## Introduction

This assignment is worth **6 or 7%** of your grade. This is an **individual assignment** (so individual, I put *your* student ID on it) and you should not share your assignment with anyone else. The **assignment is due March 31st at 11:59PM, Montreal time**.

**All submissions must go through EAS: https://fis.encs.concordia.ca/eas/**

**After you submit, go back to check EAS to confirm that you just uploaded your assignment!**

## Q1) Implementing Othello

Your assignment is to implement a variant of the game Othello according to the UML Class diagrams provided below and the original rules specified in this link:
https://www.worldothello.org/about/about-othello/othello-rules/official-rules/english

You will be evaluated primarily on how playable your game is. You will also be evaluated on having followed the provided design. Lastly, and to a lesser extent, you will be evaluated on updates to the design diagrams that you make and the justification of any changes from the original design (preferably included as UML notes, but separate text may also be appropriate…keep it under two pages of text).

In this variant of Othello, some positions on the board will be unplayable. The starting position may vary, and the rules for flipping pieces may be adjusted. The class diagrams provided will show you the minimal object oriented approach that you should start with. You will likely add classes, and potentially even methods and fields. If you do so, these changes should be documented in updated UML diagrams.

The program should draw the board using ASCII characters on the screen. Positional labelling along the top and side will allow the input of movement coordinates via keyboard. Messages detailing the result of moves and instructions for how the current player may take their turn will be output by the program at the start of every turn. This assignment description shows traditional Othello disks, but the actual implementation would show ASCII characters.

When a game starts a menu will be shown, with numeric options given:

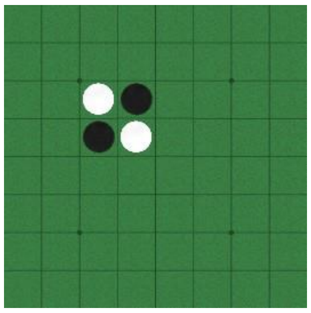
1. Quit
2. Start a New Game
3. Load a Game

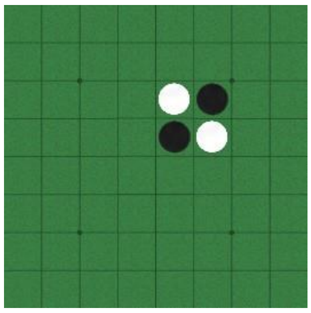
If a user chooses to quit, the program should close. When the user chooses to start a new game, two player names should be requested before starting a new game with those two players as first and second player, respectively. Should the user choose to load a game, the program should ask for a filename, load the game saved in that file, then continue from where that game left off.

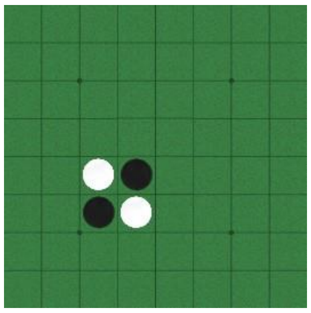
When laying out the board before starting, note that the two center squares along the top row will be unplayable. This rule does not come from standard Othello, but keep in mind that such unplayable squares will be the same for every game, irrespective of starting position.

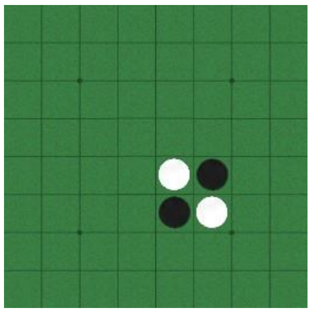
When starting a new game, the user will be offered different starting positions. They will enter a number corresponding to one of these choices:

1. An offset starting position, the user will be subsequently asked to choose a number between 1 and 4 to indicate one of the following options


1) A non-standard, offset starting position.


2) A non-standard, offset starting position.


3) A non-standard, offset starting position.


4) A non-standard, offset starting position.

2. Four-by-Four Starting Position

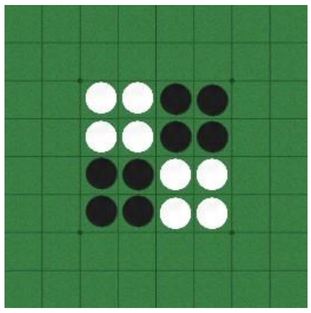


Figure 3: A non-standard, but still centered Othello starting position.

Whether through starting a new game or loading an existing game, the current player will be given options to play:

The options for the current player are as follows:

- If the current player cannot move, they may choose to save, concede the game or, if they cannot make a play to take an opponent's pieces they may forfeit their turn
- A player may choose to save, concede or make a move

Games should be saved as a text file, with first, second and current player names on the first three lines, then all the characters in the board state on the last line.

When a game is over, an appropriate message should be displayed indicating the current board state, who won.

