INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO – CAMPUS CAMPOS DO JORDÃO

BEATRIZ HELENA E SILVA PINTO

RELATÓRIO – JOGO CONSTRUÍDO COM PROGRAMAÇÃO ORIENTADA A OBJETOS

CAMPOS DO JORDÃO

2025

LISTA DE FIGURAS

LISTA DE SIGLAS

IFSP Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

GDD Game Design Document

CPP C ++ = Linguagem de programação

HPP Arquivo de interface em C++

POO Programação Orientada a Objetos

SUMÁRIO

1 INTRODUÇÃO

De acordo com Gillis (2024), a programação orientada a objetos é um paradigma centrado nos objetos, sendo que esse possui características e comportamentos próprios. Kostin (2024), explana as aplicações na qual a programação orientada a objetos pode ser usada, dentre elas, desenvolvimento web, desenvolvimento de jogos, múltiplos softwares com diferentes funcionalidades, dentre outros. Ainda, Herrity (2025) destaca as estruturas desse paradigma de programação: Classes; Atributos; Métodos; e Objetos. As classes funcionam como “blueprints” dos objetos, os modelos que demonstram como é aquela classe, mas não específica mais (Gillis, 2024). O objeto é a instância da classe, enquanto os atributos são suas características e os métodos são as coisas que o objeto pode realizar. Como exemplo, pode-se pensar num Gato, sua cor, raça, idade, nome, tamanho, são todos atributos, mas ronronar, miar, arranhar, dormir e comer, são os métodos do gato, enquanto o próprio gato é um objeto, de uma classe maior, chamada “Felinos”.

1.1 Objetivos

Desenvolver um jogo utilizando a combinação de programação orientada a objetos, a Linguagem C++ e a biblioteca – em C - Raylib.

1.2 Justificativa

Como parte dos estudos de Programação Orientada a Objetos, um jogo, oferece uma visão ampla de como funciona a estrutura da orientação a objetos e permite o uso de seus conceitos e pilares, durante sua concepção. Dessa forma, a construção de um jogo, mesmo que simples, visa ampliar e permite que se aplique os conhecimentos sobre esse componente curricular.

1.3 Aspectos Metodológicos

Para concepção desse trabalho, além da documentação feita por meio deste relatório, foram elaborados o GDD (Game Design Document) simplificado do jogo, o esquema UML de classes e por fim a elaboração dos códigos.

1.4 Aporte Teórico

Como aporte teórico para elaboração desse projeto, consideram-se:

- As aulas teóricas;

- DEITEL, Harvey M.; DEITEL, Paul J. C++: como programar. Tradução de Edson Furmankiewicz. Revisão técnica de Fábio Luis Picelli Lucchini. 3. reimpr. São Paulo: Pearson Education do Brasil, 2010. (Disponível na biblioteca do Campus)
- O conteúdo disponibilizado no site oficial da biblioteca Raylib, que contém instruções e tutoriais para o uso da biblioteca e suas funções.

2 GDD- GAME DESIGN DOCUMENT

2.1 Título e Resumo do Jogo

O título do jogo será Lost Astronaut, o jogo 2D consiste na queda constante de um astronauta até seu rover, no caminho ele busca coletar diversas estrelas e evitar ser atingidos por asteroides.

2.2 Jogos semelhantes

O jogo foi inspirado na fase Sunken Ghost Ship de Super Mario World 2, apresentando uma versão mais simples da “queda” do personagem principal, na qual ele desvia de objetos que lhe causam danos e coleta PowerUPs e moedas.

2.3 Objetivos e Obstáculos

O jogo contém puramente obstáculos de cenários e como objetivo principal, o jogador deverá permanecer vivo pelo tempo de jogo, a princípio 70 segundos, mas dinâmico, podendo ser alterado pelo jogador.

