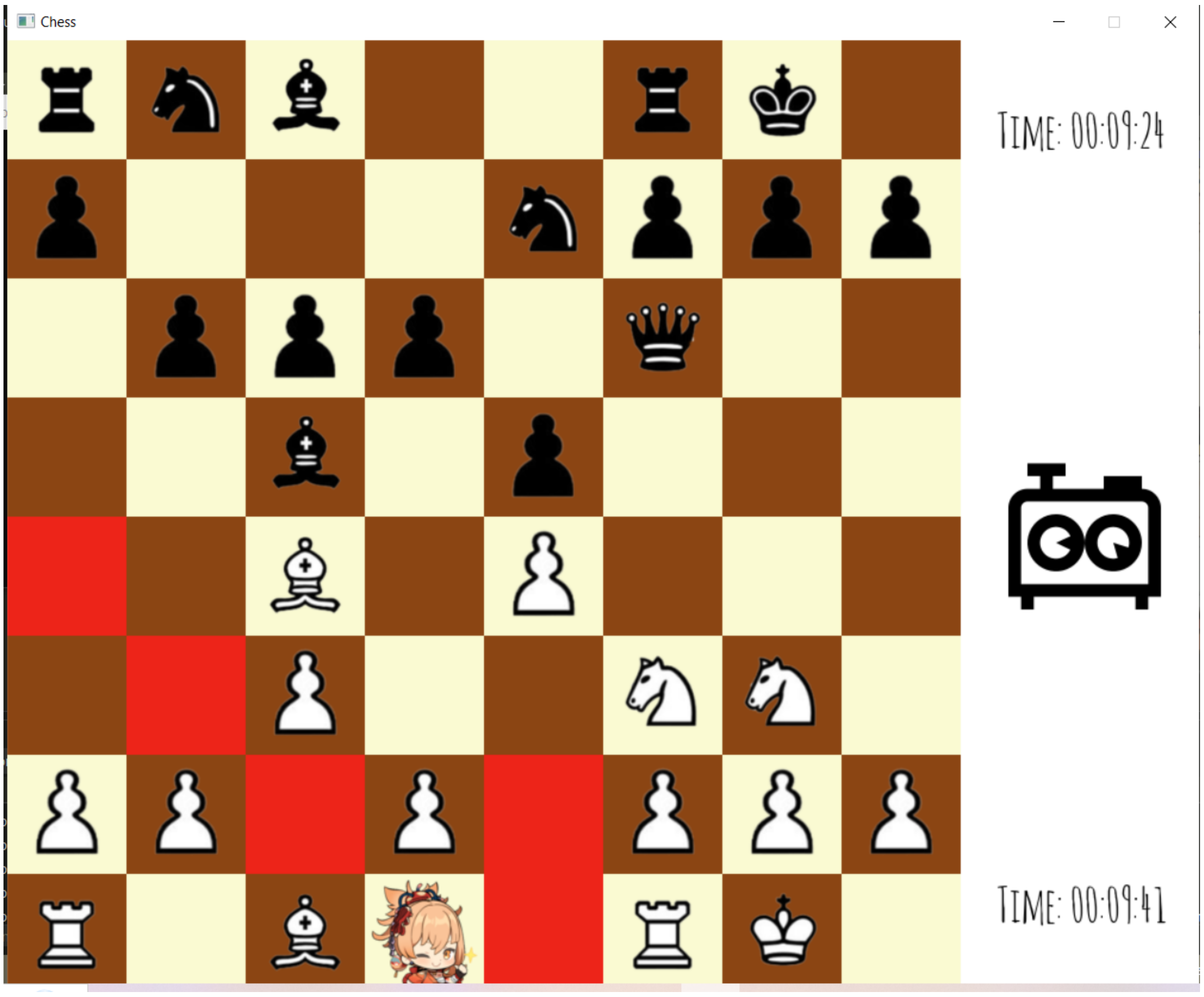
Design Overview for Epic Chess

Name: Le Minh Kha

Student ID: 104179506

Summary of Program

The project I have been working on is called `Epic C#hess` game provides support for playing a traditional chess game, where players can take turns making moves. Legal moves are highlighted on the board and the game prevents players from making illegal moves, especially if their king is already in check. In addition, the game also supports special moves such as castling, as well as detecting when a player's king is in checkmate.
The current project in D level only support Player vs Player mode, Player vs AI is set to be available in the HD custom program.



Required Roles

Describe each of the classes, interfaces, and any enumerations you will create. Use a different table to describe each role you will have, using the following table templates.

Role name: Board

Responsibility	**Type Details**	**Notes**
	Field type, parameter and return types	
Board	public class Board	Represents the chess board
Piece	public abstract class Piece	Represents a chess piece
Piece.Color	public enum Color { White, Black }	Represents the color of a piece
Current_Turn	Piece.Color Current_Turn { get; private set; }	Holds the current turn public
LegalMoves	public List<(int, int)> LegalMoves { get { return legalMoves; } }	Legal moves for the selected piece

Coordinate	public Piece Coordinate(int row, int col)	Accesses piece at coordinates, avoid obtaining piece outside of the array.
Draw	public void Draw()	Draws pieces on the board using a nested loop
Select_Piece	public bool Select_Piece(int mouseX, int mouseY, int squareSize)	Handles piece selection
Piece_Chosen	public void Piece_Chosen()	Calculating legal moves for the selected piece
Deselect_Piece	public void Deselect_Piece()	Deselects the selected piece

Role name: Piece

Responsibility	**Type Details**	**Notes**
	Field type, parameter and return types	
Piece	public abstract class Piece	Represents a chess piece
PieceColor	public Color PieceColor { get; set; }	Holds the piece color

Row	public int Row { get; set; }	Holds the row position of the piece
Col	public int Col { get; set; }	Holds the column position of the piece
LegalMoves	public List<(int, int)> LegalMoves { get { return legalMoves; } }	Legal moves for the selected piece
Coordinate	public Piece Coordinate(int row, int col)	Accesses piece at coordinates, avoid obtaining piece outside of the array.
Draw	public void Draw()	Draws pieces on the board using a nested loop
Move	public virtual void Move(int Rowed, int Columned, Board board)	Moves the piece on the board
Is_Valid_Move	public abstract bool Is_Valid_Move(int Rowed, int Columned, Board board)	Determines if a move is valid, then Move() can update coordinate accordingly.
Is_Selected	public bool Is_Selected(int mouseX, int mouseY, int square_Size)	Checks if a piece is selected
Has_Moved	public bool Has_Moved { get; set;}	Indicates if the piece has moved before

Role name: GameManager

Responsibility	**Type Details**	**Notes**
	Field type, parameter and return types	
GameManager	public class GameManager	Manages the game states and controls the game
_instance	private static GameManager _instance	Holds the singleton instance of GameManager
Menu	public Menu Menu { get; private set; }	Control the Menu class

timeFormat	Choose_Format timeFormat { get; private set; }	Represents the time format choice menu, control through timeFormat class.
ChooseAI	public Choose_AI_Opponent ChooseAI { get; private set; }	represents the AI opponent choice menu, controlled from chooseAI opponent class
GameState	public GameState GameState { get; private set; }	Represents the current game state
UserInputHandler	public UserInputHandler UserInputHandler { get; private set; }	Handles user input for game events
Select_Piece	public bool Select_Piece(int mouseX, int mouseY, int squareSize)	Handles piece selection
GameManager()	private GameManager()	Private constructor to ensure the singleton pattern
Instance	public static GameManager Instance	Handle everything through one single instance in singleton fashion

