OOP PROJECT: CHESS

**Group 93**: Vu Chau Anh Dinh, Manh Duc Huy Le, Minh Duc Kieu

**Student ID**: A1896523, A1874365 , A1899335

Use Case Description

Chess is one of the classic board games. This project would rebuild it to a PvP (Player to Player) version of this game.

Players would move their pieces, capture component pieces and try to checkmate by their own strategy and following the basic rules of chess during the gameplay. The created moves are not limited, however, there are some cases of repeated moves can result in a draw. Thus, the project aims to bring the original chess to the computer.

Class List

1. **Chessboard**: Represents the chessboard and contains information about the arrangement of chess pieces.

2. **Player**: Represents each player’s information in the game.

3. **GameManager**: Represents the management of the game flow, including players' turns, move validations, and checking for game over conditions.

4. **GameController**: Represents user input and communicates with the GameManager to execute moves.

5. **ChessPiece**: Represents all types of chess piece and contains information about their color, and position on the board.

6. **Move**: Represents a chess move and contains information about the piece moved and the target position

7. **LoggingBoard**: Represents a digital board displaying record of 3 closest moves of both players

Data and Function Members

**Chessboard:**

Attributes:

- squares: vector

Functions:

- initializeBoard()

- isSquareEmpty()

- movePiece(): a virtual function

**Player:**

Attributes:

- isWhite: bool

- name: string

- pieces: vector

Functions:

- addPiece()

- removePiece()

- makeMove()

**GameManager:**

Attributes:

- currentPlayer: Player\*

- board: Chessboard

Functions:

- isCheck(): bool

- isCheckmate(): bool

- isInProgress(): bool

- makeMove(): bool

**GameController:**

Attributes:

- game: Game

Functions:

- startGame(): void

- endgame(): void

**ChessPiece:**

Attributes:

- type: PieceType

- position: Position

- isWhite(): bool

Functions:

- isValidMove(): bool  
  
Relationships between Classes

- The Chessboard class is utilized by the GameManager class to manage the game status.

- The Player class represents the participants in the game.

- The GameManager class keeps track of the players and manages their turns during the game.

- ChessPiece is the parent class of Pawn, King, Queen, Bishop, Knight, Rook.

- Pawn, King, Queen, Bishop, Knight and Rook inherit the attributes and functions of the ChessPiece class.

- The GameController class handles user input and communicates with the GameManager to execute moves.

- The LoggingBoard class is not related to any other class in the system.

Project Task List and Timeline

**Week 8**: Brainstorm ideas, discuss and figure out the game concept, including how player controls and play, what will be displayed, what option can be chosen during gameplay, etc. Also, define what is essential work code and what is potential. (All members)

**Week 9**: Coding, debugging and testing parts of the game, working individually, include:

* SFML classes for displaying chess boards and pieces in the original state of a game (CA)
* Game main logic(win, lose, draw) and special feature: Promote and Castling (KD)
* Building chess pieces (DH)

**Week 10 - 11**: Combine personal work, debug and test the united work, test and run the game (All members)

\*Week 11: Adding optional and extending features if time allows (All members)

User Interaction Description

Displaying

**Objects:**

- Chessboard:

* Create an 8x8 grid as the game board.
* Alternating black and white squares for the chessboard pattern number and letter the grid from 1-8 and a-h, a1 is black

- Chess Pieces:

* Create graphical representations for each type of chess piece (pawn, rook, knight, bishop, queen, king).
* Place these pieces on their initial squares, following standard chess rules.the piece is clickable, the square will change color if the piece is chosen.

**Start menu:**

- "New Game": Start a new game.

- “Resume Game”: Play a saved game

- “Exit”: Close the gameplay terminal

**Game State Display:**

- Show an icon to highlight whose turn it is.

- Indicate an invalid move(in check, not possible move).

- Display game results when the game ends (win, lose, draw).

**Game menu:**

- Create an in-game menu to pause the game and return to the start menu

**Error Handling:**

- Inform the player if they make an invalid move.

User Interaction

**Mouse Interaction:**

- Click on a chess piece to select it.

- Click again or to another piece to deselect a piece or cancel the move.

- Click on a highlighted square to move the selected piece there.

**Game State Handling:**

- Continuously update 3 closest moves of both players.

- Undo and Redo option

**Menus and Buttons:**

- Implement click handlers for menu items and buttons.

- Ensure the player can navigate menus and return to the game

**Confirmation Dialogs:**

- Display confirmation dialogs for actions like quitting the game or starting a new game to avoid accidental actions. (Sure to quit?)

Unit Testing and Debugging Plan

Testing

**Menus:** Testing 2 menus: Start menu and in-game menu, test if the chosen option function right (e.g. On startMenu, the “Resume Game” would display the closest saved game and player can continue to play that game) (W11) (Chau Anh)

**GameBoard**: Test if the printed chess board has the right size, color, and arrangement of black-white squares. As well as the original positions of all pieces (W10) (Chau Anh)

**Player:** Test if player input is available and would output as their intention in the game (W10) (Duc)

**ChessPiece:** Test if pieces move and are captured in the right logic, following the rules of chess (W10) (Huy)

**GameManager:**

Test if the moves are recorded as wanted and if it is able to keep track of the game to follow the rules and logic of chess (e.g. invalid moves when being checkmated)

Test if reacting right to the game status i.e whether the game is start or end

(W10) (Duc)

Debugging

If there is a crash or misinformation in the game during the code-building phase, the member in charge of that specific part would use different methods to identify the bug and debug the program if possible. If the program cannot be debugged, rewriting can be considered as a solution. Otherwise, the team would discuss and find support from tutors.

