CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

CAMPUS CONTAGEM - INFORMÁTICA II

Raphael Mendes Fernandes

Marcus Vinícius Coimbra

Túlio Dornelas

RELATÓRIO TÉCNICO REFERENTE AO TRABALHO

“Dots and boxes”

CONTAGEM - MG

2025

Raphael Mendes Fernandes
Marcus Vinícius Coimbra
Túlio Dornelas

Relatório técnico apresentado à disciplina de Laboratório de Linguagens e Técnicas de Programação I, do Centro Federal de Educação Tecnológica de Minas Gerais, ministrada pelo professor Alisson Rodrigo dos Santos.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Este trabalho tem como objetivo o desenvolvimento do jogo clássico **"Dots and Boxes"**, utilizando a linguagem C++ em conjunto com a biblioteca gráfica **SFML (Simple and Fast Multimedia Library)**. A proposta visa a consolidação de conceitos de **programação orientada a objetos**, além da aplicação prática de técnicas voltadas ao desenvolvimento de jogos digitais.

O projeto consiste na criação de um jogo interativo para dois jogadores — um humano e um bot — em que, a cada turno, os jogadores traçam linhas entre pontos com o intuito de formar quadrados. Quando um jogador fecha um quadrado, ele conquista aquele espaço e ganha um ponto. O jogo termina quando todas as linhas possíveis forem preenchidas, vencendo quem tiver maior pontuação.

Objetivos do projeto:

- Aplicar a Programação Orientada a Objetos (POO) para estruturar o sistema em classes modulares e reutilizáveis;
- Utilizar a biblioteca **SFML** para renderização gráfica, interação com o mouse e reprodução de sons;
- Implementar um **bot oponente** com lógica simples capaz de tomar decisões estratégicas;
- Criar um sistema visual de pontuação, controle de turno e tela de fim de jogo;
- Reforçar boas práticas de organização, clareza e legibilidade do código-fonte.

Este relatório apresenta o processo de desenvolvimento do projeto, a estrutura e funcionalidades implementadas, os testes realizados e as considerações finais a respeito dos resultados alcançados.

2. IMPLEMENTAÇÕES E INTENÇÕES

O desenvolvimento do projeto **Dots and Boxes** teve como base a aplicação dos princípios da programação orientada a objetos, com foco na clareza estrutural, organização do código e uso funcional da biblioteca gráfica **SFML**. Todas as

funcionalidades do jogo foram modeladas com classes específicas que representam os elementos principais da mecânica — como linhas, quadrados, jogadores e o tabuleiro — permitindo uma separação lógica e reutilizável do código.

A estrutura do jogo foi dividida em cinco classes centrais:

- **Linha**: responsável por representar as linhas interativas do tabuleiro, permitindo detectar o clique do jogador e alterar visualmente o estado da linha.
- **Quadrado**: representa a área entre quatro linhas; quando todas as suas linhas são preenchidas, o quadrado é atribuído ao jogador da vez.
- **Tabuleiro**: gerencia todas as linhas e quadrados, além de lidar com a atualização visual da interface.
- **Player**: armazena os dados do jogador (ou bot), como pontuação e controle dos quadrados conquistados.
- **Jogo**: concentra a lógica principal de funcionamento, alternância de turnos, chamadas ao bot, verificação de fim de jogo e gerenciamento da janela.

As intenções por trás da implementação foram:

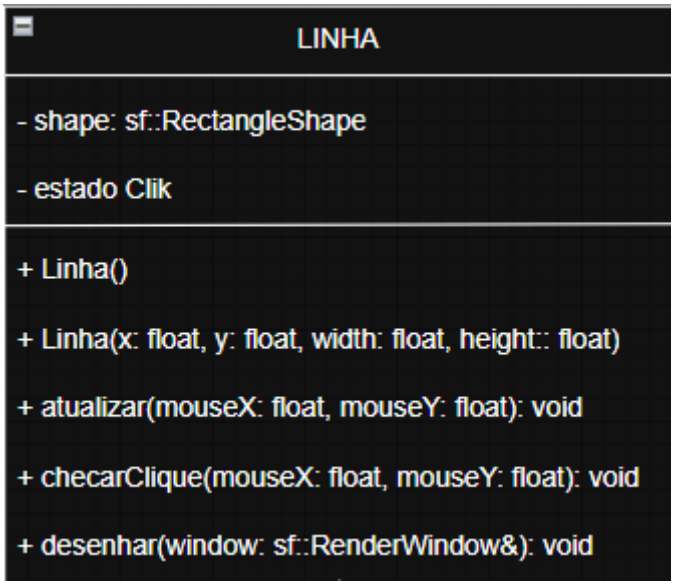
- **Modularidade**: permitir que cada classe tenha responsabilidades bem definidas, facilitando alterações futuras no comportamento do jogo ou no layout.
- **Interatividade fluida**: destacar visualmente as linhas ao passar o mouse e garantir resposta sonora para ações importantes (como traçar uma linha ou marcar ponto).
- **Desempenho simples e eficaz**: mesmo com uma lógica de bot básica, foi implementada uma estratégia funcional — o bot tenta completar quadrados e, se não for possível, escolhe uma linha aleatória.
- **Interface limpa**: com botões acessíveis, indicadores visuais de vitória/derrota, e um botão de reinício para recomeçar a partida com facilidade.
- **Experiência completa de jogo**: com ciclo de jogadas, controle de pontuação, som, lógica de IA e condição de término, o jogo está apto a ser jogado do início ao fim sem falhas estruturais.