Role name: AI

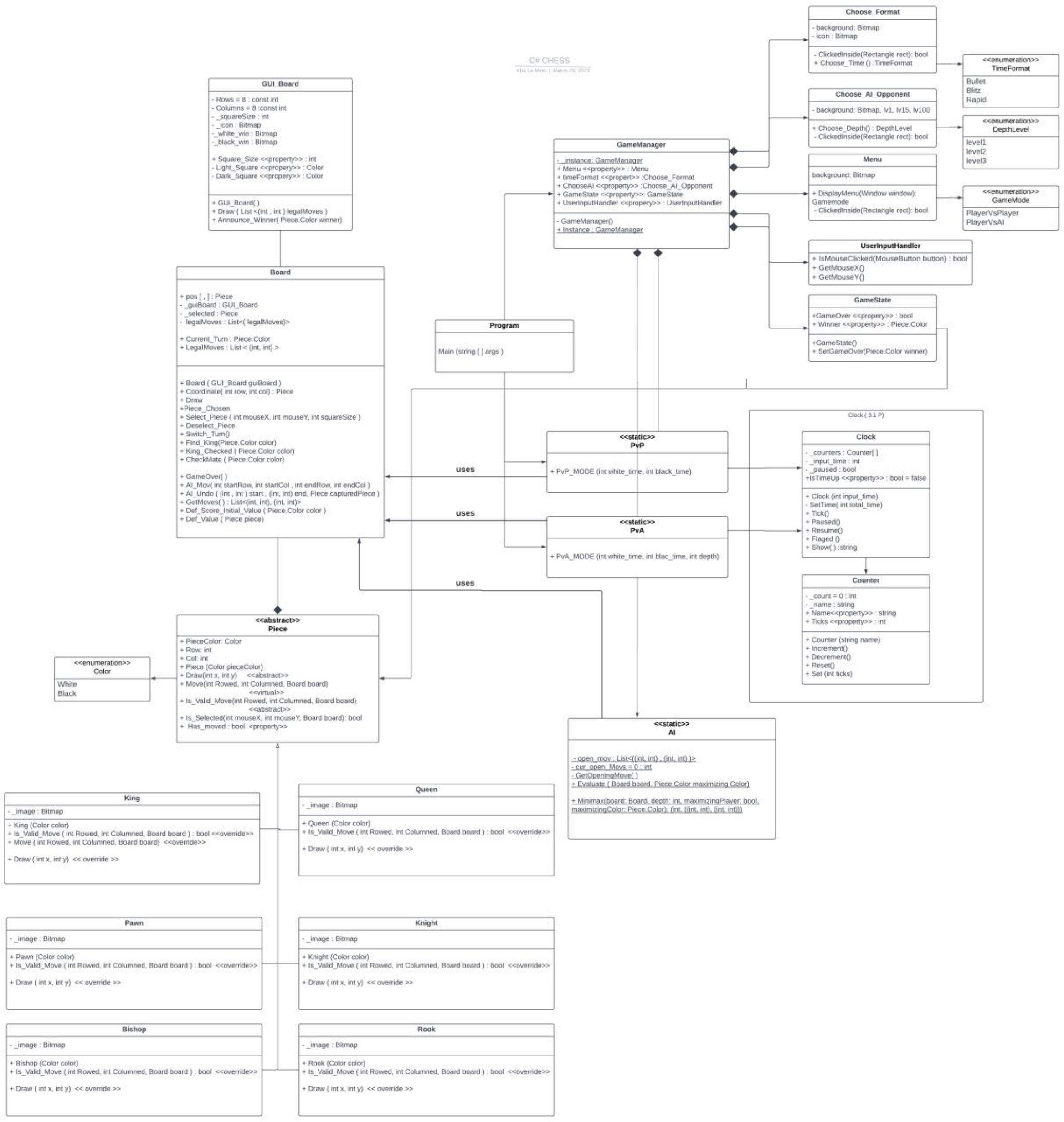
Responsibility	**Type Details**	**Notes**
	Field type, parameter and return types	
AI	public static class AI	Represents the Artificial Intelligence for the game
open_mov	private static List<((int, int), (int, int))> open_mov	Holds a list of predetermined opening move to instruct the AI, in this case it is the King’s Indian Opening.
cur_open_Movs	private static int cur_open_Movs	Tracks the current index of opening moves
GetOpeningMove	public static ((int, int), (int, int)) GetOpeningMove()	Gets the next opening move from the list

Evaluate	public List<(int, int)> LegalMoves { get { return legalMoves; } }	Evaluates the board state for black using material as the primary variable of the equation.
Minimax	public Piece Coordinate(int row, int col)	Performs the Minimax algorithm for AI decision-making. The algorithm call it self recursive to evaluate all of the possible moves at depth 3 and return a static evaluation base on material.
Draw	public void Draw()	Draws pieces on the board using a nested loop
Move	public virtual void Move(int Rowed, int Columned, Board board)	Moves the piece on the board
Is_Valid_Move	public abstract bool Is_Valid_Move(int Rowed, int Columned, Board board)	Determines if a move is valid, then Move() can update coordinate accordingly.
Is_Selected	public bool Is_Selected(int mouseX, int mouseY, int square_Size)	Checks if a piece is selected
Has_Moved	public bool Has_Moved { get; set;}	Indicates if the piece has moved before

Table 1: <<enumeration name>> details

Value	Notes
TimeFormat	Bullet Blitz Rapid
DepthLevel	level1 level2 level3
GameMode	PlayerVsPlayer PlayerVsAI
Color	White Black