2.4 Fluxo de Telas

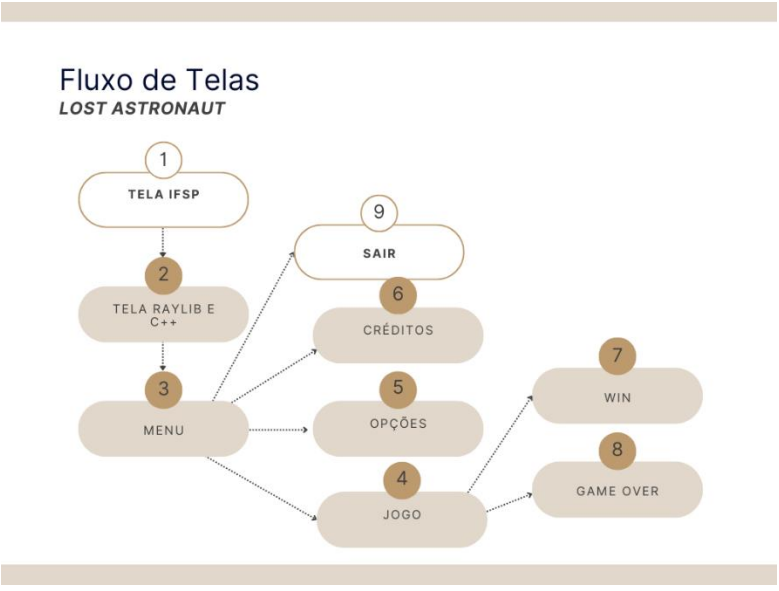

Figura 1 - Fluxo de Telas do Jogo

2.5 Diagrama UML de Classes

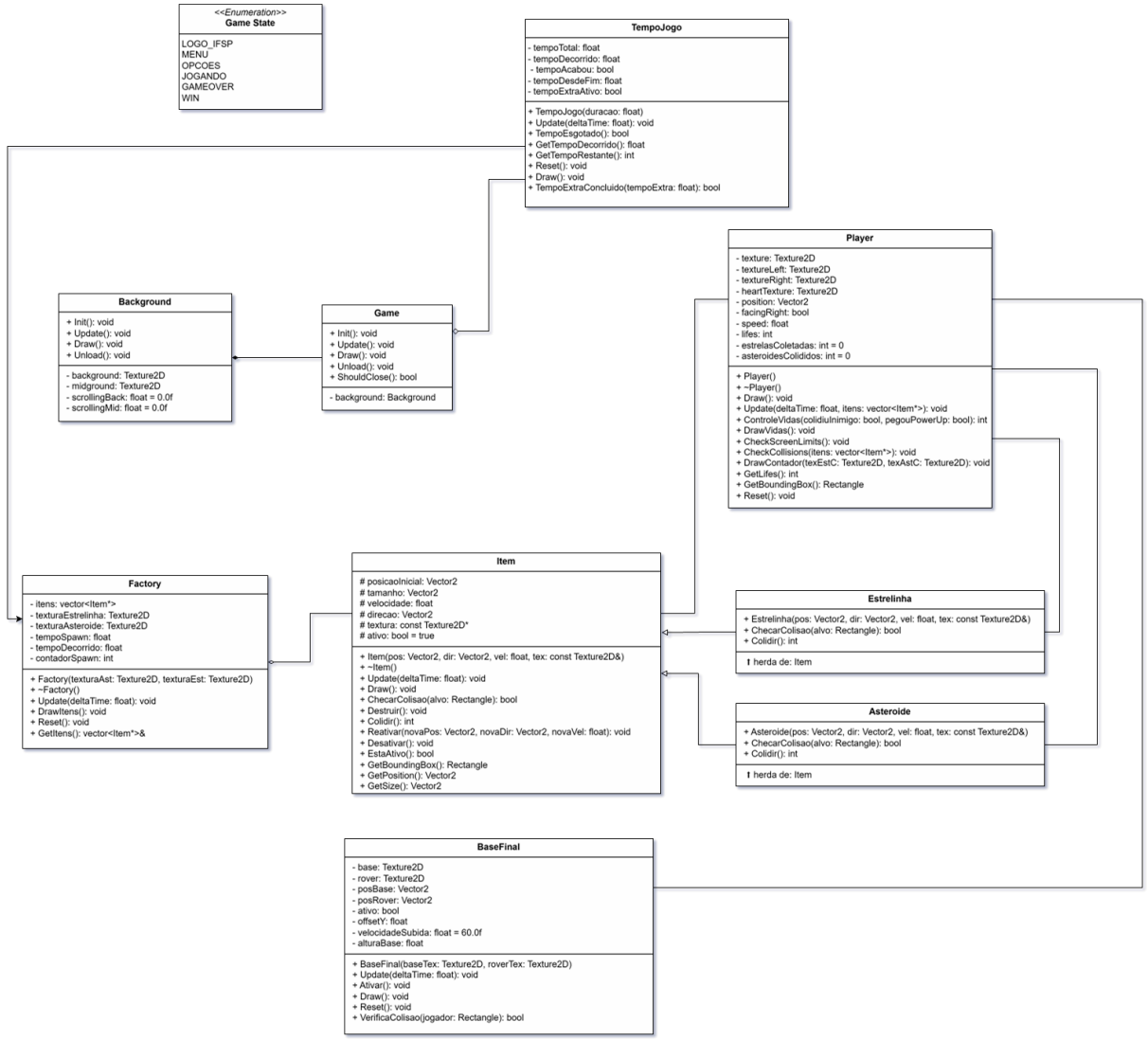

Figura 2- Diagrama UML de Classes

3 A CONSTRUÇÃO DA ESTRUTURA DE PASTAS

Como IDE para esse projeto, foi usado o CodeBlocks, a partir dele foi criada um projeto do tipo “Console Aplication”, o qual gera uma estrutura de pastas próprias, com os caminhos prévios de pastas. Acrescentaram-se também as pastas de Assets do projeto, divididas em Fontes, Sons e Imagens, os arquivos de código permaneceram em SRC e o executável em Bin/Debug.

```
LostAstronaut/
├── src/
│   ├── main.cpp
│   ├── classes.hpp
│   └── classes.cpp
├── assets/
│   ├── images
│   ├── fonts
│   └── sounds
├── bin/debug
│   └── executável
├── obj/
├── doxygen/   //não utilizado
├── include/   //padrão, mas não utilizados, arq hpp junto cpp
├── build/     //usado para armazenar o exe do background
├── lostaAstronaut.cbp
└── README.md
```

4. OS ASSETS

4.1 Imagens

Todas as imagens utilizadas passaram por tratamento de tamanho, remoção de background (utilizando Photoshop) e redimensionamento das imagens para facilitar seu uso no projeto.

4.2 Sons e músicas

A música de fundo e os sons utilizados são efeitos sonoros disponíveis para uso sem licença, o qual foram baixados e utilizados da forma que estavam.

4.3 Fontes