O código foi desenvolvido com o propósito de ser facilmente compreendido por qualquer programador familiarizado com C++ e SFML, demonstrando o uso

de técnicas de encapsulamento, ponteiros, composição e atualização visual em tempo real.

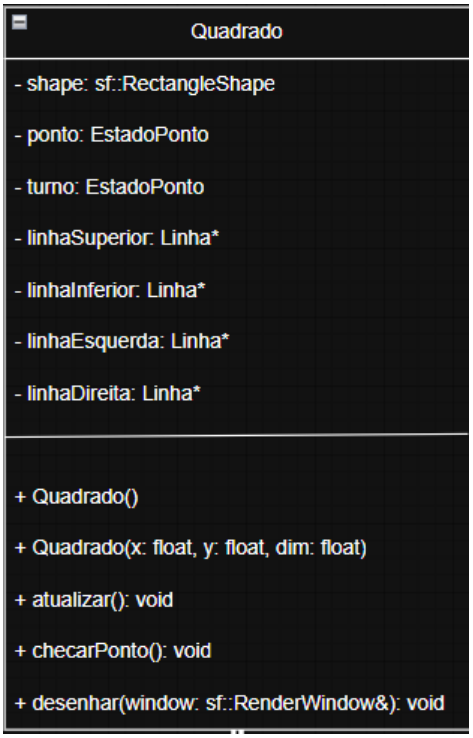
2.1 Estrutura de Classes

Figura 1 – UML da classe Linha



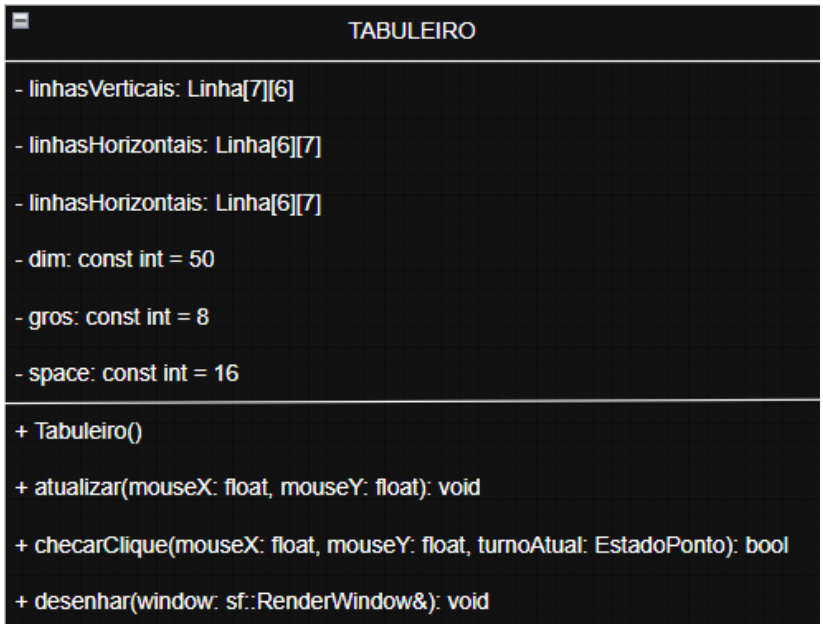
A classe Linha é responsável por representar graficamente as linhas do tabuleiro, que podem ser horizontais ou verticais. Cada linha é construída utilizando um objeto sf::RectangleShape, fornecido pela SFML, e possui um estado interno representado por um enum denominado Clik, que indica se a linha está vazia ou já foi clicada por um jogador. A classe fornece funcionalidades para detectar colisões com o cursor do mouse, alterar seu estado quando clicada e desenhar a linha na janela do jogo. Ela constitui a unidade básica de interação do jogador com o tabuleiro.

