A red blue and white star with a white flower

AI-generated content may be incorrect.

**TRIBHUVAN UNIVERSITY**

**INSTITUE OF ENGINEERING**

**PULCHOWK CAMPUS**

A REPORT ON

**CHESS WITH C++**

**SUBMITTED BY:**

**Bipin Dhungel 081BCT027**

**Neerjal Paudel 081BCT045**

**Nugaw Ratna Shakya 081BCT048**

Group: **B**

**SUBMITTED TO:**

Department of Electronics and Computer Engineering

ACKNOWLEDGEMENT

We would like to extend our heartfelt gratitude to **Assistant Professor Daya Sagar Baral** for his constant guidance, motivation, and encouragement throughout the duration of this project. His expert advice, constructive feedback, and patient supervision have been invaluable in helping us refine our ideas, overcome technical challenges, and successfully implement the **SFML Chess Game**. His ability to explain complex concepts with clarity and his dedication to student success greatly inspired us to give our best effort at every stage.

We also wish to express our sincere thanks to the **Department of Electronics and Computer Engineering** for providing the resources, academic foundation, and a supportive learning environment that enabled us to work effectively on this project. The department’s commitment to fostering technical skills and problem-solving abilities among students has been a key factor in the successful completion of this work.

Furthermore, we are grateful to all the faculty members of the department for their guidance and encouragement during our academic journey. We also acknowledge the valuable programming knowledge, design principles, and problem-solving skills gained during coursework, which were essential in implementing the advanced game logic, graphical rendering, and user interaction features of this project.

Lastly, we thank our peers and team members for their cooperation, contributions, and collaborative spirit, which made the development process both technically productive and personally rewarding.

TABLE OF CONTENT

Objectives

The primary objectives of this project are:

1. **To design and develop a fully functional, multiplayer chess game** using C++ and the Simple and Fast Multimedia Library (SFML), capable of supporting two players on the same system with real-time interaction.
2. **To implement all standard chess rules** including legal moves, check, checkmate, stalemate, pawn promotion, castling, and en passant, ensuring a complete and authentic gameplay experience.
3. **To apply object-oriented programming principles** for structuring the game logic into modular, reusable, and maintainable components.
4. **To create a visually appealing and interactive graphical user interface** that allows players to intuitively move pieces, receive visual feedback, and enjoy a smooth gaming experience.
5. **To demonstrate practical application of programming, problem-solving, and debugging skills** in the context of a complex real-time game project.
6. **To successfully test and verify the multiplayer functionality** so that two human players can complete a full game of chess under all possible rule scenarios without errors or glitches.

**Introduction**

Chess, often referred to as the “game of kings,” is a centuries-old strategy game renowned for its depth, complexity, and enduring appeal. In the digital age, chess applications have made the game more accessible, enabling players to engage in matches on various platforms.

This project focuses on the **development of a complete, offline multiplayer chess application** in **C++**, utilizing the **SFML** library for graphics rendering, input handling, and user interaction. The game is designed for **two players sharing a single system**, taking alternate turns to move pieces until a valid end condition—such as checkmate, stalemate, or draw—is reached.

The implementation strictly follows **FIDE’s official chess rules** and supports advanced moves such as castling, pawn promotion, and en passant. The system architecture is built around **object-oriented programming concepts**, with dedicated classes for the board, pieces, and game controller, ensuring code reusability and easier maintenance.

Through careful testing, the project achieved **full multiplayer functionality**, allowing players to successfully complete a game under all valid move conditions. This work demonstrates not only proficiency in programming and game logic design but also the practical integration of graphics libraries to create an interactive and engaging user experience.

**WORKING mechanism**

**2.1 main.cpp**

**Purpose:** Entry point of the program.

* A screen shot of a computer program

  AI-generated content may be incorrect.**ChessGame game;** → Constructs the main game object (calls the constructor in ChessGame.cpp).
* **game.run();** → Begins the game loop, which keeps running until the window is closed.

