuda não só a treinar pessoas que já tenham algum conhecimento na área de programação, mas também pode ajudar iniciantes a aprender de maneira mais divertida.

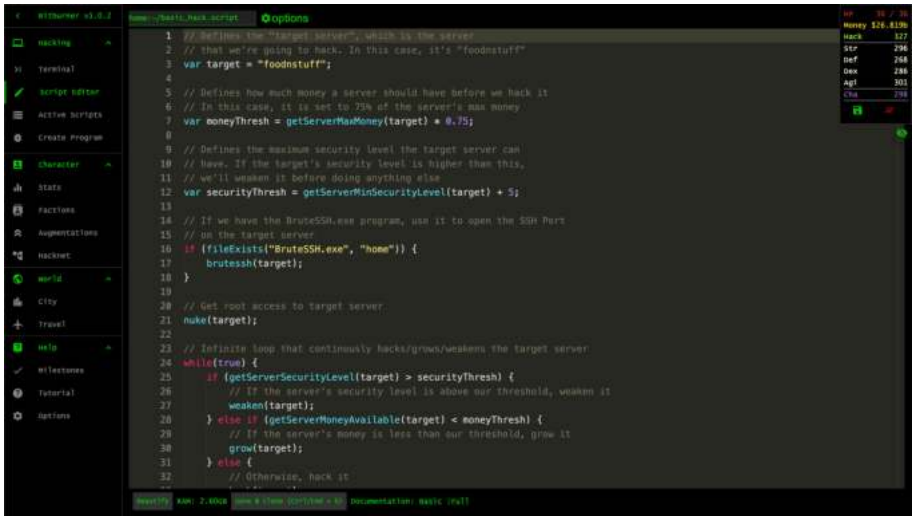

Figura 12. Imagem do jogo *Bitburner* (Fulcrum-Games, 2021)

3.5.3. *Coding Google Doodle*

O *Coding Google Doodle*, como reportado por Google Doodle Team (2017), é um jogo feito para ajudar as crianças a desenvolver a lógica de programação. Diferente de outros jogos do gênero, este não envolve escrever código em alguma linguagem específica, mas sim criar uma sequência de comandos para que o protagonista do jogo, nesse caso um coelho, consiga pegar todas as cenouras, como se vê na Figura 13. Portanto, este jogo é indicado para pessoas que ainda estão sem intimidade com a programação para que antes de escreverem linhas de código, comecem a melhorar a própria lógica de resolução de problemas.

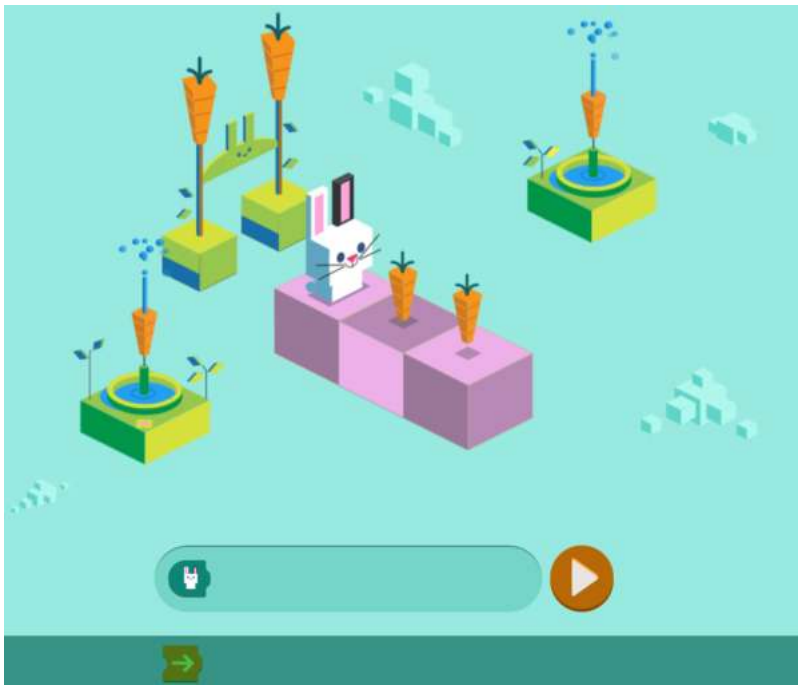

Figura 13. Imagem do *Coding Google Doodle*

3.5.4. *Else Heart.Break()*

Como apresentado por Svedäng (2015), *Else Heart.Break()* é um jogo de aventura – uma história fantástica ambientada em um mundo totalmente dinâmico e interativo. Em vez de quebra-cabeças rígidos, você pode aprender (com a ajuda de outros personagens no jogo) como a realidade do jogo pode ser alterada por meio da programação e como qualquer problema pode ser resolvido da maneira que você achar mais adequada, como mostra a Figura 14.

Figura 14. Imagem do jogo *Else Heart.Break()* (Svedäng, 2015)

No quadro 1 pode-se observar as diferenças entres os jogos citados nessa seção, no que diz respeito as plataformas onde podem ser jogados e aos conceitos abordados. Esses trabalhos foram muito importantes para entender as várias formas de usar gamificação, utilizando conceitos diversos de criação do jogo, como história e estilo artístico.

Titulo	Plataforma	Conceitos
The farmer Was Replaced	Desktop	básicos e intermediários
Bitburner	Desktop	do básico ao avançado
Coding Google Doodle	WEB	logica de programação
Else Heart.Break()	Desktop	do básico ao avançado
Resquício	Desktop	básicos e intermediários

Quadro 1. Diferenças entre jogos

4. Desenvolvimento

Nesta seção será apresentado o que foi desenvolvido utilizando os conceitos mostrados nas seções anteriores.

