Team Member Names:
Abhinav Gupta
Alexa Rinard
Varun Manjunath

Title: Mancala Game

Work Done:

The main function of the game is yet to be implemented. There are two types of players HumanPlayer and ComputerPlayer.Each of these players are classes. Each of the players is assigned a player number. The HumanPlayer is assigned a player number of 1 and the ComputerPlayer is assigned a player number of 2. The HumanPlayer class and the computerPlayer class both have methods makeMove.The HumanPlayer has to enter a number between 0 and 5 both inclusive. When the human player enters a number the Board class playCup method is called with the parameters chosenNum(The number chosen by the player) and player number(For HumanPlayer it is 1 and for ComputerPlayer it is 2).The computer player chooses a random number between 6 and 13(both inclusive).Once the computerPlayer chooses a random number then the Board class playCup is called in a similar fashion to how the HumanPlayer calls the playCup method. The Board class playCup method chooses a cup from a list of cups obtained from the Cup class. There are a total of 14 cups, where cups numbered 0 to 6 are assigned to the HumanPlayer and cups numbered 7-13 are assigned to the computerPlayer.This function playCup chooses the appropriate cup based on the chosenNum .Initially, all the cups have a bead count of 4. These chosen cup stones are emptied and then each of the beads from this cup is dropped in subsequent cups. There is a game class that is the main point of execution for the program.The main class comprises of the game constructor that calls the factory class PlayerFactory which creates an instance of HumanPlayer and computerPlayer.The playGame method in game class contains the logic to run the game.

Patterns Use:
The playGame method of the game class calls the makeMove method of the players class(HumanPlayer and computerPlayer) and this demonstrates the command design pattern. The getinstance method in Board class is called by the Game constructor of Game class and this demonstrates singleton design pattern. A simple factory pattern is demonstrated by the PlayerFactory class that returns an instance of either the HumanPlayer or computerPlayer.

Work to be done:
We are yet to implement the logic of when a stone lands in a Mancala and the player gets another turn. Also, the stealing logic of when the last stone ends up in an empty player's cup is yet to be implemented.

Changes or issues:
We will be moving out from an MVC design pattern of using Java Swing as the frontend of the application to a client-server type of architecture implementation. For the frontend we are using Spring framework to display the board for the player to play and for the backend, we are using Java.

Plan for next iteration

Question: Provide an estimate of how much more work needs to be done for your team to have implemented the design that you presented in Project 5 (with any design changes that may have occurred).
Answer: Apart from the minor tweaks, we are majorly left with the backend and frontend integration of our program. The integration is expected to result in a slight change in our API call parameters and data binding between the model and the view.

Question: What are your plans for the final iteration to get to the Project 7 delivery?
Answer: Integration effort is estimated to be a few hours and after that we plan to work on enhancing the user experience with a better UI design and a few minor features logically written in the backend.

Question: What do you plan to have done by 4/27 when the project is due?
Answer: Our plan is to deliver a working web-based Mancala board game that 2 users (or a user and a computer player) can play. We plan to include all the core functionalities/rules that are followed in a typical Mancala game.

Class Diagram:
https://lucid.app/lucidchart/ea88e036-8cc9-4052-bc83-e9e68d3040d1/edit?invitationId=inv_4e2bd4c5-7887-47da-ad98-c17160c19356