Figura 2 – UML da classe Quadrado



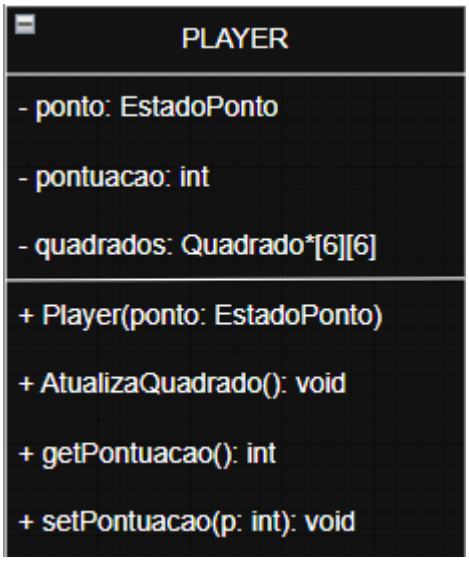
A classe Quadrado representa cada célula do tabuleiro, ou seja, cada possível área que pode ser conquistada pelos jogadores. Um quadrado é definido por quatro ponteiros para objetos da classe Linha (superior, inferior, esquerda e direita), e possui dois atributos adicionais: um para indicar se ele já foi preenchido (ponto) e outro para guardar o turno do jogador que o preencheu. Sua principal função é verificar se todas as suas linhas foram clicadas, o que indica que ele foi completado. Quando isso acontece, o quadrado é atribuído ao jogador correspondente e desenhado na cor que representa esse jogador. Assim, a lógica de pontuação e controle de território do jogo está diretamente associada à classe Quadrado.

Figura 3 – UML da classe Tabuleiro



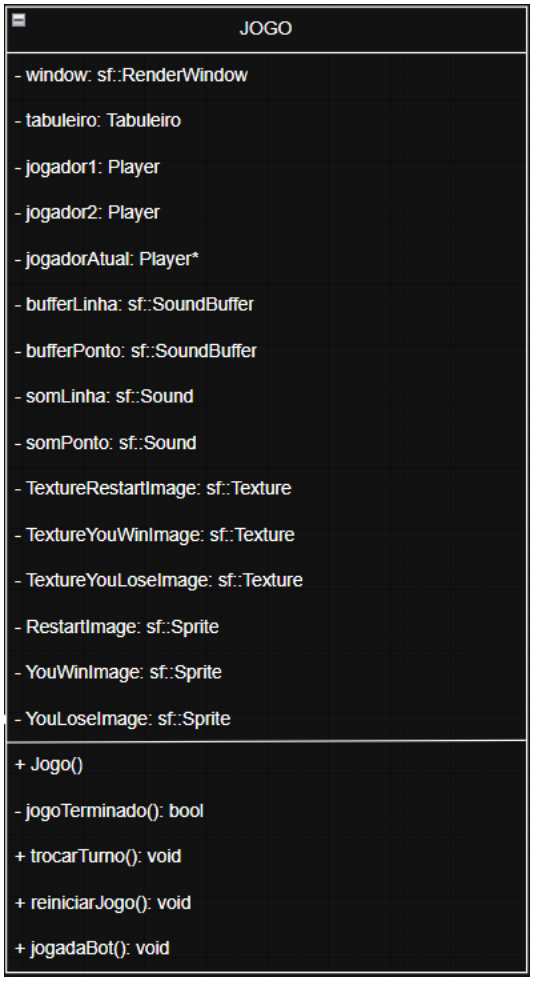
A classe Tabuleiro atua como o componente estruturante do jogo, sendo responsável por organizar e inicializar todas as linhas e quadrados que compõem o jogo. Ela contém matrizes para armazenar as linhas horizontais, verticais e os quadrados, além de atributos auxiliares que definem o espaçamento entre os elementos, a grossura das linhas e as dimensões do tabuleiro. A função do tabuleiro é garantir que todos os elementos estejam posicionados corretamente, lidar com eventos do mouse, repassar interações às linhas e quadrados, e verificar se algum quadrado foi completado após uma jogada. Com isso, o tabuleiro centraliza a lógica gráfica e estrutural da interação do jogo.

