Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Implementação do Labirinto do Rato Cego usando SDL e C++

Autor: Ian Fillipe Pontes Ferreira
Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF
2023

Ian Fillipe Pontes Ferreira

Implementação do Labirinto do Rato Cego usando SDL e C++

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2023

Ian Fillipe Pontes Ferreira
Implementação do Labirinto do Rato Cego usando SDL e C++/ Ian Fillipe Pontes Ferreira. – Brasília, DF, 2023-
74 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Desenvolvimentos de jogos. 2. Aprendizagem Baseada em Problemas. I. Prof. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação do Labirinto do Rato Cego usando SDL e C++

CDU 02:141:005.6

Ian Fillipe Pontes Ferreira

Implementação do Labirinto do Rato Cego usando SDL e C++

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 13 de dezembro de 2023:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Ricardo Ramos Fragelli
Convidado 1

Prof. Dr. Bruno César Ribas
Convidado 2

Brasília, DF
2023

"Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrepende."
(Leonardo da Vinci)

Resumo

A integração de jogos eletrônicos pode oferecer aos alunos oportunidades inovadoras de aprendizado, promovendo a capacidade de observar, analisar, ponderar e construir conhecimento de maneira ativa. Nesse contexto, o presente trabalho propôs uma nova implementação para o jogo Labirinto do Rato Cego com o objetivo de apoiar o ensino de programação. Utilizando a metodologia da Aprendizagem Baseada em Problemas, o jogo original apoiava os alunos na aplicação de conceitos fundamentais de programação, como lógica computacional, algoritmos de busca, tomada de decisões e estruturas de dados. A nova implementação, realizada em C++ e SDL2, incorpora funcionalidades que permitem aos alunos criar seus próprios labirintos, desenvolver o código para a movimentação do rato e interagir com a interface do jogo para visualizar sua locomoção no labirinto. Essa abordagem visou aperfeiçoar o Labirinto do Rato Cego original para linguagens de programação e bibliotecas mais modernas.

Palavras-chave: jogos eletrônicos. nova implementação. Labirinto do Rato Cego. ensino de programação. Aprendizagem Baseada em Projetos.

Abstract

The integration of electronic games can offer students innovative learning opportunities, promoting the ability to observe, analyze, ponder and actively construct knowledge. In this context, this paper proposed a new implementation of the Labirinto do Rato Cego game with the aim of supporting the teaching of programming. Using the Problem-Based Learning methodology, the original game supported students in applying fundamental programming concepts such as computer logic, search algorithms, decision-making and data structures. The new implementation, made in C++ and SDL2, incorporates features that allow students to create their own mazes, develop the code for moving the mouse and interact with the game's interface to visualize their movement through the labyrinth. This approach aimed to improve the original Blind Mouse Maze for more modern programming languages and libraries.

Key-words: electronic games. new implementation. Labirinto do Rato Cego. Problem-Based Learning. teaching of programming.

Lista de Figuras

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

PBL	Aprendizagem Baseada em Problemas, do inglês *Problem-Based Learning*
APC	Algoritmos e Programação de Computadores
TCC	Trabalho de Conclusão de Curso
IDE	*Integrated Development Environment*
SDL	*Simple DirectMedia Layer*
UnB	Universidade de Brasília
FGA	Faculdade UnB Gama

Sumário

Introdução

Os jogos eletrônicos tornaram-se parte integrante da sociedade moderna, cativando pessoas de todas as idades. Segundo uma pesquisa realizada pela Newzoo[1] em 2022, o mercado global de jogos eletrônicos atingiu uma receita de US$ 184,4 bilhões, o que destaca o impacto e a popularidade desse meio de entretenimento.

De acordo com Prensky (2001), os métodos tradicionais de ensino podem apresentar limitações ao engajar e educar efetivamente as gerações que cresceram em meio à era da tecnologia digital. Diante disso, a adoção do novas estratégias vem sendo exploradas, como o uso de jogos eletrônicos no contexto educacional com o intuito de proporcionar uma maior motivação e engajamento dos estudantes, ao permitir a exploração ativa, a aplicação prática do conhecimento e o desenvolvimento de habilidades de resolução de problemas.

Uma área que vêm utilizando jogos eletrônicos como forma de ensino é o aprendizado de programação. Essa abordagem tem se mostrado eficaz e envolvente para engajar os alunos nesse estudo. Dois exemplos são o Robocode[2] e o CodinGame[3], que são jogos que incentivam os alunos a desenvolver habilidades de resolução de problemas, trabalho em equipe e pensamento criativo, ao mesmo tempo em que aprimoram suas habilidades de programação. Em ambos os casos, os jogos eletrônicos proporcionam um ambiente imersivo e motivador, estimulando os alunos a explorar e experimentar, tornando o aprendizado de programação mais atraente e efetivo.

Nesse contexto, o presente trabalho propõe a utilização da teoria da Aprendizagem Baseada em Problemas (PBL, do inglês *Problem-Based Learning*) como abordagem pedagógica para o ensino de programação, utilizando uma nova implementação do jogo Labirinto do Rato Cego, criado pelo professor Dr. Ricardo Ramos Fragelli. Esta reimplementação do jogo visa retomar a iniciativa, que não está mais disponível, mas que demonstrou sucesso em promover, de forma lúdica, o aprendizado de programação e a interação social entre estudantes.

Como definido por Fragelli: “Eu desenvolvi um desafio chamado Labirinto do Rato Cego, no qual os estudantes deveriam programar o cérebro do rato para que ele pudesse sair de um labirinto”(QUEIROZ, 2021). Nesse cenário, a metodologia PBL busca promover a aprendizagem ativa, em que os alunos são incentivados aprender conceitos de programação. Logo, o jogo tem como objetivo despertar o interesse dos alunos, incentivando-os a aplicar seus conhecimentos teóricos e buscar soluções criativas para o desafio proposto.

[1] <https://newzoo.com/resources/blog/the-games-market-in-2022-the-year-in-numbers>
[2] <https://robocode.sourceforge.io>
[3] <https://www.codingame.com/start>

Nesse contexto, emerge uma questão fundamental: *Como o Labirinto do Rato Cego pode potencializar o ensino de programação, promovendo não apenas a aquisição de habilidades técnicas, mas também o desenvolvimento de competências cognitivas e sociais?* Esta indagação norteia a pesquisa e fornece uma base para o desenvolvimento do jogo.

Objetivos

O objetivo deste trabalho é uma reimplementação do Labirinto do Rato Cego utilizando a linguagem de programação C++ e a biblioteca SDL (Simple DirectMedia Layer).

Os objetivos específicos de desenvolvimento englobam:

- desenvolver a interface do Labirinto do Rato Cego;
- implementar vários ratos se movendo ao mesmo tempo;
- desenvolver o componente de criação de labirintos;
- implementar o método de interação entre o código criado pelo jogador e o labirinto;

Estrutura do trabalho

O presente trabalho está estruturado em quatro capítulos. No capítulo FUNDAMENTAÇÃO TEÓRICA serão explorados os conceitos fundamentais relacionados ao jogo Labirinto do Rato Cego. No capítulo METODOLOGIA serão detalhados todos os processos anteriores ao desenvolvimento do jogo, incluindo as ferramentas e tecnologias utilizadas, bem como os critérios adotados para a pesquisa bibliográfica de trabalhos correlatos, que serviram de embasamento para a criação do jogo. No capítulo RESULTADOS serão apresentados os resultados obtidos ao longo do desenvolvimento do jogo, demonstrando os recursos e funcionalidades implementados. Por fim, o capítulo CONSIDERAÇÕES FINAIS consolidará as reflexões e conclusões decorrentes do trabalho realizado, além de delinear possíveis direções para futuras pesquisas e aprimoramentos no Labirinto do Rato Cego.

1 Fundamentação Teórica

Neste capítulo são abordados os conceitos teóricos e trabalhos correlatos que são fundamentais para o desenvolvimento do Labirinto do Rato Cego, objeto de estudo deste trabalho. Serão apresentados os principais conceitos que direcionam a pesquisa, bem como os estudos anteriores que inspiraram e contribuíram para o desenvolvimento deste trabalho.

1.1 *Problem-Based Learning*

De acordo com Barrows e Tamblyn (1980), a Aprendizagem Baseada em Problemas (PBL, do inglês *Problem-Based Learning*) é uma estratégia de ensino que coloca o aluno como protagonista do seu próprio aprendizado, ao enfrentar desafios que estimulam o raciocínio crítico e a solução de problemas.

Segundo Barrows e Tamblyn (1980), o PBL envolve a apresentação de um problema desafiador aos estudantes, que devem investigar e adquirir conhecimentos relevantes para solucioná-lo. Nesse processo, os alunos são incentivados a trabalhar em equipe, a buscar informações de diversas fontes e a integrar conceitos de diferentes disciplinas, a fim de desenvolver uma compreensão e solução do problema.

O PBL é uma abordagem que se baseia em uma visão construtivista do conhecimento e enfatiza o desenvolvimento de habilidades cognitivas essenciais para a formação integral dos alunos. Segundo Hardless, Nilsson e Nuldén (2005), a teoria construtivista na educação enfatiza a importância de elementos como ação, significação, conflitos cognitivos, socialização, autonomia e interdisciplinaridade. Esses elementos são fundamentais em ambientes educacionais que adotam uma abordagem que centraliza o processo de aprendizagem no estudante.

O estudo de Barrows, considerado o pioneiro no desenvolvimento do PBL, começou na década de 1960, quando ele estava trabalhando na Universidade McMaster, no Canadá. Ele buscava encontrar uma alternativa ao tradicional modelo de ensino de Medicina, que se baseava principalmente em aulas expositivas. Barrows acreditava que os estudantes de medicina deveriam aprender através da aplicação prática do conhecimento, em vez de apenas receber informações passivamente (BARROWS; TAMBLYN, 1980).

Os resultados do estudo de Barrows sobre a PBL foram muito positivos. Os alunos envolvidos nesse método de ensino apresentaram maior motivação, desenvolveram habilidades de resolução de problemas e tiveram um melhor desempenho em testes de aplicação do conhecimento. Além disso, os alunos relataram uma maior satisfação com o processo

de aprendizagem, pois se sentiam mais engajados e responsáveis por sua própria educação, já que nesse processo o papel do professor era o de facilitador, fornecendo orientação e *feedback* aos estudantes à medida que eles progrediam (BARROWS; TAMBLYN, 1980).

O PBL tem sido aplicado em diferentes áreas do conhecimento, desde a educação até a engenharia. Sua abordagem tem se mostrada vantajosa, pois auxilia no aumento do senso de responsabilidade dos estudantes, que agora precisam ter vontade e disciplina para estudar e aprender por conta própria, e incentiva os estudantes a investigarem mais a fundo os problemas apresentados a fim de encontrar soluções práticas para eles.

1.2 Labirinto do Rato Cego

O Labirinto do Rato Cego é um projeto criado e implementado pelo professor Dr. Ricardo Ramos Fragelli, que tem como objetivo motivar e tornar a aprendizagem mais significativa para os estudantes de disciplinas iniciais de programação, através da teoria de Aprendizagem Significativa e da Aprendizagem Baseada em Problemas (PBL) (FRAGELLI; VAINSTEIN, 2011).

Fragelli se motivou a fazer esse trabalho porque percebeu que as disciplinas iniciais de programação possuíam um alto fator de desmotivação, já que os alunos chegavam com um pensamento de que ao final do semestre teriam a capacidade de fazer um software comercial, porém percebiam que o padrão de aulas expositivas que estavam submetidos fazia com que não tivessem contato com a criação de uma aplicação significativa que incorporasse o conteúdo aprendido. E esse fato era refletido na dificuldade de aprendizado em disciplinas subsequentes, que necessitam de um conhecimento sólido nos conceitos básicos da programação (FRAGELLI; VAINSTEIN, 2011).

No desafio proposto pelo Labirinto do Rato Cego, os estudantes devem desenvolver um programa de computador que simule as decisões que um agente inteligente deve tomar para encontrar a saída de um labirinto. Complementar a isso, é proposto ao final do desafio uma competição entre os estudantes, na qual o grupo vencedor é aquele que desenvolve o programa que consegue fazer o rato sair do labirinto no menor tempo.

Fragelli iniciou a implementação do Labirinto do Rato Cego em 2005, destinado aos estudantes da disciplina de Algoritmos e Programação de Computadores (APC) em uma instituição particular do Distrito Federal. Este projeto foi posteriormente avaliado no contexto de uma universidade federal, a Faculdade UnB Gama (FGA), no ano de 2010, conforme mencionado por Fragelli e Vainstein (2011).

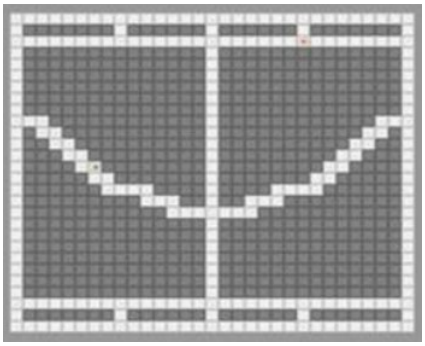
1.2.1 Implementação do projeto

No intuito de tornar o trabalho proposto pelo Labirinto do Rato Cego um aprendizado contínuo, os estudantes recebiam no início um rato que tomava decisões aleatórias com todos seus módulos já bem estruturados. Com o passar das aulas, os grupos iam tendo o aprendizado proposto pela disciplina de APC e iam modificando o programa ao longo do semestre, no intuito de desenvolverem a inteligência do rato, fazendo com que ele saísse do labirinto de uma forma cada vez mais eficiente (FRAGELLI; VAINSTEIN, 2011).