Project Title: Video Game Inventory Management System  
Use Case Description  
The Video Game Inventory Management System is a program designed to help video game store owners  
manage their inventory efficiently. The system will enable store owners to add, delete, and update physical and  
digital video games, manage customer information, and track transactions. The project's goal is to provide an  
intuitive and functional user interface that allows store owners to manage their inventory with ease.

**Class List**  
1. Game: Represents a video game and has attributes like title, developer, publisher, release date, genre, and  
rating.  
2. PhysicalGame: Represents a physical copy of a video game and inherits from the Game class. It has  
additional attributes like the condition of the disc and its availability status.  
3. DigitalGame: Represents a digital copy of a video game and inherits from the Game class. It has additional  
attributes like the platform it can be played on and the download link.  
4. Inventory: Represents the inventory of the video game store and contains all the physical and digital games.  
5. Customer: Represents a customer of the video game store and has attributes like name, email, and a unique  
customer ID.

**Data and Function Members** (UML can be used here as well)  
Game:  
Attributes:  
- title: string  
- developer: string  
- publisher: string  
- release\_date: date  
- genre: string  
- rating: float  
Functions:  
- play(): a virtual function that can be overridden by the derived classes to provide different implementations  
of the function.

PhysicalGame:  
Attributes:  
- condition: string  
- availability\_status: bool  
Functions:  
- get\_condition(): returns the condition of the disc  
- is\_available(): returns the availability status of the game

DigitalGame:  
Attributes:  
- platform: string  
- download\_link: string  
Functions:  
- get\_platform(): returns the platform the game can be played on  
- get\_download\_link(): returns the download link of the game

Inventory:  
 Attributes:  
- physical\_games: vector of PhysicalGame  
- digital\_games: vector of DigitalGame  
Functions:  
- add\_game(game): adds a game to the inventory  
- remove\_game(game): removes a game from the inventory  
- update\_game(game): updates the details of a game in the inventory  
- search\_game(title): searches for a game by its title in the inventory  
- display\_inventory(): displays the inventory of physical and digital games

Customer:

Attributes:  
- name: string  
- email: string  
- customer\_id: int  
Functions:  
- get\_name(): returns the name of the customer  
- get\_email(): returns the email of the customer  
- get\_customer\_id(): returns the customer ID of the customer

**Relationships between Classes** (UML can be used here as well)  
- Game is the parent class of PhysicalGame and DigitalGame.  
- PhysicalGame and DigitalGame inherit the attributes and functions of the Game class.  
- The Inventory class contains both physical and digital games.  
- The Customer class is not related to any other class in the system.

Project Task List and Timeline  
1. Design the class structure and relationships (x days) (group member x)  
2. Implement the Game, PhysicalGame, and DigitalGame classes (x days) (group member x)  
3. Implement the Inventory and Customer classes (x days) (group member x)  
4. Implement the user menu and user interface (x days) (group member x)  
5. Implement the file input/output functions (x days) (group member z,y)  
6. Write unit tests for all classes and functions (x days) (group member x,y)  
7. Debug and refine the code (x days) (group member x,y,z)  
8. Write user documentation (x days) (group member x,y,z)  
User Interaction Description  
The user will interact with the program through a user-friendly menu that provides options for adding, deleting,  
and updating games in the inventory, managing customer information, and viewing transaction history. The menu  
will be implemented using prompts that will allow the user to select an option by entering a corresponding number  
or character.  
The program will also use error messages and confirmation prompts to guide the user through the program and  
ensure that the input is valid. For example, if the user tries to add a game that already exists in the inventory, the  
program will display an error message and prompt the user to enter a different game.  
The program will allow the user to view the inventory of physical and digital games by selecting the corresponding  
option from the menu. The user can search for a game by its title and view its details, including its availability  
status, condition, platform, and download link.  
The user can also add and manage customer information by selecting the customer management option from the  
menu. The program will allow the user to add a new customer, update customer information, and view a list of all  
customers in the system.  
Unit Testing and Debugging Plan  
The program will be tested using a combination of unit testing and organized input/output testing. The unit tests  
will cover each function of the classes and will ensure that the program performs as expected. The input/output  
tests will verify that the program reads and writes data correctly from the file and that the program handles user  
input and output correctly.  
The program will also use exception handling to handle unexpected inputs or situations and prevent crashes or  
errors. The program will log any errors or issues encountered during runtime and will provide error messages or  
prompts to guide the user through the program.  
Makefile and README files will be included in the project, which will provide clear instructions on how to compile,  
run and test the program. The Makefile will handle dependencies, and the debugging and release builds will be  
separated. The README file will explain how to use the program and will list any known issues or limitations of  
the program.