4.1. GDD

Segundo Motta e Junior (2013), o documento de design de jogo, ou GDD (*game design document*) é uma ferramenta textual produzida por um game designer que descreve todas

as características de um jogo, desde informações básicas de premissa, conceitos, passando por personagens e cenários, informações mais detalhadas como projeto de *levels* e até sons.

A utilização do GDD foi essencial para centralizar as informações do jogo em um só documento, fazendo com que todos os envolvidos no projeto estivessem a par do que estava sendo desenvolvido. As subseções a seguir abordam as especificações que foram definidas no GDD para o jogo.

4.1.1. Visão Geral

O jogo se passa em um mundo fictício, onde a sociedade humana se reconstrói após uma guerra nuclear destruir o planeta terra como conhecemos.

Após se formar como Técnico de Reparo na prestigiosa Grande Academia, você é enviado pelo Reino a uma vila remota no extremo norte, onde a recepção aos forasteiros é fria e o descontentamento com o governo é notável. Armado com seu Aparelho de Conserto, você precisará resolver problemas mecânicos da era pré-guerra e ganhar a confiança dos moradores da vila. Guiado pela esposa do padeiro, uma das poucas vozes amigáveis na vila, você rapidamente descobre que uma rebelião contra o Reino está ganhando força. Suas escolhas e alianças moldarão seu destino: permanecer fiel ao Reino ou se unir aos rebeldes na luta pela liberdade. O objetivo do jogador é ajudar os aldeões, resolvendo exercícios de programação para concluir as missões.

4.1.2. Requisitos do jogo digital

Foram definidos alguns requisitos mínimos para serem implementados no jogo, garantindo uma jogabilidade confortável para o usuário:

- Movimentação do jogador: Permitir que o jogador se movimente livremente pelo ambiente do jogo;
- Janela de diálogo: Uma interface que mostre os diálogos com os NPCs de maneira clara e visual;
- Interface do editor de código: Uma interface que permita que o usuário escreva e edite os códigos dentro do jogo;
- Interação com NPC: Uma mecânica para o usuário interagir com os NPCs ao pressionar um botão;
- Interação com objeto: Uma mecânica para o usuário interagir com os objetos ao pressionar um botão;
- Controle das *quest*: Implementação de uma lógica que acompanhe o progresso do jogador dentro do jogo.
- Cenário e ambientação de jogo: Modelagem e elaboração do cenário em que o jogo se passa;
- História: Desenvolvimento da história e enredo do jogo;
- Cinemática inicial: Um vídeo curto de introdução ao jogo, mostrando o cenário e contextualizando a história;
- Menu: Uma tela que permita o jogador ver seu progresso no jogo e consultar os controles básicos.

4.1.3. Enredo

Para consertar os objetos da era pré-guerra, o Reino disponibiliza um técnico graduado da Grande Academia que é enviado para prestar serviços nas cidades e vilarejos. A Grande Academia é uma instituição fundada depois da guerra para formar diversos tipos de trabalhadores do mais alto nível, situada na capital do reino *Bjorgoren*. O protagonista estudou e se formou em um dos cursos da Grande Academia, responsável por formar os Técnicos de Reparo.

Entretanto, todos os técnicos que se formarem são distribuídos de maneira aleatória em diferentes pontos do Reino. O protagonista por seu azar (ou sorte), foi mandado para uma vila no extremo norte, que por ser bastante remota, tem várias dificuldades internas. Como todo bom técnico, o protagonista tem seu próprio aparelho de conserto, uma espécie de bracelete que usará para consertar os objetos da pré-guerra. Chegando na vila ele logo percebe que seus habitantes não são muito receptivos com estrangeiros e que para ganhar sua confiança terá que ajudá-los em uma série de desafios.

O protagonista descobre também que alguns habitantes têm ódio do governo do Reino, com uma rebelião emergindo na vila. Como o protagonista é um funcionário do Reino, ele será alvo de várias ameaças e conflitos. Ele descobre algumas verdades e terá que tomar decisões, se fica fiel ao Reino ou se junta à Rebelião. Após adquirir a confiança dos aldeões e descobrir mais sobre suas histórias, o protagonista decide se juntar à rebelião. O jogo se encerra após um diálogo com o líder da rebelião, onde o protagonista é aceito no grupo rebelde.

4.1.4. Exercícios de programação

Para avançar no jogo, o jogador precisa resolver exercícios de programação. Os exercícios foram baseados no projeto de Neto (2023) e contemplam as partes mais básicas da escrita de código, podendo se estender para conceitos mais avançados.

O jogo conta com três exercícios de programação básica. No primeiro, o jogador deve completar um trecho de código para que dois números sejam somados. O segundo exercício exige que o jogador verifique se um número é positivo, negativo ou zero, utilizando uma estrutura de seleção. Já no terceiro exercício, o jogador precisa fazer uma média ponderada de três números com pesos destintos. A Figura 15 mostra o primeiro exercício que precisa ser resolvido pelo jogador.