Para a fonte escolhida para o jogo, foi utilizada uma fonte pronta, com direitos livres a qual combinada com o tema do jogo.

5 A ESTRUTURA DO JOGO

5.1 O Background

Primeira etapa a qual foi construída, com auxílio de tutoriais do youtube e do próprio site oficial da biblioteca raylib, o qual conta com uma wiki. É composto por duas imagens, uma correspondente ao plano de fundo representando o “Céu” do planeta onde está o astronauta e outra, correspondente as diversas estrelas do fundo. Ambas usam uma função de movimentação em loop, a qual faz com que ao atingir o tamanho total uma nova imagem seja criada, estão em velocidades diferentes para dar a impressão de movimento.

5.2 O Personagem

Idealizado a partir de uma imagem de astronauta simples e “fofa” da internet, foi redimensionado para seu tamanho atual. Tem posição vertical (Y) constante, a princípio os movimentos verticais foram feitos, mas deixados de fora na versão final. (A versão dos códigos de movimento vertical ainda está no código final, porém comentada). Além da movimentação lateral, também foi criada uma fórmula simples, que inverte o personagem horizontalmente quando o jogador aciona as teclas de movimento.

5.3 Os Itens

Foram considerados 3 itens essenciais para o jogo: os corações, os quais representam as vidas, as estrelinhas, que são os PowerUPs do jogo e, por último, os asteroides que são os “inimigos”.

