Modern 3D Simulated Chess Game integrated with AI agents

Github: https://github.com/kyle-paul/modern-chess-3d.git
Youtube: https://youtu.be/0tOOrvqqWPE

Project Overview

This is a chess game set in a simulated 3D environment, built on a core foundation of OpenGL. While Vulkan is not currently implemented, it is planned for future updates, as the backend renderer is designed to be agnostic. The game supports two-player mode for human opponents as well as gameplay against AI agents, with three difficulty levels: easy, medium, and hard. Players can move pieces using either the keyboard or direct mouse selection within the 3D scene. Additionally, players can customize the board color, adjust environmental lighting, and modify the camera view to analyze the match from different angles.

Requirements Specification

1. Game Modes:

- 2-Player Mode:
    - Two players can play against each other on the same device.
    - The game should manage turns automatically, alternating between Player 1 (white) and Player 2 (black).
- Player vs AI Mode: The user plays against an AI-controlled opponent with three selectable difficulty levels:
    - Easy: The AI makes random or less optimal moves with little to no strategy.
    - Medium: The AI uses moderate strategic calculations with some errors.
    - Hard: The AI uses advanced strategy, attempting to play with minimal mistakes.
    -

2. User Interface (UI): create a user-friendly interface using a C++ GUI library that includes:

- A chessboard where pieces can be moved.
- Turn indicators showing whether it's white or black's turn.
- Notifications for check, checkmate, or stalemate.
- Buttons to:
    - Start a new game.
    - Choose game mode (2-player or Player vs AI).
        - Select AI difficulty level before starting a Player vs AI game.
    - Reset the current game.
    - Undo the last move.
    - Redo the last undone move.
    - Save the current game state.

- Load a previously saved game.
- Access game settings for customization.
    - Board Color: Options to select different colors or themes for the chessboard.
    - Piece Design: Options to change the design or style of the chess pieces (e.g., classic, modern, cartoon).
    - Sound Effects: Options to enable or disable sound effects for moves, check, checkmate, and background music. Provide a selection of different sound themes.
    - Background Music: Options to choose background music or disable it. Include a volume control for sound effects and music.
- Exit the game.

3. Chess Board Representation

- Implement a graphical chessboard using a C++ GUI library such as **SFML**, SDL, or Qt.
- Players can interact with pieces by selecting, dragging, and dropping them.
- Highlight legal moves when a player selects a piece.
- Visual feedback should be provided for check, checkmate, and stalemate situations.
- In 2-player mode, alternate between Player 1 (white) and Player 2 (black) automatically after each move.
- In Player vs AI mode, alternate between the player's move and the AI's move. The AI should make a move immediately after the player completes theirs.

4. Game Logic

- Implement the rules of chess, including:
    - Movement for all pieces (pawns, knights, bishops, rooks, queens, and kings).
    - Special moves like castling, en passant, and pawn promotion.
    - Detection of check, checkmate, stalemate, and draw.
- Ensure that only valid moves are allowed.
- Automatically detect and end the game with a notification in case of checkmate or stalemate.

AI Opponent:

- Implement three levels of AI difficulty:
    - Easy: The AI makes quick, random moves without much evaluation.
    - Medium: The AI uses the Minimax algorithm with a shallow depth, making reasonable moves with occasional mistakes.
    - Hard: The AI uses Minimax with Alpha-Beta pruning, evaluating a deeper move tree to play strategically, reducing mistakes.
- AI should make moves efficiently and within a reasonable time for all difficulty levels.

Game State Management

- Track the current state of the board and the game.

- Handle turn management, ensuring proper transitions between the player's turn and the AI's turn or between two human players.
- Implement an Undo feature that allows players to revert the last move:
    - Store a history stack of moves (using a suitable data structure) to facilitate undo operations.
    - Implement a Redo feature to reapply the last undone move:
        - Use a second stack to manage redone moves.
        - Provide the ability to reset the game to its initial state at any time.

Save and Load Functionality
- Implement the ability to save the current game state to a file: save the positions of all pieces, the current player's turn, and any other relevant game state information.
- Implement the ability to load a previously saved game: read from the saved file and restore the game state to the last saved point, allowing players to continue from where they left off.