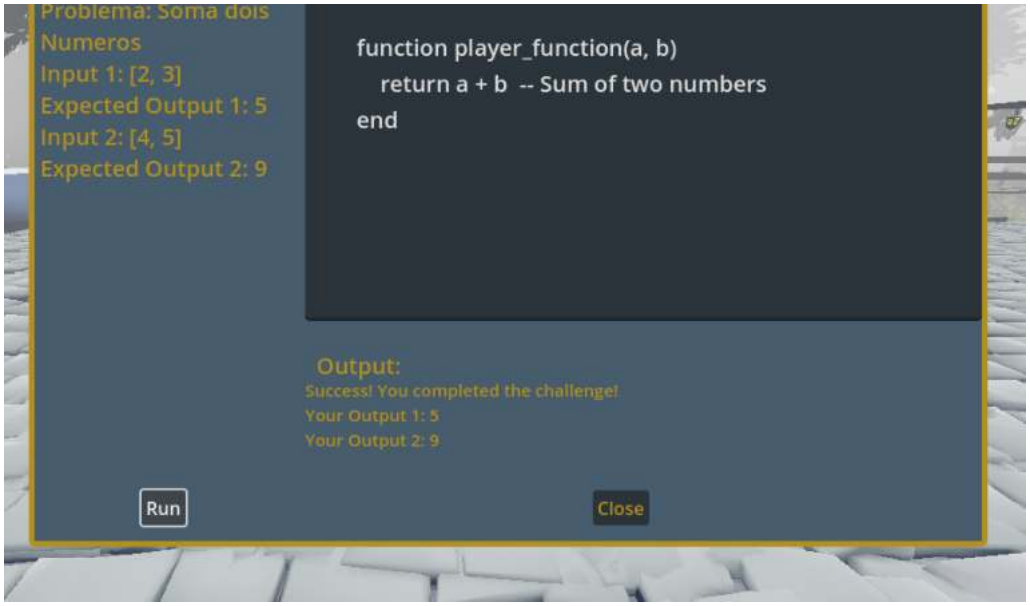

Figura 15. Exercício de soma de dois números

4.1.5. Personagens

Os personagens do jogo são responsáveis por preencher o cenário e passar os objetivos para o protagonista. Com a exceção do protagonista, os demais personagens são NPCs como na Figura 16. O quadro 2 descreve o papel de cada um dos personagens.

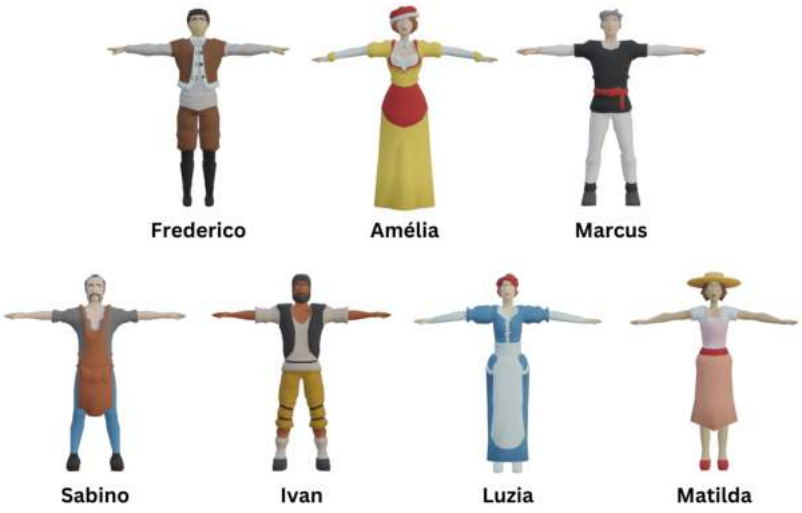

Figura 16. NPCs que habitam a vila

Nome	Papel
Augusto	Protagonista do jogo.
Frederico	Pessoa de influência na vila, líder da revolução.
Amélia	Recepciona o protagonista na vila. Esposa do padeiro.
Marcus	Padeiro da vila.
Sabino	Ferreiro da vila.
Ivan	Pescador, passa o dia pescando. De noite ajuda sua esposa na estalagem da vila.
Luzia	Esposa do pescador, administra a estalagem da vila.
Matilda	Passa o dia pescando junto com seu irmão Ivan. Vende peixes no centro da vila.

Quadro 2. Personagens do jogo

4.1.6. Ambientação

O jogo se passa em uma vila nevada com algumas casas. Alguns aldeões ficam nas ruas, cuidando de seus afazeres. As construções têm um estilo germânico e rodeiam o centro da vila. Maquinários e equipamentos dos tempos pré-guerra reforçam o cenário de abandono e dificuldade. Alguns lampiões iluminam a vila, destacando o isolamento e o abandono por parte do Reino. A Figura 17 mostra parte da vila.

Figura 17. Exemplo da ambientação do jogo

4.1.7. Estilo de câmera

A câmera em terceira pessoa é amplamente utilizada em jogos para proporcionar ao jogador uma visão externa de seu personagem, permitindo uma percepção completa do ambiente e dos arredores. Esse estilo de câmera, geralmente posicionado atrás ou ligeiramente acima do personagem controlado, oferece uma visão mais ampla do cenário, facilitando a navegação e a interação com o espaço tridimensional.

Segundo Tavinor (2009), a perspectiva em terceira pessoa permite ao jogador observar o ambiente e prever ações com mais clareza, aprimorando a interação entre jogador e ambiente virtual. Esse tipo de câmera é especialmente vantajoso em jogos de ação e aventura, nos quais o jogador precisa de uma visão que permita avaliar tanto o movimento do personagem quanto dos inimigos e obstáculos ao redor, melhorando a imersão e o controle espacial, sem comprometer a percepção do espaço ao redor, como na Figura 18. A perspectiva em terceira pessoa foi julgada a mais adequada pelo jogo conter um foco grande na ambientação e interação com objetos que estão espalhados pelo cenário.

