Project 7 Update

Status Summary

Project Title:
MancalaAI

Team Members:
John Danekind

Work Done:
As of Friday, April 12th, I have completed 11 classes and 3 interfaces. The code written so far creates a GUI for the board, updates the board via an observer, and distributes stones on the board according to some of the main rules of Mancala.

Changes or Issues Encountered
The main challenge that I am running into now is having the user click on the board and having this update the board. I am having a hard time getting this functionality integrated with the rest of the code and the observer. So right now, the gameplay you see in the example video is just two players randomly selecting pits and playing until the winner is declared.

Patterns:

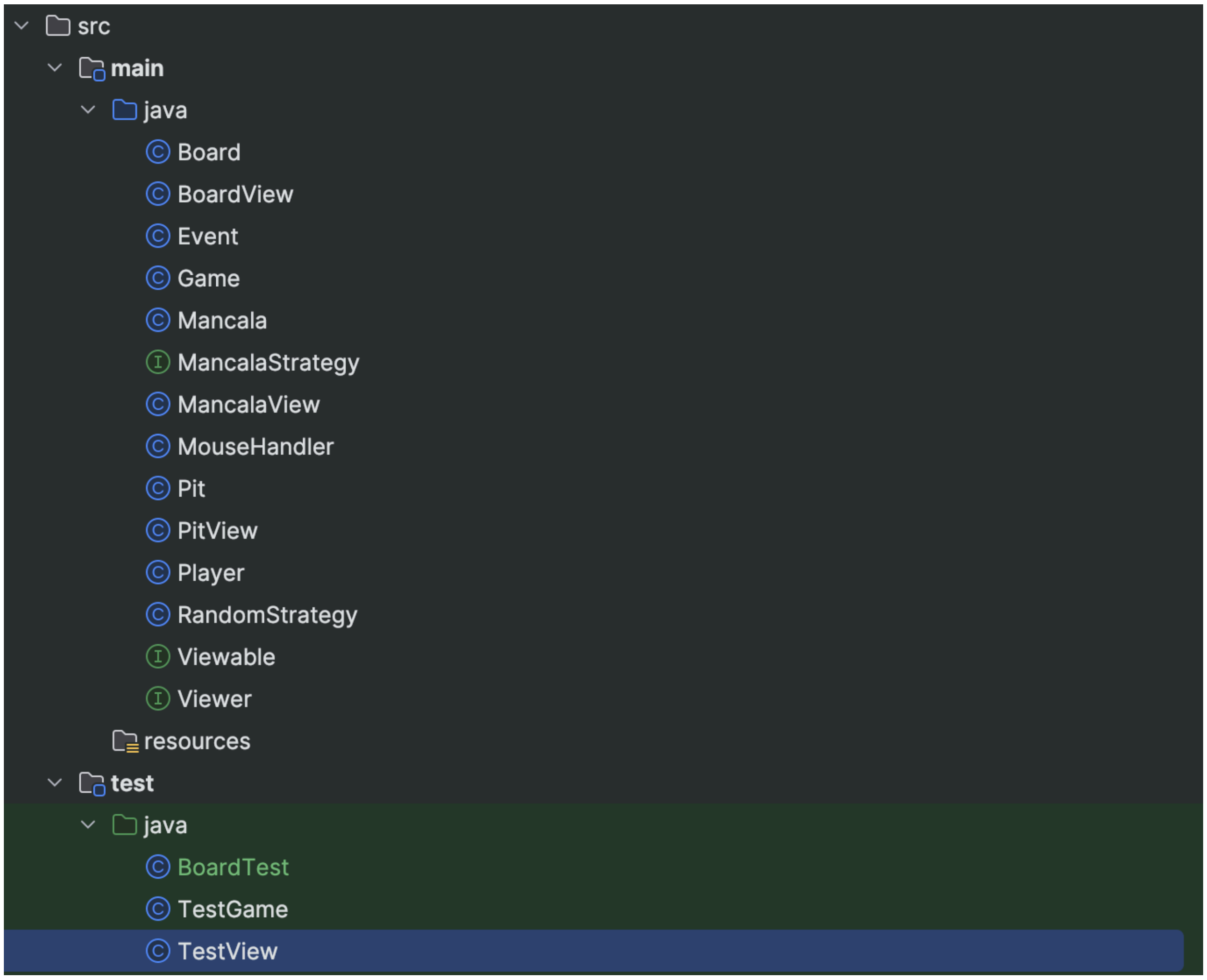
Observer
In the code, we have a Board class and a BoardView class. There are also two interfaces, Viewable and Viewer. The board class represents the actual data structure used for the board while BoardView is responsible for creating and displaying the GUI of the board. Viewable is an interface with 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:
The strategy part of my code is not fully implemented yet. However, as of now, I have a MancalaStrategy interface with a method called getNextMove that takes in the board and the player as well as a RandomStrategy class. Later, I plan on adding more strategy classes that will implement minimax with alpha-beta-pruning at various depth levels.