No programa do Labirinto do Rato Cego, a comunicação entre o rato e o labirinto era feita por arquivos de texto. O labirinto deveria informar a posição onde o rato estava em um determinado momento, e o rato deveria informar em outro arquivo qual é a sua decisão, ou seja, para onde ele queria se mover. Com essa comunicação, o labirinto sempre sabia onde o rato estava e para onde ele gostaria de ir, podendo permitir ou não esse caminho, e essa comunicação continuava até a saída ser encontrada ou até o limite de novecentos e noventa e nove decisões ser atingido. Essa comunicação por arquivos de texto foi escolhida para que os estudantes pudessem desenvolver os ratos utilizando qualquer linguagem de programação (FRAGELLI; VAINSTEIN, 2011).

Quando o Labirinto do Rato Cego era executado, o primeiro passo era escolher o labirinto em que o rato se movimentaria. A Figura 1 mostra um exemplo de labirinto.

Figura 1 – Exemplo de labirinto no trabalho original.



Fonte: Fragelli e Vainstein (2011).

Uma vez escolhido o labirinto, os ratos eram executados e era gerado o arquivo de texto contendo as informações sobre o labirinto e os percursos dos ratos. A partir da leitura do arquivo, era possível visualizar o movimento dos ratos e verificar ao final quem ganhou, a partir da quantidade de passos, movimentos e o tempo, como pode ser visto na Figura 2.

Como resultado da aplicação em turmas de APC, de 2005 a 2006, foi observando a mudança de comportamento dos estudantes durante as aulas, podendo-se verificar que

Figura 2 – Visualização do jogo no trabalho original.

Fonte: Fragelli e Vainstein (2011).

os alunos demonstraram maior interesse nas aulas presenciais e fizeram associações entre o conteúdo apresentado e sua aplicação na confecção da inteligência do rato. E isso levou a discussões frequentes em sala de aula, promovendo uma atmosfera mais produtiva. Em adição a esses resultados, a partir de 2007 foram utilizados fóruns de discussão online para estudar as expectativas dos estudantes em relação ao curso, à disciplina e ao andamento dos trabalhos durante o semestre. Os fóruns apresentaram bons resultados, com os alunos destacando a implementação do Labirinto do Rato Cego como uma forma de assimilar melhor o aprendizado, ajudar no desenvolvimento de criatividade e transformar a matéria em algo mais interessante e instigante. Em 2010, esse experimento foi feito como uma atividade de extensão universitária na FGA, que contou com estudantes mais avançados, dando a possibilidade dos estudantes novatos interagirem com veteranos e se familiarizarem com tópicos mais avançados de programação (FRAGELLI; VAINSTEIN, 2011).

1.3 Jogos eletrônicos

Segundo Salen e Zimmerman (2003), pesquisadores de jogos, um jogo eletrônico é uma atividade com regras definidas, que envolve a interação do jogador com um sistema computacional.

Os jogos eletrônicos possuem uma variedade de características que os distinguem

de outras formas de mídia e entretenimento. Uma das características centrais é a interatividade, que permite aos jogadores assumir o controle e tomar decisões que fazem a experiência de jogar ser diferente para cada um.

Outra característica importante dos jogos eletrônicos é a ludicidade. Johan Huizinga, em seu livro "Homo Ludens", discute o conceito de jogo como uma atividade intrinsecamente humana. Os jogos eletrônicos, como forma moderna de jogo, incorporam elementos de desafio, competição e diversão, proporcionando aos jogadores uma experiência lúdica e imersiva (HUIZINGA, 2019).

Outro autor relevante, que aborda o tema de jogos como forma de ensino, é Marc Prensky, que usou o termo "nativos digitais" para descrever a geração que cresceu imersa em tecnologia digital, defendendo a ideia de que os jogos eletrônicos podem ser uma forma eficaz de aprendizagem para esses indivíduos (PRENSKY, 2001). Essas contribuições teóricas ajudam a fundamentar o estudo e a compreensão dos jogos eletrônicos como um fenômeno cultural, artístico e educacional.

1.3.1 Desenvolvimento de jogos eletrônicos

A Figura 3 mostra as etapas do desenvolvimento de um jogo, sendo elas: concepção, pré-produção, produção e pós-produção, que ocorre após o lançamento do jogo. Porém, essas etapas podem sofrer variações de acordo com a empresa ou a equipe, por exemplo, a inclusão de etapas de lançamentos de versões *alpha* e *beta*, para validar a qualidade do jogo.

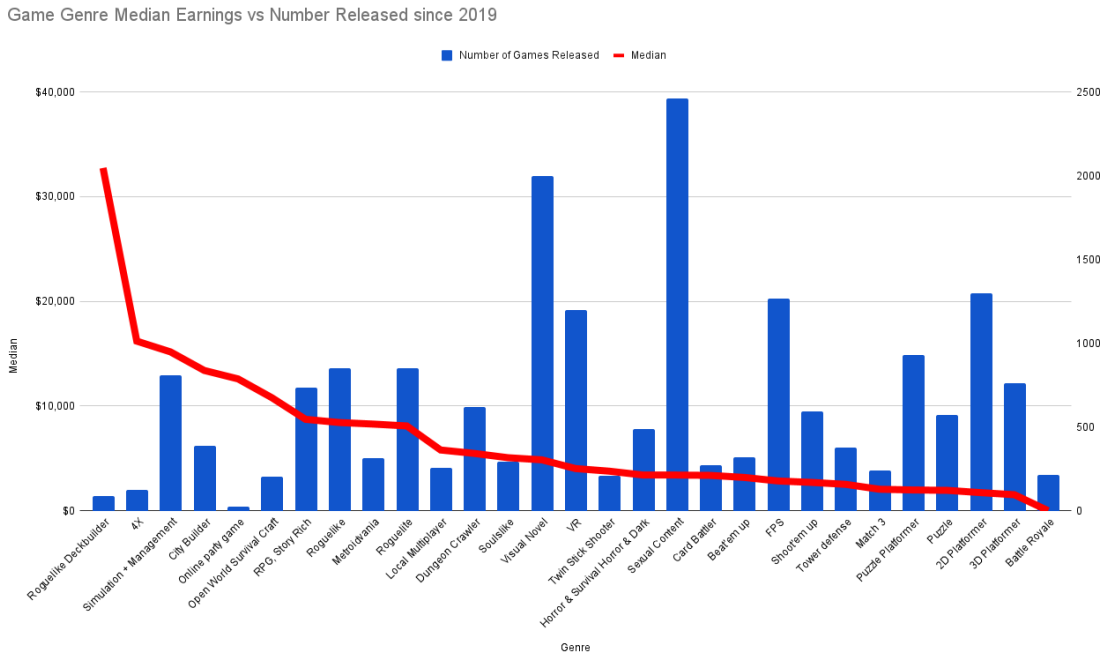
Figura 3 – Etapas do desenvolvimento de jogos.

Fonte: IMD (2016).

O processo de desenvolvimento de um jogo eletrônico geralmente começa com a concepção da ideia. Nesta fase, a equipe de desenvolvimento se reúne para realizar o *game design*, que é o processo de criar a estrutura, regras e elementos de um jogo, com o objetivo de proporcionar uma experiência envolvente e significativa para os jogadores. A pesquisa de mercado e a análise de tendências também desempenham um papel importante na definição da ideia central (FULLERTON, 2008). Nesse sentido, exemplificando essa situação, a Figura 4 representa um gráfico que serve como análise da tendência de

jogos que vêm sendo feitos por estúdios independentes, aonde as barras azuis mostram a quantidade de jogos *indies* lançados por gênero entre os anos de 2019 e 2022, e a linha vermelha mostra a estimativa média de vendas para cada gênero.

Figura 4 – Ganho médio de jogos *indies* X quantidade de jogos.



Fonte: Zukalous (2022).

Após a definição da ideia, a equipe passa para a fase de pré-produção. Nessa etapa, são realizados estudos de viabilidade, análises de recursos necessários e definição dos requisitos do jogo. Também é comum criar protótipos rápidos para testar e validar as mecânicas de jogo, a fim de identificar possíveis melhorias e ajustes.

Uma vez concluída a pré-produção, inicia-se a fase de produção propriamente dita. Essa fase envolve a implementação das mecânicas de jogo, desenvolver os níveis, criar personagens, efeitos visuais e sonoros, entre outros aspectos.

Uma parte fundamental do desenvolvimento de jogos eletrônicos é a programação. A equipe de desenvolvimento utiliza linguagens de programação específicas, como C++, C# e Java, para criar o código que controla a lógica do jogo e a interação com o jogador.

Aliado a programação, uma das abordagens é o uso de motores de jogo, como o Unity[1] e o Unreal Engine[2]. Essas ferramentas oferecem um conjunto de recursos e funcionalidades que simplificam o processo de desenvolvimento, permitindo que os desenvolvedores se concentrem mais na criatividade e no design.

[1] <https://unity.com/pt>
[2] <https://www.unrealengine.com/pt-BR>

No aspecto visual, a equipe de arte é responsável por criar os elementos visuais do jogo, incluindo a concepção de personagens, ambientes, animações, efeitos especiais e interfaces de usuário, no intuito de ser criado uma identidade visual única e atrativa para o jogo. Softwares especializados, como Photoshop[3], Illustrator[4], Maya[5] ou Blender[6], são utilizadas na indústria de jogos para criar *assets* visuais de alta qualidade.

Durante o processo de desenvolvimento, é importante realizar testes regulares para identificar e corrigir problemas. A equipe de teste é responsável por explorar o jogo em busca de *bugs*, falhas de desempenho, problemas de balanceamento e possíveis melhorias. Essa etapa de refinamento ajuda a garantir a qualidade final do jogo antes de seu lançamento.

Em resumo, o desenvolvimento de um jogo eletrônico é um processo abrangente e multifacetado, que envolve a concepção da ideia, a pré-produção, a produção, os testes e a pós-produção, sendo essa última etapa responsável por ajustes e otimizações depois que o jogo é lançado, garantindo a melhoria contínua do jogo.

1.4 Trabalhos Correlatos

Com o intuito de fundamentar o estudo realizado, trabalhos correlatos foram pesquisados, a partir de palavras e frases chaves. Encontrado essas pesquisas, uma segunda seleção a partir do resumo foi feita, separando aqueles que mais se relacionassem com o estudo proposto pelo presente Trabalho de Conclusão de Curso (TCC). Toda a metodologia de pesquisa bibliográfica se encontra na Seção 2.1.

A pesquisa feita por Farias et al. (2019) propõe o uso de jogos digitais como estratégia de ensino para a aprendizagem de programação *front-end*. Foram utilizados três jogos digitais: CSS Diner, Flexbox Defense e Flexbox Froggy, que foram selecionados por suas mecânicas e jogabilidade, além da possibilidade de trilhas de aprendizagem personalizadas. Os resultados da pesquisa indicaram que os jogos digitais foram eficazes na promoção de uma aprendizagem significativa. Este trabalho demostrou uma semelhança significativa com o Labirinto do Rato Cego, pois todos esse jogos necessitam que o jogador programe de fato para que se tenha sucesso no jogo.

Chang, Chung e Chang (2020) propõem uma abordagem de Aprendizagem Baseada em Jogos, com foco no ensino de programação para estudantes universitários, usando o jogo Programmer Adventure Land, o qual visa melhorar o desempenho dos alunos ao lidar com problemas práticos que estimulam o pensamento lógico. O estudo utiliza estratégias de Aprendizagem Baseada em Problemas e um sistema de pontuação para incentivar

[3] <https://www.adobe.com/br/products/photoshop/landpa.html>
[4] <https://www.adobe.com/br/products/illustrator/campaign/pricing.html>
[5] <https://www.autodesk.com.br/products/maya>
[6] <https://www.blender.org>

a aprendizagem autodirigida. Da mesma forma, o Labirinto do Rato Cego também utiliza a Aprendizagem Baseada em Problemas para ensinar programação. No entanto, enquanto o Programmer Adventure Land é um jogo que oferece uma experiência interativa, no Labirinto do Rato Cego é necessário que o jogador programe a movimentação do rato, de modo que a experiência do jogador não é interativa.

O estudo proposto por Natucci et al. (2020) investigou o uso do jogo educacional Robocode para ensinar habilidades de programação e habilidades do século XXI, como pensamento computacional e trabalho em equipe. O estudo envolveu uma competição nacional com a participação de estudantes e ex-alunos de diferentes instituições educacionais do Brasil, e os resultados indicaram que o Robocode contribuiu significativamente para o desenvolvimento dessas habilidades. Em comparação, o programa Labirinto do Rato Cego possui uma capacidade de promover tais aprendizados, já que ele também promove um sistema de competição, no sentido de fazer o algoritmo mais eficiente para tirar o rato do labirinto, enquanto que, no RoboCode, o algoritmo visa a criação do melhor robô para a batalha.

Ince (2021) concentra-se nas percepções dos alunos sobre a plataforma CodinGame para aprender programação, e a pesquisa mostra que o CodinGame é considerado uma plataforma educacional agradável e conveniente para aprender programação, aumentando a motivação para o estudo de programação, especialmente durante a pandemia. Tanto o Labirinto do Rato Cego quanto o CodinGame são voltados para a aprendizagem de programação, usando uma abordagem lúdica para envolver os alunos e desafiá-los a escrever códigos para resolver problemas.

Mohanarajah e Sritharan (2022) propõem uma abordagem de Aprendizagem Baseada em Jogos para ensinar programação usando o jogo educacional Shoot2Learn. O estudo examina se essa abordagem melhora o desempenho acadêmico e mostra que esse jogo, no formato *fix-and-play*, ajuda a superar desafios no aprendizado de programação e motivam os alunos a jogar e aprender. O estudo difere-se do Labirinto do Rato Cego, em principal, no quesito de jogabilidade, no qual este promove a codificação em si, já no Shoot2Learn, ele promove um estudo interativo com perguntas.