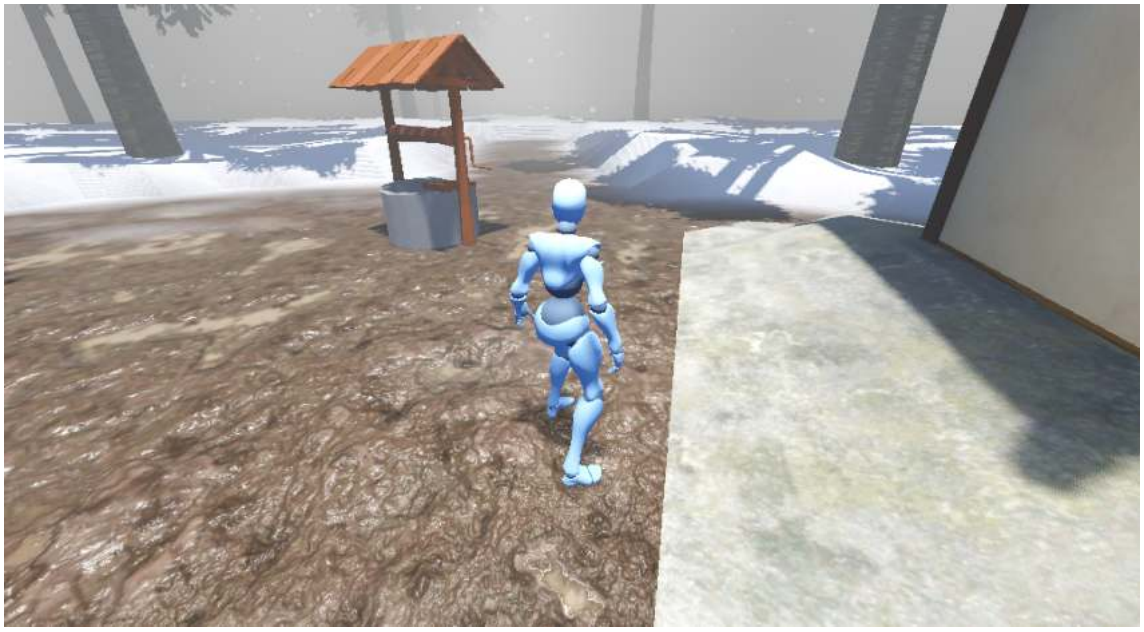

Figura 18. Exemplo de câmera em terceira pessoa

4.1.8. Jogabilidade

A jogabilidade permite ao jogador movimentar o personagem principal com as teclas W, A, S e D, realizar saltos com a barra de espaço e interagir com NPCs e objetos utilizando a tecla F. A câmera é controlada pela movimentação do mouse, oferecendo uma visão dinâmica do ambiente, enquanto a navegação nos menus e interações são realizadas com o botão esquerdo do mouse. Durante as missões que envolvem codificação, o jogador utiliza o teclado para digitar os códigos necessários, proporcionando uma experiência intuitiva e fluida que combina exploração, interação e resolução de desafios programáticos.

4.1.9. Mecânicas

No jogo, o progresso do jogador depende da resolução de códigos em linguagem Lua. Ao interagir com determinados objetos, o jogador acessa um editor de código que apresenta o problema, os *inputs* a serem testados e os *outputs* esperados, incentivando a compreensão e solução do desafio de forma estruturada. Esse editor integra um interpretador de linguagem Lua desenvolvido especialmente para projetos na *Godot Engine*, permitindo

que o código seja executado diretamente no ambiente do jogo. Dessa forma, o jogador não apenas resolve enigmas, mas também aprende e pratica conceitos fundamentais de programação, como estruturas de controle e manipulação de dados, enquanto interage e avança na narrativa do jogo.

4.1.10. Arte e design visual

O estilo escolhido para o jogo foi *Cartoon-Low poly*, um híbrido entre dois estilos comumente usados em jogos digitais 3D. Ambos os estilos influenciam na modelagem dos objetos contidos no jogo, que não buscam um nível alto de fotorrealismo.

O Low Poly é um estilo que consiste em diminuir o número de polígonos ou faces de um objeto 3D visando alcançar um visual mais leve, sem perda de identidade, como ilustrado na Figura 19. É utilizado não só para otimização de jogos digitais, mas como escolha artística.