5.4 As Colisões e as Vidas

Na idealização do jogo a ideia era manter apenas 3 corações, no entanto, ao testar o jogo, (ainda não todo implementado) ficou claro que ficava muito complexo sobreviver o tempo de jogo, logo foram implementadas 5 vidas, uma mesma textura, desenhada 5 vezes com uma borda igual entre elas. A lógica é feita por meio das

funções de colisão e checar vidas, ao colidir, a função de checagem verifica o item (estrelinha ou asteroide) e aplica o seguinte:

- Se as vidas do jogador forem iguais a 5 e a colisão com a estrelinha: a estrela é desativada e destruída e nada mais é feito.
- Se as vidas do jogador forem estiverem entre 1 e 4 e a colisão com a estrelinha: a estrela é desativada e destruída e uma vida é adicionada ao total, redesenhando o coração que já foi perdido. Também, é acrescido 1 a contagem total de estrelas.
- Se as vidas do jogador forem iguais a 1 e a colisão com o asteroide: o asteroide é desativado, destruído, a última vida é retirada do jogador (vidas = 0) e é chamada a tela de game over.
- Se as vidas do jogador forem estiverem entre 2 e 5 e a colisão com o asteroide: o asteroide é desativado, destruído e é removida uma vida do jogador. Também é acrescido 1 a contagem total de asteroides.

5.5 O Temporizador

Pensado para ser dinâmico, ou seja, mudar de acordo com a vontade do jogador, a contagem geral do jogo o inicia na primeira tela, e é reiniciada para o jogo, podendo mudar de acordo com o que for digitado no menu de opções.

5.6 As Contagens

Não estavam previstas, mas se tornaram interessantes durante os testes de funcionamento da função “Checar Colisões”, contam as estrelas coletadas (se a vida não estiver máxima, ou seja, igual a 5), e os asteroides independente da quantidade de vidas do personagem.

5.7 O Rover

Ponto final do jogo, é ativada quando o tempo definido – padrão de 60 segundos, ou definido pelo jogador - é atingido, o Rover e a base “sobem” se movendo em Y para a posição final.

5.8 Splash: Vitória e GameOver

Criadas em tamanho proporcional, simples, são chamadas/desenhadas quando os estados de jogo respectivos a elas são ativados, juntamente com um som característico.

6 A LÓGICA E OS PASSOS PARA CONSTRUÇÃO DO JOGO

A construção do jogo foi realizada seguindo uma sequência que teve o objetivo de facilitar o desenvolvimento, a organização do código e as mudanças que poderiam ocorrer durante o desenvolvimento. Para isso, a partir do fluxo de telas, definiu-se uma série de estados para controlar o progresso entre as telas do jogo.

6.1 Estados do jogo

Cada estado do jogo corresponde a uma tela ou momento distinto da execução, permitindo a troca de textura, ações e sons específicos para cada etapa.

- **Menu Principal**: Permite escolher entre as opções de iniciar o jogo, acessar opções, ver créditos ou sair.
- **Opções**: Permite configurar o tempo de jogo.
- **Jogando**: Estado em que o jogo roda, atualizando o personagem, itens, tempo e colisões.
- **Game_Over**: Tela exibida quando o jogador perde todas as vidas, ou seja, perde o jogo.
- **Win_Game**: Tela exibida quando o jogador completa o objetivo, ou seja, sobrevive o tempo total de jogo.

6.2 Fluxo e controle

O fluxo entre os estados é controlado primeiramente por tempo, mas não somente por ele, também pelas entradas do jogador (usuário). Além disso, foram implementados controles que permitem a mudança de telas automática ao final do jogo.

6.3 Atualização do jogo

Quando o estado “JOGANDO” está em andamento, o método Update do personagem é chamado para atualizar a posição e checar colisões com os itens. Os itens possuem comportamentos que são próprios e estados também como ativos/inativos para garantir a dinâmica correta.

6.4 Renderização

Cada estado tem sua rotina de desenho (Draw), que exibe elementos como o background em movimento, o personagem, os itens, as vidas e as contagens, e as telas finais (game over e win) com seus respectivos sons.

6.5 Tratamento de entradas

Além da movimentação lateral do personagem, foram implementadas funções para capturar a entrada do usuário na tela de opções para permitir a digitação do tempo desejado para o jogo, embora, a visualização dessa tela tenha ficado um pouco estranha.

6.6 A Classe Factory para os Itens

Para criar, controlar e gerenciar os itens do jogo (os asteroides e as estrelinhas), foi implementado a classe Factory. Essa classe centraliza a criação de objetos, definindo quando e onde cada item aparece na tela – no caso de forma aleatória de diversos pontos da tela -, e controla o fluxo deles na tela — ativar, desativar, resetar e destruir. Isso facilita a manutenção do código e a criação dos itens, inclusive quanto as proporções entre eles, também, foi a forma mais adequada (dentre as pesquisadas) para permitir um controle desses itens, evitar a criação repetida de objetos e bugs durante a execução.