Class Diagram


C# CHESS
GUI_Board
- Rows = 8 : const int
- Columns = 8 :const int
- _squareSize : int
- _icon : Bitmap
-_white_win : Bitmap
-_black_win : Bitmap
+ Square_Size <<property>> : int
- Light_Square <<propery>> : Color
- Dark_Square <<propery>> : Color
+ GUi_Board( )
+ Draw (List <(int , int) legalMoves)
+ Announce_Winner(Piece.Color winner)
Board
+ pos [,] : Piece
- _guiBoard : GUI_Board
- _selected : Piece
- legalMoves : List<(legalMoves)>
+ Current_Turn : Piece.Color
+ LegalMoves : List < (int, int) >
+ Board (GUI_Board guiBoard)
+ Coordinate(int row, int col) : Piece
+ Draw
+Piece_Chosen
+ Select_Piece (int mouseX, int mouseY, int squareSize)
+ Deselect_Piece
+ Switch_Turn()
+ Find_King(Piece.Color color)
+ King_Checked (Piece.Color color)
+ CheckMate (Piece.Color color)
+ GameOver( )
+ AI_Mov(int startRow, int startCol , int endRow, int endCol)
+ AI_Undo ((int , int) start , (int, int) end, Piece capturedPiece)
+ GetMoves( ) : List<(int, int), (int, int)>
+ Def_Score_Initial_Value (Piece.Color color)
+ Def_Value (Piece piece)
Program
Main (string [ ] args)
GameManager
- _instance: GameManager
+ Menu <<property>> : Menu
+ timeFormat <<propert>> :Choose_Format
+ ChooseAI <<property>> :Choose_AI_Opponent
+ GameState <<property>>: GameState
+ UserInputHandler <<propery>> : UserInputHandler
- GameManager()
+ Instance : GameManager
Choose_Format
- background: Bitmap
- icon : Bitmap
- ClickedInside(Rectangle rect): bool
+ Choose_Time () :TimeFormat
<<enumeration>> TimeFormat
Bullet
Blitz
Rapid
Choose_AI_Opponent
- background: Bitmap, lv1, lv15, lv100
+ Choose_Depth() : DepthLevel
- ClickedInside(Rectangle rect): bool
<<enumeration>> DepthLevel
level1
level2
level3
Menu
background: Bitmap
+ DisplayMenu(Window window): Gamemode
- ClickedInside(Rectangle rect): bool
<<enumeration>> GameMode
PlayerVsPlayer
PlayerVsAI
UserInputHandler
+ IsMouseClicked(MouseButton button) : bool
+ GetMouseX()
+ GetMouseY()
GameState
+GameOver <<propery>> : bool
+ Winner <<property>> : Piece.Color
+GameState()
+ SetGameOver(Piece.Color winner)
uses
uses
uses
<<static>> PvP
+ PvP_MODE (int white_time, int black_time)
<<static>> PvA
+ PvA_MODE (int white_time, int blac_time, int depth)
Clock (3.1 P)
Clock
- _counters : Counter[ ]
- _input_time : int
- _paused : bool
+IsTimeUp <<property>> : bool = false
+ Clock (int input_time)
- SetTime(int total_time)
+ Tick()
+ Paused()
+ Resume()
+ Flaged ()
+ Show( ) :string
Counter
- _count = 0 : int
- _name : string
+ Name<<property>> : string
+ Ticks <<property>> : int
+ Counter (string name)
+ Increment()
+ Decrement()
+ Reset()
+ Set (int ticks)
<<abstract>> Piece
+ PieceColor: Color
+ Row: int
+ Col: int
+ Piece (Color pieceColor)
+ Draw(int x, int y) <<abstract>>
+ Move(int Rowed, int Columned, Board board) <<virtual>>
+ Is_Valid_Move(int Rowed, int Columned, Board board) <<abstract>>
+ Is_Selected(int mouseX, int mouseY, Board board): bool
+ Has_moved : bool <property>>
<<enumeration>> Color
White
Black
<<static>> AI
- open_mov : List<((int, int) , (int, int))>
- cur_open_Movs = 0 : int
- GetOpeningMove( )
+ Evaluate (Board board, Piece.Color maximizing Color)
+ Minimax(board: Board, depth: int, maximizingPlayer: bool, maximizingColor: Piece.Color): (int, ((int, int), (int, int)))
King
- _image : Bitmap
+ King (Color color)
+ Is_Valid_Move (int Rowed, int Columned, Board board) : bool <<override>>
+ Move (int Rowed, int Columned, Board board) <<override>>
+ Draw (int x, int y) << override >>
Queen
- _image : Bitmap
+ Queen (Color color)
+ Is_Valid_Move (int Rowed, int Columned, Board board) : bool <<override>>
+ Draw (int x, int y) << override >>
Pawn
- _image : Bitmap
+ Pawn (Color color)
+ Is_Valid_Move (int Rowed, int Columned, Board board) : bool <<override>>
+ Draw (int x, int y) << override >>
Knight
- _image : Bitmap
+ Knight (Color color)
+ Is_Valid_Move (int Rowed, int Columned, Board board) : bool <<override>>
+ Draw (int x, int y) << override >>
Bishop
- _image : Bitmap
+ Bishop (Color color)
+ Is_Valid_Move (int Rowed, int Columned, Board board) : bool <<override>>
+ Draw (int x, int y) << override >>
Rook
- _image : Bitmap
+ Rook (Color color)
+ Is_Valid_Move (int Rowed, int Columned, Board board) : bool <<override>>
+ Draw (int x, int y) << override >>

Note: this class diagram is the finished one which include even some of the classes that belong to HD design (i.e. Game Manager and its support class, the AI mode). I will not be elaborating it, yet!

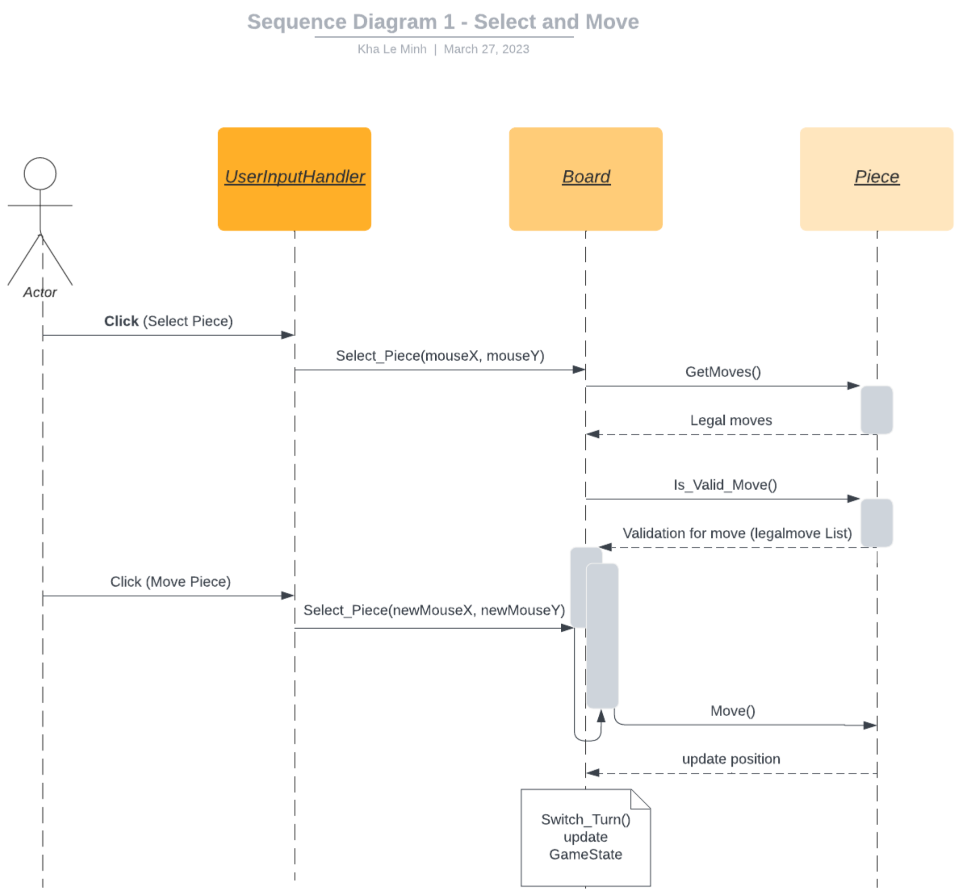
If the image happens to crack or broke upon zooming, here is the alternative link to the software I used to design the chart: https://lucid.app/lucidchart/f325cc8b-05c1-4fd1-9dd0-

42a1ffe13559/edit?viewport_loc=-158%2C1608%2C2560%2C1116%2CT7JIHdY0eP.Q&invitationId=inv_4cc24a8c-2289-4018-97de-722d316dc97d

Sequence Diagram

Sequence diagram of player click on the piece and move:

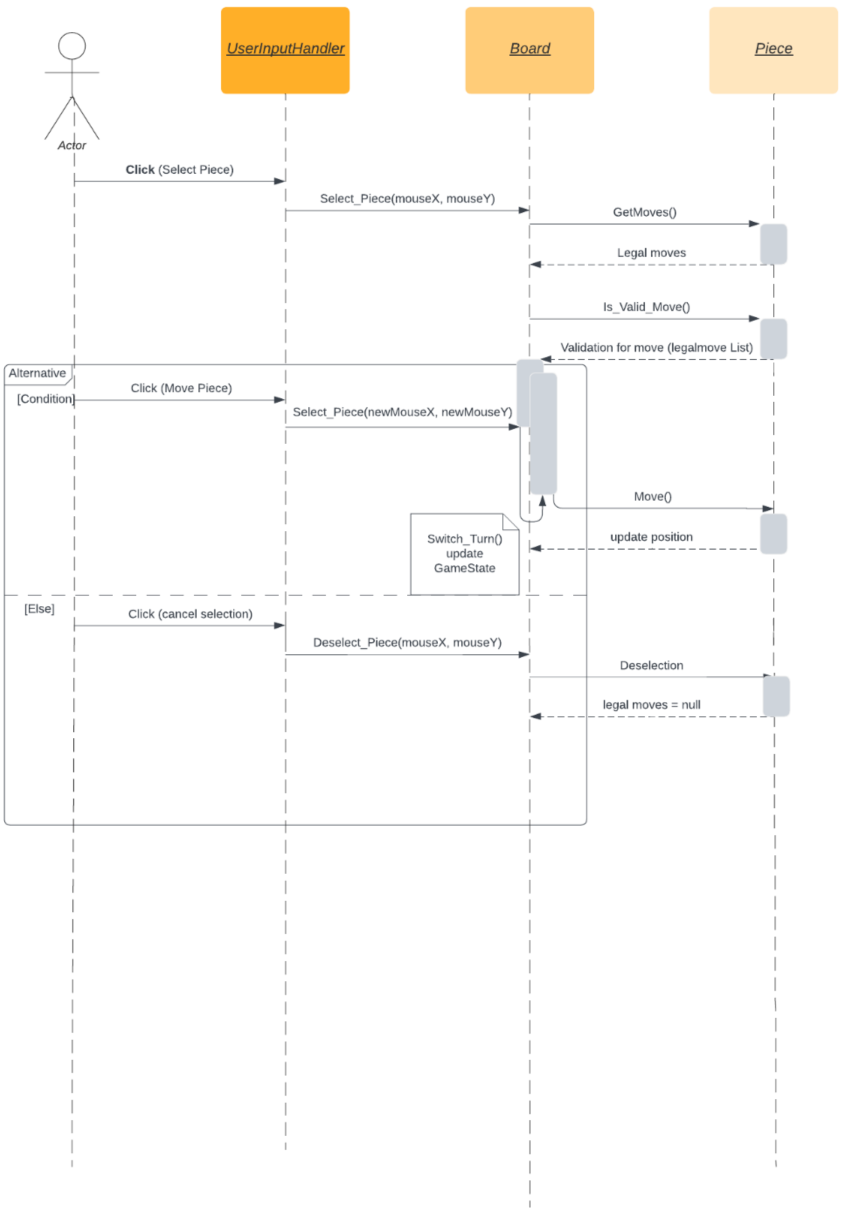
Link for better inspection: https://lucid.app/lucidchart/64012d45-7fd6-49d9-b28a-a9a23ca2e8ed/edit?viewport_loc=-27%2C-23%2C2139%2C1029%2C0_0&invitationId=inv_87f12063-ddaf-4dd9-bd78-0ac52c8469e0


Sequence Diagram 1 - Select and Move
Kha Le Minh | March 27, 2023
UserInputHandler
Board
Piece
Actor
Click (Select Piece)
Select_Piece(mouseX, mouseY)
GetMoves()
Legal moves
Is_Valid_Move()
Validation for move (legalmove List)
Click (Move Piece)
Select_Piece(newMouseX, newMouseY)
Move()
update position
Switch_Turn() update GameState

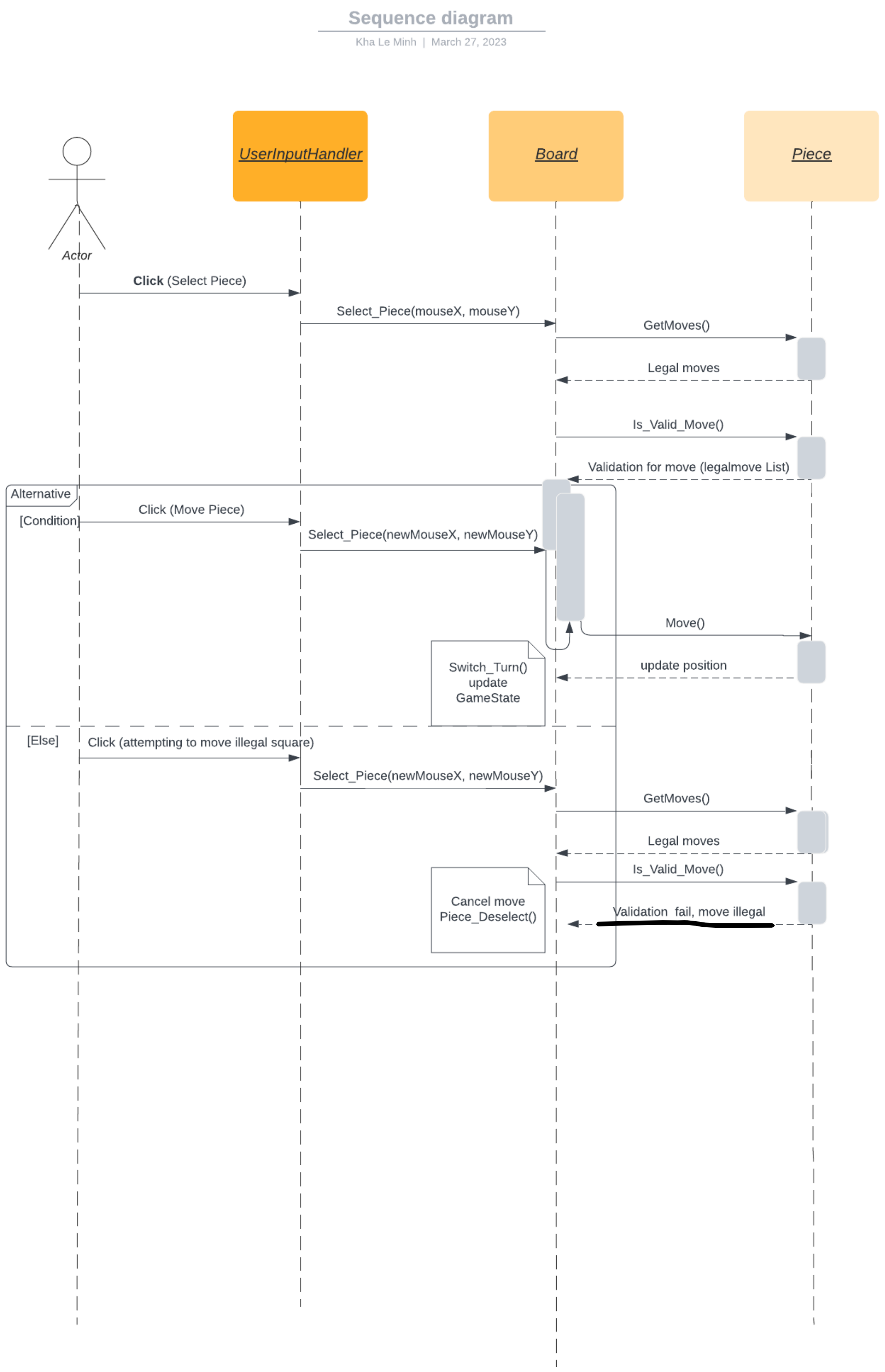
Sequence diagram including deselecting a piece:

Sequence Diagram 1 - Select and Move

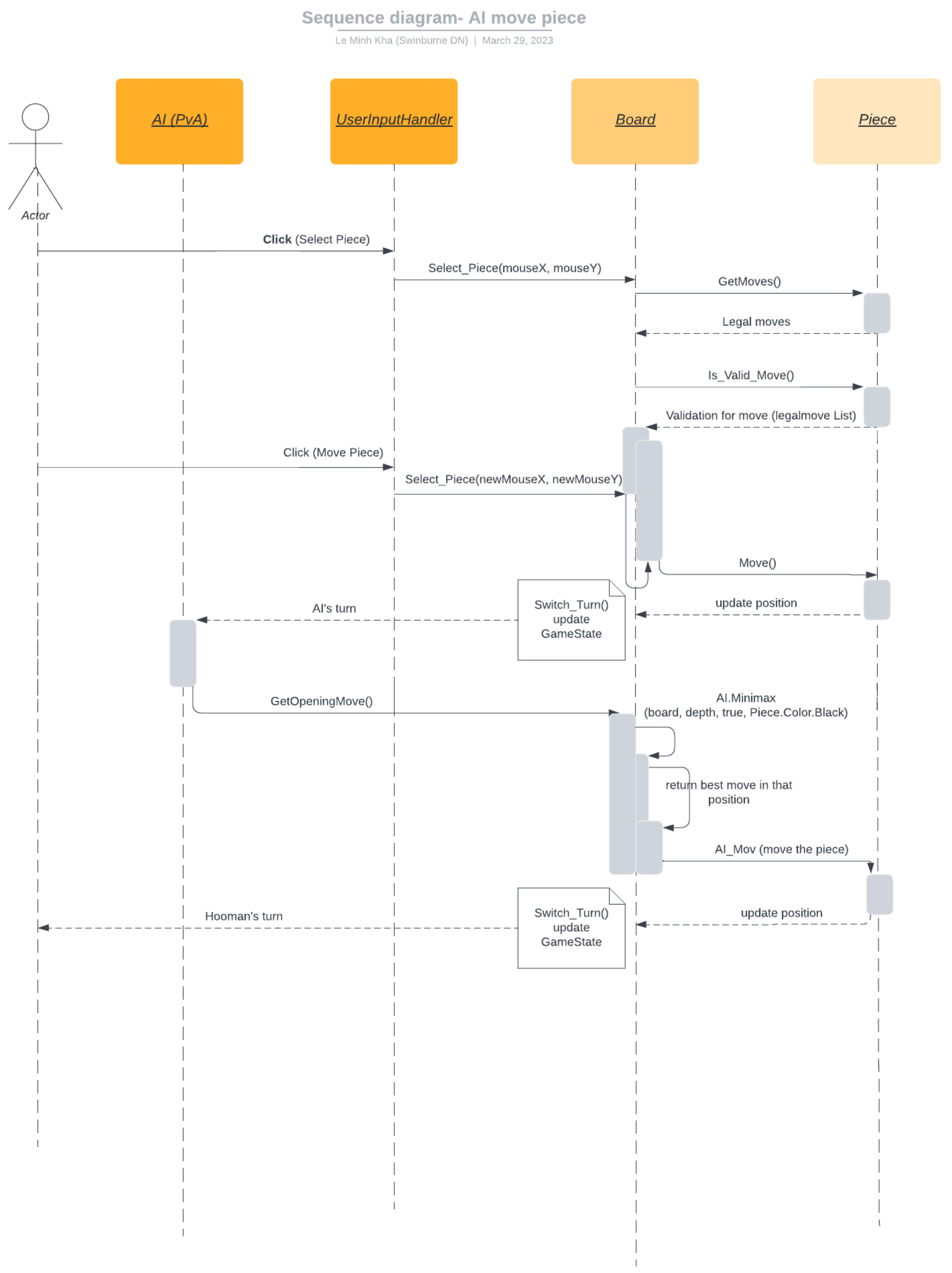
Kha Le Minh | March 27, 2023



Sequence diagram for attempting to move to an illegal square:


Sequence diagram
Kha Le Minh | March 27, 2023
Actor
UserInputHandler
Board
Piece
Click (Select Piece)
Select_Piece(mouseX, mouseY)
GetMoves()
Legal moves
Is_Valid_Move()
Validation for move (legalmove List)
Alternative
[Condition]
Click (Move Piece)
Select_Piece(newMouseX, newMouseY)
Move()
update position
Switch_Turn() update GameState
[Else]
Click (attempting to move illegal square)
Select_Piece(newMouseX, newMouseY)
GetMoves()
Legal moves
Is_Valid_Move()
Cancel move Piece_Deselect()
Validation fail, move illegal

Sequence diagram AI using minimax algorithm to move a chess piece:



Sequence diagram- AI move piece
Le Minh Kha (Swinburne DN) | March 29, 2023
Actor
AI (PvA)
UserInputHandler
Board
Piece
Click (Select Piece)
Select_Piece(mouseX, mouseY)
GetMoves()
Legal moves
Is_Valid_Move()
Validation for move (legalmove List)
Click (Move Piece)
Select_Piece(newMouseX, newMouseY)
Move()
update position
Switch_Turn() update GameState
AI's turn
GetOpeningMove()
AI.Minimax (board, depth, true, Piece.Color.Black)
return best move in that position
AI_Mov (move the piece)
update position
Switch_Turn() update GameState
Hooman's turn

Link for better inspection:

https://lucid.app/lucidchart/7a0547c8-9a0d-43eb-8db4-69c18bf9095e/edit?viewport_loc=-208%2C-376%2C2220%2C911%2C0_0&invitationId=inv_df51c41c-9565-4871-9613-346b915c2bd7

Question 2: Provide a summary of your program — What does it do? What are some of the key features

`Epic C#hess` uses Splash Kit is a graphical chess application that allows users to play chess against another player (Player vs Player) or against an AI (Player vs AI). The key features of the program include:

Game Modes:
Player vs Player: Two players can play against each other in a local multiplayer setup.
Player vs AI: Will be available in HD project design.

A visually appealing chessboard with clear and distinct chess piece icons, making it easy for players to recognize and interact with the game pieces.
A menu screen that allows users to choose between different game modes.
In-game timer for both players, displayed on the screen, to keep track of the time each player has spent on their moves.

Chess mechanics:
The program supports some standardized chess rules, such as valid piece movements, capturing, and checkmate detection.
User Input:
The game handles mouse input for selecting and moving chess pieces.
The players can interact with the chessboard using simple click actions for a very smooth gameplay experience.

==

Question3: Describe the main roles: enumerations, classes & interfaces.

Enumerations:

Piece.Color: An enumeration representing the colour of a chess piece (White or Black).
TimeFormat: An enumeration representing the time format for each side (Bullet, Blitz, Rapid).
DepthLevel: An enumeration representing the AI depth level (level1, level2, level3).
GameMode: An enumeration representing the game mode (PlayerVsPlayer, PlayerVsAI).
Classes:

Board: The class that manage all of the mechanic of a conventional chess game, it represents the chessboard and its game state, including the positions of the pieces and the current turn. It also contains methods to handle piece movements, turn switching, and checkmate detection.

Piece: An abstract class representing a generic chess piece, with properties such as color, position, and methods to handle piece-specific movements.
King, Queen, Rook, Bishop, Knight, Pawn: These classes inherit from the parent Piece class, each representing a specific type of chess piece and their unique movement rules.

Clock: Reference from Clock 3.1 P with some additional methods, it represents the in-game timer for each player, with methods to handle ticking, pausing, and displaying the remaining time.

GUI_Board: A class responsible for rendering the graphical elements of the chessboard, including the squares, pieces, and any additional visual elements like legal moves and the winner announcement.

PvA and PvP: These static classes define the main game loop and handle the logic for each game mode (Player vs AI and Player vs Player, respectively).

Some of the notable methods that might need some extra explanations are:

Draw () Method:

The Draw () method is responsible for drawing all the pieces on the chessboard. This method uses a for loop in a for loop to iterate over all the cells on the board. It checks if a piece is present at the cell, and then calls the Draw () method of that piece to draw it on the GUI board. The x and y coordinates of the piece are calculated by multiplying the cell row and column by the size of a single square on the GUI board.