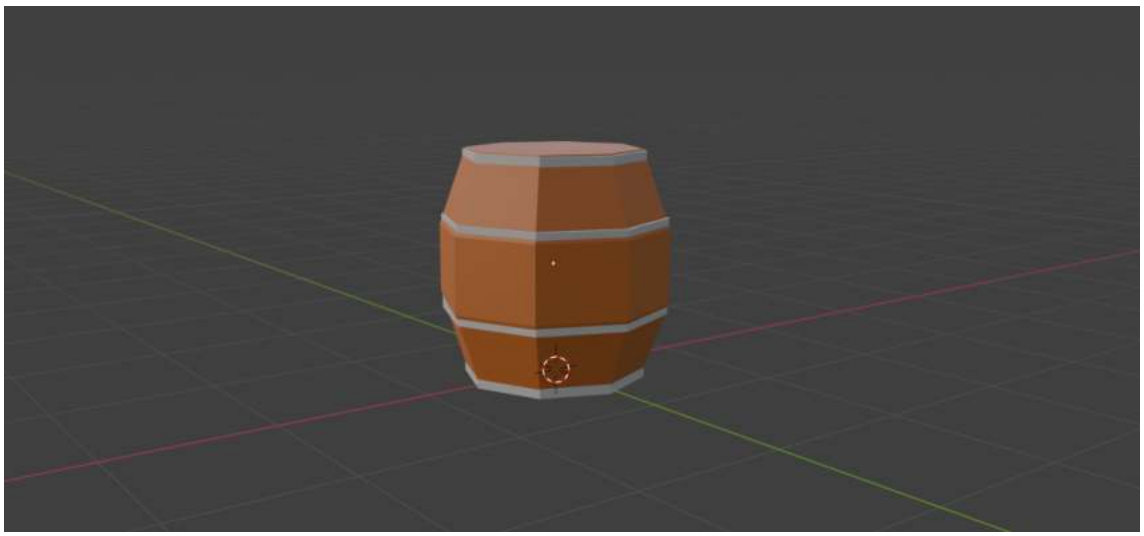

Figura 19. Exemplo de um objeto Low Poly

Já o estilo cartoon é derivado das histórias em quadrinhos e desenhos animados. Tem como principais características a pintura com cores sólidas e o contorno preto das extremidades Pinheiro (2023).

4.1.11. Interface do usuário

A tela de interação com NPC é ativada quando o jogador se aproxima de um NPC e utiliza a tecla de interação. A proximidade é detectada por um elemento do tipo área 3D ao redor do personagem, que identifica se o objeto próximo possui um método de interação. No caso dos NPCs, essa interação aciona um elemento *Control* que gerencia uma tela de diálogo, permitindo ao jogador se comunicar e receber informações relevantes para a progressão no jogo. Esse sistema garante que as interações ocorram de forma natural e intuitiva, melhorando a imersão e facilitando o acesso às informações fornecidas pelos

NPCs, como na Figura 20. A mesma logica é utilizada para a tela do editor de código, como pode-se ver na Figura 21.

Figura 20. Exemplo de interface de dialogo com NPC

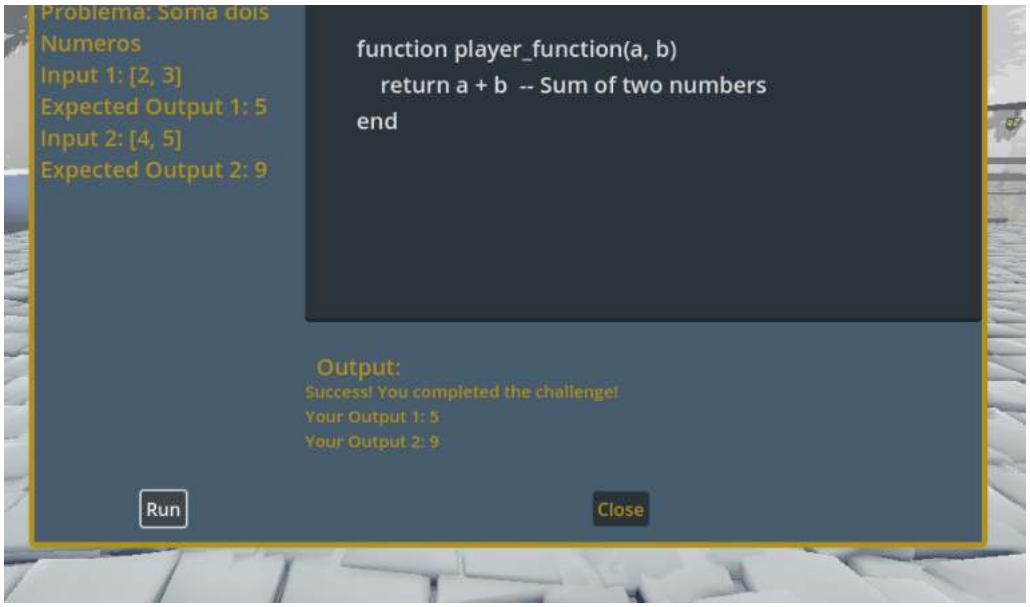

Figura 21. Exemplo do editor de código

4.2. Modelagem e Adaptação de Assets

No desenvolvimento deste projeto, foi necessário criar, animar e texturizar diversos modelos 3D. Alguns desses modelos foram elaborados especificamente para atender às necessidades do jogo, enquanto outros foram obtidos de repositórios online e posteriormente adaptados para o projeto, em um processo conhecido como flipagem de assets. Essa adaptação envolveu a animação de modelos de personagens e a pintura dos modelos seguindo uma paleta de cores definida. Um total de 210 modelos 3D foram preparados.

A paleta de cores utilizada foi desenvolvida exclusivamente para este trabalho, de modo a garantir coesão estética entre os elementos visuais. Essa paleta pode ser visualizada na Figura 22.

