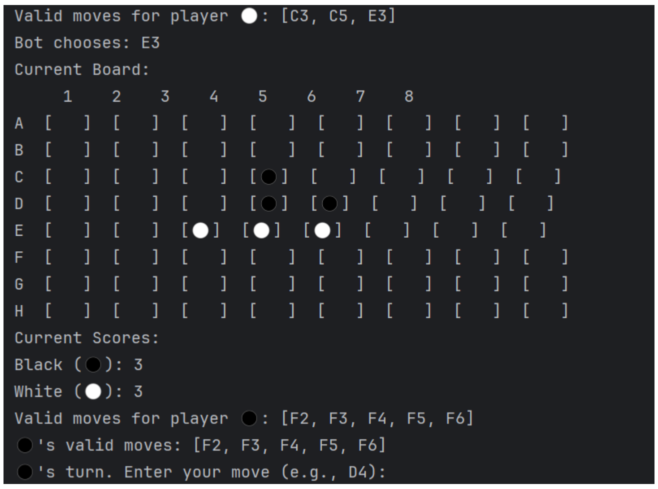
2024/1/19

PLEASE RUN THE REVERSIGAMEFX FILE IN ORDER TO RUN

Project Purpose:

The purpose of this project is to create a simulator of one of my favorite past-times, Reversi. The objective of Reversi is to have as many pieces of your color as possible on the board. The way to take pieces is by putting your piece in front of an enemy piece that forms a line with an already existing piece of your color. This creates a sort of sandwich: the filling being the enemy piece and the bread being your own pieces. You then flip all the enemy pieces in the filling to your own color. An annoying thing, however, is that if you have a very large filling, it can take a long time to flip all the enemy colors. Therefore, I have created a computer version of this game that automatically flips the colors.

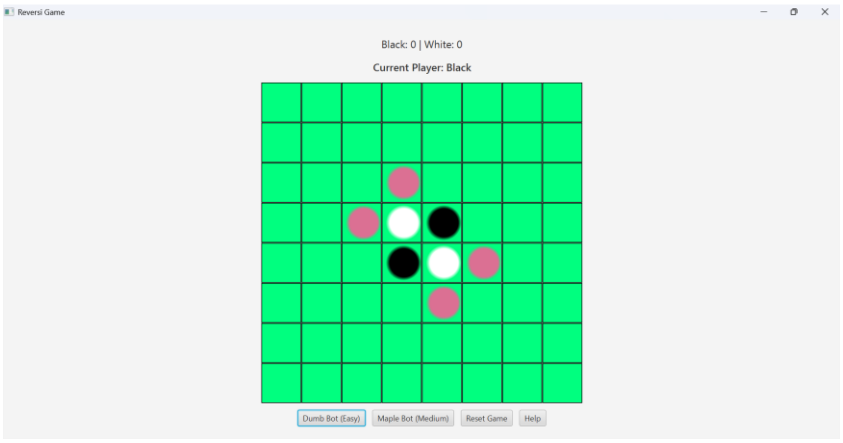
In my draft, I have previously printed out the boards directly onto the console:



```
Valid moves for player ⚪: [C3, C5, E3]
Bot chooses: E3
Current Board:
    1    2    3    4    5    6    7    8
A  [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
B  [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
C  [   ] [   ] [   ] [⚫] [   ] [   ] [   ] [   ]
D  [   ] [   ] [   ] [⚫] [⚫] [   ] [   ] [   ]
E  [   ] [   ] [⚪] [⚪] [⚪] [   ] [   ] [   ]
F  [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
G  [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
H  [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
Current Scores:
Black (⚫): 3
White (⚪): 3
Valid moves for player ⚫: [F2, F3, F4, F5, F6]
⚫'s valid moves: [F2, F3, F4, F5, F6]
⚫'s turn. Enter your move (e.g., D4):
```

Screenshot of the Draft Output

However, it is obvious to see that this looks a bit awkward, especially with how the board is misaligned. Therefore, with my final submission of this project, with the power of JavaFX, I have created an aesthetically pleasing application for my Reversi game.



Screenshot of the Final Output

CLASSES | FUNCTIONS | EXPLANATIONS

Please note that what I am trying to do here is just implement the logic I created in my Draft onto a JavaFX application. However, please note that there was some logic that I had to implement in ReversiGameFX given that my code faced difficulties with the skip option.

