
	Academic Year:	2022/2023	Semester:	Spring	
	Course Code:	CMP 1242	Course Title:	Basic Programming	

Project Statement

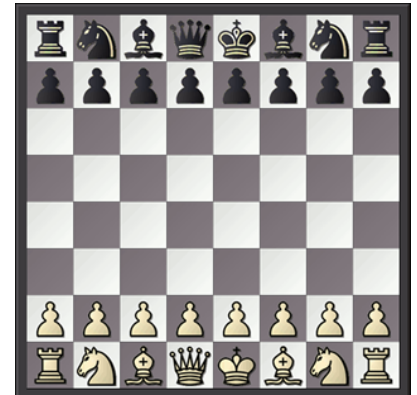
Multiplayer Chess Game

You are required to design and implement a **multiplayer chess game**.

It is required to apply all of **the fundamental concepts of OOP** including abstraction, polymorphism, inheritance, and encapsulation. In addition, we encourage you to make use of interfaces in your project.

Description:

Chess is played on a board of 64 squares alternate between two colors: light (e.g. white) and dark (e.g. black).



A player's army consists of 16 pieces that begin play on the two ranks closest to that player. There are six different types of pieces; **king**, **rook**, **bishop**, **queen**, **knight**, and **pawn**. Each has a different appearance and moves.

Assuming the traditional chess rules with some modifications that are **highlighted**.

Requirements:

I- General:

- The game is played between two opponents with different colors of pieces (commonly black and white)
- White moves first, after which the player's alternate turns are under fixed rules until the end of the game. A player can't make two successive moves.
- When a player selects a piece, all possible moves should be highlighted in green, while impossible moves should be highlighted in red.

- The eaten pieces of each participant should be displayed on the sidebar after they have successfully eaten a piece.
- It is important that your game correctly implements the win condition of chess, which is to force your opponent's king into checkmate (a position where it is unable to avoid capture).

a) **Pawn**

- A pawn can move only forward; it can never retreat.
- A pawn captures (eats) the squares diagonally or the piece in front.
- On its first move, a pawn can move either one or two squares from its starting position.
- When a pawn reaches the opposite end of the board, it can be promoted to any other piece except for a king. This means that your program should include functionality to allow players to choose which piece their pawn will be promoted to.

b) **Rook (Castle)**

- Each player has two rooks (castles) that can move vertically or horizontally to any unobstructed square.

c) **King**

- Each king can move one square in any direction.
- Your program should include functionality to allow players to perform a "castling" move, which allows a player to move their king two spaces towards a rook, and then move the rook to the other side of the king. This move can only be performed if all these conditions are met:
 - The king and rook involved have not yet been moved in the game.
 - There are no pieces between them.
 - The king is not in check.

d) **Bishop (Elephant)**

- Each player has two bishops.
- The elephant (bishop) can only move three steps on the diagonal and bypass any pieces. The elephant also has a new special move, which is to move one square on the horizontal axis so it changes the diagonal color.

e) **Queen**

- Each player has one queen, which combines the powers of the rook and bishop and is thus the most mobile and powerful piece.

f) **Knight**

- Each player has two knights.
- The knight can move in an L shape, with **two** squares horizontally or vertically, and then **three** squares in the perpendicular direction.

II- Bonus:

- **Timer:** Create a timer that counts down from a set time before starting the game. Each player will have a set amount of time to complete their moves. When a player starts their turn, their timer will begin to count down. The timer will stop when the player makes their move, and then the opponent's timer will start counting down. If a player's timer reaches 0 before they make their move, they will lose the game. It's important for the player to keep track of the time remaining and make strategic moves to ensure that you don't run out of time.
- **User account**
 - The user can create new account credentials "username and password".
 - If the credentials are incorrect, display an error message.
 - Upon login, the user can either:
 - i. Play a new game, when starting a new game, prompt the user to enter the name of each player. (Also prompt the user to set the timer, in case you implemented the bonus).
 - ii. Retrieve the scores of previous games, including the players' names and the winner, as well as the time elapsed to complete the game. "Each account shall have its own recorded games"
 - Save the result of each game so that a logged account can retrieve it later.
 - No database is required; you can use files to save and read the game history.

GitHub:

- Here is the link for your GitHub repository ([Link](#)).
- To form a team, the leader must first accept the assignment and create the team. Once the team has been created, the other members can join the team by accepting the team invitation. Be sure not to join multiple teams.
- All codes must be committed and pushed to your GitHub repo.
- In your repo, Create a folder called '1_deliverables' and upload any non-coding files to this folder.
- The Readme page of your repo shall contain a clear explanation of your project, preferably screenshots or gifs from the game.

- Convert the Readme page to a static website using GitHub pages.
- Feel free to create videos for your project and upload them to the [department's YouTube channel](#).

Final Presentation

- Each team shall document all the project steps to be presented in the final presentation.
- Create a Gantt chart to document the timeline of your project.
- Create a UML diagram to visually represent the structure of your program.
- Document the responsibility of each team member. You may use [Trello](#), [GitHub Issues](#), [Jira](#), etc.... to manage your project.

Milestones:

Date	Deliverables	Submission
April 25 th , 2023	UML diagram	GitHub
May 2 nd , 2023	GUI wireframe	GitHub
May 11 th , 2023	Final project code	GitHub
May 15 th , 2023	Final presentation	In class discussion

We are looking forward to seeing your innovative and creative design work.

Good Luck!