Figura 4 – UML da classe Player



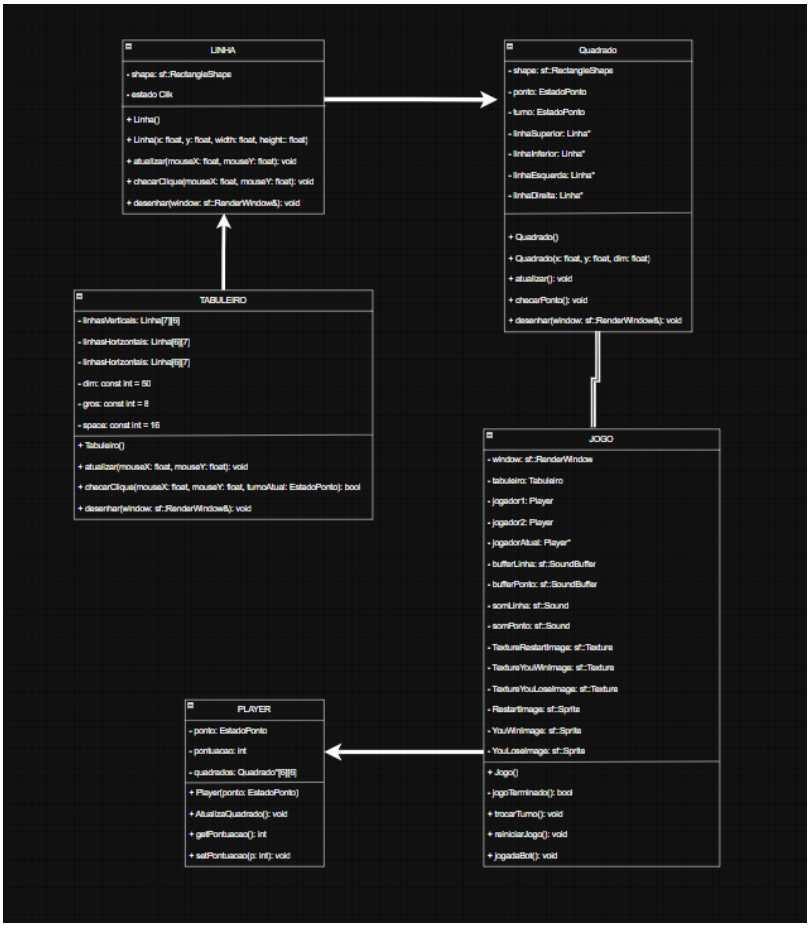
A classe Player representa cada jogador da partida, podendo ser o jogador principal ou o oponente controlado por inteligência artificial. Ela possui um atributo que identifica o jogador por meio de um enum (EstadoPonto), um atributo de pontuação, e um ponteiro para a matriz de quadrados do tabuleiro. Sua função principal é verificar, após cada jogada, quantos quadrados foram conquistados por aquele jogador, atualizando sua pontuação conforme necessário.

Figura 5 – UML da classe Jogo



A classe Jogo funciona como a controladora principal de todo o sistema. Ela agrega todos os componentes mencionados anteriormente, incluindo o tabuleiro, os dois jogadores, os objetos gráficos como janelas, fontes e sons, e os elementos visuais de interface como sprites e botões. A responsabilidade da classe Jogo é coordenar o fluxo completo da aplicação, controlando o loop principal do jogo, alternando turnos entre os jogadores, processando entradas do mouse, verificando condições de vitória e exibindo mensagens visuais ao final da partida. Além disso, ela oferece mecanismos para reiniciar o jogo e gerenciar a interação com o usuário de forma fluida e contínua.

Figura 6 – Diagrama geral da UML do projeto



Do ponto de vista das relações entre classes, a classe Jogo possui instâncias diretas de Player e Tabuleiro, e gerencia a renderização na janela sf::RenderWindow. A classe Tabuleiro, por sua vez, contém as matrizes de Linha e Quadrado, enquanto cada Quadrado contém ponteiros para quatro objetos da classe Linha. Já a classe Player interage diretamente com os quadrados do tabuleiro para atualizar a pontuação, a partir da verificação de quais células foram conquistadas após uma jogada.

Em resumo, a estrutura de classes foi pensada para promover clareza, modularidade e facilidade de manutenção. A separação de responsabilidades entre as classes, o uso correto do encapsulamento e a modelagem orientada a objetos permitiram a construção de um código organizado e escalável, com forte integração entre lógica de jogo e interface gráfica. Esse modelo também oferece uma base sólida para futuras expansões, como a adição de novos modos de jogo, melhorias na inteligência artificial ou ajustes visuais

2.2 Funções e Procedimentos

As funções utilizadas no projeto são:

Classe Linha

- **atualizar(mouseX, mouseY)**
  Destaca a linha quando o mouse passa por cima dela.
- **checarClique(mouseX, mouseY)**
  Verifica se a linha foi clicada e marca como usada.
- **desenhar(window)**
  Desenha a linha na tela com a cor correspondente ao seu estado.

