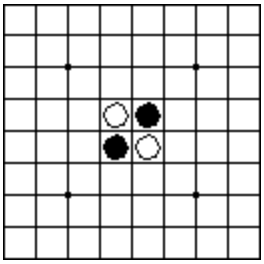
OTHELLO!

Our team members are Julie Barron, Ambika Goel, Hayley Hansson and Allison Patterson. We have created a computer version of the game Othello.

About the Game

The game always starts off with the setup below. As the player, the objective of the game is to have the majority of your color discs on the board by the end of the game. Each player takes 32 discs and chooses one colour to use throughout the game.



A move consists of "outflanking" your opponent's disc(s), then *flipping* the outflanked disc(s) to your color. Outflanking means to place a disc on the board so that your opponent's row(s) of disc(s) is/are blocked at each end by a disc of your color.

Rules:

1. Black always moves first.
2. You may forfeit your turn if there is no way to make a move.
3. A disc can outflank any number of discs in any number of directions at the same time - horizontally, vertically or diagonally.
4. You may not skip over your own colour disc to outflank an opposing disc.
5. Discs may only be outflanked as a direct result of a move and must fall in the direct line of the disc placed down.
6. All discs outflanked in any one move must be flipped, even if it is to the player's advantage not to flip them at all.
7. A player who flips a disc which should not have been turned may correct the mistake as long as the opponent has not made a subsequent move. If the opponent has already moved, it is too late to change and the disc(s) remain as is.
8. Once a disc is placed on a square, it can never be moved to another square later in the game.
9. If a player runs out of discs, but still has an opportunity to outflank an opposing disc on his or her turn, the opponent must give the player a disc to use. (This can happen as many times as

the player needs and can use a disc).
10. When it is no longer possible for either player to move, the game is over. Discs are counted and the player with the majority of his or her color discs on the board is the winner.
http://www.site-constructor.com/othello/othellorules.html

Goals

Our minimum deliverable was to create a game that works on a computer by simply running the program. Our maximum deliverable was to have a downloadable program that allows people to play Othello on their own personal computers.

Design Story / Chosen Structures

For this project, the information we needed to store is a location on the board and its subsequent status (empty, black, or white). We believed we could accomplish this with a dictionary of tuples mapping to values. We wanted to create a visual interface for our game's user. We had multiple options in order to achieve this goal, namely: pygame, vpython, and Tkinter. Pygame would be a fine choice, but it seems unnecessarily advanced as it fits better with games that follow a storyline or take place in a virtual world. Indeed, it fulfills the requirements for Othello but make use of a more complex program than necessary. Vpython is a rendering tool for 3D objects and graphs. It allows the user to create geometric objects and displays them after in a 3D space in a window. Vpython would also allow us to create an Othello game but the main purpose of vpython is to create 3D object which is unnecessary. Tkinter is a GUI, or graphical user interface, a module used in Python. There are elements called widgets that make up a GUI. Although there are basic widgets such as Button, Circle and Frame, there are also more complex widgets like Canvas, Text, and Scrollbar. Tkinter's flow of execution is determined by user actions rather than by the programmer, which is what we want. We expected Tkinter to be best-suited to our needs, which it was.