Select_Piece () Method:

The Select_Piece () method is responsible for handling piece selection and deselection. This method takes the mouse x and y coordinates and the size of a single square on the GUI board as input parameters. The method first checks if the clicked cell is valid and then gets the clicked piece using the Coordinate () method. If a piece is already selected, the method checks if the clicked cell is a legal move for the selected piece. If it is, then the Move () method of the selected piece is called to move the piece to the new cell, and the turn is switched to the other player using the Switch_Turn() method. If the clicked cell is not deemed a legal move, the method will then checks if another piece is selected and if it is, then deselects it using the Deselect_Piece () method. If no piece is currently selected, the method selects the clicked piece if it belongs to the current player.

Piece_Chosen () Method:

The Piece_Chosen () method is called when a piece is selected and is responsible for generating a list of legal moves for the selected piece. This method for loop in a for loop to iterate over all the cells on the board and checks if the selected piece can move to that cell using the Is_Valid_Move () method of the piece. If the move is legal, the cell coordinates are added to the LegalMoves list.

CheckMate () Method:

The CheckMate () method is responsible for detecting if the current player is mated or not. The method first checks if the king of the current player is under attack using the King_Checked () method. If the king is not being checked, then the method returns false. Else, the method iterates over all the pieces of the current player and checks if any of them can move to a cell where the king is not under attack. If that move is found, the method returns false, and the game continues. Else, the method returns true, indicating that the current player is in checkmate, and the game is over.

Question4: Describe the main responsibilities for the classes and interfaces.

Classes and its responsibility:

Board:

Manages the chessboard state, including pieces' positions and the current turn.
Handle piece movement and updates
Switch turns.
Check for checkmate conditions.

GUI_Board:

Renders the GUI of the chessboard.
Draw squares, pieces, and other visual elements on the board.
Highlight legal moves and announce the winner.

Piece (abstract):

Represents a generic chess piece with shared properties and methods.
Store piece's color, position, and movement status
Define methods for movement validation and updating positions
Provide a foundation for derived classes to implement specific behaviour
King, Queen, Rook, Bishop, Knight, Pawn: Inherits from Piece and implements specific movement rules and visuals for each chess piece.
Override movement validation methods to implement piece-specific rules

Override drawing methods to display the correct piece visuals

Clock:

Manages the in-game timer for each player.
Count down the remaining time for each player.
Pause and resume the timer based on the current turn.
Display the remaining time for both players.

AI

Generate AI moves based on the current board state.
Evaluate the board state to determine the best move.
Implement search minimax search algorithms to determine best outcome.
Interact with the game environment by providing chosen moves.

PvA (Player vs AI):

Set up the game environment for a human player to compete against an AI opponent.
Manage the game loop, user input, and AI moves.
Handle game state updates, such as checking for checkmate or flagged clocks.
Display game information, including the chessboard, legal moves, and clocks.

PvP (Player vs Player):

Set up the game environment for two human players to compete against each other.
Manage the game loop and user input for both players.
Handle game state updates, such as checking for checkmate or flagged clocks.
Display game information, including the chessboard, legal moves, and clocks.

GameManager

Provide a centralized access point.

The GameManager is implemented as a singleton class, ensuring there's only one instance of it throughout the game. This enables other classes to access game components and game state information easily.

Question 5 : Show your plans to your tutor, lecturer, help desk staffers, and/or friends to get some feedback.

Reference for the chess pieces' image as my tutor requested:

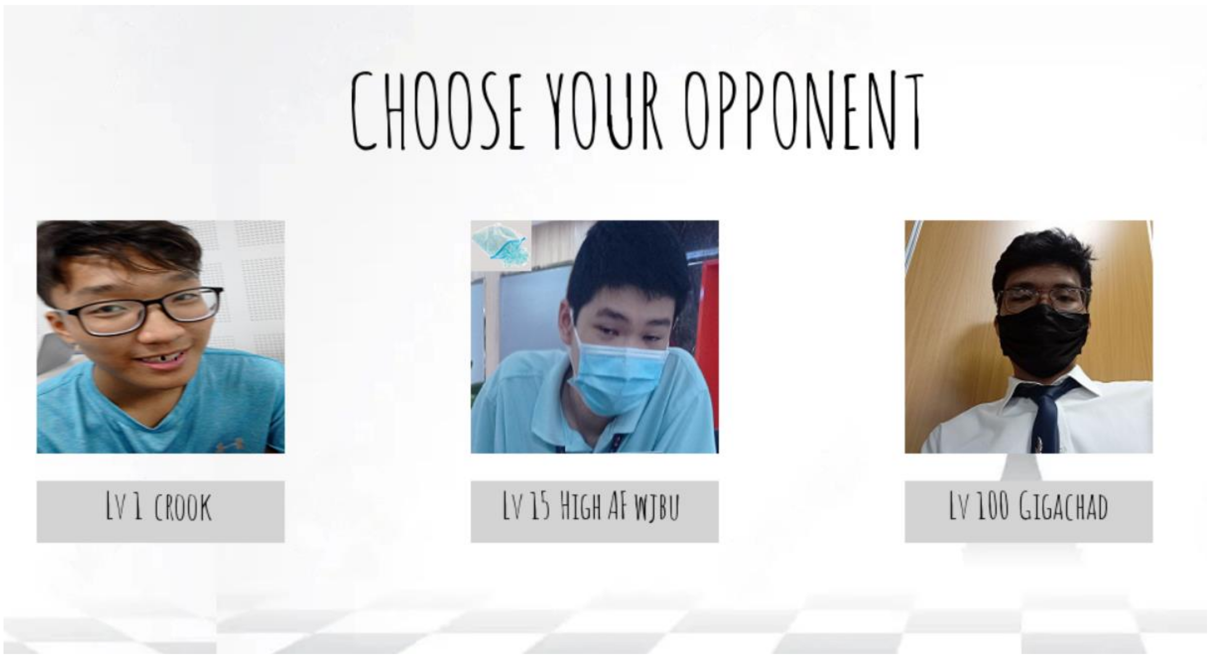
Brown, A. (n.d.). *Chess piece queen king game chess pieces transparent - 2D chess piece PNG, PNG Download - Kindpng*. KindPNG.com. Retrieved March 28, 2023, from https://www.kindpng.com/imgv/ToiTwm_chess-piece-queen-king-game-chess-pieces-transparent/

HD sections: enhancements, and improvement:

Q2:

1. Creating an Artificial Intelligent class to play as an opponent against the player.

This is a 2 phase mission, converting the pseudo code to usable C# code that match with my game and creating a way for the AI to move the piece on the chess board in its turn.
+ Minimax algorithm: It takes in the current board position, the depth to which the algorithm should evaluate, a boolean value indicating whether the AI player is maximizing or minimizing, and the color of the AI player. The method recursively evaluates all possible moves up to the given depth, and returns the best move it finds. The `depth` parameter is a flexible one since it could be pass from the choose opponent class to determine the intelligent of the AI, the higher the depth the more accurate the algorithm search creating a better response from the AI.



