CS 2401 – Spring 2022

## Project 6 – A Game

# OTHELLO!


The final project for CS2401 will be a single large project which is to be submitted in three stages and therefore counts as three projects. For this project we will be implementing a game of OTHELLO, with the final product being a game that can play an intelligent game of OTHELLO against a human opponent.

This game **must** be derived from the author's game class. The files are *game.h* and *game.cc*. I have also included a file called *colors.h* which was created by a former student who has given us permission to use it. It allows you to adjust the colors of the screen during a text-based console or ssh session. I have altered the *play* function in the game class, by commenting out some of the code, so that it will work for the first phase of the project. By the end we will return to the original author's game class with only a couple of little alterations.

The game class creates a map for us in how the project is developed, and at the end we will find that most of the "AI" has already been written for us in this parent class. If you look at *game.h*, you will see that there are virtual functions that "must be overridden" and some that "may optionally be overridden." Eventually you will write a child version for all the mandatory overrides, but you probably do not need to override any of the optional ones. (Although the *winning* function is an exception – his version does not work.)

The rules of this game appear on a separate sheet. Basically, the game consists of two-sided playing pieces and the goal is to out-flank your opponent which allows you to "capture" his pieces by flipping them over to your color. Pieces are outflanked when you position a piece in such a way that one or more of your opponent's pieces in a row are between two of your pieces. Every move consists of putting a single piece onto the board. The pieces are never moved from their original location, and the game continues until there are no more moves available to either player. At that time, the player with the most pieces showing his or her color wins.

The first stage is the **design stage**. In this part you decide how you will represent the pieces and how you will display the board. Good grades are given for the quality of the design, the attractiveness of the board, and the ease of the user interface. This first stage should be derived from the game class. You know that you will be creating a child class for Othello, and that that class will have some way of storing the board. The board should be a two-dimensional array of spaces, pieces, or pointers[1] to spaces or pieces, where the spaces are another class which you have written. This board becomes the principle private member of the Othello class. The spaces class should be able to store all the attributes that a space (or you can call it a piece) might have – emptiness, black, white, as well as mutators and accessor functions to transform a piece/space from one state to another.

You should then implement your design to the stage where I can see the board displayed and be allowed to make one initial move. The first step in doing this is to write your space/piece class, which will **not** be derived from anything, but can change states. (A function called "flip" might be useful.) Then, when you write your Othello class, which will be derived from the author's game class, the best first step, after declaring your board, **is to create stubs for all the author's purely virtual functions.** (A stub is a function with an empty implementation, which exists merely to validate a call, or to allow the program to compile.) For this stage you should implement the *display_status*, *make_move*, and an *is_legal* which returns true if the move is any of the four allowable initial moves. Notice that the author expects the move to be entered and passed around as a string, and you need to stay with this as it is an essential part of the design. Think of the string as merely a container for characters. (Remember you will have inherited the *get_user_move* function – just go ahead and use it.)

Blackboard submission of this 50-point stage (6a) is due at 11:59 p.m. on Wednesday April 13th.

[1] Pointers make this a lot harder, so I'm not recommending this for any but the most adventuresome.

In the second stage you implement a two-player game which allows two humans to play the game against each other on the computer. (You will have one of the humans "be" the computer in the terms of the author's game class.) All rules should be enforced. This stage will involve a much more extensive *is_legal* function, since now that function must embody all the rules of the game. This is the hardest stage of this project so you will want to start on this as soon as possible.

Blackboard submission of this 50-point stage (6b) is due at 11:59 p.m. on Sunday April 24th.

In the final stage the computer will play an "intelligent" game against a human opponent. Again, all rules should be enforced, and the computer should not cheat. This is frequently called the "AI" stage of the game, and you will find that much of the AI work is done by the author's game class.

Electronic submission of this 50-point stage (6c) is due at 1:00 pm on Friday April 29th.