Plans / Division of Work

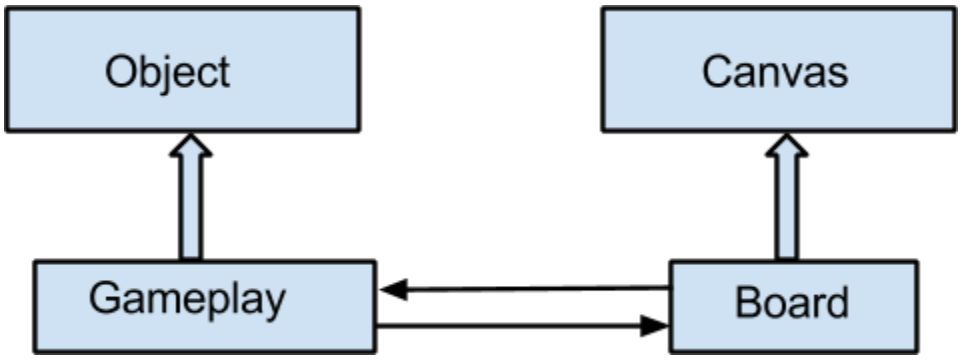
Model:	Detailing game rules (valid moves, etc.) Establishing protocol once a valid move has occurred (i.e. flipping proper tokens)
View:	Establishing the game atmosphere and frequency of updates (i.e. the board)
Control:	Establishing user's interface with the game (how to make a move, etc.)

Given the sequential nature of these sections, we decided divide the responsibility of leading the model and the view and control between members of our group. The model section was headed by Julie and Hayley. The view and control section was headed by Allison and Ambika. The plan was to create a 2D visual array of the board before the Model was written. The visual would be used to validate that the Model makes logical sense. The way we divided the workload allowed us to work independently or in our subgroups in order to move forward in the project. Github quickly became a valuable tool that helped us keep each other updated on the progress we made as we began to edit with more frequency.

Abstraction and Language

We have two classes. One is called Gameplay which handles the logic. The other is called Board which handles our visual interface. Board inherits Canvas from Tkinter. Within Gameplay there are several methods that check the 8 directions the user is clicking on and updates the dictionary. Within Board a method creates a grid for the user to visualize the coordinate system. Another method within Board uses the dictionary to update the appearance of the board. The Board has an “event” which is a click by the user. This event is an input into the Gameplay class. Gameplay receives the event which matches a key in its dictionary and performs methods on the associated keys. Board receives the updated dictionary. This tells which values have changed and what to update on the visual representation of the board.

UML Class Diagram



Final Design Refinement

We stayed with the design of a dictionary of tuples mapping to values which were strings.

We experimented with list of lists because it would make visualizing the board easier when testing the different methods in the model. This caused issues when we attempted to access and manipulate the x and y coordinates of the keys. Initially we had decided on using VPython as our visual interface but we decided a 3D rendering of the board was unnecessary, so we switched to using Tkinter. We considered using other GUIs but we needed something with the smallest learning curve for the given amount of time we had left. At some point we had two classes within our Model but we were unsure how to access methods from other classes at that point in the project, so we combined the two classes. (Eventually we did learn how to communicate between classes by having to integrate the class the GUI was storing information in and the class the information was being processed in.) The event supplies a tuple of integers which interfaces with the tuple in Gameplay's dictionary which are integers as well.

Bug Report

We had a few significant bugs towards the end of the project, but all except for one were extremely easy to find and fix. The one we had persisting problems with was when a single dictionary key, (5,1), having its value set to its own coordinates instead of "black", "white", or "E". We found this error by strategically placing print statements in all of the 8 direct methods and the two other methods used at the time. By moving the statements around we were able to narrow it down to a faulty for loop that was in place in each of the direct methods. The dictionary mapping coordinates to their colors was being run through in a random order, but the same order each time. The last coordinate key in the loop happened to be (5,1), and that value was being retained and replacing the value in the last key value pair that was run through the loop. In order to make the code more efficient, simple, and interpretable, we decided to change the way it ran completely by adding more methods to be called from inside the directs. This alleviated the need for the problematic for loops, and solved our problem.

Conclusion

In retrospect, we should have accomplished the model and view simultaneously, but also finished the view after a week or so to allow the model to be tested with the GUI. That said, we still managed to achieve our minimum deliverable of creating a playable virtual version of Othello plus a simple website so people can download our code and play our game. None of us understood classes very well causing us to struggle a lot with this project. We developed great techniques for and enjoy debugging large chunks of code. In the end we learned a lot despite being very simple. We now feel comfortable trying larger more complicated projects.