Project 7 Final

Information for Graders Summary

Project Title:
MancalaAI

Team Members:
John Danekind

Patterns:

Observer
In the code, I have a Board class and a BoardView class. There are also two interfaces, Viewable and Viewer. The board class represents the actual data structure used for the board while BoardView is responsible for creating and displaying the GUI of the board. Viewable is an interface with methods that are responsible for updating the board and the state of the game. Viewer is simply an interface with an update method. The board class implements viewable and has an array list of viewers and each time an event happens, we update the viewers. BoardView implements viewer and within the class, we have the update method taking the event as an argument. So, through the use of Board, BoardView, Viewer, and Viewable, all information regarding the board and the state of the game is passed around via an observer pattern.

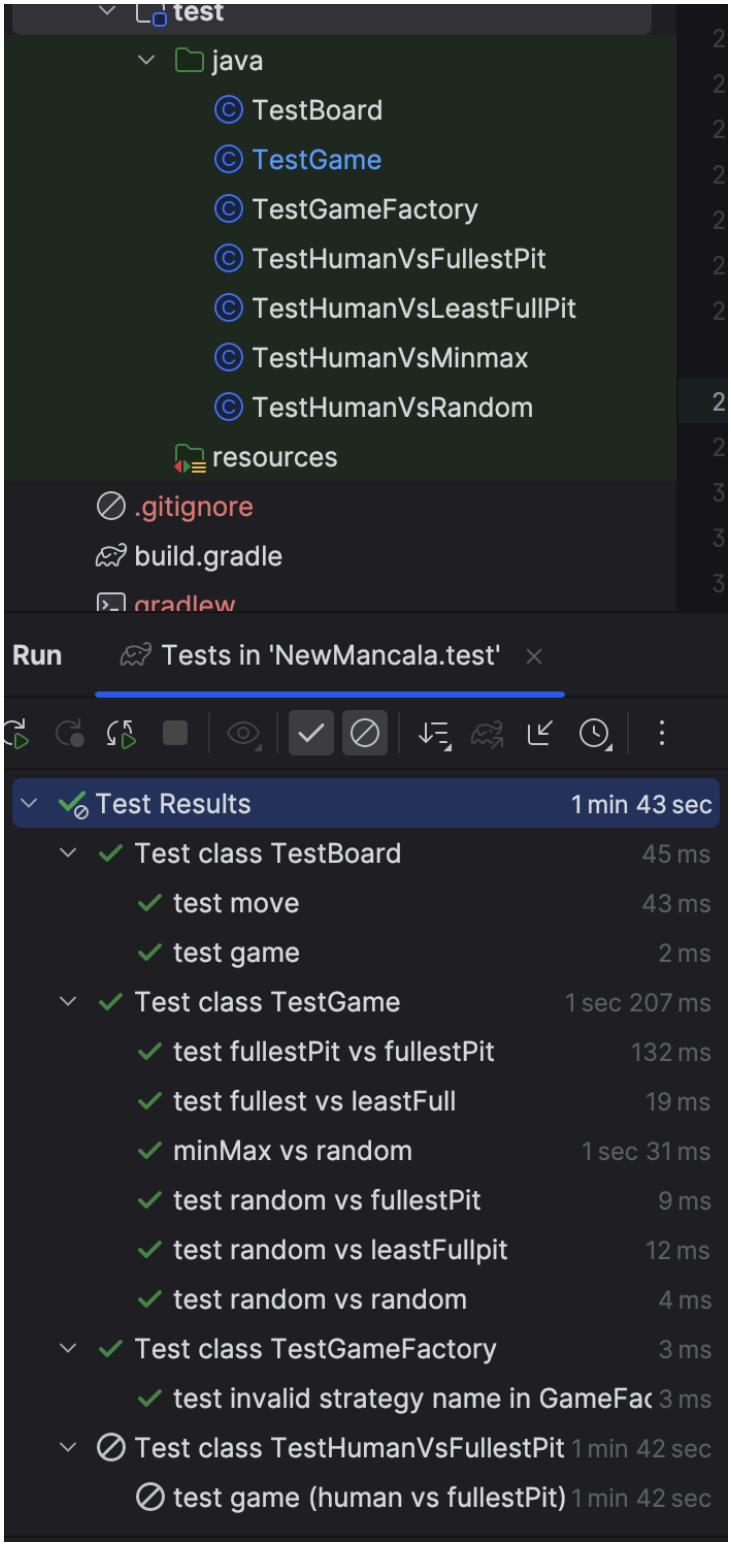
Strategy:
I wanted to test different strategies a player could use to find their next move. So, I felt that a strategy pattern would be perfect for this. I have a Mancala strategy interface with a method called getNextMove(Board board, Player player). This method is used to implement four different strategies, fullestPit, leastFullPit, minMix, and random. Fullest pit simply selects the pit with the highest number of stones in it, least full selects the pit with the least number of stones in it, random will randomly select a pit, and minimax performs the minimax algorithm.