Classe Quadrado

- **checarPonto()**
  Verifica se o quadrado foi completado (todas as linhas clicadas).
- **atualizar()**
  Atualiza o estado do quadrado se ele for fechado.
- **desenhar(window)**
  Desenha o quadrado colorido se ele tiver sido conquistado.

Classe Tabuleiro

- **checarClique(mouseX, mouseY, turno)**
  Verifica se alguma linha foi clicada e se houve ponto.
- **atualizar(mouseX, mouseY)**
  Atualiza os destaques visuais das linhas com base no mouse.
- **desenhar(window)**
  Desenha todas as linhas e quadrados na tela.

Classe Player

- **AtualizaQuadrado()**
  Conta quantos quadrados o jogador conquistou.
- **getPontuacao()**
  Retorna a pontuação atual do jogador.
- **setPontuacao(p)**
  Define manualmente a pontuação do jogador.

Classe Jogo

- **jogadaBot()**
  Faz uma jogada automática para o bot.
- **reiniciarJogo()**
  Reinicia todo o estado do jogo.
- **trocarTurno()**
  Alterna entre o turno do jogador e do bot.
- **FimDoJogo()**
  Verifica se o jogo acabou e mostra o resultado.
- **open()**
  Executa o jogo: eventos, jogadas, atualizações e tela.

2.3 Organização do Código e Detalhes Técnicos

O código foi estruturado com foco na modularidade e clareza, utilizando a programação orientada a objetos para representar os elementos principais do jogo. As classes Linha, Quadrado, Tabuleiro, Player e Jogo têm responsabilidades bem definidas e se comunicam por meio de composição.

A classe Jogo concentra o controle da execução, alternância de turnos, lógica do bot, eventos de mouse, controle de sons e fim de jogo. Já o Tabuleiro cuida da interface visual e da verificação de cliques nas linhas.

A inteligência artificial do bot segue uma lógica simples: tenta completar quadrados com três lados preenchidos e, se não for possível, realiza uma jogada aleatória. Esse comportamento é suficiente para tornar o jogo desafiador, mas acessível.

Sons foram integrados com a SFML para reforçar ações importantes (traçar linha e marcar ponto). Os elementos gráficos são todos desenhados dinamicamente, com dimensões ajustadas por constantes, facilitando possíveis mudanças no layout.

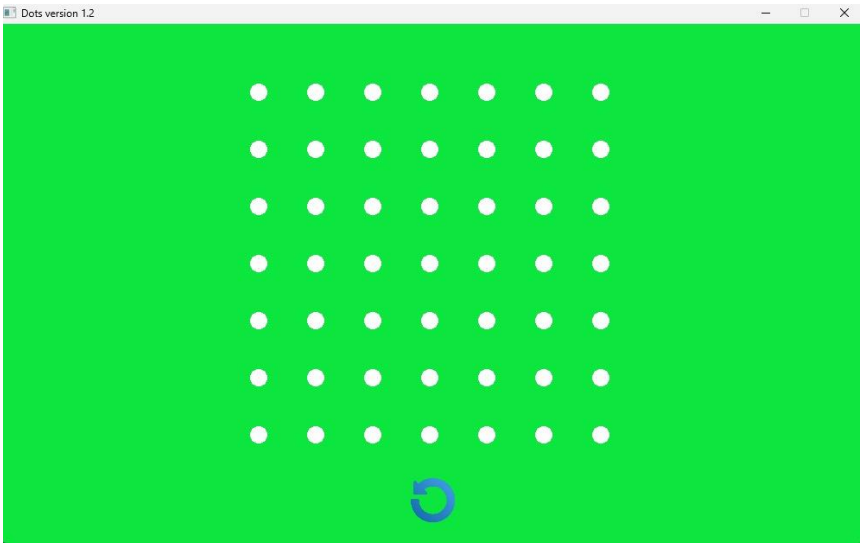
3. TESTES

Testes realizados:

- Interação visual do mouse com as linhas

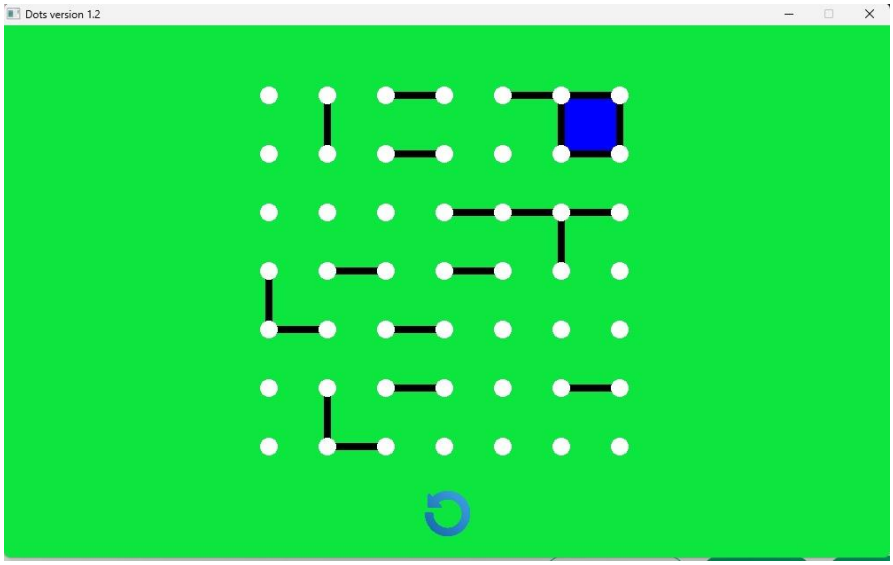
- Preenchimento correto dos quadrados e atribuição de pontos
- Comportamento lógico do bot (movimento estratégico e aleatório)
- Verificação correta de fim de jogo
- Sons funcionando nas ações correspondentes
- Botão de reinício funcional

FIGURA 7 – Visão do tabuleiro com linhas e quadrados destacados



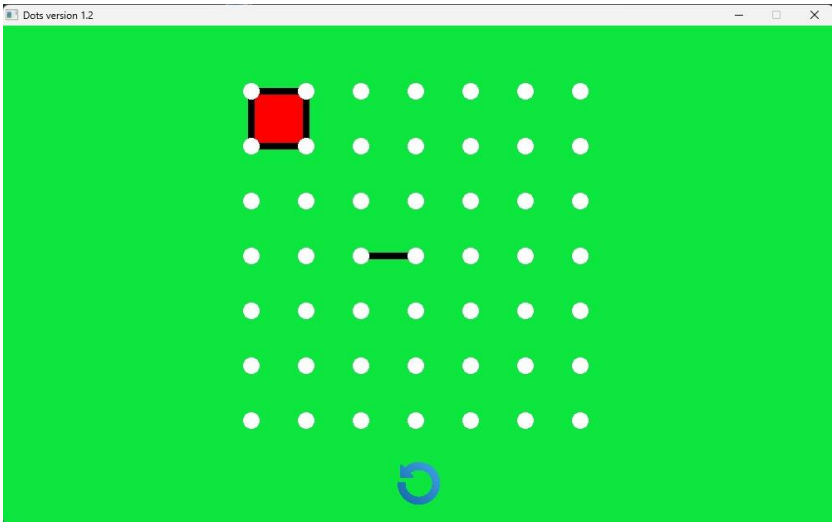
Nesta imagem, vemos o estado inicial do tabuleiro, com todas as linhas disponíveis e quadrados em branco. Quando o cursor do mouse se aproxima de uma linha, ela é automaticamente destacada, oferecendo um feedback visual claro de que a linha está interativa e pode ser clicada.

FIGURA 8 – Quadrado completado pelo jogador 1 (azul)



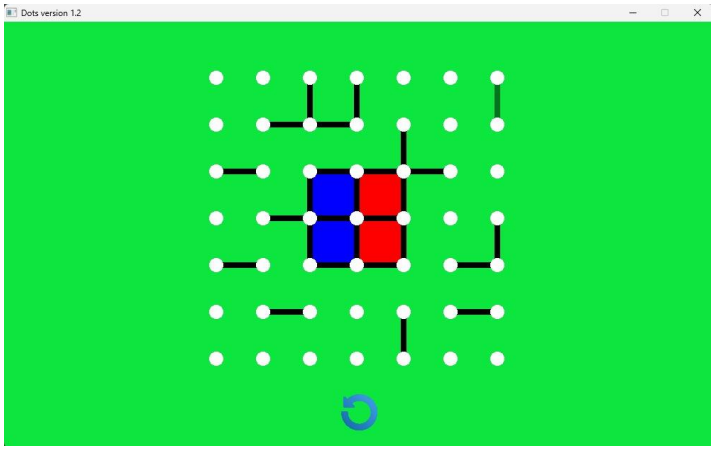
Após uma jogada válida, o quadrado é completado pelo jogador 1 e pintado automaticamente na cor azul. A imagem demonstra que o sistema reconhece corretamente a conquista de um quadrado e associa a cor ao jogador atual.

