What's the point of this exercise?
The purpose of this exercise is for you to demonstrate your code design and implementation skills in the context of a toy Unity project.

Things we are looking for:
• Completion of all listed requirements
• **Intentional code.** We want to see that you think about the context and requirements of a specific problem and make intentional decisions about how to structure your code for that problem.
• We'll be reading your code with a focus on the following aspects:
  – **Clean and readable code style:** Is it easy to understand what the code is doing?
  – **Separation of concerns:** Does your code separate unrelated things, and group related things together?
  – **Encapsulation:** Do you structure your code in a way that attempts to prevent invalid use?
  – **Don't repeat yourself:** Do you factor out shared behavior into appropriate constructs and abstractions (functions, classes, MonoBehaviours, interfaces, etc)?
  – **Use language and engine constructs appropriately:**
    ∗ When and why do you choose to use a struct vs. a plain C# class vs. a MonoBehaviour?
    ∗ When and why do you choose to use polymorphism, interfaces, events, etc.
  – **Consistency:** Are you consistent about these choices across your solution?
  – **Performance:** Do you take steps to reduce unnecessary work and allocations?

Things we are NOT looking for:
  • We are not looking for a polished user experience. Nice user interactions are super important for real apps, but they take a long time! We don't want you to spend a lot of time on things we won't be judging you on, and we're really focused on your code, so only add extra nice things if you've fulfilled all the requirements, you're happy with your code, and you still feel like you want to spend more time on it.
  • We are not looking for perfection, or a right answer. We don't think there's one right solution or one right code style or anything like that, and we're not looking for code that looks exactly like ours. The thing we care most about is that you can pay attention to the context and demands of a specific problem and make intentional decisions about how to structure + design your code to solve that problem.

The Game

The toy project we're asking you to implement is based on Chess. Our simplified version of Chess has the following rules:
  • Each player takes turns making 1 single move at a time.
  • The player controlling the white pieces will go first.
  • The game ends when a player captures the opposing player's King (There is no check or checkmate in this simplified version of Chess).

• A piece is captured when a piece controlled by the opposing player is moved onto the same square as the piece being captured. The captured piece is removed from play.
• Each piece has different rules for how it may move, enumerated below:

Pawn

• A pawn moves straight forward (towards the other team's side of the board) one square, if that square is vacant. If it has not yet moved, a pawn also has the option of moving two squares straight forward, provided both squares are vacant. Pawns cannot move backwards.
• A pawn, unlike other pieces, captures differently from how it moves. A pawn can capture an enemy piece on either of the two squares diagonally in front of the pawn (but cannot move to those squares if they are vacant).

Knight

• A knight moves to the nearest square not on the same rank, file, or diagonal. (This can be thought of as moving two squares horizontally then one square vertically, or moving one square horizontally then two squares vertically—i.e. in an "L" pattern.) The knight is not blocked by other pieces: it jumps to the new location.

King

• A king moves 1 square in any direction: horizontally, vertically, or diagonally.

Figure 1: Pawn

Figure 2: Knight

Figure 3: King

What you are provided

- Unity Project that contains a Chess board with pieces already on the board.
- A Board component that finds and manages references to all the tiles on it, and publishes events for tiles that are clicked or hovered with the mouse cursor.
- A Tile component that contains its position.
- A Piece component that contains its position.

Requirements

Game Requirements

- Only allow movement of the pieces for the player whose turn it is.
- Implement the movement rules for all 3 pieces.
- Captured pieces are removed from the board.
- Do not allow any more movements to take place after the game end condition has been met.

Implementation Requirements

- Usage of any C# or Unity events. These could be either the events provided in the Board class or any other events you deem appropriate.