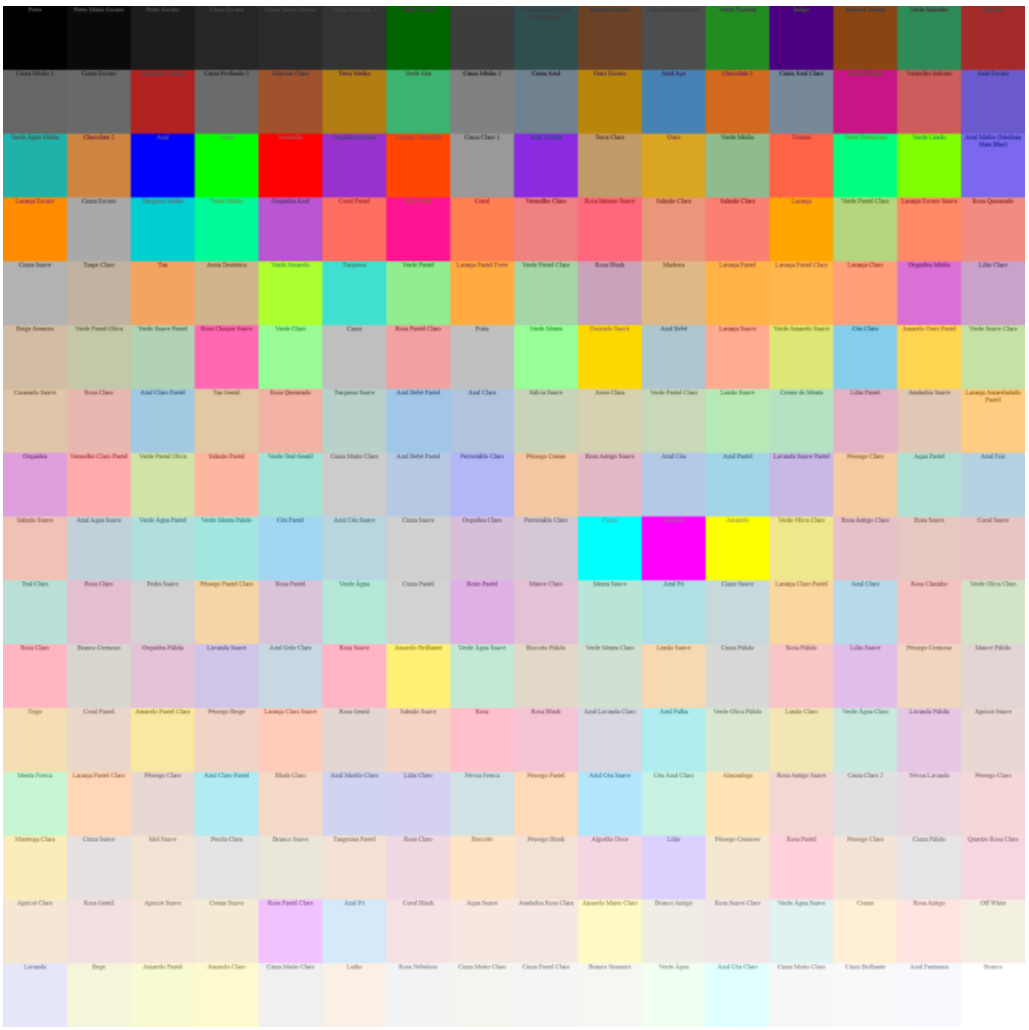

Figura 22. Paleta de cores (Fonte: Estácio, L. B. 2024)

4.2.1. Assets Criados para o Projeto

Os seguintes modelos foram desenvolvidos especificamente para o jogo:

- Personagens: boy_6; boy_12; girl_6; girl_12.
- Cenário e Objetos: arvore_pinus (14 variações); arvores_araucaria (14 variações); cadeira_lowpoly; casa1; casa2; casa3; casa4; colher; faca; fornalha; garfo; lampiao_lowpoly; notebook; poco_lowpoly; prato_com_bolo; terreno_vila; xícara.

4.2.2. Assets Adaptados de Repositórios Online

Para complementar o ambiente do jogo, foram utilizados diversos modelos 3D de fontes externas. Esses assets foram devidamente creditados e adaptados para harmonizar com o estilo artístico do projeto. As fontes e os respectivos assets estão listados a seguir:

Craftpix (Craftpix, 2024), incluem-se, entre outros:

- Bag_1;
- Bag_2;
- Bench;
- Fense_1;
- Gate_1;
- Lamp_1;
- Road_breek_1;
- Axe.

Itch.io - Ilay15 (ilay15, 2024), incluem-se:

- Book;
- Bucket1;
- Bulova;
- Cart;
- Hammer3.

Itch.io - Quaternius (Quaternius, 2024), incluem-se:

- Arch;
- Arch_bars;
- Arch_Door;
- Arch_Door_bottompivot;
- Bag_Coins.

Low Poly Assets e *Sjolle* (LowPolyAssets, 2024) e (sjolle, 2024), incluem-se:

- barrel01;
- bed_double;
- bed_single;
- bed01;
- bedmattress01.

Com essa abordagem, buscou-se diversificar os elementos do jogo, aproveitando recursos existentes e otimizando o processo de desenvolvimento. Essa combinação de criação autoral e reutilização de modelos externos permitiu atingir um resultado visual mais coeso e eficiente.

4.3. Implementação

Esta subseção abordará mais a fundo alguns conceitos e ferramentas essenciais que foram implementados no desenvolvimento do projeto.

4.3.1. Arquitetura de *software*

A arquitetura de *software* do projeto foi definida antes do início da implementação, como observada na Figura 23. Nela, o modelo do protagonista tem seu *script* em *Player.gd*, que contém as lógicas de movimentação e interação. Esse *script* permite as interações com NPCs e objetos, que por sua vez também têm seus *scripts* contendo as lógicas das *quests* e controle de diálogos. O editor de código (*CodeEditor.gd*) também está vinculado ao interpretador Lua.