FIGURA 9 – Quadrado completado pelo bot (vermelho)



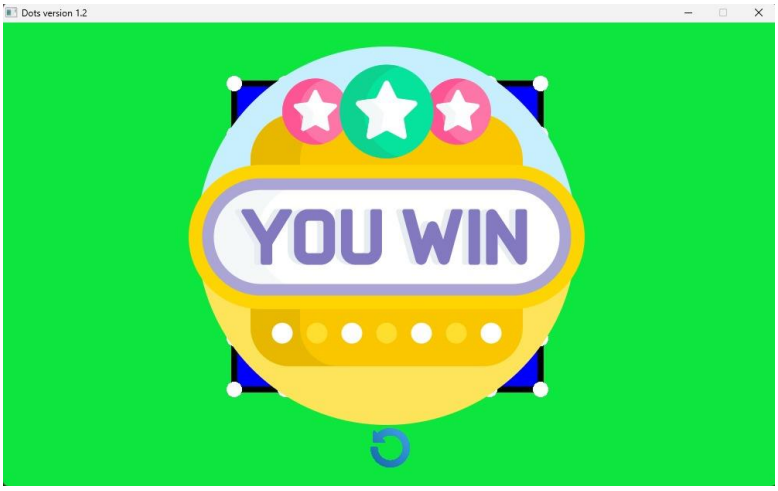
Nesta etapa do jogo, o bot realiza uma jogada que resulta na conclusão de um quadrado. A interface atualiza imediatamente a cor do quadrado para vermelho, confirmando que o sistema diferencia corretamente as jogadas de cada jogador.

FIGURA 10 – Bot realizando jogada automática



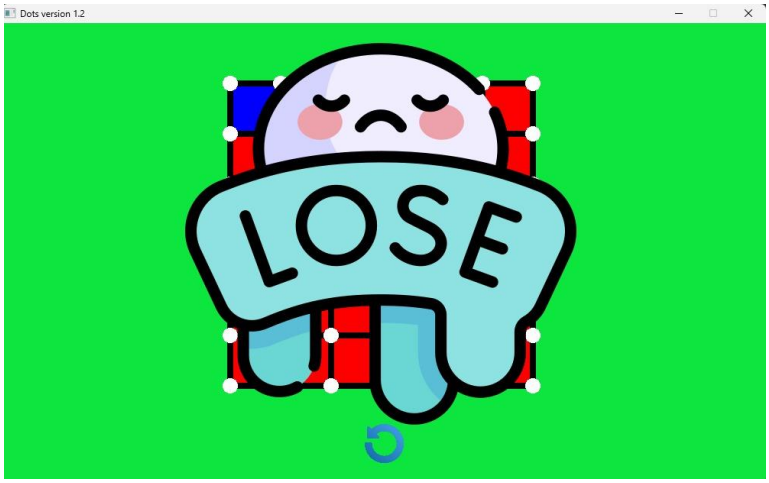
A imagem mostra o momento da jogada automática do bot, onde uma linha é preenchida sem intervenção do usuário. Isso comprova que a função `jogadaBot()` está funcionando como esperado e responde de forma dinâmica ao andamento do jogo.

FIGURA 11 – Tela de fim de jogo com mensagem "You Win"



Com todos os quadrados conquistados e o jogador 1 com a maior pontuação, a tela final exibe a mensagem de vitória. A interface encerra a partida e comunica o resultado de forma clara, conforme a lógica implementada na função `FimDoJogo()`.