6.7 O Temporizador (TempoJogo)

A classe TempoJogo gerencia a contagem de tempo do jogo, funcionando como um contador de tempo e não de forma regressiva, atualizando o tempo restante em cada frame e parando quando atinge o tempo configurado. Como já dito, esse temporizador é dinâmico, podendo ser configurado pelo jogador na tela de opções,

permitindo ajustar a duração da partida. Ao final do tempo, o jogo dispara a transição para a tela de vitória, ativando os elementos finais como o rover e a base.

7 DESAFIOS, OBSERVAÇÕES E CONCLUSÃO

Durante o desenvolvimento do jogo, foram encontrados diversos desafios; Um dos principais desafios foi o gerenciamento dos objetos em tela, mesmo com uma ideia inicial a implementação não foi simples, resultando em problemas no desenho das texturas, da criação de itens, ajuste de quantidade e velocidade para que o jogador pudesse completar o jogo e outros. No entanto, foram encontrados dificuldades, especialmente, na criação e o reaproveitamento dos itens (asteroides e estrelinhas), que demandou a implementação da classe Factory, com ajuda de um tutorial em C principalmente (Making the most COMMON "First Game" In Raylib with C) disponível no youtube que ajudou a entender e escrever o código, além dos códigos da wiki do próprio raylib.

Outro ponto que gerou várias dúvidas e bugs foi a implementação da lógica de colisões e controle de vidas, principalmente por detalhes que muitas vezes passam despercebidos por quem não é familiar com a linguagem e a biblioteca, o problema de múltiplas contagens ao colidir com asteroides, foi um desses pontos, a correção de perda das vidas de forma indevida, só foi possível por conta do fórum GameMaker.io, o qual tem vários tópicos sobre problemas com colisão e que demonstrando que o problema era a destruição dos itens. Ajustar o temporizador para ser configurável pelo jogador também apresentou certa complexidade, o que acabou deixando a tela feia no final, pois demandou a integração da interface de opções com a lógica principal do jogo, além da validação da entrada do usuário, então levou um tempo para ficar pronta.

A criação de classes e interfaces, o principal para esse jogo, e a integração entre eles foi mais um desafio, por várias vezes os códigos não conversaram e levou tempo para ajustar as chamadas e inclusões.

Por outro lado, a familiaridade com o C++ ajudou no projeto, embora a raylib seja em C e muitos exemplos e projetos estejam nessa linguagem, a proximidade tornou mais simples entender os códigos, além disso, o jogo modelo no git_hub e página CheatSheet no site oficial, facilitaram a busca pelos comandos da biblioteca e, também, serviram de base. O youtube também conta com vários tutoriais – principalmente em inglês) o que facilita o entendimento e ajuda no desenvolvimento.

Então, pode-se dizer o projeto possibilitou a aplicação prática dos conceitos de programação orientada a objetos, e ainda permitiu o conhecimento de uma nova biblioteca, que além da programação possui gerenciamento de recursos gráficos e sonoros, tudo isso também foi combinado com desenvolvimento de lógica interativa – de forma a incluir e pensar no usuário, nesse caso: o jogador. O uso da biblioteca Raylib facilitou o desenvolvimento como um todo, os comandos simples a quantidade de recursos fez com o projeto fluísse melhor. Ainda, infelizmente, o tempo não possibilitou a inclusão de mais recursos, como outras funções para o jogo (aceleração, aumento de asteroides e outros powerups) e interfaces mais interessantes.

BIBLIOGRAFIA

DATE, Christopher J. **Introdução a sistemas de bancos de dados.** Elsevier Brasil, 2004.

HEUSER, Carlos Alberto. **Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS.** Bookman Editora, 2009.

MACÁRIO, Carla Geovana do N.; BALDO, Stefano Monteiro. **O modelo relacional**. Instituto de Computação Unicamp. Campinas, p. 1-15, 2005.

MACHADO, Felipe Nery Rodrigues. **Banco de Dados–Projeto e Implementação.** Saraiva Educação SA, 2020.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de gerenciamento de banco de dados-3**. AMGH Editora, 2008.

ROB, Peter; CORONEL, Carlos. **Sistemas de banco de dados. Projeto, implementação** ed. 2011.