Por fim, o estudo proposto por Moradi e Noor (2022) utiliza o *framework* de jogos educacionais Immersivio para ensinar computação gráfica e conclui que o jogo aumentou significativamente a motivação e o desempenho dos participantes. O estudo enfatiza a importância de considerar tanto as perspectivas pedagógicas quanto técnicas ao projetar jogos educacionais. Esse estudo se assemelha ao Labirinto do Rato Cego, pois ambos utilizam jogos educacionais para motivar a aprendizagem por meio da teoria de Aprendizagem Baseada em Problemas.

2 Metodologia

Este capítulo visa detalhar os procedimentos adotados para atingir os objetivos do trabalho, dividindo-se em seis seções. A primeira aborda os métodos para levantar estudos sobre o Labirinto do Rato Cego. Em seguida, são descritas as principais ferramentas e o ambiente de desenvolvimento. Finalizando com os tópicos de planejamento, estrutura do código e diagrama de classes.

2.1 Pesquisa bibliográfica

Para realizar a pesquisa bibliográfica deste trabalho foram utilizadas diversas plataformas de pesquisa, incluindo o Google Acadêmico[1], *ResearchGate*[2] e Periódicos CAPES[3]. É importante destacar que as buscas foram realizadas por meio da conexão de internet disponibilizada pela Universidade de Brasília (UnB), o que permitiu acesso amplo às publicações disponíveis nessas plataformas.

Com o objetivo de encontrar estudos e publicações relevantes relacionadas ao desenvolvimento de jogos para a aprendizagem de programação e jogos educacionais, foram selecionadas frases-chave específicas para cada área. Para garantir a atualidade das publicações, a maioria das buscas foi filtrada por período de publicação mais recente. A Tabela 1 apresenta as palavras-chave utilizadas em cada site, juntamente com o respectivo número de publicações encontradas.

Após realizar cada uma das buscas, foram analisados os títulos dos artigos, com o objetivo de fazer uma primeira seleção. Em seguida, foram lidos os resumos de todos os textos selecionados e escolhidos aqueles que melhor se adequavam ao escopo do Labirinto do Rato Cego. Dessa forma, foram separados os artigos que estão apresentados na Seção 1.4, como pode ser observado na Tabela 2.

Essa etapa de pesquisa bibliográfica foi fundamental para a construção de uma base teórica sólida e embasada nas mais recentes pesquisas e teorias relacionadas ao tema do trabalho.

[1] <https://scholar.google.com>
[2] <https://www.researchgate.net>
[3] <https://www-periodicos-capes-gov-br.ezl.periodicos.capes.gov.br/index.php?>

Tabela 1 – Mapeamento da pesquisa bibliográfica feita de trabalhos correlatos.

Site	**Frase-chave**	**Qtd. de resultados**	**Período de busca**
Google Acadêmico	*Problem-Based Learning with games*	≈17.500	Desde 2019
Google Acadêmico	*Game Programming Simulation coding games for learning programming for software students*	≈ 16.200	Desde 2019
ResearchGate		≈ 1000	Qualquer período
Google Acadêmico	*Flexbox Froggy*	≈ 26	Desde 2019
Google Acadêmico	*RoboCode*	≈ 371	Desde 2019
ResearchGate	*RoboCode*	78	Qualquer período
Periódicos CAPES	*RoboCode*	34	Qualquer período
Periódicos CAPES	*CodinGame*	3	Qualquer período

Fonte: o Autor.

2.2 Ferramentas e ambiente de execução

O ambiente de desenvolvimento escolhido para o projeto foi o Microsoft Visual Studio 2022 na versão 17.6.5. O Visual Studio[4] é uma IDE (*Integrated Development Environment*) que fornece uma ampla gama de recursos e ferramentas para o desenvolvimento de software, como recursos avançados de depuração, edição de código e gerenciamento de projetos.

A linguagem de programação escolhida para o desenvolvimento do jogo foi o C++ em sua versão C++20, que é a versão contida no Microsoft Visual Studio 2022. Essa escolha se deve à sua eficiência, poder e ampla adoção na indústria de jogos. O C++ oferece um alto nível de controle sobre o hardware e a memória do sistema, o que é essencial para a criação de jogos de alto desempenho. Além disso, a ampla disponibilidade de bibliotecas e *frameworks* para jogos em C++ o torna uma escolha sólida para o desenvolvimento de jogos.

A biblioteca *Simple DirectMedia Layer*[5] (SDL) é uma biblioteca multiplataforma amplamente utilizada para o desenvolvimento de jogos em C++. Ela fornece uma API simples e intuitiva para lidar com gráficos, áudio, entrada de usuário e outras funcionalidades essenciais para jogos. A escolha da SDL2 para o desenvolvimento do jogo se se deu, dentre outros motivos, pela sua facilidade de uso, eficiência e suporte multiplataforma, o que permite que o jogo seja executado em diferentes sistemas operacionais.

[4] <https://visualstudio.microsoft.com/pt-br>
[5] <https://www.libsdl.org>

Tabela 2 – Textos selecionados como trabalhos correlatos.

Título	Autor(es)	Ano de publicação
The Impact of Problem-Based Serious Games on Learning Motivation	Meisam Moradi e Nurul Fazmidar Binti Mohd Noor	2022
Influence of problem-based learning games on effective computer programming learning in higher education	Chiung-Sui Chang, Chih-Hung Chung e Julio Areck Chang	2020
Shoot2Learn: Fix-and-Play Educational Game for Learning Programming; Enhancing Student Engagement by Mixing Game Playing and Game Programming	Selvarajah Mohanarajah e Thambithurai Sritharan	2022
Ensino de Programação *Front-end* através de jogos digitais: Um relato de experiência na Escola de Programação do LAIS/HUOL	Fernando Farias, Danieli Rabelo, Ricardo Valentim e Isabel Nunes	2019
O uso do jogo *Robocode* para desenvolvimento de carreiras em STEM e habilidades do século XXI: um estudo de caso nacional	Gabriel Natucci, Jhasmani Cruz, Marcos Borges e Regina Moraes	2020
Students' Perceptions on Learning Programming with CodinGame	Ebru Yilmaz Ince	2021

Fonte: o Autor.

No ambiente Windows, o WSL2[6] (Subsistema do Windows para Linux) foi empregado para criar um arquivo FIFO e executar comandos que possibilitam a comunicação entre dois códigos distintos. Essa abordagem, facilitada pelo WSL2, permite a integração de funcionalidades do sistema operacional Windows com a compatibilidade e recursos do ambiente Linux.

O versionamento do código-fonte foi realizado pelo Git, um sistema de controle de versão amplamente utilizado na indústria de desenvolvimento de software. Ele permite o

[6] <https://learn.microsoft.com/pt-br/windows/wsl/install>

controle de alterações no código, o trabalho colaborativo e a criação de diferentes versões do projeto. Para hospedar o repositório do projeto, foi utilizado o GitHub[7], uma plataforma online para hospedagem de projetos baseados em Git que oferece recursos para o compartilhamento, colaboração e gerenciamento.

O desenvolvimento do jogo ocorreu em um sistema com as seguintes especificações de hardware: *Desktop* Windows 10 com processador AMD Ryzen 5 1600 Six-Core 3.2 GHz, 16 GB de memória RAM e placa de vídeo NVIDIA GeForce RTX 2060.

2.3 Escolha da biblioteca SDL2 com C++

No início do desenvolvimento deste projeto, foram cuidadosamente analisadas as opções disponíveis em termos de escolha entre uma *engine* de jogos ou uma biblioteca. A decisão de utilizar C++ em conjunto com a biblioteca SDL2 foi fundamentada principalmente no conhecimento prévio do autor em C++ e na versatilidade da SDL2 como uma biblioteca multiplataforma.

A escolha do C++ se justifica não apenas pelo seu amplo uso na indústria de jogos, mas também pelos recursos avançados que oferece, como manipulação direta de memória, suporte à programação orientada a objetos e a capacidade de otimização de código. A considerável comunidade de desenvolvedores de jogos que utiliza C++ contribui para uma base sólida de recursos, bibliotecas e frameworks, facilitando o acesso a soluções prontas e melhores práticas (BRANCO, 2022).

A SDL2, por sua vez, proporciona uma abstração eficiente de hardware e gráficos, simplificando o desenvolvimento e garantindo consistência em diferentes plataformas. Sua natureza multiplataforma é crucial, permitindo que os desenvolvedores alcancem uma audiência abrangente, incluindo diversos sistemas operacionais, consoles de videogame e dispositivos móveis.

Ao optar por C++ e SDL2, a intenção é estabelecer uma base sólida para a durabilidade do jogo. A robustez e eficiência do C++ facilitam a manutenção contínua do código, possibilitando atualizações, correções de bugs e expansões sem comprometer o desempenho. Além disso, a SDL2, com seu suporte a várias plataformas, contribui para a longevidade do jogo, assegurando que ele permaneça acessível a um público amplo mesmo diante da evolução constante das tecnologias.

Vale ressaltar que a escolha de C++ e SDL2 também está alinhada com o contexto da continuidade do projeto Labirinto do Rato Cego. Considerando o perfil dos estudantes da FGA, que frequentam disciplinas como Introdução aos Jogos Eletrônicos (FGA) ou Introdução ao Desenvolvimento de Jogos (campus Darcy Ribeiro), as quais utilizam a

[7] <https://github.com>

biblioteca SDL2, essa escolha simplifica a manutenção e a adição de novas funcionalidades por estudantes interessados na área de desenvolvimento de jogos.

2.4 Abordagem de desenvolvimento solo

O conceito de *Personal Extreme Programming* (PXP), conforme apresentado por Dzhurov, Krasteva e Ilieva (2009), incorpora princípios específicos que se alinham com as nuances do desenvolvimento de software em um contexto individual. Além de destacar a adaptação dos princípios do Extreme Programming (XP), o PXP inclui dois pilares fundamentais:

- **Comunicação Consigo Mesmo**: Em ambientes tradicionais de desenvolvimento, a comunicação efetiva ocorre entre membros da equipe. No PXP, o desenvolvedor precisa garantir uma comunicação clara consigo mesmo, documentando decisões, pensamentos e planejamentos de maneira compreensível para futuras referências.
- **Planejamento Pessoal Iterativo**: Enquanto o XP tradicional enfatiza o planejamento em equipe, o PXP ajusta esse conceito para a realidade individual. Isso envolve ciclos de planejamento mais curtos e adaptações mais frequentes para acomodar as mudanças nas prioridades do desenvolvedor.

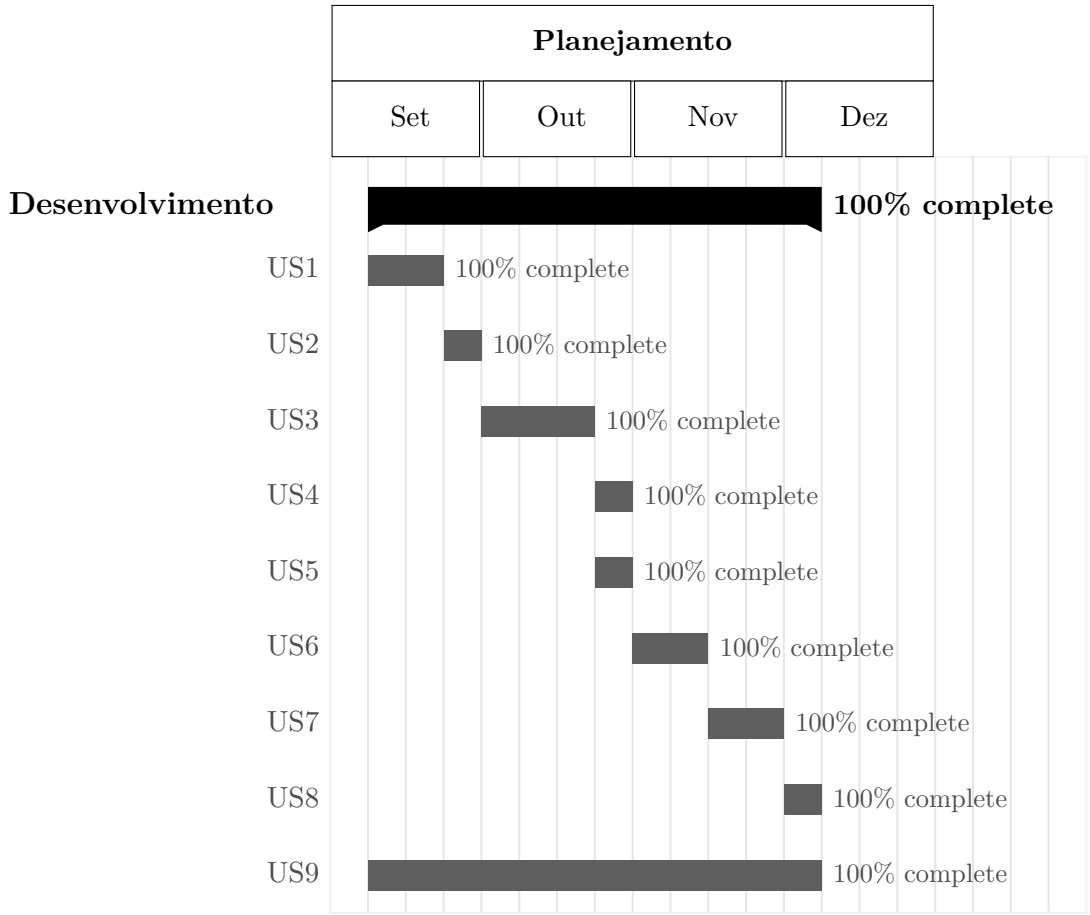
A partir desse princípios fase de planejamento foi construída. A Figura 5 apresenta o planejamento em *sprints* semanais para realaizar o desenvolvimento do Labirinto do Rato Cego.