Factory:
Since I wanted to test the different strategies and how they performed in Mancala, I knew that the game would have to be configured in different ways. Because of this, I created a gameFactory class. This game factory has a method called createGame that takes in player one's name, player two's name, the strategy for each player, and a boolean called hasTextBoardView, and another boolean hasBoardView. With this, when I create a game, I can use the factory and input the player names I want, the strategies I want, and if I want the GUI and or text version of the game.

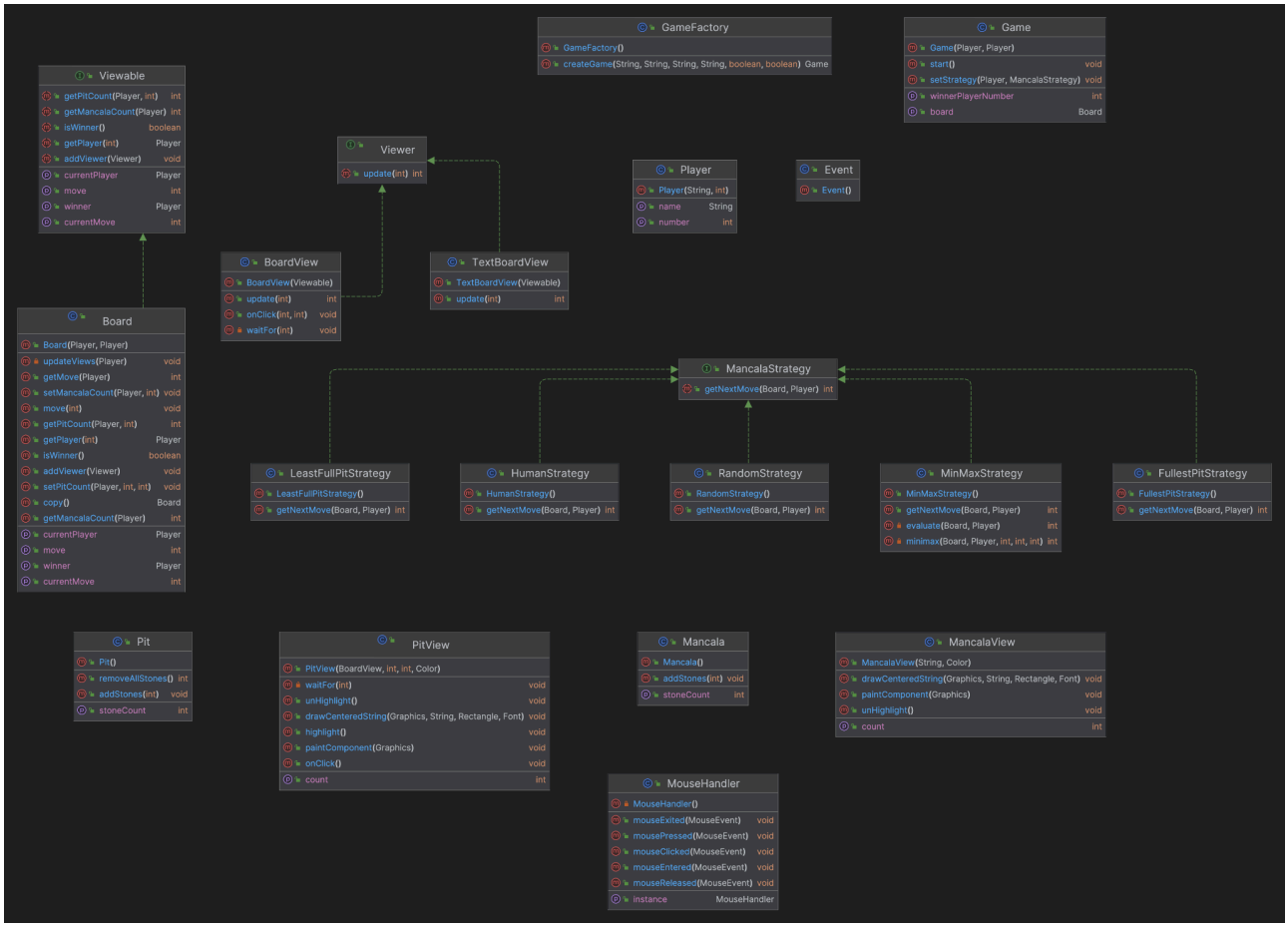
Singleton:

Since there is a GUI environment for the player, and moves happen when the human player clicks somewhere on the board, I needed a class to handle everything with the mouse. I called this class mouse handler and it implements the MouseListener interface, built into Java. However, we only want one instance of the mouse handler because if multiple are created, events will happen more than once in the GUI environment. So, I made it a singleton with: `private static final MouseHandler mouseHandlerInstance = new MouseHandler();` this ensures that an event caused by a mouse click will only happen once.

Screenshot of Test Coverage


test
java
TestBoard
TestGame
TestGameFactory
TestHumanVsFullestPit
TestHumanVsLeastFullPit
TestHumanVsMinmax
TestHumanVsRandom
resources
.gitignore
build.gradle
gradlew
Run
Tests in 'NewMancala.test'
Test Results 1 min 43 sec
Test class TestBoard 45 ms
test move 43 ms
test game 2 ms
Test class TestGame 1 sec 207 ms
test fullestPit vs fullestPit 132 ms
test fullest vs leastFull 19 ms
minMax vs random 1 sec 31 ms
test random vs fullestPit 9 ms
test random vs leastFullpit 12 ms
test random vs random 4 ms
Test class TestGameFactory 3 ms
test invalid strategy name in GameFac 3 ms
Test class TestHumanVsFullestPit 1 min 42 sec
test game (human vs fullestPit) 1 min 42 sec

UML Diagram



BDD

Board Class

move Function

Behavior: When a player makes a move by selecting a pit, the stones from that pit should be distributed counterclockwise to the subsequent pits, skipping the opponent's Mancala, and landing one stone in each pit until no stones remain. The selected pit should become empty after the move.
Scenario: Given a board with an initial pit configuration, when a player selects a pit and makes a move, the stones should be distributed correctly, and the selected pit should become empty.
NOTE: This is tested in the TestBoard class in the testMove() method.

isWinner Function

Behavior: Determines the winner based on the stone count in each player's Mancala. If all pits of one player are empty, the player with more stones in their Mancala wins.
Scenario: Given a board state where one player has more stones in their Mancala, the function should correctly identify that player as the winner.