**2.2 ChessGame.h**

**Purpose:** Declares all game components.

**Enumerations**

* A black screen with white text

  AI-generated content may be incorrect.These enums make the code more **readable** and **type-safe**.

**Piece Structure**

* Stores **piece type**, **color**, and **sprite** for rendering.
* A close-up of a computer screen

  AI-generated content may be incorrect.hasMoved tracks if a piece has moved (important for castling).

**ChessGame Class**

Declares:

* **Variables:**  
  board (2D vector of Piece), currentTurn, menuState, moveLog, rotateBoard, sounds, textures, fonts, etc.
* **Functions:**
  + **Initialization:** initializeBoard(), loadTextures(), loadSounds().
  + **Rendering:** drawBoard(), drawPieces(), drawMenu().
  + **Input Handling:** handleMouseClick(), handleKeyPress().
  + **Game Logic:** isValidMove(), updateGameState(), movePiece(), canCastle(), isEnPassantValid().
  + **Move Logging:** logMove(), savePGN().

**2.3 ChessGame.cpp【7†source\*\***

**Initialization**

**Constructor**

* A screen shot of a computer code

  AI-generated content may be incorrect.Creates **800×800 SFML window**.
* Calls setup functions for board, textures, and sounds.
* Loads font, tries multiple fallback paths.

**initializeBoard()**

* **A black text on a white background

  AI-generated content may be incorrect.**Creates an **8×8 grid** of empty pieces.
* Places **white** pieces on rows 0–1 and **black** pieces on rows 6–7:

**loadTextures()**

* Loads "chess\_pieces.png" into piecesTexture.
* Cuts each sprite from the sheet using coordinates (spriteX, spriteY).
* Assigns textures to pieces with Piece::setTexture().

**loadSounds()**

* Loads "move.mp3" and "capture.mp3".
* Assigns them to moveSound and captureSound.

**Menu Handling**

**drawMenu()**

* If font loads:
  + Draws title "SFML Chess".
  + Draws menu items "Single Player", "Multiplayer", "Quit".
* If font fails:
  + Uses rectangles instead of text.
* Highlights the currently selected menu option.

**handleMenuClick()** & **handleKeyPress()**

* **Arrow keys** change selectedMenuItem.
* **Enter key** starts Multiplayer, shows message for Single Player (not implemented), or quits.
* **Mouse clicks** on menu items are also supported.

**Game Logic**

**isValidMove()**

* Checks:
  + Move is inside board (isInBounds()).
  + Target square is not occupied by same color.
  + King won’t be in check after move (wouldBeInCheck()).
* Calls specific move functions like:
  + isValidPawnMove()
  + isValidRookMove()
  + isValidBishopMove()
  + isValidQueenMove()
  + isValidKnightMove()
  + isValidKingMove()

**Special Rules:**

* **canCastle()** → Ensures neither king nor rook moved, path clear, no check in between.
* **isEnPassantValid()** → Checks if pawn can capture via en passant.
* **Pawn Promotion** → Promotes pawn to Queen when it reaches last rank.

**isInCheck() & isSquareAttacked()**

* Locates king position.
* Checks if any opponent piece can legally move to king’s square.

**updateGameState()**

* If no valid moves:
  + If in check → Checkmate
  + Else → Stalemate
* If still playable:
  + Updates to Check or Playing.

**Moving Pieces**

**movePiece()**

* Logs move (logMove()).
* Plays sound based on capture or normal move.
* Handles:
  + **Castling** → Moves rook with king.
  + **En Passant** → Removes captured pawn.
  + **Promotion** → Changes pawn sprite to Queen.
* Updates board state and en passant targets.

**Drawing Functions**

**drawBoard()**

* Draws alternating **light** and **dark** squares.
* Highlights en passant target with **blue**.

**drawPieces()**

* Draws all sprites from the board.

**drawSelection()**

* Highlights selected piece in **yellow**.
* Draws **green circles** for valid moves.
* Marks king in **red** if in check.