2.5 Estrutura de pastas e arquivos

A estrutura de diretórios foi planejada de maneira a garantir uma organização de arquivos no intuito de facilitar a navegação e a manutenção dos recursos e do código-fonte da aplicação. Os principais diretórios do projeto são:

- `assets/`
  Este diretório é responsável por armazenar todos os recursos externos ou arquivos de dados necessários para o jogo. Ele contém os subdiretórios: `fonts/`, que contém a fonte de texto utilizada no jogo; `maps/`, que abriga os labirintos do jogo; `movements/`, que guarda os arquivos de movimentação do rato; e `rat/`, que armazena as imagens relacionadas ao rato.
- `include/`
  Este diretório contém arquivos de cabeçalho, que definem as interfaces e as declarações de classes e funções usadas no jogo. Nela existem os arquivos:

Figura 5 – Planejamento de histórias de usuários.



Fonte: o Autor.

  - `cell.h`
  - `configSelection.h`
  - `drawRanking.h`
  - `engine.h`
  - `gameDesign.h`
  - `gameObject.h`
  - `mapRender.h`
  - `maze.h`
  - `rat.h`
  - `ratInstance.h`
  - `ratSelection.h`

- `src/`
  Este diretório contém os arquivos de código-fonte do jogo. Ele inclui os arquivos:

- `cell.cpp`
- `configSelection.cpp`
- `drawRanking.cpp`
- `engine.cpp`
- `gameDesign.cpp`
- `main.cpp`
- `mainMap.cpp`
- `mapRender.cpp`
- `maze.cpp`
- `rat.cpp`
- `ratInstance.cpp`
- `ratSelection.cpp`

Além da estrutura de pastas, a aplicação possui um Makefile para criar e executar o projeto em um sistema baseado em Linux. Esse arquivo contém instruções para que a aplicação seja compilada e produza o executável.

Para o ambiente Windows, existem dois arquivos de solução: `LabirintoRatoCego.sln` e `CriadorLabirinto/CriadorLabirinto.sln`. Esses arquivos são destinados ao Microsoft Visual Studio, permitindo a compilação da aplicação, o gerenciamento das configurações de compilação e dependências, e execução dos componentes Interface do Labirinto do Rato Cego e Criador de Labirintos.

Além disso no diretório `RatoCego/` está o arquivo `ratoCego.cpp` que é utilizado no componente Rato Cego, para criar o arquivo de movimentação do rato através da comunicação com o código do jogador.

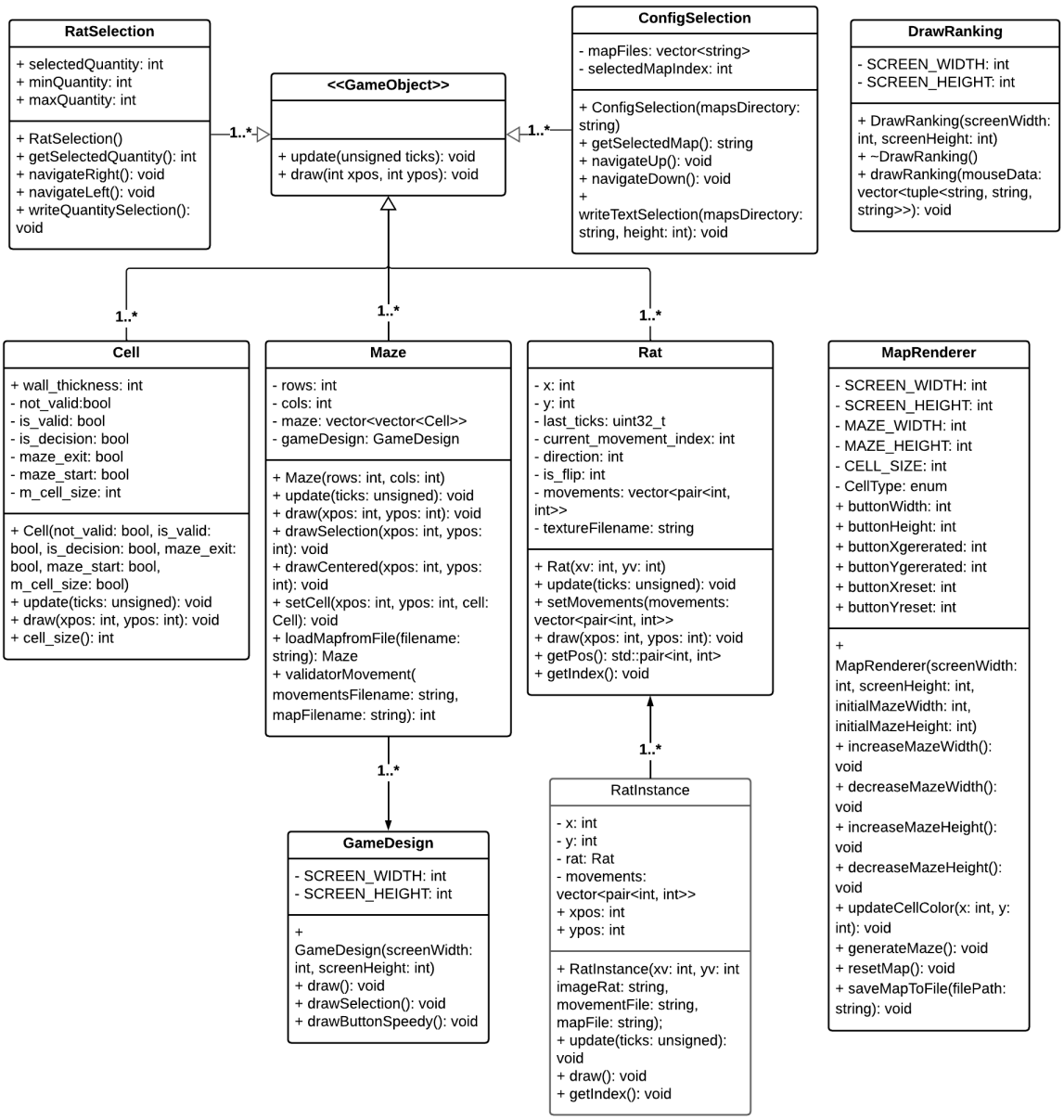
Para orientações detalhadas sobre a execução do Labirinto do Rato Cego, consulte o Apêndice A.

2.6 Diagrama de Classes

Na Figura 6, apresenta-se o diagrama de classes do Labirinto do Rato Cego, o qual é composto por 10 classes distintas. Essas classes desempenham papéis específicos no sistema, e o diagrama oferece uma representação visual da estrutura e das relações entre as elas, proporcionando uma compreensão mais aprofundada da arquitetura do Labirinto do Rato Cego.

A classe `GameObject`, definida em `gameObject.h`, é uma classe base abstrata que fornece uma interface para objetos do jogo. Ela contém duas funções virtuais: `update()`

Figura 6 – Diagrama de Classes.



Fonte: o Autor.

e `draw()`. A função `update()` é responsável por sincronizar e atualizar o estado do jogo, enquanto a função `draw()` é usada para renderizar o objeto na tela.

A classe `Cell`, definida no arquivo `cell.h`, encapsula a representação de uma célula em um jogo, incorporando propriedades distintas que a categorizam como válida, inválida, decisão, saída de labirinto ou ponto de início do labirinto.

A classe `ConfigSelection`, proveniente de `configSelection.h`, gerencia a seleção de mapas e a escolha do arquivo de movimentação para cada rato no jogo.

A classe `DrawRanking`, encontrada em `drawRanking.h`, tem como responsabilidade

desenhar o ranking na tela com base nos dados fornecidos. Esses dados incluem o nome do rato, o número de movimentos e a imagem associada a cada rato.

A classe `GameDesign`, presente em `gameDesign.h`, é responsável por desenhar as telas de escolha do labirinto e a tela onde ocorre a movimentação dos ratos no mapa.

A classe `MapRenderer`, definida em `mapRender.h`, é responsável por criar toda a tela e configuração do componente Criador de Labirintos.

A classe `Maze`, declarada em `maze.h`, representa um labirinto dentro do jogo. A classe gerencia a estrutura geral do labirinto, mantendo uma matriz de objetos `Cell`. Ela inclui funções para atualizar e desenhar o labirinto, bem como para carregar um labirinto de um arquivo.

A classe `Rat`, definida no arquivo `rat.h`, representa o personagem dentro do labirinto. Ela define a lógica de posição e movimento do rato de um jogador.

A classe `RatInstance`, definida em `ratInstance.h`, representa uma instância específica de um objeto rato no jogo. Armazena informações sobre a posição do rato, sua imagem, movimentos e o mapa associado, permitindo a renderização simultânea de vários ratos na tela.

A classe `RatSelection`, definida no arquivo `ratSelection.h`, lida com a seleção da quantidade de ratos no jogo, sendo também responsável por desenhar a tela de seleção correspondente.

3 Resultados

O Labirinto do Rato Cego é um trabalho composto por três componentes: a Interface do Labirinto do Rato Cego, o Rato Cego e o Criador de Labirintos. Neste capítulo será realizada uma análise detalhada dos resultados obtidos no desenvolvimento de cada um desses componentes, visando destacar as funcionalidade individuais de cada parte do projeto e a sinergia entre elas. Ao final são descritos os principais usos dos componentes da biblioteca SDL2.

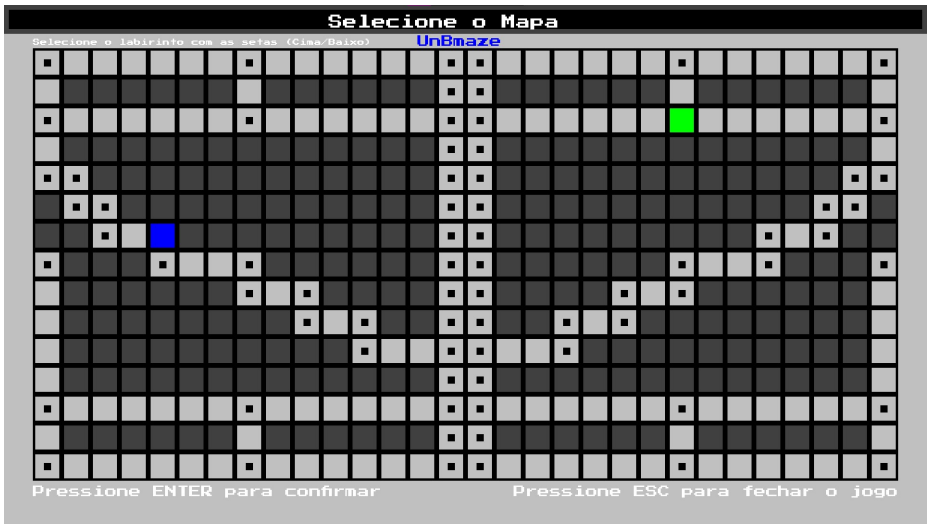
3.1 Interface do Labirinto do Rato Cego

A interface do Labirinto do Rato Cego proporciona aos jogadores uma visualização gráfica para selecionar configurações pré-jogo, como o labirinto, a quantidade de jogadores e o arquivo de movimento de cada rato. Por meio dessa interface, os jogadores também acompanham a movimentação de todos os ratos no labirinto, tentando encontrar a saída, e conferem o *ranking* final com a colocação de cada participante.

3.1.1 Escolha do labirinto

A Figura 7 mostra a primeira tela que os usuários veem ao executar a interface do Labirinto do Rato Cego, que é a tela de seleção de mapas. Nela, os jogadores podem selecionar entre os diversos labirintos disponíveis, visualizando tanto o nome quanto o desenho correspondente.

Figura 7 – Tela de seleção do labirinto.



Fonte: o Autor.

A escolha do mapa é feito por meio da classe `ConfigSelection`, que gerencia o acesso a arquivos dentro de um diretório dando, nesse caso, acesso a lista de mapas disponíveis no diretório `./assets/maps`. A classe utiliza a biblioteca `filesystem` do C++ para percorrer os arquivos no diretório e armazenar os caminhos completos dos mapas em um vetor chamado `mapFiles`. O Código 1 mostra como o construtor da classe popula a lista de mapas automaticamente, e o índice selecionado é inicializado como 0, representando o primeiro mapa na lista.

Código 1 – Construtor da classe `ConfigSelection` populando o vetor `mapFiles`.

```
1  ConfigSelection::ConfigSelection(const std::string &mapsDirectory)
2  {
3      for (const auto &entry : fs::directory_iterator(mapsDirectory))
4      {
5          if (entry.is_regular_file())
6          {
7              mapFiles.push_back(entry.path().string());
8          }
9      }
10
11     selectedMapIndex = 0;
12 }
```

Fonte: o Autor.

Durante a execução do programa, a tela de seleção de mapas é exibida em um *loop*, permitindo que o jogador navegue pelos mapas usando as teclas de seta para cima e para baixo. O nome do mapa selecionado e um *preview* do labirinto correspondente são exibidos na tela. O jogador pode confirmar a seleção pressionando a tecla `Enter` ou fechar o jogo pressionando a tecla `Esc`.

3.1.2 Seleção da quantidade de ratos e do arquivo de movimentação

Após a escolha do mapa, os jogadores realizam a seleção da quantidade de ratos no labirinto, variando de um a seis, e dos arquivos de movimentação de cada rato no Labirinto do Rato Cego.

A classe `RatSelection` gerencia essa interação, fornecendo métodos para navegar entre as opções disponíveis. Os jogadores podem aumentar ou diminuir a quantidade de ratos pressionando as teclas de seta para direita e para esquerda, respectivamente. A quantidade selecionada é exibida visualmente na tela, proporcionando *feedback* imediato ao jogador.