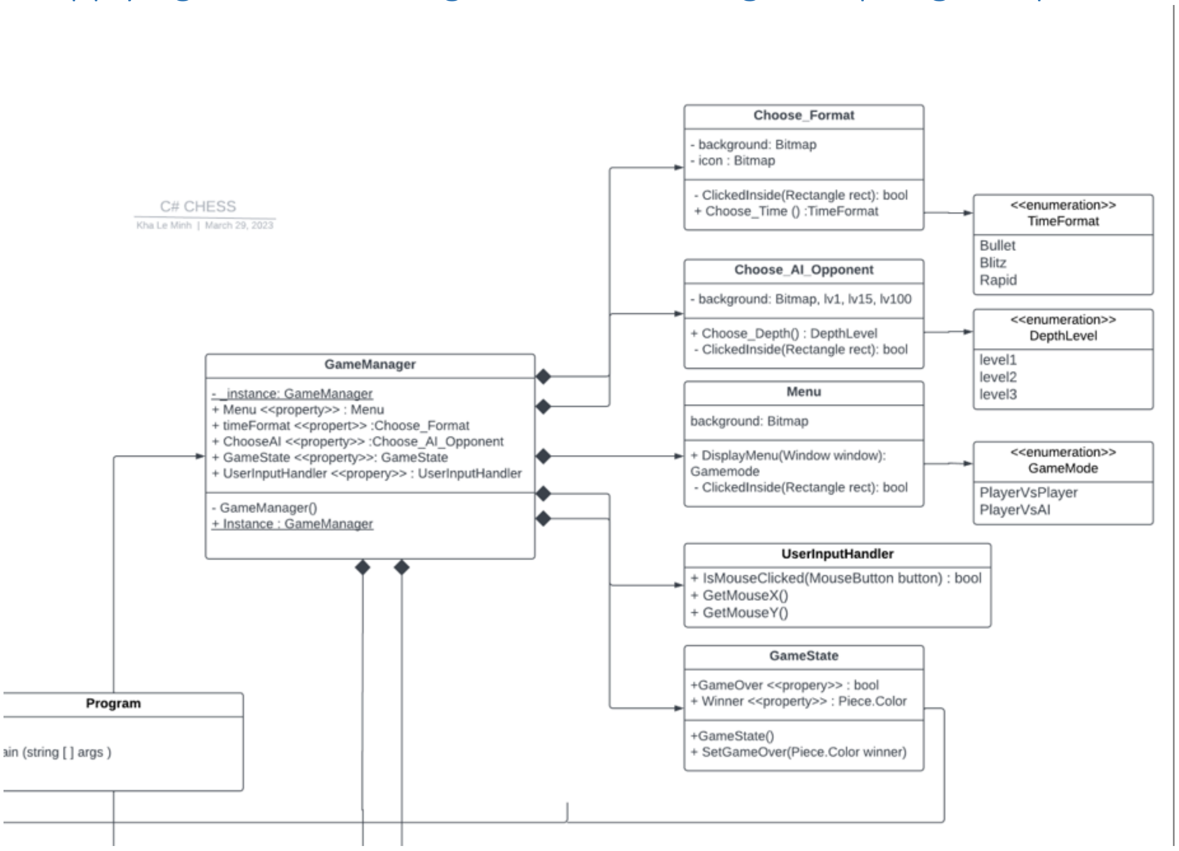
The Evaluate() method is the backbone of the minimax algorithm, it is used to calculate a score for the current board position, the current version of the AI is using a very simple method of subtracting the black player's score from the white player's score, or in other word, returning the static evaluation base on the material of the position.

This method of returning a static evaluation of the board is very passive and does not search far, making the AI sometimes make some repetitive moves. Hence, I create a list containing a pre-determined moves for the AI to follow, in chess its call an opening book move and it allows the AI’s piece to move closer to enemy piece, making it easier to capture because of the material based evaluation. I chose the move order to be an opening called the King’s Indian in chess.

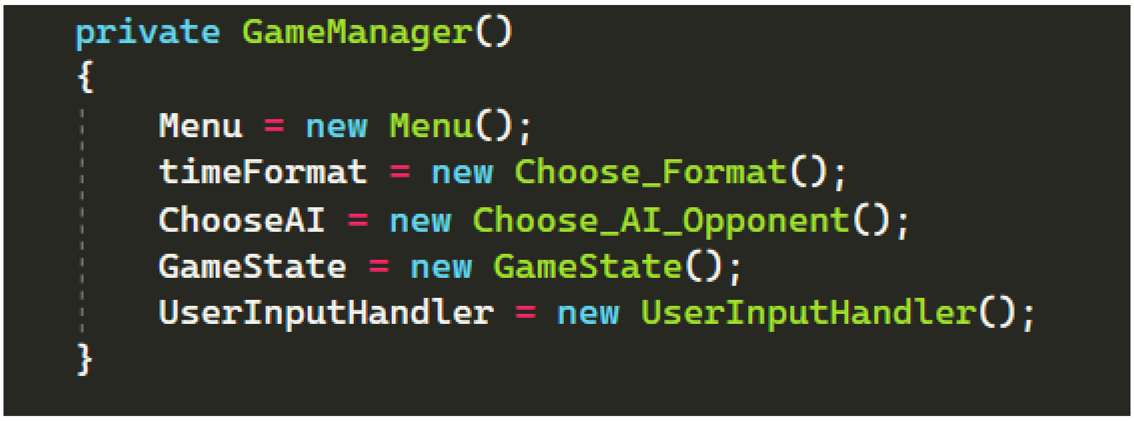
Reference:

Minimax algorithm guide: https://www.youtube.com/watch?v=l-hh51ncgDI
The pseudo code for Minimax algorithm: https://pastebin.com/VSehqDM3

2. Applying a Game Manager class to manage the pre-game phase.

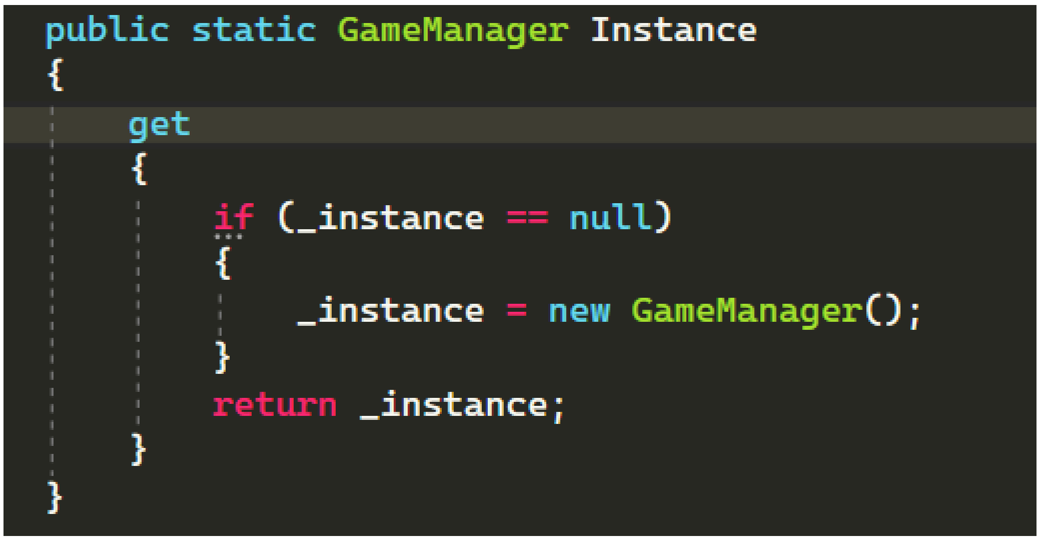

C# CHESS
Kha Le Minh | March 29, 2023
Choose_Format
- background: Bitmap
- icon : Bitmap
- ClickedInside(Rectangle rect): bool
+ Choose_Time () :TimeFormat
<<enumeration>> TimeFormat
Bullet
Blitz
Rapid
Choose_AI_Opponent
- background: Bitmap, lv1, lv15, lv100
+ Choose_Depth() : DepthLevel
- ClickedInside(Rectangle rect): bool
<<enumeration>> DepthLevel
level1
level2
level3
GameManager
- _instance: GameManager
+ Menu <<property>> : Menu
+ timeFormat <<propert>> :Choose_Format
+ ChooseAI <<property>> :Choose_AI_Opponent
+ GameState <<property>>: GameState
+ UserInputHandler <<propery>> : UserInputHandler
- GameManager()
+ Instance : GameManager
Menu
background: Bitmap
+ DisplayMenu(Window window): Gamemode
- ClickedInside(Rectangle rect): bool
<<enumeration>> GameMode
PlayerVsPlayer
PlayerVsAI
UserInputHandler
+ IsMouseClicked(MouseButton button) : bool
+ GetMouseX()
+ GetMouseY()
GameState
+GameOver <<propery>> : bool
+ Winner <<property>> : Piece.Color
+GameState()
+ SetGameOver(Piece.Color winner)
Program
ain (string [ ] args)