**Input Handling**

**handleMouseClick()**

* Converts mouse coordinates to board position.
* If no piece selected → selects a piece of current player.
* If piece selected → tries move and updates turn.

**switchTurn()**

* Changes currentTurn between White and Black.
* Rotates board (rotateBoard).

**Move Logging**

**logMove()**

* Converts moves to chess notation (e.g., "e4", "Nf3").
* Appends to moves.txt.
* savePGN() → Saves complete PGN format.

**Main Loop (run())**

* A screenshot of a computer program

  AI-generated content may be incorrect.Processes events (sf::Event).
* Draws either the main menu or game board based on menuState.
* Keeps game running at 60 FPS.

**3. How the Game Works**

**Start Menu**

* Player chooses **Multiplayer** to start.
* **Single Player** placeholder (future AI feature).

**Playing**

* Click to select a piece → valid moves highlighted.
* Click destination square → piece moves if legal.
* Special moves handled automatically.

**End**

* Checkmate or Stalemate triggers game end.
* Press ESC to return to menu

Implementation   
**1. Install SFML**

1. Go to **SFML download page**:  
   https://www.sfml-dev.org/download.php
2. Under **Visual C++ 17 (2022)**, download **SFML 2.5.1 (or latest)** — choose **32-bit** or **64-bit** depending on your project.
3. Extract the .zip to a location, e.g.:

makefile

CopyEdit

C:\SFML-2.5.1\

**2. Create a New Project in Visual Studio 2022**

1. Open **Visual Studio 2022**.
2. Go to **File → New → Project**.
3. Search for **"Console App"** and select **C++ Console Application**.
4. Name your project (e.g., ChessGame) and click **Create**.

**3. Add Your Files**

1. In **Solution Explorer**, right-click the **Source Files** folder → **Add → Existing Item**.
   * Add main.cpp and ChessGame.cpp.
2. Do the same for **Header Files** folder.
   * Add ChessGame.h.
3. Make sure all your code files are in the project.

**4. Configure SFML in Visual Studio**

You need to tell Visual Studio **where SFML’s include files and libraries are**, and **link** them.

**4.1 Open Project Properties**

* Right-click your project in Solution Explorer → **Properties**.
* Make sure **Configuration** is set to:

pgsql

CopyEdit

Configuration: All Configurations

Platform: x64 (if you downloaded 64-bit SFML)

**4.2 Add Include Directory**

1. Go to **Configuration Properties → VC++ Directories**.
2. In **Include Directories**, add:

makefile

CopyEdit

C:\SFML-2.5.1\include

**4.3 Add Library Directory**

1. In **Library Directories**, add:

vbnet

CopyEdit

C:\SFML-2.5.1\lib

**4.4 Link the SFML Libraries**

1. Go to **Configuration Properties → Linker → Input**.
2. In **Additional Dependencies**, add these for **Debug** mode:

sfml-graphics-d.lib

sfml-window-d.lib

sfml-system-d.lib

sfml-audio-d.lib

And for **Release** mode:

sfml-graphics.lib

sfml-window.lib

sfml-system.lib

sfml-audio.lib

**5. Copy the SFML DLLs**

* From C:\SFML-2.5.1\bin, copy these files into **your project’s output folder** (e.g., ...\Debug or ...\Release):

sfml-graphics-d-2.dll

sfml-window-d-2.dll

sfml-system-d-2.dll

sfml-audio-d-2.dll

*(For Release build, use files without -d in the name.)*

**6. Add Your Assets**

Your code references images, fonts, and sounds:

piecesTexture.loadFromFile("...\\images\\chess\_pieces.png");

moveBuffer.loadFromFile("...\\sounds\\move.mp3");

Make sure:

* Create an **images** and **sounds** folder in your project directory.
* Put your chess piece PNG and sound files there.
* Update paths in your code if needed.