A Figura 8 ilustra a primeira parte de escolha da quantidade de ratos. A informação visual é apresentada no centro da tela, indicando a quantidade atual de ratos selecionada. Instruções adicionais são exibidas na parte inferior da tela, orientando o jogador sobre como confirmar a seleção pressionando a tecla `Enter` ou fechar o jogo utilizando a tecla `Esc`.

Figura 8 – Tela de seleção da quantidade de ratos.

Fonte: o Autor.

Uma vez que a quantidade de ratos é definida, o processo de seleção de arquivos de movimentação para cada rato é iniciado. Cada rato é associado a um arquivo de movimentação que contém as instruções para sua navegação no labirinto. Durante esse processo, o jogador pode percorrer a lista de arquivos de movimentação disponíveis no diretório `./assets/movements` usando as teclas de seta para cima e para baixo. O arquivo de movimentação selecionado para cada rato é exibido na tela, proporcionando uma visão clara das escolhas feitas.

A Figura 9 mostra a segunda parte de seleção de arquivos de movimentação. O jogador pode confirmar a escolha pressionando `Enter` ou fechar o jogo pressionando `Esc`.

Durante a iteração sobre os ratos e a subsequente seleção dos arquivos de movimentação, o programa armazena os caminhos completos dos arquivos escolhidos no vetor `movementFiles`. Cada iteração acrescenta um novo elemento ao vetor, refletindo a escolha específica para aquele rato em particular. Dessa forma, ao final do processo de seleção, o vetor `movementFiles` contém os caminhos de todos os arquivos de movimentação escolhidos para cada rato.

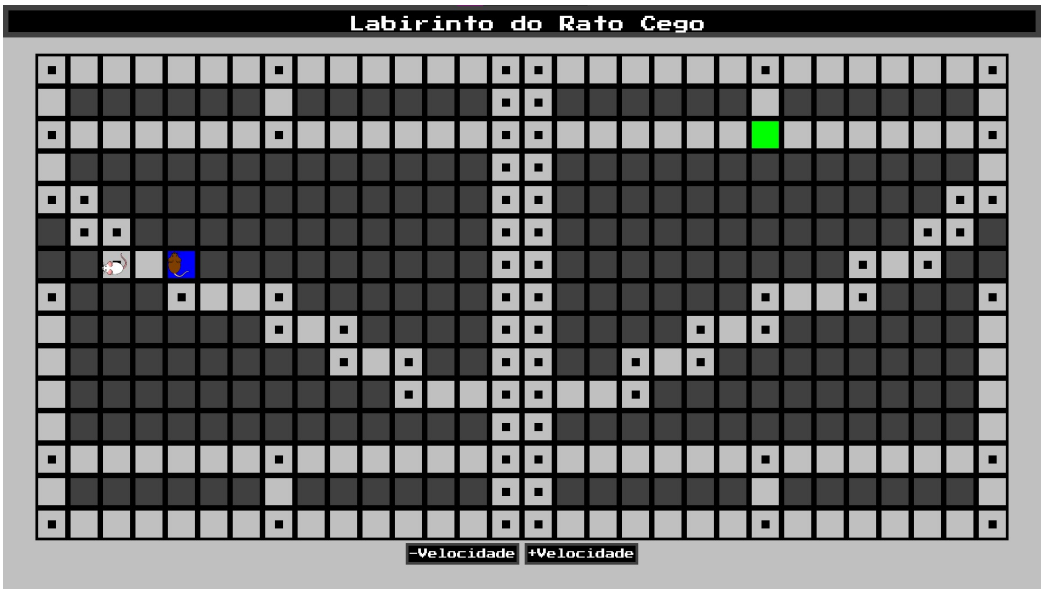
Figura 9 – Tela de seleção do arquivo do movimentação de cada rato.

Fonte: o Autor.

3.1.3 Movimentação dos ratos no labirinto

Com as escolhas de quantidade de ratos e movimentos realizadas, os jogadores visualizam todos os ratos movendo-se simultaneamente no labirinto, partindo do mesmo ponto de início (bloco azul) até alcançarem a saída (bloco verde) ou atingirem o limite de 999 movimentos. A Figura 10 mostra dois ratos em movimento, apresentando também botões na parte inferior para ajustar a velocidade do movimento.

Figura 10 – Movimentação simultânea de dois ratos no labirinto.



Fonte: o Autor.

No início do jogo, é carregado o mapa selecionado por meio da classe `Maze`. Esta classe é responsável por representar e desenhar o labirinto na interface gráfica.

Os ratos no jogo são representados como instâncias da classe `RatInstance`, sendo cada uma delas associada a uma imagem visual, um arquivo de movimentação e o caminho completo do mapa selecionado. Durante a fase de inicialização do jogo, o programa realiza iterações sobre os arquivos de movimentação previamente escolhidos. Nesse processo, são lidos o nome do rato, o número total de movimentos e as coordenadas iniciais (linha e coluna) do rato no labirinto.

Com base nessas informações, uma nova instância da classe `RatInstance` é criada e adicionada ao vetor `rats`. O Código 2 ilustra como cada nova instância do rato é gerada e incorporada ao vetor `rats`. Além disso, o código demonstra a inclusão das informações, como o nome do rato, a quantidade de movimentos e a imagem do rato, em um vetor de tuplas. Esse vetor de tuplas será utilizado para estabelecer o *ranking* no final do jogo.

Código 2 – Criação das instâncias do rato.

```
1  for (size_t i = 0; i < movementFiles.size(); ++i) {
2      std::ifstream movementsFile(movementFiles[i]);
3
4      std::string ratName;
5      int n;
6      int col, row;
7
8      movementsFile >> ratName;
9      movementsFile >> n;
10     movementsFile >> row >> col;
11
12     RatInstance newRat(col, row, ratImages[i], movementFiles[i],
   ↪   mapSelection.getSelectedMap());
13     rats.push_back(newRat);
14
15     mouseData.push_back(std::make_tuple(ratName, std::to_string(n),
   ↪   ratImages[i]));
16
17     movementsFile.close();
18 }
```

Fonte: o Autor.

A lógica dos movimentos do rato é derivada a partir do arquivo de movimentação associado a cada indivíduo. Cada arquivo compreende o nome do rato, o número total de movimentos realizados e os pares de coordenadas que representam esses movimentos. O primeiro par de coordenadas indica o ponto inicial no labirinto. O Código 3 apresenta

a função `loadMovementsFromFile()` da classe `RatInstance`, responsável pela leitura do arquivo de movimentação do rato e pelo armazenamento dos movimentos em um vetor de pares.

Código 3 – Função de leitura do arquivo de movimentação do rato.

```
1  std::vector<std::pair<int, int>> RatInstance::LoadMovementsFromFile(const
   ↪  std::string& movementsFilename)
2  {
3      std::ifstream movementsFile(movementsFilename);
4      if (!movementsFile.is_open())
5      {
6          throw std::invalid_argument("Failed to open movements file: " +
   ↪  movementsFilename);
7      }
8
9      std::string ratName;
10     int n;
11
12     movementsFile >> ratName;
13     movementsFile >> n;
14
15     std::vector<std::pair<int, int>> movements;
16     int col, row;
17     for (int i = 0; i < n; i++) {
18         movementsFile >> row >> col;
19         movements.push_back(std::make_pair(col, row));
20     }
21
22     movementsFile.close();
23     return movements;
24 }
```

Fonte: o Autor.

Com o armazenamento dos movimentos em um vetor de pares, a movimentação dos ratos é controlada pelo método `update()` da classe `RatInstance`, que é chamado repetidamente dentro do *loop* principal do jogo. Este método, por sua vez, faz uso da classe `Rat`, na qual cada rato é representado visualmente. A classe `Rat` contém a lógica para para executar os movimentos definidos no vetor de pares.

O Código 4 mostra a função `update()` da classe `Rat`, responsável por atualizar a posição do rato com base nas coordenadas de movimentação. A função verifica se há mais movimentos a serem processados e, em caso afirmativo, calcula o tempo decorrido desde a última atualização. Se esse tempo ultrapassar 1000 milissegundos, a posição do rato é

atualizada para a próxima coordenada na lista de movimentos. Além disso, a direção do rato é ajustada com base na diferença entre as coordenadas atual e nova, determinando se o rato está se movendo para a direita, esquerda, para cima ou para baixo. A variável `is_flip` é alternada para realizar a inversão do rato, cujo intuito é promover a sensação de movimento.

Código 4 – Função para atualizar a posição atual do rato no labirinto.

```
1  void Rat::update(unsigned ticks)
2  {
3      if (current_movement_index < movements.size()) {
4          if (!last_ticks) {
5              last_ticks = ticks;
6          } else if (ticks - last_ticks > 1000) {
7              int new_x = movements[current_movement_index].first;
8              int new_y = movements[current_movement_index].second;
9              is_flip = (is_flip) ? false : true;
10
11             // Calcular a direção com base nas coordenadas
12             int delta_x = new_x - x;
13             int delta_y = new_y - y;
14             if (delta_x > 0) {
15                 direction = 270; // Direita
16             } else if (delta_x < 0) {
17                 direction = 90; // Esquerda
18             } else if (delta_y > 0) {
19                 direction = 0; // Baixo
20             } else if (delta_y < 0) {
21                 direction = 180; // Cima
22             }
23
24             x = new_x;
25             y = new_y;
26
27             last_ticks = ticks;
28             current_movement_index++;
29         }
30     }
31 }
```

Fonte: o Autor.

A função `draw()` da classe `Rat` é responsável por renderizar visualmente cada rato na tela. A posição do rato é atualizada dinamicamente por meio da função `update()`, e o método `draw()` utiliza as coordenadas atualizadas para exibir a representação gráfica do rato na interface do Labirinto do Rato Cego.

3.1.3.1 Renderização do labirinto

O labirinto, onde ocorre a movimentação do rato, é renderizado pelas classes `Maze` e `Cell`.

A classe `Maze` é responsável por criar e manter a estrutura do labirinto, que é representada como uma matriz bidimensional de objetos `Cell`. Cada célula na matriz corresponde a uma unidade do labirinto e contém informações cruciais, como se é uma célula válida para navegação, se é uma decisão, se é uma saída ou se é o ponto de partida. A classe `Cell` define a aparência visual de cada célula, incluindo detalhes como a espessura das paredes, as cores associadas a diferentes tipos de células e a representação gráfica de blocos de decisões e pontos de início e de fim.

A construção do labirinto é executada por meio da função `loadMapfromFile()` na classe `Maze`, que lê as informações de um arquivo de mapa. Este arquivo contém dados como as dimensões do labirinto, o tamanho das células e os valores atribuídos a cada célula. A função interpreta esses dados, utilizando-os para gerar a representação interna do labirinto na forma de uma matriz de células.

Durante a leitura do arquivo, a função `loadMapfromFile()` itera sobre cada posição na matriz do labirinto, lendo o valor correspondente do arquivo e atribuindo as propriedades apropriadas à célula naquela posição. Por exemplo, se o valor lido for 0, a célula é marcada como não válida; se for 1, é marcada como válida; se for 2, é marcada como uma célula de decisão; se for 3, é marcada como uma saída; e se for 4, é marcada como o ponto de partida. As propriedades são então utilizadas para criar instâncias da classe mintinlinesmalltalkCell e atribuídas à posição correspondente na matriz do labirinto. O Código 5 mostra a definição da função.

Código 5 – Função `loadMapfromFile()` para a leitura e definição do labirinto.

```
1  Maze Maze::loadMapfromFile(const std::string &filename)
2  {
3      std::ifstream file(filename);
4      if (!file.is_open())
5      {
6          throw std::invalid_argument("Failed to open file: " + filename);
7      }
8      int rows, cols, cell_size;
9      file >> cols >> rows >> cell_size;
10     Maze maze(rows, cols);
11     for (int i = 0; i < rows; i++)
12     {
13         for (int j = 0; j < cols; j++)
14         {
15             int cellValue;
```

```
16              if (file >> cellValue)
17              {
18                  bool not_valid = false;
19                  bool is_valid = false;
20                  bool is_decision = false;
21                  bool maze_exit = false;
22                  bool maze_start = false;
23
24                  if (cellValue == 0)
25                      not_valid = true;
26                  if (cellValue == 1)
27                      is_valid = true;
28                  if (cellValue == 2)
29                      is_decision = true;
30                  if (cellValue == 3)
31                      maze_exit = true;
32                  if (cellValue == 4){
33                      maze_start = true;
34                  }
35
36                  maze.setcell(i, j, Cell(not_valid, is_valid, is_decision,
    ↪  maze_exit, maze_start, cell_size));
37              }
38              else
39              {
40                  file.close();
41                  throw std::invalid_argument("Invalid cell value in file: " +
    ↪  filename);
42              }
43          }
44      }
45
46      file.close();
47      return maze;
48  }
```

Fonte: o Autor.

O labirinto é composto por cinco tipos diferentes de células. A **célula de início** do Labirinto, destacada pela cor azul, representa o ponto no qual a exploração do labirinto é iniciada. É o ponto de partida para jogadores e seus algoritmos de percurso.

Por outro lado, a **célula de saída** do Labirinto, identificada pela cor verde, é o objetivo final. Ao alcançar essa célula, os desafiantes e seus programas de navegação atingem o sucesso no labirinto, completando o percurso.

As **células válidas**, representadas pela cor cinza claro, compõem o caminho percorrido no labirinto. Jogadores podem atravessar essas células, utilizando-as para navegar pelo labirinto e avançar em direção à saída.

As **células não válidas**, coloridas em cinza escuro, indicam posições inválidas no labirinto. Elas representam áreas intransponíveis, criando obstáculos que devem ser contornados durante a exploração.