As per the diagrams, implement the classes Game, Board, Position and Player. A Game object should have a Board object and the Board object should have Players. The Position class should be a parent class of PlayablePosition. The Position class should have a virtual method named ”canPlay()” that returns a boolean value indicating whether the position is playable or not. The initial implementation in the Position class should return ”false” since it represents an unplayable position. The return value should be ”true” for empty playable positions in the PlayablePosition subclass. The use of polymorphism is required and recommended in this design.

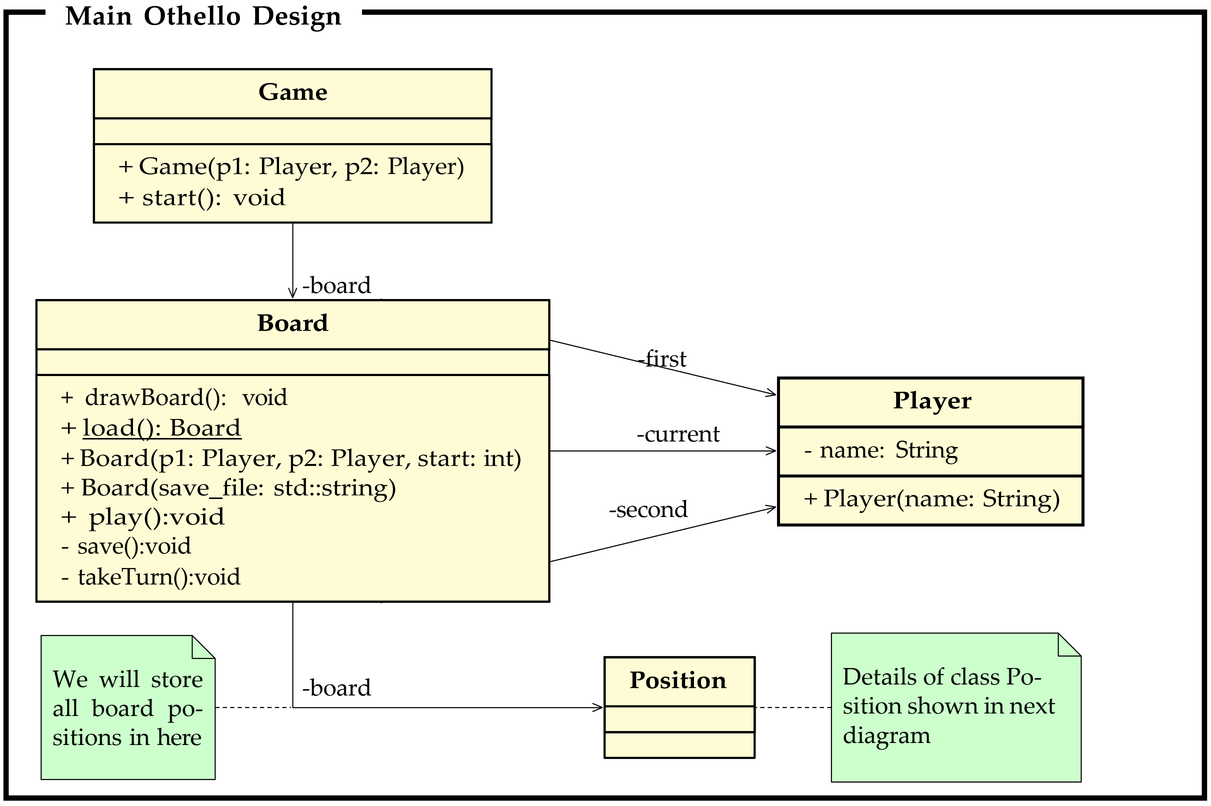
There is some latitude on how one could approach the implementation, but one would want a play() method that loops. A makeMove method and methods to determine whether a move converts any pieces will be needed. A method to check if there are any valid moves left will be needed. Think about the game problem and rules and include any additional methods, mindful of what should be public and private.

Classes should use Constructors that make clear how the life-cycle of the elements of the game (or their construction) will work.

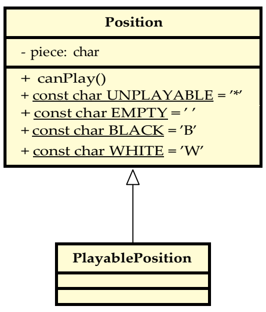
Consider the UML Notes when implenting, as there may be some guidance there. In particular, it has been noted that access/mutator methods are not shown/can be interpreted based on fields.

Start with these diagrams and base your design on the shown structure. If you add classes, fields or methods, include the updated diagram in your submission along with a brief (less than 500) word desription of that has changed.

While not shown in the diagrams, assume appropriate getters and setters, and interpret



## Othello Position Inheritance


Position
- piece: char
+ canPlay()
+ const char UNPLAYABLE = '*'
+ const char EMPTY = ' '
+ const char BLACK = 'B'
+ const char WHITE = 'W'
PlayablePosition