Technology Stack:
- C++ for Core Logic: Write all game logic, AI algorithms, and game state management in C++.
- AI Algorithms: Implement AI using Minimax with Alpha-Beta pruning to handle decision-making for medium and hard difficulty levels.
- GUI Library: Use SFML, SDL, or Qt to build the user interface, render the chessboard, and manage input events.
- Save Game:
    - Serialize the current game state, including piece positions, turn information, and any other necessary data.
    - Write this data to a file in a format such as JSON, XML, or a simple text format.
- Load Game:
    - Open a file dialog to allow the user to select a saved game file.
    - Deserialize the data from the file and restore the game state accordingly, updating the chessboard and any other relevant UI elements.

Design Document

1. Rendering pipeline

I developed my game using OpenGL as the primary renderer, focusing on abstracting the rendering process for greater flexibility and modularity. For instance, I encapsulated the creation of vertex arrays, vertex buffers, index buffers, and shader programs into reusable abstractions. To render an object, I simply add a new shader program to the `shaderLibrary` variable in the `Game` class and bind the appropriate VAO and VBO for rendering meshes in 3D space.

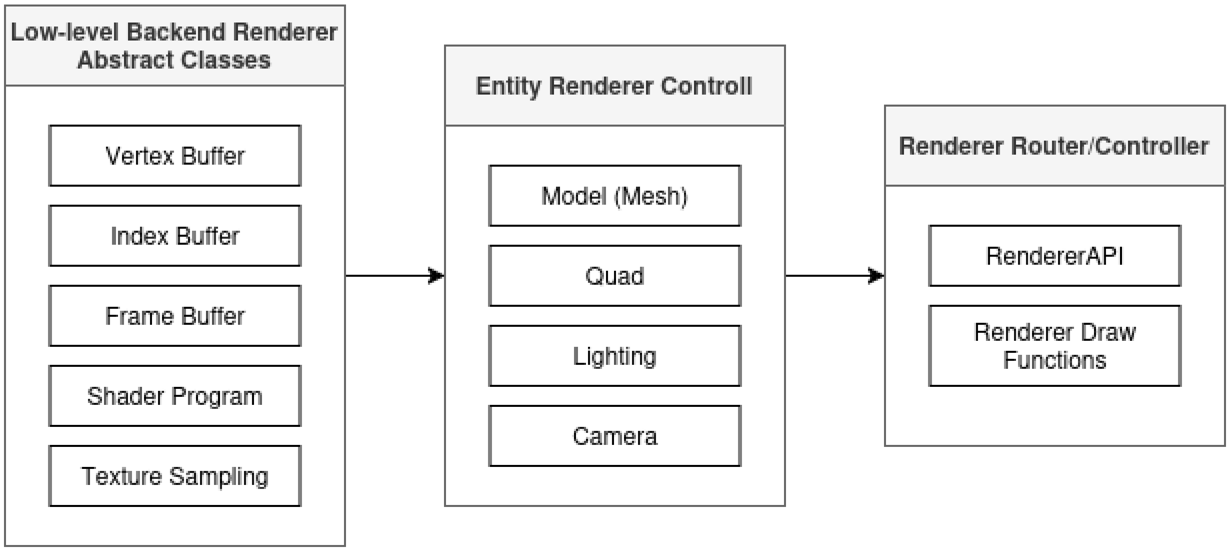
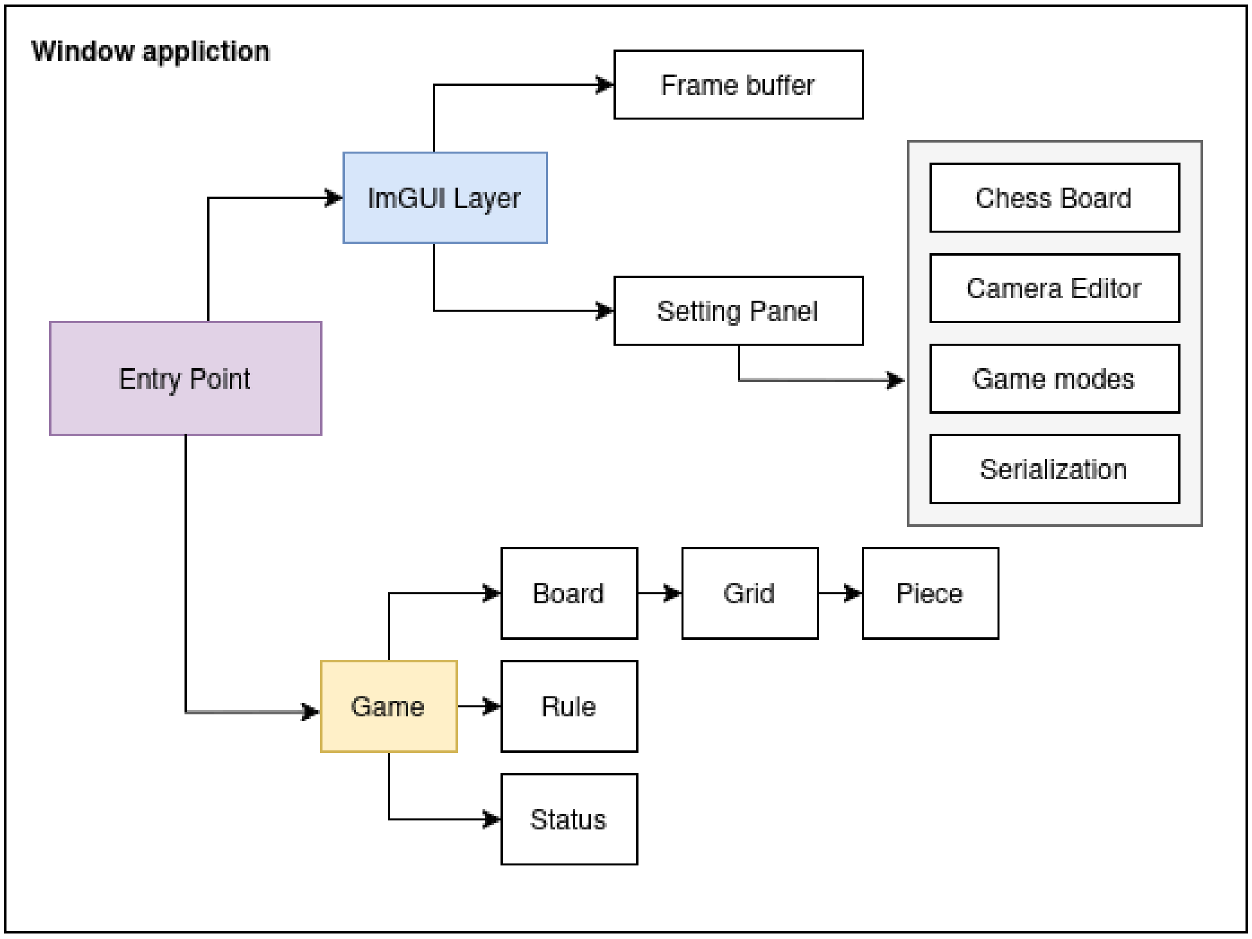