Por fim, as **células de decisão** apresentam uma característica única. Com a cor cinza claro e um quadrado preto no centro, essas células representam estados de decisão no labirinto. Elas não só fazem parte do caminho percorrido, mas também indicam pontos de escolha, onde o algoritmo dos jogadores deverão tomar decisões de qual direção eles querem que o rato se mova.

A Figura 11 exemplifica todos os tipos de células que existem no labirinto.

Figura 11 – Células do labirinto na ordem: não válida, válida, decisão, saída, início.

Fonte: o Autor.

3.1.3.2 Renderização do rato

A função `draw()`, da classe `Rat`, é responsável por renderizar a imagem do rato na tela. Os parâmetros `xpos` e `ypos` indicam as coordenadas onde o rato será desenhado. Dentro da função, são definidas constantes para a largura (`ratWidth`) e altura (`ratHeight`) da imagem do rato.

Em seguida, a variável `flip` é inicializada como `SDL_FLIP_NONE`, que representa nenhuma inversão na renderização. Se a condição `is_flip` for verdadeira, indicando que o rato deve ser invertido horizontalmente, `flip` é ajustado para `SDL_FLIP_HORIZONTAL`.

A textura do rato é carregada usando a função `engine::loadTexture()`, que retorna um `shared pointer` para a textura. Em seguida, um objeto `SDL_Rect` chamado `destRect` é criado, representando o retângulo de destino onde a imagem do rato será desenhada.

Finalmente, a função `SDL_RenderCopyEx()` é utilizada para renderizar a textura do rato no retângulo de destino. A direção e o centro de rotação são especificados pelos parâmetros `direction` e `center`, respectivamente. A variável `flip` é usada para aplicar a inversão horizontal, se necessário.

No arquivo `main.cpp` é declarado um vetor de *strings* denominado `ratImages`, o qual armazena os caminhos das imagens representativas de diferentes tipos de ratos. A

ordem dos elementos neste vetor determina a sequência em que os ratos estarão disponíveis para os jogadores. Dessa forma, o rato associado à primeira posição no vetor será atribuído ao primeiro jogador.

O jogo apresenta uma variedade de seis cores distintas para os ratos, todas escolhidas no site PNGWING[1]. É importante ressaltar que essas imagens estão licenciadas para uso exclusivamente não comercial, de acordo com os termos estabelecidos. A Figura 12 mostra todas as cores disponíveis no jogo. Cada jogador tem uma cor fixa: o jogador 1 é sempre representado pela cor branca, o jogador 2 pela cor marrom, o jogador 3 pela cor preta, o jogador 4 pela cor vermelha, o jogador 5 pela cor cinza e o jogador 6 pela cor bege.

Figura 12 – *Sprite* do rato com as 6 cores disponíveis no jogo.

Fonte: PNGWING.

Cada jogador é associado a uma cor específica, proporcionando uma identificação visual consistente ao longo das partidas. Essa atribuição de cores é fixa, garantindo uma experiência visual uniforme para os jogadores. Para maior clareza, a correspondência entre jogadores e cores é a seguinte: o jogador 1 é sempre representado pela cor branca, o jogador 2 pela cor marrom, o jogador 3 pela cor preta, o jogador 4 pela cor vermelha, o jogador 5 pela cor cinza e o jogador 6 pela cor bege.

3.1.4 *Ranking* do jogo

Ao término da movimentação dos ratos, é exibida a tela de *ranking* do jogo, apresentando de forma ordenada o nome de cada rato, a quantidade de passos realizados e a imagem representativa de cada jogador. A Figura 13 ilustra a tela de *ranking*, destacando em verde o participante que conquistou o primeiro lugar.

Como existe a possibilidade dos jogadores escolherem um arquivo de movimentação que não corresponda ao mapa, seja tentando acessar uma célula inválida ou fornecendo coordenadas fora dos limites do labirinto, a classe `Maze` incorpora a função `validatorMovement()` para realizar essa verificação. Esta função retorna `-1` ao detectar qualquer inconsistência entre a movimentação especificada e o layout do labirinto. Como resultado, o rato não aparece no mapa durante a movimentação, e sua posição no

[1] <https://www.pngwing.com>

Figura 13 – Tela de *ranking*.

Fonte: o Autor.

ranking é exibida na última posição em vermelho. A Figura 14 ilustra a tela de ranking exibindo um rato com movimentação inválida, destacando a identificação visual desse estado.

Figura 14 – Tela de *ranking* exibindo um rato com movimentação inválida.

Fonte: o Autor.

3.2 Rato Cego

O componente Rato Cego representa a comunicação entre o arquivo `ratoCego.cpp` e o código criado pelo jogador. O jogador, ao desenvolver seu código, utiliza o comando do Código 6 para realizar esse comunicação.

Código 6 – Comando para realizar a comunicação entre o `ratoCego.cpp` e o código criado pelo jogador.

```
1 codigo_jogador < tmpfifo | ./ratoCego "../assets/maps/mapa_escolhido.txt" >
  ↪ tmpfifo
```

Fonte: o Autor.

Este comando estabelece uma conexão entre o código do jogador, escrito em qualquer linguagem, e o código do Rato Cego (`ratoCego.cpp`), que é implementado em C++. A comunicação ocorre através de um canal de entrada e saída especial chamado FIFO (*First In*, *First Out*), representado pelo arquivo `tmpfifo`, que é criado usando o comando `$ mkfifo`.

Dividindo o comando do Código 6, cada parte significa:

- `código_jogador < tmpfifo`: O operador `<` é usado para redirecionar a entrada padrão do `código_jogador`. Isso significa que o código do jogador irá ler os dados do `tmpfifo` como sua entrada.
- `|` (`pipe`): Este operador conecta a saída padrão do `código_jogador` à entrada padrão do `./ratoCego`.
- `./ratoCego "../assets/maps/mapa_escolhido.txt"`: Este é o código do Rato Cego, que recebe como entrada o arquivo de texto que representa o labirinto.
- `> tmpfifo`: O operador `>` é usado para redirecionar a saída padrão do `./ratoCego` para o `tmpfifo`. Isso significa que as mensagens escritas pelo código do Rato Cego na saída padrão serão colocadas no FIFO.

Portanto, a comunicação ocorre da seguinte maneira: o código do Rato Cego escreve dados no FIFO e o código do jogador lê os dados do FIFO, e o que o código do jogador escreve é passado para o código do Rato Cego através do `|` (`pipe`). Isso cria uma forma de comunicação eficiente entre os dois processos, permitindo que eles troquem informações enquanto estão em execução. Essa abordagem é útil para que o jogador possa escrever o código para movimentar o rato em qualquer linguagem de programação, desde

que ela tenha funções que permita a leitura de dados da entrada padrão e a escrita de dados na saída padrão.

Durante a execução, o código do jogador interage dinamicamente com o código do Rato Cego, que mapeia o labirinto e trata as escolhas do usuário. Como resultado dessa interação, um arquivo de texto é gerado. Este arquivo contém informações essenciais, incluindo o nome do rato, a quantidade total de movimentos realizados e todas as coordenadas de movimentação ao longo do labirinto.

O Código 7 mostra a saída do *prompt* de comando após a interação dos dois códigos, do jogador e do Rato Cego, exibindo informações relevantes, como largura e altura do labirinto, número de pontos de decisão, coordenadas atuais, escolhas feitas pelo código do usuário e, ao final, o nome escolhido.

Código 7 – Exemplo de saída do *prompt* de comando após a interação do código do jogador e do código do Rato Cego.

```
1  ================================================
2  =           LABIRINTO DO RATO CEGO             =
3  ================================================
4
5  Largura: 5 - Altura: 5 - Tamanho da celula: 60
6  Numero de pontos de decisao: 7
7  Ponto Final: 4 4
8  ================================================
9
10 Posicao atual: 0 0
11 Possiveis direcoes: SE
12 Direcao escolhida: E
13 OK - Movendo para o proximo ponto
14 Posicao atual: 0 2
15 Possiveis direcoes: SEW
16 Direcao escolhida: E
17 OK - Movendo para o proximo ponto
18 Posicao atual: 0 4
19 Possiveis direcoes: SW
20 Direcao escolhida: S
21 OK - Movendo para o proximo ponto
22 Posicao atual: 2 4
23 Possiveis direcoes: NSW
24 Direcao escolhida: N
25 OK - Movendo para o proximo ponto
26 Posicao atual: 0 4
27 Possiveis direcoes: SW
28 Direcao escolhida: S
29 OK - Movendo para o proximo ponto
```

```
30 Posicao atual: 2 4
31 Possiveis direcoes: NSW
32 Direcao escolhida: S
33 OK - Movendo para o proximo ponto
34 ================================================
35 =        FIM DO LABIRINTO DO RATO CEGO       =
36 ================================================
37
38 O rato chegou ao ponto final em 13 passos.
39
40 Escolha o nome do rato (o nome nao deve conter espacos ou caracteres especiais)
41 Nome do rato: ratoAleatorio
```

Fonte: o Autor.

Já o Código 8 representa o arquivo de texto gerado depois da comunicação entre código do jogador e do código do Rato Cego, arquivo este que será usando na função `loadMovementsFromFile()` da classe `RatInstance`, apresentada no Código 3.

Código 8 – Exemplo de arquivo de texto gerado depois da comunicação entre o código do jogador e o código do Rato Cego.

```
1  ratoAleatorio
2  13
3  0 0
4  0 1
5  0 2
6  0 3
7  0 4
8  1 4
9  2 4
10 1 4
11 0 4
12 1 4
13 2 4
14 3 4
15 4 4
```

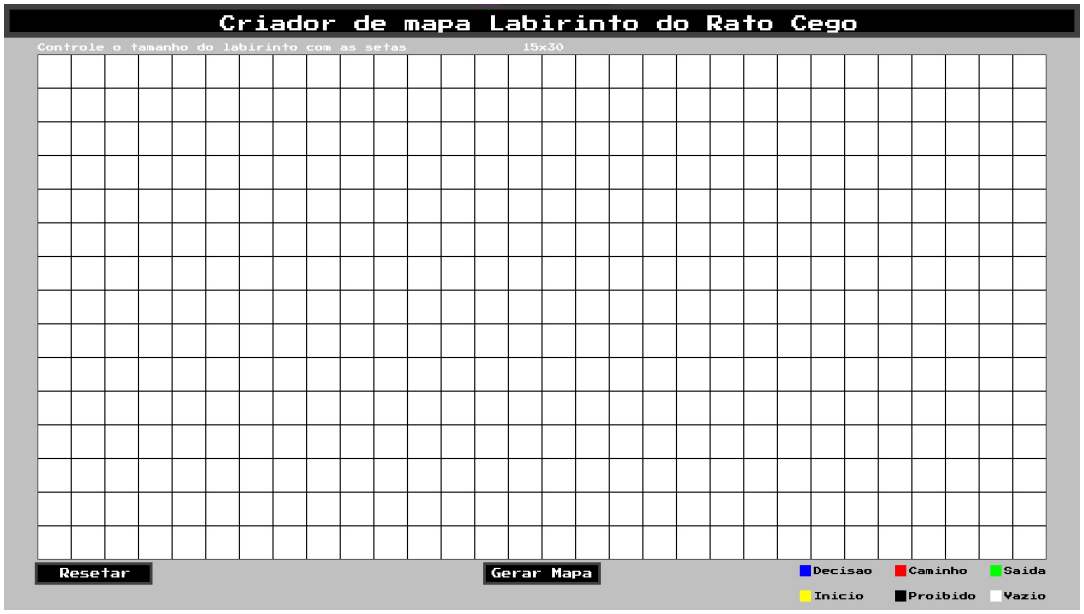
Fonte: o Autor.

No Apêndice B são apresentados os códigos de referência destinados a movimentar o rato de maneira aleatória. Esses códigos têm como objetivo oferecer suporte ao jogador na elaboração de seus próprios códigos.

3.3 Criador de Labirintos

Através do componente de criação de labirintos, definido na classe `MapRenderer`, os usuários podem elaborar seus próprios mapas. É possível selecionar a altura e largura do mapa e desenhar o labirinto, definindo pontos de decisão, caminhos, pontos de início e fim, e caminhos proibidos. As Figuras 15 e 16 mostram a criação no estado inicial, vazio, e um mapa elaborado, respectivamente. As figuras também mostram que existem os botões para gerar o mapa e resetar o mesmo, a legenda com o significado de cada cor e o texto informativo que o tamanho do mapa é controlado pelas setas do teclado.

Figura 15 – Criador de Labirintos vazio.

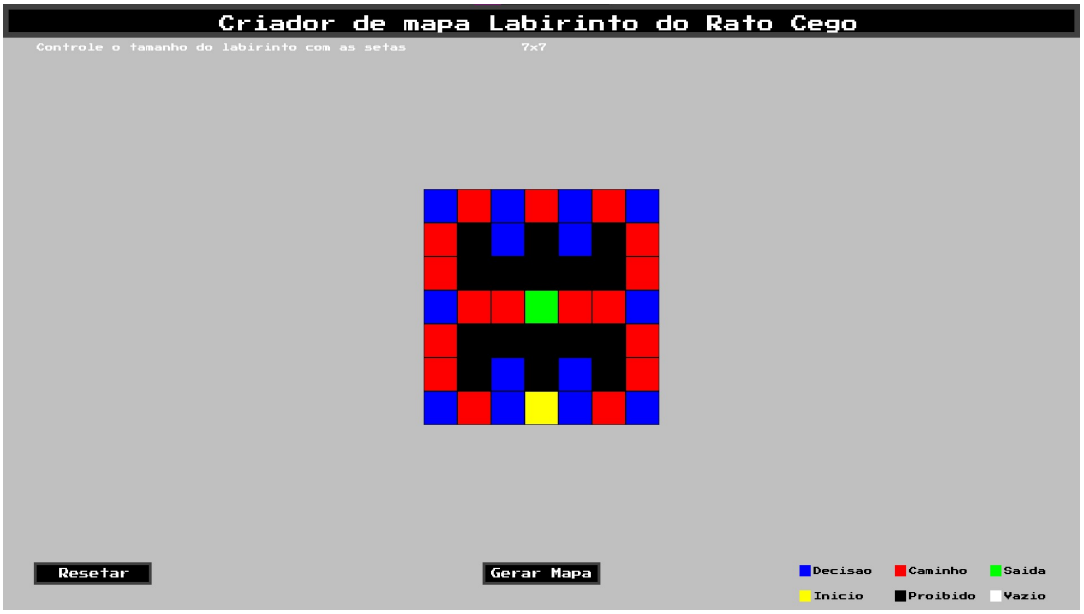


Fonte: o Autor.