BoardView Class

update Function

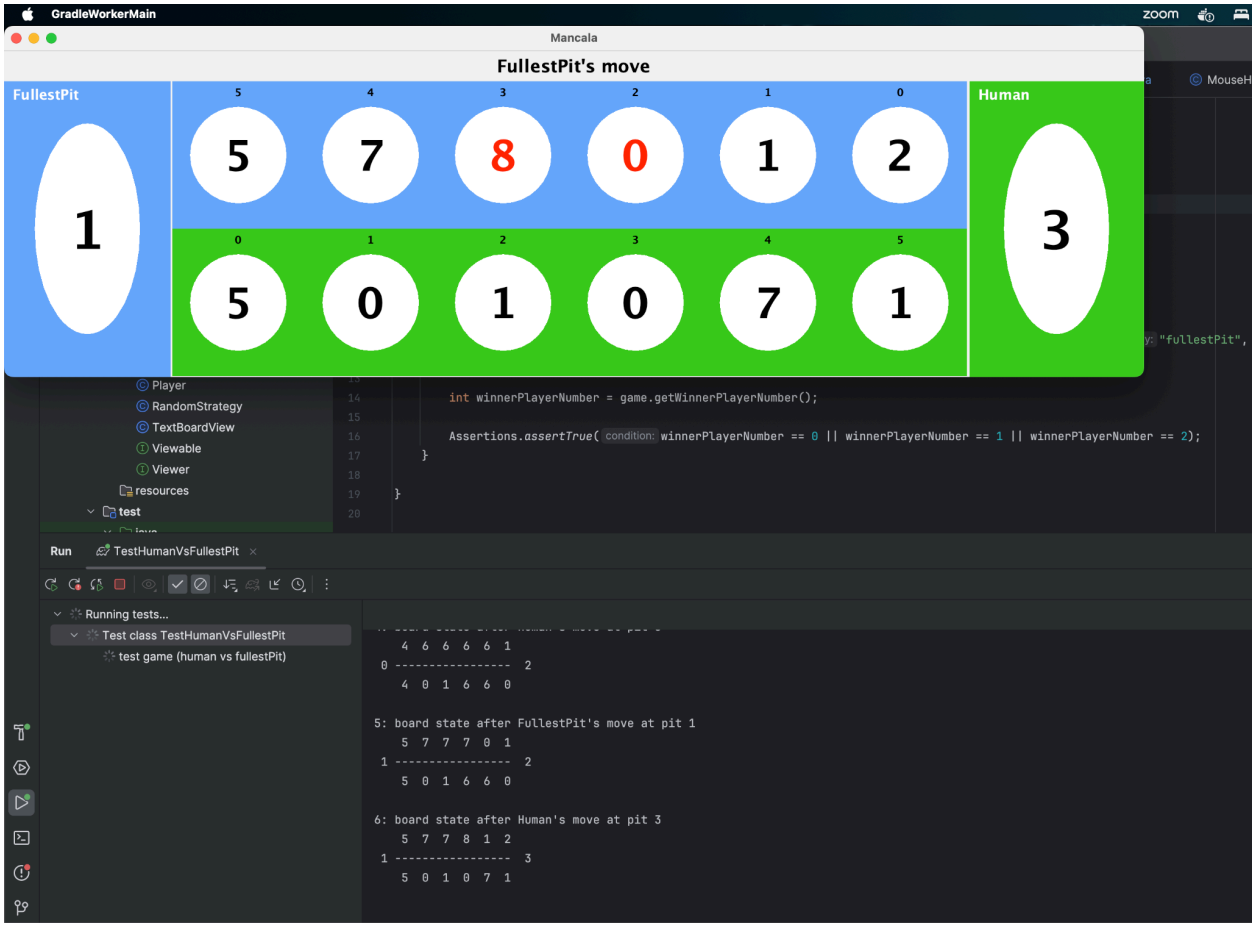
Behavior: Updates the graphical view of the board when the board state changes. It should reflect the stone counts in each pit and Mancala accurately.
Scenario: When the stone counts in the board's pits and Mancalas change, the graphical view should update accordingly.

Game Class

start Function

Behavior: Starts the game and continues until there is a winner. Players take turns making moves according to the game rules.
Scenario: When the game starts, players alternate making moves until one player wins. The function should end the game and declare the winner correctly.

Observers


GradleWorkerMain
Mancala
FullestPit's move
FullestPit
1
5 4 3 2 1 0
5 7 8 0 1 2
0 1 2 3 4 5
5 0 1 0 7 1
Human
3
Player
RandomStrategy
TextBoardView
Viewable
Viewer
resources
test
int winnerPlayerNumber = game.getWinnerPlayerNumber();
Assertions.assertTrue(condition: winnerPlayerNumber == 0 || winnerPlayerNumber == 1 || winnerPlayerNumber == 2);
Run
TestHumanVsFullestPit
Running tests...
Test class TestHumanVsFullestPit
test game (human vs fullestPit)
4 6 6 6 6 1
0 ---------- 2
4 0 1 6 6 0
5: board state after FullestPit's move at pit 1
5 7 7 7 0 1
1 ---------- 2
5 0 1 6 6 0
6: board state after Human's move at pit 3
5 7 7 8 1 2
1 ---------- 3
5 0 1 0 7 1