As can be seen from this UML class diagram, I created classes to include a welcome menu, a option to choose time format, and option to choose an AI opponent, all of which is managed through the single Game Manger class in a singleton design pattern, making it really easy to manage and change features which is really reminiscence of the command-processor in swin-adventure.



```
    private GameManager()
    {
        Menu = new Menu();
        timeFormat = new Choose_Format();
        ChooseAI = new Choose_AI_Opponent();
        GameState = new GameState();
        UserInputHandler = new UserInputHandler();
    }
```

Static method, Instance, which returns the single instance of the class. This ensures that only one instance of the class can be created and accessed throughout the application.



```
public static GameManager Instance
{
    get
    {
        if (_instance == null)
        {
            _instance = new GameManager();
        }
        return _instance;
    }
}
```

Q3:
Like the previous response, I have already implemented the singleton design pattern for my class. Here are some design pattern that could be implemented into the chess game:

Strategy Pattern: Can be used to encapsulate algorithms for piece movement, scoring, and other game mechanics. Each piece can be given a different strategy that governs its behavior on the board.

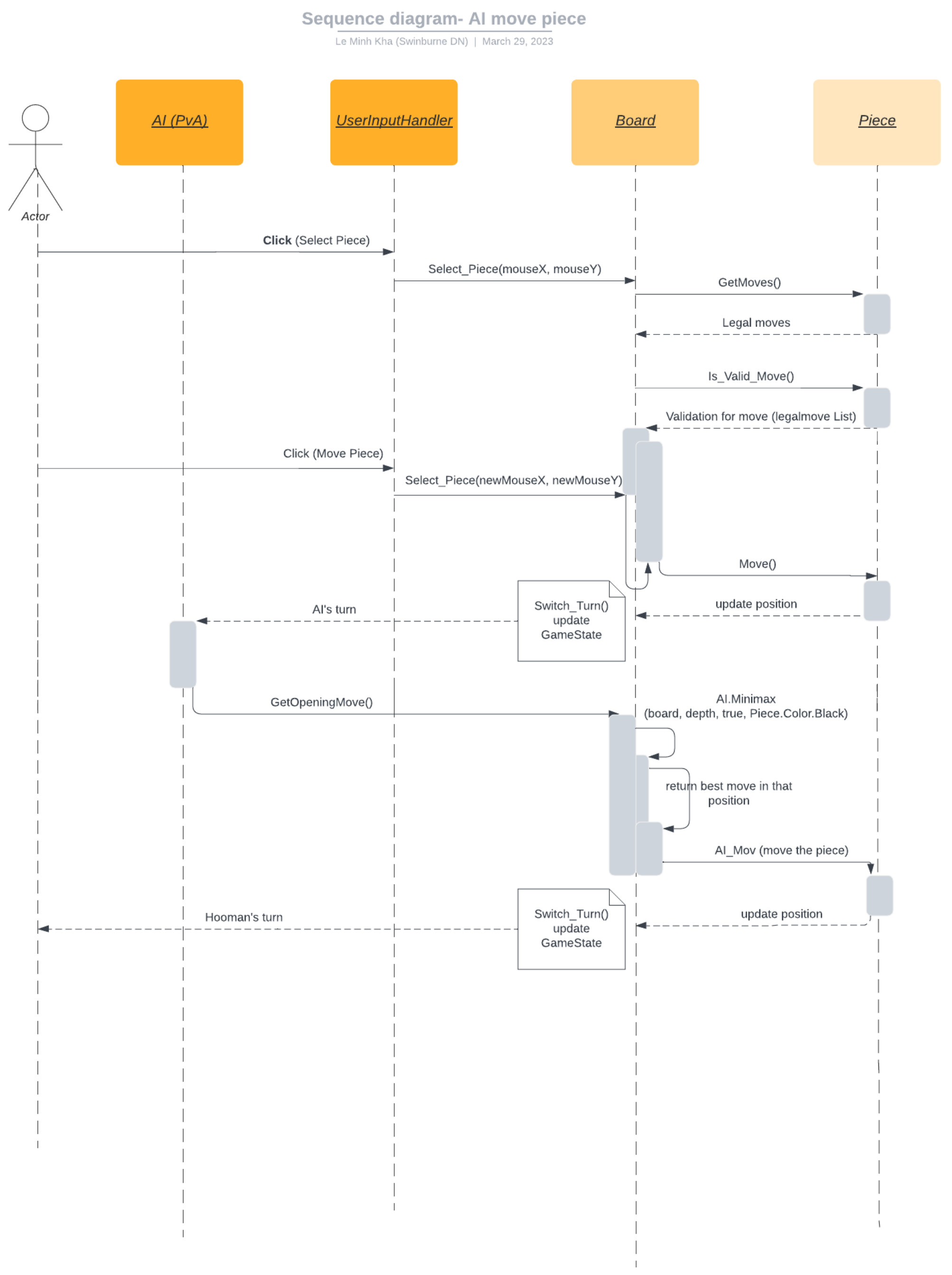
Command Pattern: Can be used to encapsulate user actions (i.e., moving a piece) as objects. This allows the GameManager to execute user actions and also can undo them if needed.

Q4:
The design pattern I have implemented is the singleton because it is the most simple and practical to implement of all the proposed design pattern into my code. Firstly, It would be a mess and hard to keep track of the classes that allow player to choose game mode and such in pre-game. Therefore, applying singleton through game manager just make a lot of sense.

The strategy pattern is typically implemented for a family of algorithms that can be interchanged at runtime. However, in the case of my chess game, the rules of the game are predetermined as fixed and cannot be interchanged at runtime. Hence, applying the strategy pattern may cause unnecessary complication without actually adding any benefits.

As for the command pattern, it is usually used to encapsulate requests or operations as objects, allowing them to be treated as variables and passed as arguments. However, in the case of the chess game, there are not really any explicit commands to encapsulate as objects. Instead, the game logic is determined by the sequence of user inputs and board state.

Q5: Updated additional sequence diagrams:


Sequence diagram- AI move piece
Le Minh Kha (Swinburne DN) | March 29, 2023
Actor
AI (PvA)
UserInputHandler
Board
Piece
Click (Select Piece)
Select_Piece(mouseX, mouseY)
GetMoves()
Legal moves
Is_Valid_Move()
Validation for move (legalmove List)
Click (Move Piece)
Select_Piece(newMouseX, newMouseY)
Move()
update position
AI's turn
Switch_Turn() update GameState
GetOpeningMove()
AI.Minimax (board, depth, true, Piece.Color.Black)
return best move in that position
AI_Mov (move the piece)
update position
Hooman's turn
Switch_Turn() update GameState

Q6:

It is quite evident that the HD program is much more complex than the D level program due to the implementation of the minimax algorithm and the singleton design pattern through Game Manager class. The ability to choose which game mode, time format you want to play make the game much more dynamic.

Moreover, the implementation of the minimax algorithm to adapt to the game of chess is a demanding process that requires a high level of expertise. However, the potential benefits of this implementation are noteworthy, as it enables players to engage in challenging gameplay against AI opponents, even in the absence of human counterparts.

Similarly, the singleton pattern, although relatively uncomplicated, provides significant advantages to the program by making the code more comprehensible and facilitating its extension in the future.