Os elementos do mapa são representados numericamente no arquivo de saída, onde diferentes valores são atribuídos com base nos tipos de células do labirinto. Para facilitar a leitura e interpretação do arquivo, os valores numéricos são associados a categorias específicas, como caminhos proibidos ou vazios (0), caminhos válidos (1), pontos de decisão (2), saída (3) e ponto de início (4). Dentro do Criador de Labirintos, cada tipo de célula é associado a uma cor específica, e sua modificação ocorre através de cliques do *mouse* em cada uma delas.

Além da representação do mapa, o arquivo de saída inclui informações sobre a quantidade de pontos de decisão no labirinto, seguido pela especificação das coordenadas e direções associadas a cada ponto de decisão. Cada ponto de decisão é caracterizado pela sua posição (linha e coluna) e pelas direções em que é possível prosseguir a partir desse ponto (Norte, Sul, Leste, Oeste). Essa abordagem permite uma representação clara e detalhada da topologia do labirinto, destacando as escolhas disponíveis em cada ponto

Figura 16 – Exemplo do Criador de Labirintos preenchido.



Fonte: o Autor.

de decisão.

No processo de especificação das coordenadas e direções associadas a cada ponto de decisão, o código realiza uma análise da disposição do labirinto. Ao encontrar uma célula marcada como um ponto de decisão ou ponto de início, o sistema salva não apenas as coordenadas dessa célula, mas também determina as direções disponíveis a partir desse ponto.

Para cada ponto de decisão ou ponto de início identificado, o código realiza uma verificação em todas as direções principais: Norte, Sul, Leste e Oeste. Ele examina as células adjacentes na vertical e horizontal, procurando por caminhos válidos que possam ser seguidos a partir do ponto em questão. Se um caminho válido é encontrado em uma determinada direção, a letra correspondente a essa direção (N para Norte (*North*), S para Sul (*South*), E para Leste (*East*), e W para Oeste (*West*)) é registrada no arquivo de saída.

Essa abordagem cria uma representação textual das opções disponíveis em cada ponto de decisão, fornecendo informações sobre as direções acessíveis a partir desse ponto específico no labirinto. Por exemplo, se um ponto de decisão estiver localizado em determinada posição, e existirem caminhos disponíveis para o Norte e Leste, o arquivo de saída refletirá essas opções com as letras “N”e “E”associadas a esse ponto.

O Código 9 representa um trecho do método `saveMapToFile()`, presente na classe `MapRenderer`. Este trecho é estruturado para percorrer cada linha e coluna da representação matricial do labirinto, dada por `mapMaze[row][col]`. Ao encontrar uma célula de

decisão ou o ponto de partida, o código registra as coordenadas (`row`, `col`) e, em seguida, determina as direções possíveis a partir desse ponto.

Código 9 – Parte do método `saveMapToFile()` que realiza a especificação das direções associadas a cada ponto de decisão.

```
1  for (int row = 0; row < MAZE_HEIGHT; ++row) {
2      for (int col = 0; col < MAZE_WIDTH; ++col) {
3          if (mapMaze[row][col] == CELL_DECISION || mapMaze[row][col] ==
   ↪  CELL_START) {
4              file << row << " " << col << ' ';
5
6              for (int i = row - 1; i >= 0; --i) {
7                  if (mapMaze[i][col] == CELL_EMPTY || mapMaze[i][col] ==
   ↪  CELL_FORBIDDEN) break;
8                  if (mapMaze[i][col] == CELL_DECISION || mapMaze[i][col] ==
   ↪  CELL_EXIT || mapMaze[i][col] == CELL_START)
9                  {
10                     file << "N";
11                     break;
12                 }
13             }
14
15             for (int i = row + 1; i < MAZE_HEIGHT; ++i) {
16                 if (mapMaze[i][col] == CELL_EMPTY || mapMaze[i][col] ==
   ↪  CELL_FORBIDDEN) break;
17                 if (mapMaze[i][col] == CELL_DECISION || mapMaze[i][col] ==
   ↪  CELL_EXIT || mapMaze[i][col] == CELL_START)
18                 {
19                     file << "S";
20                     break;
21                 }
22             }
23
24             for (int j = col + 1; j < MAZE_WIDTH; ++j) {
25                 if (mapMaze[row][j] == CELL_EMPTY || mapMaze[row][j] ==
   ↪  CELL_FORBIDDEN) break;
26                 if (mapMaze[row][j] == CELL_DECISION || mapMaze[row][j] ==
   ↪  CELL_EXIT || mapMaze[row][j] == CELL_START)
27                 {
28                     file << "E";
29                     break;
30                 }
31             }
32
33             for (int j = col - 1; j >= 0; --j) {
```

```
34                 if (mapMaze[row][j] == CELL_EMPTY || mapMaze[row][j] ==
   ↪  CELL_FORBIDDEN) break;
35                 if (mapMaze[row][j] == CELL_DECISION || mapMaze[row][j] ==
   ↪  CELL_EXIT || mapMaze[row][j] == CELL_START)
36                 {
37                     file << "W";
38                     break;
39                 }
40             }
41
42             file << '\n';
43         }
44     }
45 }
```

Fonte: o Autor.

O Código 10 representa um exemplo de arquivo de texto gerado pelo Criador de Labirintos. O arquivo inicia com a especificação das dimensões do labirinto, indicando a largura, altura e o tamanho das células. Em seguida, segue-se a representação numérica do mapa, em que cada célula é mapeada para um valor numérico correspondente ao seu tipo. O arquivo também inclui a contagem total de pontos de decisão no labirinto, seguida pela descrição detalhada de cada ponto de decisão acompanhada das direções. Por fim, o arquivo finaliza com as coordenadas do ponto de saída do labirinto.

Código 10 – Arquivo de texto exemplo gerado pelo Criador de Labirintos.

```
1  5 5 60
2  4 1 2 1 2
3  1 0 1 0 1
4  2 1 2 1 2
5  1 0 1 0 1
6  2 1 2 0 3
7  7
8  0 0 SE
9  0 2 SEW
10 0 4 SW
11 2 0 NSE
12 2 2 NSEW
13 2 4 NSW
14 4 0 NE
15 4 2 NW
16 4 4
```

Fonte: o Autor.

O arquivo gerado pelo Criador de Labirintos é usado no componente Rato Cego e na função definida no Código 5.

3.4 Biblioteca SDL2

O código presente no arquivo `engine.cpp` contém as funções relacionadas aos mecanismos do jogo, a inicialização e ao gerenciamento de gráficos usando a biblioteca SDL2.

O `namespace engine` encapsula a funcionalidade do mecanismo de jogo. Ele inclui diversas variáveis estáticas e funções para controlar a janela, o renderizador e o *cache* de textura.

A duas primeiras funções definidas são a `init()` e a `close()`. A primeira é responsável pela inicialização da biblioteca SDL, pela criação de uma janela e pela configuração de um renderizador, enquanto que a segunda é usada para limpar e liberar recursos antes de encerrar o jogo. Ela destrói a janela e o renderizador, define seus ponteiros como nulos e chama as funções de limpeza da SDL e SDL_image.

O Código 11 implementa a função `loadTexture()`, que é importante para renderizar imagens, pois ela evita a múltipla carga de texturas. Nesse sentido, a função primeiro verifica se a textura solicitada já está carregada e armazenada em *cache*. Em caso afirmativo, ela retorna a textura armazenada. Caso contrário, ela carrega a imagem usando a biblioteca SDL_image, cria uma `SDL_Texture` a partir da textura carregada, armazena-a no *cache*, que é um dicionário do tipo `unordered_map`, para uso futuro, e a retorna a textura como um `shared_pointer`, que é um ponteiro inteligente que fornece gerenciamento automático de memória para objetos alocados dinamicamente.

Código 11 – Textura sendo retornada como `shared_pointer`.

```
1  std::shared_ptr<SDL_Texture> loadTexture(const std::string&filename)
2  {
3      // Check if the texture is already loaded
4      auto it = gTextureCache.find(filename);
5      if (it != gTextureCache.end()) {
6          return it->second;
7      }
8
9      // Load the texture
10     SDL_Surface* loadedSurface = IMG_Load(filename.c_str());
11     if (!loadedSurface) {
12         std::cerr << "Failed to load texture image '" << filename << "'!
   ↪  SDL_image Error: " << IMG_GetError() << '\n';
13         return nullptr;
```

```
14      }
15
16      // Create texture from surface
17      SDL_Texture* texture = SDL_CreateTextureFromSurface(gRenderer,
    ↪   loadedSurface);
18      if (!texture) {
19          std::cerr << "Failed to create texture from surface! SDL Error: " <<
    ↪   SDL_GetError() << '\n';
20          SDL_FreeSurface(loadedSurface);
21          return nullptr;
22      }
23
24      // Store the texture in the cache
25      std::shared_ptr<SDL_Texture> sharedTexture(texture, SDL_DestroyTexture);
26      gTextureCache[filename] = sharedTexture;
27
28      // Free the loaded surface
29      SDL_FreeSurface(loadedSurface);
30
31      return sharedTexture;
32  }
```

Fonte: o Autor.

Para a renderização de textos, é empregado a função `renderText()`. Essa função desempenha o papel de exibir textos na tela, fazendo uso da biblioteca de renderização de fontes `SDL_ttf`. É importante destacar que, para manter uma consistência visual, foi estabelecido o padrão de utilizar a fonte `PressStart2P-Regular.ttf`, obtida no Google Fonts[2], disponibilizada sob a licença *Open Font License*, que permite a livre utilização, modificação e distribuição da fonte, seja de forma pessoal ou comercial.

Ainda dentro do **`namespace engine`**, existe os também o **`namespace draw`** e o **`namespace screen`**. O primeiro contém funções de desenho, sendo a função `rect()` usada para desenhar um retângulo preenchido a partir da posição, da largura, da altura e da cor especificada, e a função `clearRect()`, usada para desenhar um retângulo preenchido com uma cor branca, limpando a área especificada. O segundo *namespace* fornece funções relacionadas ao gerenciamento da tela. A função `clear()` define a cor de desenho do renderizador como branca e limpa a tela inteira, e a função `show()` atualiza a tela apresentando o conteúdo renderizado, sendo que a primeira função é chamada no início e a segunda é chamada no final de cada quadro.

[2] <https://fonts.google.com/specimen/Press+Start+2P>

4 Considerações Finais

Através da pesquisa e desenvolvimento deste trabalho, foi possível perceber o potencial do Labirinto do Rato Cego como uma ferramenta educacional para estimular a aprendizagem ativa e prática de programação. A própria trajetória do autor reflete a proposta do Rato Cego, já que, inicialmente, não possuía experiência prévia em desenvolvimento de jogos.

Como é comum no desenvolvimento de jogos, também houve algumas dificuldades ao longo do desenvolvimento. Uma das maiores barreiras foi a criação da tela de seleção de arquivos de movimentação para cada jogador. Lidar com a renderização de todos os ratos simultaneamente, cada um com seu arquivo de movimento específico, revelou-se um desafio técnico significativo também. Superar essas dificuldades exigiu uma abordagem criativa e aprendizado contínuo.

A implementação bem-sucedida da interface do jogo, a capacidade de movimentação simultânea de múltiplos ratos e a criação de labirintos são aspectos cruciais que enriquecem a experiência do usuário. A interação efetiva entre o código desenvolvido pelos jogadores e o código de mapeamento do labirinto possibilita uma aplicação prática e imersiva para o aprendizado de programação.

Diante do objetivo geral e específicos de desenvolvimento alcançados, a nova implementação do Labirinto do Rato Cego, utilizando SDL e C++, visa assegurar sua continuidade por mais tempo ao adotar tecnologias mais modernas em comparação com o jogo original, que foi desenvolvido em C e tinha a interface executada em Macromedia Flash MX Professional.

É importante ressaltar que o Labirinto do Rato Cego original não era apenas uma ferramenta para o ensino de habilidades técnicas. Ele também demostrou sucesso no desenvolvimento de competências cognitivas, como o raciocínio lógico e a criatividade, além de habilidades sociais, incentivando a colaboração e a troca de conhecimentos entre os participantes. Essa eficácia foi evidenciada pelo experimento conduzido pelo criador Fragelli em colaboração com Vainstein (FRAGELLI; VAINSTEIN, 2011). E com essa nova implementação, surge a oportunidade de testá-la em ambientes práticos com o intuito de reproduzir o mesmo sucesso alcançado pelo trabalho original.

4.1 Trabalho futuros

O desenvolvimento do Labirinto do Rato Cego proporcionou uma base sólida para futuras expansões e aprimoramentos. Diversos aspectos podem ser explorados para enri-

quecer ainda mais a experiência dos usuários e potencializar os benefícios educacionais do jogo.

Uma extensão natural do Labirinto do Rato Cego seria a implementação de um algoritmo de geração aleatória de labirintos. Isso permitiria que os desafios propostos aos jogadores fossem variados, oferecendo uma gama mais ampla de situações e problemas a serem resolvidos. A introdução de labirintos gerados aleatoriamente também contribuiria para aumentar a diversidade de estratégias e soluções desenvolvidas pelos alunos, enriquecendo o aprendizado.