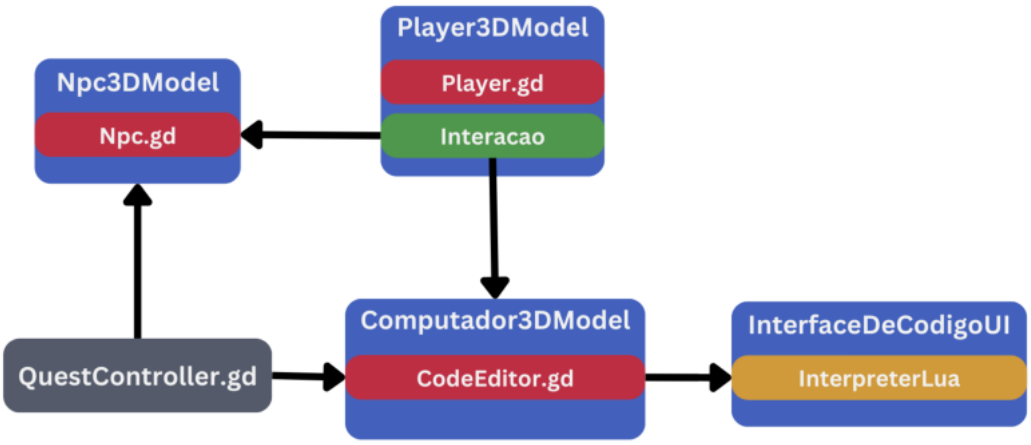

Figura 23. Ilustração da arquitetura de software utilizada

4.3.2. Programação em GDScript

Em *Godot Engine*, para colocar elementos na cena de jogo, são utilizados *Nodes*. Como descrito por da Silva e Yepes (2018), *Nodes* são blocos de construção fundamentais para criar um jogo. Um *Node* pode executar uma variedade de funções especializadas. No entanto, qualquer *Node* fornecido sempre possui os seguintes atributos:

- Tem um nome;
- Tem propriedades editáveis;
- Pode receber um retorno de chamada para processar cada quadro;
- Pode ser estendido (para ter mais funções).
- Pode ser adicionado a outros nodes como filho.

Quando organizados dessa maneira, os nodes se tornam uma árvore. Esses *Nodes* podem ser associados a um *script* que os permite ter funcionalidades a mais, como movimentação, interação e mudança de estado.

4.3.3. Interpretador Lua

Para permitir que o jogador execute linhas de código durante o jogo, foi utilizado um interpretador da linguagem Lua. Este interpretador é uma biblioteca que pode ser baixada dentro do próprio *Godot Engine* e usada gratuitamente para qualquer projeto. A Figura 24 mostra um trecho de código com uma função que verifica quando o botão *run* é clicado e faz com que o código escrito pelo jogador seja interpretado e executado, validando assim a resposta fornecida para o exercício.

```
func _on_run_pressed():
    var lua_code = $TextEdit.text
    # 1. Create a Lua state
    var lua_state = LuaState.new()
    # 2. Import Lua and Godot APIs into the state (open standard libraries)
    lua_state.open_libraries()
    # 3. Wrap the player's code inside a globally declared Lua function
    # Extract the player's code from the wrapped version if needed
    var start_pos = lua_code.find("function player_function(a, b)") + "function player_function(a, b)\n".length()
    var end_pos = lua_code.rfind("end")
    var player_code = lua_code.substr(start_pos, end_pos - start_pos).strip_edges()

    # 3. Update the wrapped code
    _update_wrapped_code(player_code)
    # 4. Run the Lua code using `LuaState.do_string()` to define the function
    var result = lua_state.do_string($TextEdit.text)

    # 5. Check if there was a Lua error
    if result is LuaError:
        $Output.text = "Error in Lua code: " + str(result)  # Display Lua error and exit
        print("Lua Error: ", result)
        return
    # 6. Access the global _G table via `LuaState.globals`
    var globals = lua_state.globals
    # Check if the 'player_function' exists in globals
```

Figura 24. Trecho do código do interpretador Lua

4.4. Resultados e discussão

Após as implementações, o resultado foi uma versão demonstrativa do jogo, com início, meio e fim. Ao iniciar o jogo, o jogador é apresentado ao cenário por uma série de textos e imagens aéreas da vila, ilustrado pela Figura 25.

Figura 25. Introdução ao jogo

Após a sequência introdutória, o jogador toma controle do personagem principal, podendo se movimentar livremente e explorar a vila a procura de aldeões que necessitem de ajuda do Técnico de Reparos. A Figura 26 mostra um diálogo entre o NPC e o jogador, onde o NPC solicita o reparo do poço da vila.

Figura 26. NPC pedindo ajuda do protagonista

O jogador precisa então ir até o poço da vila e interagir com o objeto. Ao fazer isso, a janela de edição de código é aberta, onde o jogador consegue resolver o desafio utilizando a linguagem Lua. A Figura 27 exemplifica a resolução do desafio.

Figura 27. Jogador resolvendo o desafio

Com o exercício resolvido, o jogador recebe uma mensagem de secesso e volta para falar com o NPC que solicitou ajuda. O NPC agradece o jogador e a *quest* é encerrada ao fechar a janela de diálogo, como mostra a Figura 28.

Figura 28. NPC agradecendo a ajuda do protagonista

Ao finalizar as três *quests* existentes na vila, o jogador é apresentado com mais uma imagem aérea do cenário e uma mensagem final de agradecimento, mostrada na Figura 29. Após fechar a mensagem, o jogo é encerrado.

Figura 29. Mensagem final do jogo

5. Conclusão

O objetivo deste trabalho foi desenvolver um jogo digital no estilo RPG para gamificar o ensino da programação básica, tentando assim reduzir a desistência das matérias que abordam esse conteúdo na graduação.

O grupo pesquisou sobre jogos que ensinem algum conteúdo, o uso deles na educação, jogos do tipo RPG e outras ferramentas que se alinharam com a proposta do trabalho. Foi elaborado também o *Game Design Document* (GDD) com a história do jogo, escolhas de design, exercícios de programação a serem resolvidos, entre outros detalhes.