Diagram 1: Basic Rendering Process. This diagram presents a high-level abstraction of the rendering workflow. Behind the scenes, the API calls and implementations differ for each backend, such as OpenGL or Vulkan.

2. Chess game logic



Implementation Details

- **Language:** C++, GLSL
- **Main core:** OpenGL
- **Library used:**
    - spdlog (for debugging)
    - Glm (math library)
    - Imgui (User interface)
    - Stb (image loader)
    - ImGuiFileDialog (file explorer dialog)
    - Glfw (Window console)
- **Build system:** CMake, Ubuntu, Linux
- **Tool used**: Github, Trello

Testing and Future Improvements:

- Add the more choice of 3D chess piece models with different styles
- Train Reinforcment Learning with Pytorch C++

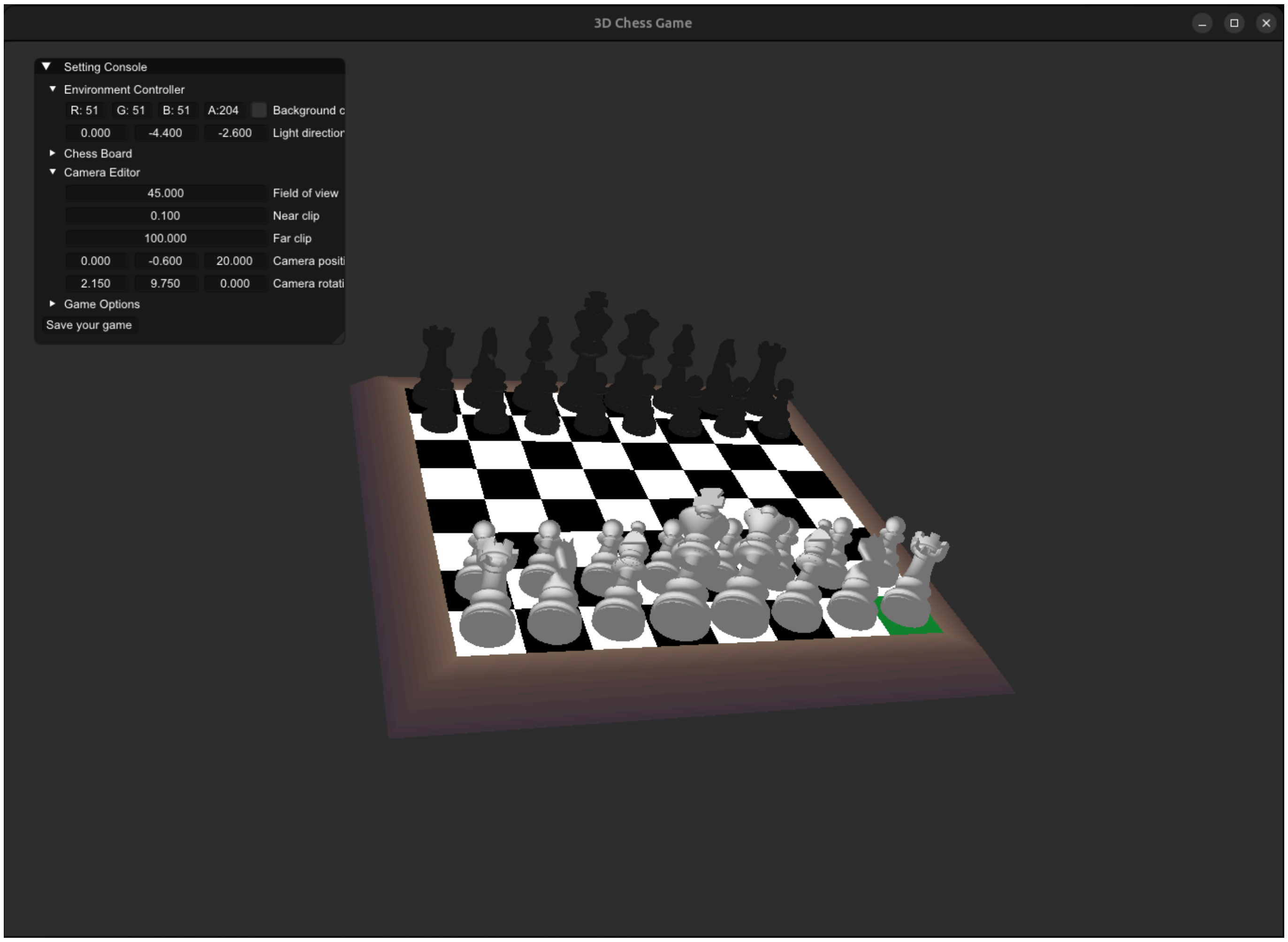
Team Members

- 1 member: Nguyen Le Quoc Bao - 24125004 - APCS 24A01

Responsibilities

- [x] Added `spdlog` for debugging the code.
- [x] Abstracted the window application.
- [x] Abstracted the camera control.
- [x] Abstracted the shading program.
- [x] Abstracted the rendering system (supports multiple backend APIs).
- [x] Abstracted the `VertexArray`, `VertexBuffer`, and `IndexBuffer`.
- [x] Abstracted the environment (lighting, background).
- [x] Abstracted the square on the board, including rotation.
- [x] Created the game state.
- [x] Abstracted the mesh loading.
- [x] Added all chess players to the board and controlled the lighting.
- [x] Abstracted the game layers (Game → Board → Square → Piece → Grid).
- [x] Added selection using keyboard events.
- [x] Abstracted the game rules.
- [x] Added the feature to select entities by clicking with framebuffer.
- [x] Added an Undo moves feature using a stack.
- [x] Added a mode to play against the machine (Minimax).
- [x] Added UI elements for users.
- [x] Created an easy mode (random moves).
- [x] Implemented the Minimax algorithm.
- [x] Added Alpha-Beta pruning for deeper searches (hard mode).
- [x] Designed user options (landscape, board color, lighting).
- [x] Debugged rules for checkmate and stalemate.

- [x] Added the Redo feature.
- [x] Added sound effects using OpenAL.
- [x] Implemented game state serialization and deserialization.
- [x] Added the File Dialog for file selection.



```
R: 51   G: 51   B: 51   A:204   Background c
0.000   -4.400   -2.600   Light directior
45.000   Field of view
0.100   Near clip
100.000   Far clip
0.000   -0.600   20.000   Camera positi
2.150   9.750   0.000   Camera rotati
```