FIGURA 12 – Tela de fim de jogo com mensagem "You Lose"

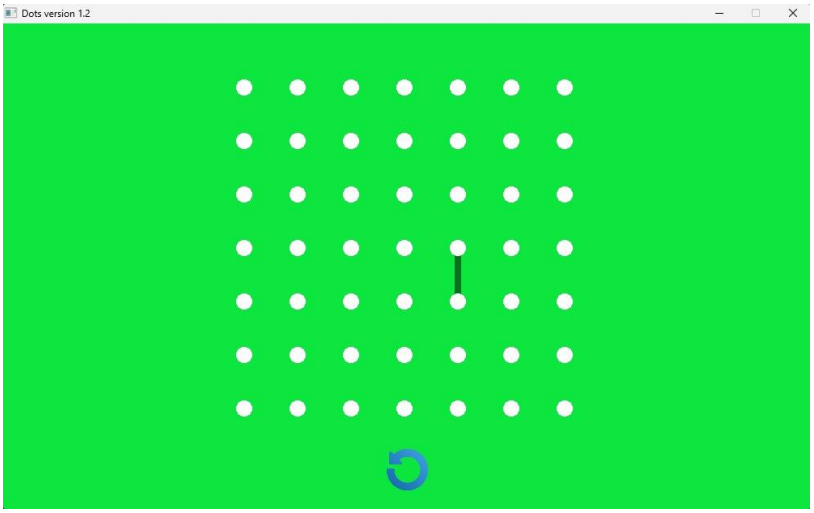


Neste exemplo, o bot vence a partida. A tela de fim exibe a mensagem de derrota, mantendo o mesmo padrão visual de encerramento, mas adaptando a mensagem de acordo com o resultado.

FIGURA 13 – Botão de reinício posicionado na interface

A imagem apresenta a localização e o estilo do botão de reinício, visível durante ou após a partida. Seu posicionamento facilita o acesso do usuário para começar uma nova partida, confirmando a funcionalidade da função `reiniciarJogo()`.

FIGURA 14 – Interação visual da linha ao passar o mouse



Quando o cursor se aproxima de uma linha ainda não utilizada, ela muda de cor automaticamente. A imagem mostra esse destaque visual, confirmando

que a função `atualizar(mouseX, mouseY)` das linhas está ativa e melhora a usabilidade do jogo.

4. CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto permitiu implementar um jogo interativo baseado no clássico “Pontos e Caixas”, utilizando a linguagem C++ e a biblioteca SFML para a interface gráfica. A estrutura orientada a objetos facilitou a organização do código, dividindo as responsabilidades entre classes específicas, o que garantiu maior modularidade e clareza no desenvolvimento.

As funções e procedimentos foram implementados para oferecer uma experiência de usuário intuitiva, incluindo a detecção precisa de cliques, atualizações visuais em tempo real e a jogada automática do bot. A interface gráfica, com destaque para as linhas ao passar o mouse e a diferenciação visual dos quadrados conquistados por cada jogador, proporcionou um feedback visual importante para o entendimento do estado do jogo.

Além disso, o sistema de pontuação e a lógica de troca de turnos foram essenciais para o funcionamento correto da dinâmica do jogo, enquanto as telas de finalização reforçaram o encerramento da partida com mensagens claras de vitória ou derrota. A presença do botão de reinício na interface oferece ao jogador a possibilidade de recomeçar o jogo facilmente, aumentando a usabilidade.

Em suma, o projeto cumpriu os objetivos propostos, unindo lógica de jogo, interação e apresentação visual em uma aplicação funcional e de fácil compreensão. Futuras melhorias podem incluir inteligência artificial mais avançada, personalização da interface e maior feedback sonoro para enriquecer ainda mais a experiência do usuário.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BRYAN, John. *Beginning C++ through game programming*. 4. ed. Boston: Cengage Learning, 2014.

LEE, Jeff. *SFML game development*. Birmingham: Packt Publishing, 2013.

STROUSTRUP, Bjarne. *The C++ programming language*. 4. ed. Boston: Addison-Wesley, 2013.

GAMEDEV.NET. *Game programming patterns*. Disponível em: https://gameprogrammingpatterns.com. Acesso em: 20 maio 2025.

SFML. *Simple and fast multimedia library – documentation*. Disponível em: https://www.sfml-dev.org/documentation/2.5.1/. Acesso em: 21 maio 2025.

SKIFFER, Mike. *Object-oriented design and patterns*. 2. ed. Hoboken: Wiley, 2010.

SILVA, João. *Introdução à programação orientada a objetos com C++*. São Paulo: Ciência Moderna, 2017.

MILLER, Jason. *Artificial intelligence for games*. 2. ed. Boca Raton: CRC Press, 2019.

MATHIS, Andrew. *Effective C++: 55 specific ways to improve your programs and designs*. 3. ed. Boston: Addison-Wesley, 2005.

AURELIANO, Pedro. *Programação de jogos digitais: fundamentos, técnicas e aplicações*. Rio de Janeiro: Elsevier, 2021.

MOURA, Carlos. *Desenvolvimento de jogos com SFML: teoria e prática*. São Paulo: Novatec, 2020.