Além disso, a adição de elementos e obstáculos aos labirintos poderia enriquecer significativamente o escopo do Labirinto do Rato Cego. Introduzir elementos como portais, áreas de teletransporte ou obstáculos que alteram dinamicamente as regras do jogo proporcionaria desafios mais complexos e estimularia a criatividade dos jogadores na elaboração de códigos mais sofisticados.

Além disso, a expansão do Labirinto do Rato Cego para múltiplas plataformas e ambientes poderia aumentar ainda mais sua acessibilidade e utilidade. Considerar versões online ou integrações com ambientes de aprendizagem virtual seriam passos valiosos para disseminar o uso do Labirinto do Rato Cego em diferentes contextos educacionais.

Referências

BARROWS, H.; TAMBLYN, R. M. *Problem-Based Learning: An Approach to Medical Education*. [S.l.]: Springer Publishing Company, 1980. (Springer Series on Medical Education). ISBN 9780826128423. Citado 2 vezes nas páginas 23 e 24.

BRANCO, C. D. C. *As 10 linguagens de programação mais utilizadas no desenvolvimento de games*. 2022. Disponível em: <https://canaltech.com.br/mercado/as-10-linguagens-de-programacao-mais-utilizadas-no-desenvolvimento-de-games-213047/>. Citado na página 34.

CHANG, C.; CHUNG, C.; CHANG, J. A. Influence of problem-based learning games on effective computer programming learning in higher education. In: *Educational Technology Research and Development*. [S.l.: s.n.], 2020. v. 68, p. 2615–2634. Citado na página 29.

DZHUROV, Y.; KRASTEVA, I.; ILIEVA, S. Personal extreme programming – an agile process for autonomous developers. In: . [s.n.], 2009. Disponível em: <https://api.semanticscholar.org/CorpusID:55590264>. Citado na página 35.

FARIAS, F. L. O. et al. Ensino de programação front-end através de jogos digitais: Um relato de experiência na escola de programação do lais/huol. In: *Anais do XXV Workshop de Informática na Escola*. [S.l.: s.n.], 2019. p. 187–196. Citado na página 29.

FRAGELLI, R. R.; VAINSTEIN, M. H. O labirinto do rato cego: Aprendizagem baseada em projeto em algoritmos e programação de computadores. In: *PAEE'2011 - Project Approaches in Engineering Education: Aligning Engineering Education with Engineering Challenges*. [S.l.: s.n.], 2011. p. 99–105. Citado 4 vezes nas páginas 24, 25, 26 e 63.

FULLERTON, T. *Game Design Workshop. A Playcentric Approach to Creating Innovative Games*. 2nd. ed. [S.l.]: Elsevier Morgan Kaufmann, 2008. ISBN 9780240809748. Citado na página 27.

HARDLESS, C.; NILSSON, M.; NULDéN, U. 'copernicus': Experiencing a failing project for reflection and learning. In: *Management Learning*. [S.l.: s.n.], 2005. p. 181–217. Citado na página 23.

HUIZINGA, J. *Homo ludens: O jogo como elemento da cultura*. 9th. ed. São Paulo: Perspectiva, 2019. ISBN 978-85-273-1170-0. Citado na página 27.

IMD, I. M. D. *Papéis e Processos no Desenvolvimento de Jogos Digitais*. 2016. Disponível em: <https://materialpublic.imd.ufrn.br/curso/disciplina/5/18/6/5>. Citado na página 27.

INCE, E. Y. Students' perceptions on learning programming with codingame. In: *International Journal of Technology in Teaching and Learning*. [S.l.: s.n.], 2021. v. 17, p. 38–46. Citado na página 30.

MOHANARAJAH, S.; SRITHARAN, T. Shoot2learn: Fix-and-play educational game for learning programming; enhancing student engagement by mixing game playing and game programming. In: *Journal of Information Technology Education:Research*. [S.l.: s.n.], 2022. v. 21, p. 639–661. Citado na página 30.

MORADI, M.; NOOR, N. F. B. M. The impact of problem-based serious games on learning motivation. In: *IEEE*. [S.l.: s.n.], 2022. v. 10, p. 8339–8349. Citado na página 30.

NATUCCI, G. C. et al. O uso do jogo robocode para desenvolvimento de carreiras em stem e habilidades do século xxi: um estudo de caso nacional. In: *IEEE*. [S.l.: s.n.], 2020. Anais do XXXI Simpósio Brasileiro de Informática na Educação, p. 362–371. Citado na página 30.

PRENSKY, M. Digital natives, digital immigrants. In: *On the Horizon*. [S.l.: s.n.], 2001. v. 9, p. 1–6. Citado 2 vezes nas páginas 21 e 27.

QUEIROZ, M. *Educação Gamificada Na Prática: Ricardo Fragelli*. 2021. Disponível em: <https://orcestra.com.br/2021/07/08/educacao-gamificada-na-pratica-ricardo-fragelli/>. Citado na página 21.

SALEN, K.; ZIMMERMAN, E. *Rules of Play: Game Design Fundamentals*. [S.l.]: MIT Press, 2003. (ITPro collection). ISBN 9780262240451. Citado na página 26.

ZUKALOUS, H. to Market a G. *WHAT GENRES ARE POPULAR ON STEAM IN 2022*. 2022. Disponível em: <https://howtomarketagame.com/2022/04/18/what-genres-are-popular-on-steam-in-2022/>. Citado na página 28.

Apêndices

APÊNDICE A – Executando o Labirinto do Rato Cego

O propósito deste apêndice é fornecer um guia para a execução dos componentes em ambientes Windows e sistemas baseados em Linux. Para alcançar esse objetivo, os dois primeiros passos essenciais são:

- **Instalação das Bibliotecas:** certifique-se de ter as bibliotecas necessárias instaladas: `SDL2`[1], `SDL2_ttf`[2] e `SDL2_image`[3,4].
- **Clone do Repositório no GitHub:** realize o clone do repositório disponível no GitHub através do link: IanFPFerreira/LabirintoRatoCego.

A.1 Sistemas baseados em Linux

A.1.1 Interface do Labirinto do Rato Cego

Passos para a execução do componente Interface do Labirinto do Rato Cego:

- **Passo 1:** estando no diretório raiz do repositório execute o makefile:

  ```
  $ make
  ```

- **Passo 2:** estando no diretório raiz do repositório execute o programa com o comando:

  ```
  $ ./RatoCego
  ```

A.1.2 Criador de Labirintos

Passos para a execução do componente Criador de Labirintos:

- **Passo 1:** estando no diretório raiz do repositório execute o makefile:

[1] Link ensinando a instalar e configurar o SDL2: <https://lazyfoo.net/tutorials/SDL/01_hello_SDL/index.php>
[2] <https://github.com/libsdl-org/SDL_ttf/releases>
[3] <https://github.com/libsdl-org/SDL_image/releases>
[4] Link ensinando a instalar e configurar bibliotecas adicionais (as bibliotecas usadas no Labirinto do Rato Cego seguem o mesmo guia): <https://lazyfoo.net/tutorials/SDL/06_extension_libraries_and_loading_other_image_formats/index.php>

```
$ make
```

- **Passo 2:** estando no diretório raiz do repositório execute o programa com o comando:

```
$ cd CriadorLabirinto && ../MapGenerator
```

A.1.3 Rato Cego

Passos para a execução do componente Rato Cego:

- **Passo 1:** acesse o diretório `/RatoCego` do repositório:

```
$ cd RatoCego/
```

- **Passo 2:** compile o arquivo `ratoCego.cpp` com o comando:

```
$ g++ ratoCego.cpp -o rc
```

- **Passo 3:** compile, se necessário, o código criado pelo jogador.
- **Passo 4:** crie o arquivo FIFO com o comando:

```
$ mkfifo tmpfifo
```

- **Passo 5:** estabeleça a comunicação entre o `ratoCego.cpp` e o código do jogador via `pipe` com o comando:

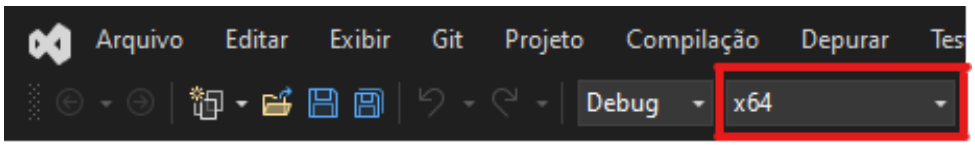
```
$ timeout 3s codigo_jogador < tmpfifo | ./rc ../assets/maps/mapa.txt > tmpfifo
```

A.2 Windows

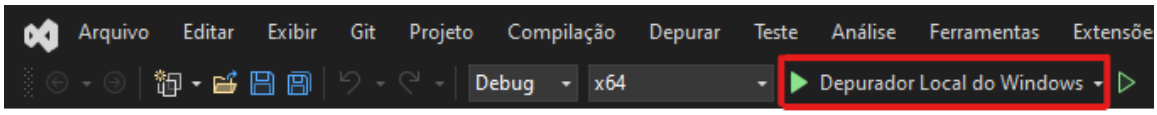
A.2.1 Interface do Labirinto do Rato Cego

Passos para a execução do componente Interface do Labirinto do Rato Cego:

- **Passo 1:** abra o projeto `LabirintoRatoCego.sln`, localizado na pasta raiz do repositório, utilizando o Visual Studio 2022.
- **Passo 2:** certifique-se de que a configuração de compilação está definida para x64.



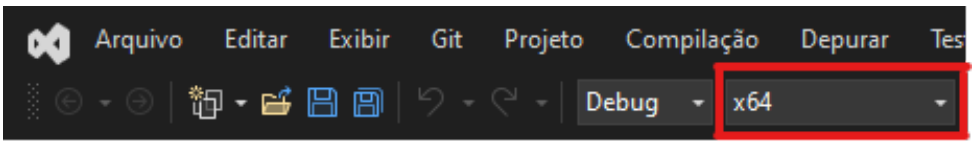
- **Passo 3:** use o Depurador Local do Windows (Local Windows Debugger) para construir e executar o jogo.



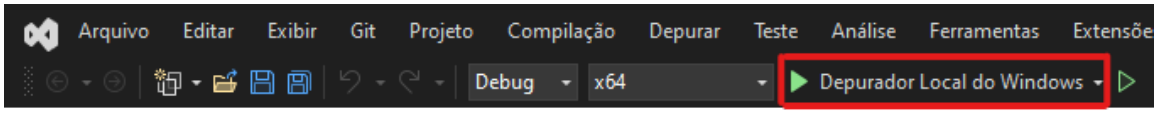
A.2.2 Criador de Labirintos

Passos para a execução do componente Criador de Labirintos:

- **Passo 1:** abra o projeto `CriadorLabirinto.sln`, encontrado no caminho `LabirintoRatoCego/CriadorLabirinto/` do repositório, no Visual Studio 2022.
- **Passo 2:** certifique-se de que a configuração de compilação está definida para x64.



- **Passo 3:** use o Depurador Local do Windows (Local Windows Debugger) para construir e executar o jogo.



A.2.3 Rato Cego

Passos para a execução do componente Rato Cego no Windows (necessário WSL2[5]):

- **Passo 1:** abra o terminal do WSL2.
- **Passo 2:** acesse o diretório `/RatoCego` do repositório:

```
$ cd RatoCego/
```

- **Passo 3:** compile o arquivo `ratoCego.cpp` com o comando:

```
$ g++ ratoCego.cpp -o rc
```

- **Passo 4:** compile, se necessário, o código criado pelo jogador.

[5] Link da própria Microsoft ensinando a configurar o WSL2: <https://learn.microsoft.com/pt-br/windows/wsl/install>

- **Passo 5:** crie o arquivo FIFO com o comando:

  ```
  $ mkfifo tmpfifo
  ```

  OBS.: Caso esse comando não funcione no terminal que está aberto o repositório, faça o comando no diretório raiz do WSL, acesse esse diretório com o comando:

  ```
  $ cd ~
  ```

  Para voltar para o diretório do jogo, o comando é:

  ```
  $ cd /mnt/letra_disco/caminho_para_o_repositorio/
  ```

  Criando a FIFO no diretório raiz, o caminho para acessá-la no passo 6 será:

  ```
  ~/tmpfifo
  ```

- **Passo 6:** estabeleça a comunicação entre o `ratoCego.cpp` e o código do jogador via `pipe` com o comando:

  ```
  $ timeout 3s codigo_jogador < tmpfifo | ./rc ../assets/maps/mapa.txt > tmpfifo
  ```

APÊNDICE B – Códigos base para o jogador movimentar o rato

Este apêndice foi criado para fornecer a estrutura de dois códigos funcionais em C++ e Python, projetados para gerar movimentos aleatórios no componente Rato Cego. Essas soluções servem como exemplos práticos para os desenvolvedores, permitindo que os jogadores evoluam o código a partir destes arquivos base.

B.1 Exemplo de implementação do rato em C++

Código 12 – Código base C++.

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    string options;
    srand(time(NULL));
    while (true) {
        cin >> options;

        if (options == "LOSE" || options == "WIN") {
            break;
        }

        cout << options[rand()%options.size()] << endl;

    }

    string ratName = "ratoAleatorioCPP";

    cout << ratName << endl;

    return 0;
}
```

Fonte: o Autor.

B.2 Exemplo de implementação do rato em Python

Código 13 – Código base Python.

```python
import random


while True:
    options = input()

    if options == "LOSE" or options == "WIN":
        break

    choice = random.randint(0, len(options) - 1)
    print(options[choice])


ratName = "ratoAleatorioPython"
print(ratName)
```

Fonte: o Autor.