Um dos maiores obstáculos da implementação foi a interação do jogador com os objetos do mundo. Inicialmente os nodos do objeto e jogador não se identificavam para que a interação acontecesse, então foi preciso criar uma lógica de área. Tanto o objeto quanto o jogador têm uma área invisível ao seu redor, permitindo que seja identificado quando o jogador se aproxima do objeto. Ao entrar na área do objeto são gerados *triggers* que mostram se esse objeto possui interação ou não.

Para garantir a efetividade do jogo desenvolvido, é necessário implementá-lo em uma turma de introdução à programação e analisar seu impacto no engajamento e aprendizado dos alunos. A partir dessa aplicação prática, será possível coletar *feedbacks*, avaliar os resultados obtidos e identificar possíveis melhorias na estrutura do jogo. Esse processo permitirá ajustes e refinamentos, tornando a ferramenta mais eficiente na aplicação dos conceitos de gamificação no ensino de programação.

Embora o trabalho tenha resultado em um protótipo jogável, que pode ser expandido futuramente, o desenvolvimento de um jogo completo demanda um tempo muito

maior do que o disponível para este projeto, considerando o período de estudo e desenvolvimento. As melhorias futuras do trabalho incluem a adição de mais desafios de programação, podendo chegar a um nível de dificuldade intermediária, a implementação de todos os requisitos citados no GDD e a implantação do jogo em uma turma do curso de ciência da computação.

Referências

Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Iwanicki, M., e Hillaire, S. (2018). Real-time rendering 4th edition.

Campbell, J. (2008). *The hero with a thousand faces*, volume 17. New World Library.

Craftpix (2024). 2D Game Assets Store & Free - CraftPix.net — craftpix.net. `https://craftpix.net/`. [Acessado 26-08-2024].

da Silva, V. P. e Yepes, I. (2018). Desenvolvimento de jogos na plataforma godot. *Anais do Encontro Anual de Tecnologia da Informação*, 8(1):102–102.

Foundation, B. (2024). About — blender.org — blender.org. `https://www.blender.org/about/`. [Acessado 14-05-2024].

Fulcrum-Games (2021). Bitburner on Steam — store.steampowered.com. `https://store.steampowered.com/app/1812820/Bitburner/`. [Acessado 25-08-2024].

Giardinetto, J. R. B. e Mariani, J. M. (2005). Jogos, brinquedos e brincadeiras: O processo ensino-aprendizagem da matemática na educação infantil. *MATEMÁTICA E EDUCAÇÃO INFANTIL*.

Godot (2024). Introduction — docs.godotengine.org. `https://docs.godotengine.org/en/stable/about/introduction.html`. [Acessado 14-05-2024].

Google Doodle Team, Google Blockly team, M. S. (2017). Celebrating 50 years of Kids Coding Doodle - Google Doodles — doodles.google. `https://doodles.google/doodle/celebrating-50-years-of-kids-coding/`. [Acessado 26-08-2024].

Group, K. (2021). glTF - Runtime 3D Asset Delivery — khronos.org. `https://www.khronos.org/gltf/`. [Acessado 24-08-2024].

Herzog, T. (2023). The Farmer Was Replaced on Steam — store.steampowered.com. `https://store.steampowered.com/app/2060160/The_Farmer_Was_Replaced/`. [Acessado 25-08-2024].

ilay15 (2024). ilay15 - itch.io — ilay15.itch.io. `https://ilay15.itch.io/`. [Acessado 08-12-2024].

LowPolyAssets (2024). LowPolyAssets - itch.io — lowpolyassets.itch.io. `https://lowpolyassets.itch.io/`. [Acessado 08-12-2024].

Lua (2024). The Programming Language Lua — lua.org. `https://www.lua.org/about.html`. [Acessado 14-05-2024].

Maryono, D., Budiyono, S., e Akhyar, M. (2022). Implementation of gamification in programming learning: literature review. *Int. J. Inf. Educ. Technol*, 12(12):1448–1457.

Motta, R. L. e Junior, J. T. (2013). Short game design document (sgdd). *Proceedings of SBGames*, 2013:115–121.

Murr, C. E. e FERRARI, G. (2020). Entendendo e aplicando a gamificação: o que é, para que serve, potencialidades e desafios. *UFSC. E-BOOK*.

Neto, W. C. B. (2023). Dinamizando a aprendizagem de programaÇÃo de computadores usando desenvolvimento orientado a testes.

Pharr, M., Jakob, W., e Humphreys, G. (2023). *Physically based rendering: From theory to implementation*. MIT Press.

Pinheiro, M. J. O. (2023). Desafios na utilização de técnicas de renderização não fotorrealistas com transferência de estilo com características do cartoon: uma revisão sistemática da literatura.

Quaternius (2024). Quaternius - itch.io — quaternius.itch.io. `https://quaternius.itch.io/`. [Acessado 08-12-2024].

sjolle (2024). sjolle - itch.io — sjolle.itch.io. `https://sjolle.itch.io/`. [Acessado 08-12-2024].

Svedäng, E. (2015). Else Heart.Break() on Steam — store.steampowered.com. `https://store.steampowered.com/app/400110/Else_HeartBreak/`. [Acessado 26-08-2024].

Tavinor, G. (2009). *The art of videogames*. John Wiley & Sons.

Zhao, D., Muntean, C. H., Chis, A. E., Rozinaj, G., e Muntean, G.-M. (2022). Game-based learning: enhancing student experience, knowledge gain, and usability in higher education programming courses. *IEEE Transactions on Education*, 65(4):502–513.