**7. Build and Run**

* Set configuration to **Debug** and platform to **x64**.
* Press **Ctrl + F5** or click **Local Windows Debugger**.
* The chess game should open in an **SFML window**.

RESULTS

**Problems Faced and Solutions**

**1. Collaboration Challenges**

**Problem:**  
While working in a group, each member was coding independently without proper integration, which led to **inconsistent coding styles**, duplicate work, and merge conflicts.

**Solution:**  
We introduced:

* A **shared GitHub repository** for version control.
* Weekly **code review meetings** to merge progress.
* A **coding style guide** so that all members followed the same structure and naming conventions.

**2. SFML Setup Difficulties**

**Problem:**  
Setting up SFML in Visual Studio 2022 was challenging for some members because of unfamiliarity with library linking, include directories, and DLL dependencies.

**Solution:**

* Created a **step-by-step setup guide** (as given earlier in this report).
* Shared a **pre-configured Visual Studio project file** so that new members could start coding immediately.

**3. Asset Loading Issues**

**Problem:**  
Game crashed or displayed empty boards when image, sound, or font files were missing or in incorrect paths.

**Solution:**

* Used **absolute paths** during early testing to avoid missing file errors.
* Later switched to **relative paths** with a fixed project folder structure (images/, sounds/ folders).
* Added fallback textures and warnings in case loading fails.

**4. Complex Chess Logic**

**Problem:**  
Implementing **special chess rules** like en passant, castling, and check detection required careful planning and debugging.

**Solution:**

* Broke down chess rules into smaller helper functions (e.g., isValidRookMove(), canCastle()).
* Added debug console logs to trace move validation.
* Tested rules incrementally before combining them.

**limitations and Future Enhancements**

**Limitations**

* **No AI Opponent** — Single Player mode is not yet implemented.
* **Basic Graphics** — No animations for moving pieces; just instant repositioning.
* **No Undo Feature** — Once a move is made, it cannot be reverted.
* **Pawn Promotion Limited** — Always promotes to Queen, with no option to choose.
* **No Online Multiplayer** — Only supports local 2-player mode.

**Future Enhancements**

* Implement **AI opponent** using Minimax or Stockfish engine.
* Add **smooth animations** for moves and captures.
* Implement **undo/redo functionality** for move corrections.
* Enhance **pawn promotion** with choice of piece.
* Add **online multiplayer** using sockets or a game server.
* Include **timer/clock** for competitive play.

**Conclusion**

The **SFML Chess Game** project successfully implements a complete two-player chess environment in **C++** with a graphical interface built using the **SFML (Simple and Fast Multimedia Library)** framework. The game allows players to interact through mouse and keyboard, supports all standard chess rules including **castling**, **en passant**, and **pawn promotion**, and provides visual feedback through highlights, board rotation, and move indicators. Audio effects enhance the gameplay experience by signaling moves and captures, while a move logging system records the game in standard notation and PGN format for later review.

The development process involved **modular object-oriented design**, with separate components for initialization, rendering, input handling, move validation, and special chess logic. The board is represented as an 8×8 matrix of Piece objects, each storing its type, color, sprite, and movement status. Careful implementation of helper functions, such as isValidMove(), isInCheck(), and updateGameState(), ensures that every move adheres to official chess rules.

Throughout development, the team encountered challenges including **collaboration difficulties**, **SFML setup issues**, and **complex special move logic**. While the current version focuses on **local multiplayer**, the codebase has been designed with extensibility in mind, allowing future integration of AI opponents, online play, animations, and advanced user interface features.

In conclusion, this project demonstrates a practical application of **C++ programming concepts, game loop design, and graphics rendering** in a real-world game development scenario. It not only provides a functional and enjoyable chess-playing experience but also serves as a foundation for further enhancements such as artificial intelligence, richer graphics, and online multiplayer support. The SFML Chess Game stands as a successful blend of technical implementation and game design principles, fulfilling its objective of delivering an engaging and fully rule-compliant digital chess environment.

REFERENCES