Class/Interface	**Function**	**Explanation**
ReversiGameFX	**The ReversiGameFX is responsible for the game mechanics, concerning JavaFX**	
	start	Initializes the main menu with the game mode selection and help option.
	startGame	Setup the game board and visualization for the selected game mode. If a bot mode was chosen, then this method also configures the respective bot behavior.
	showHelp	Displays a help dialog explaining the game modes and rules. Also provides a link to a YouTube tutorial on how to play Reversi.
	updateBoard	Updates the board visualization: show pieces, valid moves, and click event handlers.
	createPieceFill	Creates a gradient effect for board pieces or highlighted cells.

	handleCellClick	Processes cell clicks, validating move, updating the board, and handling game turns.
	checkForSkipTurn	Skips a player’s turn if no valid moves are available, switching to the other player
	isGameOver	Determines if the game has ended by checking valid moves and board state.
	switchPlayer	Switches to the other player and updates the board visualization.
	updateScores	Calculates and updates the current scores for both players
	resetGame	Resets the game state, including the board, scores, and player turn.
	handleBotMove	Execute the bot’s move with a few second delay, updating the board and switching back to the player. *Only if a Bot mode is selected.
	toggleDumbBotMode	Toggles DumbBot mode and resets the game.
	toggleMapleBotMode	Toggles MapleBot mode and resets the game
	showAlert	Displays an informational alert with a specified message.
Reversi	**The Reversi function is responsible for the game mechanics**	
	initializeBoard	Set up the board with four starting pieces in the center: two for white and two for black.
	printBoard	Displays the board in an easy-to-read format for the player, the letter-number grid. The letter represents the rows, and the number represents the columns.
	isValidMove	Checks if a given move is valid by checking if it takes an opponent’s piece.
	getValidMoves	Gets a list of the valid moves for a current player
	makeMove	Executes a given move and flips all the opponent pieces found in the sandwich

	savesMovesToFile	Saves the game history to a .txt file.
	determineWinner	Counts the number of pieces for each player and determines the winner by whoever has the greatest number of pieces.
SimpleBot	**SimpleBot is an abstract class that serves as a blueprint for the bot classes: DumbBot and MapleBot.**	
	getBotMove	Abstract method that each bot must implement according to each bot's strategy.
	convertMove	Converts a move from the letter-number format to a number.
DumbBot	**DumbBot is an easy difficulty bot that just iterates randomly through its valid moves.**	
	getBotMove	Overrided method inherited from SimpleBot. DumbBot's strategy randomly chooses a valid move.
MapleBot	**MapleBot is the medium difficulty bot that goes for corners.**	
	getBotMove	Overrided method inherited from SimpleBot. MapleBot's strategy is to go for corners if available, however, if not, then it just randomly chooses a valid move.
Scorable	**Scorable is an interface that has two functions relevant to scoring that must be implemented with whatever class signs the contract.**	
	calculateScore	Should calculate the current score for a player
	displayScore	Should display the current score for both players
BasicScoring	**BasicScoring is an implementation of the Scorable interface and implements the function in Scorable**	
	calculateScore	Overrided method from the Scorable interface.

		Checks for every instance of a player’s piece and stores the sum in the score.
	displayScore	Overided method from the Scorable interface.

Plans for Improvement:

1. Smarter Bots.

   I could create bots that are smarter, posing more of a challenge for human players to play against. Right now, most of the Bots’ algorithm is random, except in the case of MapleBot going for corners if available. New smarter bots could contain algorithms that are harder to play against.

   Specifically, I was thinking of creating a bot that takes my most used strategy:

   - In the first half of the game, try to hold the center four most pieces as possible.
   - In the second half of the game, go for corners and then wall cells.

   To implement this algorithm, I would need to identify what “first half” and “second half” mean. I am thinking that this could just be measured by how many pieces are already placed on the board. This should be a bit easy to implement given that methods relevant to scoring already count the number of pieces on the board.

2. Updating savesMovesToFile function

   I can update this function to also include perhaps what game mode is chosen giving more context to the game being played.

3. Better aesthetics and language support

   The current aesthetics already look much better than the draft version, however, the game does look a bit like an old Flash player game (RIP Flash). I can work on providing a cleaner aesthetic to the game. Also, since Reversi is very popular in Japan, perhaps implementing a feature that allowed text to be displayed in Japanese would be nice.