Othello Project Instructions

1. Project Introduction

Project Title: Othello

Main Objective: Build the main functionality of the board game Othello, play the game using a pre-written GUI, and implement various famous game theory algorithms for computer AI decision making.

General Skills:

- Practice with version control systems
- Group work collaboration
- Implementation of software design patterns
- Open-ended project planning

Specific Skills:

- Building complex trees that simulate future states
- Working with the Strategy design pattern
- Integrating your code with a graphical user interface (JavaFX)
- Using package-based organization and dependancy management with Maven

Optional "Stretch" Skills:

- Exploring complex game theory and tree pruning algorithms
- Writing JavaFX GUI code

2. General Requirements & Restrictions

Team Structure

- Teams will have 3 members
- All members must contribute, verified by peer evaluations and GitHub history

Version Control

- Teams must use GitHub project management tools
- Projects should be broken down into discrete tasks. These tasks will be managed & assigned using GitHub and tracked from planning through completion
- Regular commits/pull requests from feature branches expected

Technical Requirements

- All code must be written in Java (firm requirement)

Timeline

- First proposal due: March 7
- Feedback provided by: March 19

- Required check-in meeting by: April 11 (to review planning & design choices)
- Project code due: May 12
- Project presentations to TAs: Until May 13 (can be scheduled after May 1 with group & TA discretion)

Documentation

- All code must be documented with comments and tests
- Fully featured JavaDocs included as optional "completeness" feature
- Work with outside resources (books, websites, LLMs/Copilot) is permitted but must be cited

3. Specific Requirements for this Project

Technical Components

Students will be responsible for implementing the logic for the game Othello. This will involve designing code that can interface with a **pre-written graphical user interface (GUI)**, and designing various famous game theory algorithms to allow a computer AI to make game decisions.

User Interface

Provided with the starter code for this project will be an extensive GUI that displays the state of the Othello board, and allows users to play the game through mouse input. Running `App.java` starts the GUI. Students will **not** need to write any GUI code or learn JavaFX; they will only need to make use of the GUI to help test the complex states of their code.

File Organization

Due to the use of the GUI, the starter code will be organized into packages such that logical and GUI code will be isolated. Students will need to understand how to use **packages** and keep their code well-organized during this project.

Game Mechanics

Given a starting board, students will implement the logic for tracking players, board spaces, and possible next moves.

- The board will be given as a 2D array of `BoardSpace`s (`BoardSpace[][]`)
- It will initially be populated with the starting state of an Othello board (2 black and 2 white discs in the center)
- Black will start, finding the next available spaces that "outflanks" their opponents pieces, meaning spaces that will cause their opponent's pieces to be sandwiched between theirs.
- The turn will switch, and the game proceeds until either player cannot make a move or the board is full.

Game Modes

Through program arguments passed as a run configuration or arguments passed into the run command, the game can be played with human vs. human, human vs. computer, and computer vs computer. Additionally, the computer will be provided a decision making strategy as a program argument that will be detailed below.

Computer AI

Students will be designing various decision making algorithms that will allow a computer to make smart decisions on how to take their turns given the state of the Othello board. Students will implement 2 famous algorithms, either Minimax or Expectimax, and Monte Carlo Tree Search. Then, students will either design or research and implement another computer AI algorithm for the game.

Strategy Design Pattern

Using the Strategy Design Pattern, students can organize the various computer AI strategies within their project structure and make use of polymorphism to allow varied use of the strategies.

Decision Making Algorithms

Students will be implementing a backtracking algorithm (Minimax or Expectimax) and a heuristic search algorithm (Monte Carlo Tree Search). These algorithms are famous game theory algorithms that can be used for various other board games.

- **Minimax with Alpha-Beta Pruning**: A backtracking algorithm that involves finding optimal moves by casting one player as a *maximizer* and their opponent as a *minimizer*. This process is performed by forming a tree of possible future states. Alpha-Beta pruning refines this process by removing branches of this tree that are not within a certain threshold.
- **Expectimax**: A variation of the Minimax algorithm that instead simulates more realistic decisions based on non-optimal, probability-based decisions. Alpha-beta pruning is **optional** to implement here.
- **Monte Carlo Tree Search(MCTS)**: A heuristic search algorithm used to make optimal decisions by simulating future state outcomes. It incrementally builds a search tree by conducting random playouts and expanding nodes of interest. The process involves four steps: selection, expansion, simulation, and backpropagation, each of which builds upon a tree.

Custom Algorithm:

Students will define their **own algorithm** or research an **existing decision-making algorithm** with the same level of complexity as the previous 3 algorithms above. Some examples may include some graph-based approaches, using heuristics based on datasets of historically good moves, or probabilistic exploration of tree nodes. Students should avoid picking algorithms that require significant overhead or adjustments to the provided starter code.

Required Project Features

- Functional game of Othello with human vs. human, human vs. computer, and computer vs computer modes
- 2 decision making algorithms (1 backtracking, 1 heuristic search) implemented using the Strategy design pattern
- 1 custom/alternative algorithm (team-proposed)
- Proper use of the provided packages and file organization
- Able to run your code through the GUI

Optional Features

- Extra decision making algorithms besides the required 3
- Adjusting the GUI interface, adding animations
- Decision making algorithms that require heavy extra overhead, such as tracking past board states across turns

4. Evaluation Scheme

Code Quality

- 80% testing coverage required
- Commented code required

Documentation

- Design document with class diagrams for all included classes

Presentation

- Required sections:
  - Project design
  - Initial project plans
  - Choices made for both the custom algorithm and optional requirements
  - Project demo
  - Challenges faced/project retrospective
  - TA Q&A

Team Assessment

- Peer evaluation via Google form

5. Other Resources

- Othello Rules
- Play Othello Online
- Strategy Design Pattern
- Textbook Chapter about this assignments algorithms