Patterns I will implement later:

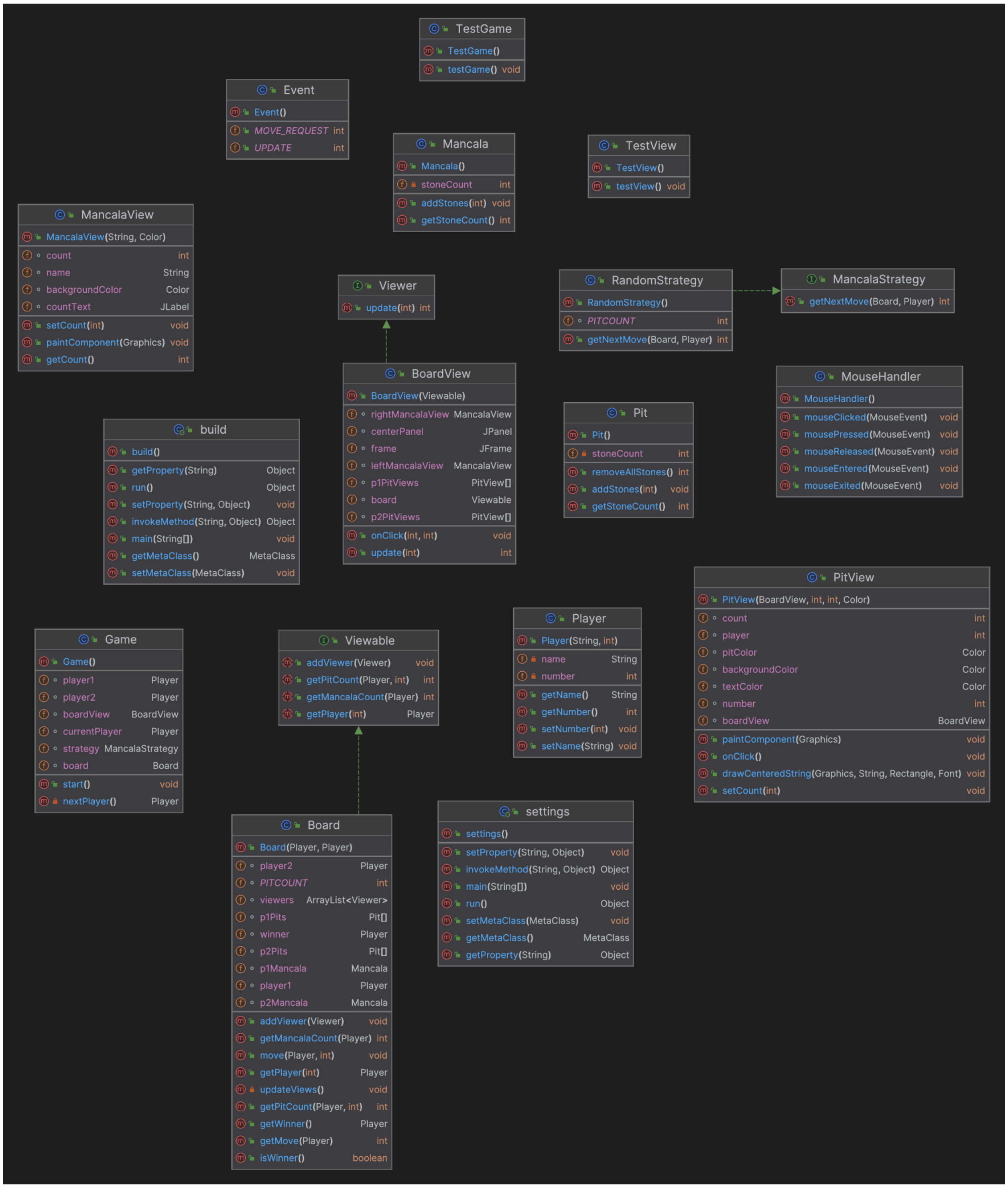
I will use the factory pattern to create a game with a given strategy. I will use the command pattern as a way to encapsulate GUI actions such as clicking on a pit or selecting a difficulty level.

Screenshot of Test Coverage



```
src
  main
    java
      Board
      BoardView
      Event
      Game
      Mancala
      MancalaStrategy
      MancalaView
      MouseHandler
      Pit
      PitView
      Player
      RandomStrategy
      Viewable
      Viewer
    resources
  test
    java
      BoardTest
      TestGame
      TestView
```

UML Diagram


TestGame
TestGame()
testGame() void
Event
Event()
MOVE_REQUEST int
UPDATE int
Mancala
Mancala()
stoneCount int
addStones(int) void
getStoneCount() int
TestView
TestView()
testView() void
MancalaView
MancalaView(String, Color)
count int
name String
backgroundColor Color
countText JLabel
setCount(int) void
paintComponent(Graphics) void
getCount() int
Viewer
update(int) int
RandomStrategy
RandomStrategy()
PITCOUNT int
getNextMove(Board, Player) int
MancalaStrategy
getNextMove(Board, Player) int
BoardView
BoardView(Viewable)
rightMancalaView MancalaView
centerPanel JPanel
frame JFrame
leftMancalaView MancalaView
p1PitViews PitView[]
board Viewable
p2PitViews PitView[]
onClick(int, int) void
update(int) int
MouseHandler
MouseHandler()
mouseClicked(MouseEvent) void
mousePressed(MouseEvent) void
mouseReleased(MouseEvent) void
mouseEntered(MouseEvent) void
mouseExited(MouseEvent) void
build
build()
getProperty(String) Object
run() Object
setProperty(String, Object) void
invokeMethod(String, Object) Object
main(String[]) void
getMetaClass() MetaClass
setMetaClass(MetaClass) void
Pit
Pit()
stoneCount int
removeAllStones() int
addStones(int) void
getStoneCount() int
PitView
PitView(BoardView, int, int, Color)
count int
player int
pitColor Color
backgroundColor Color
textColor Color
number int
boardView BoardView
paintComponent(Graphics) void
onClick() void
drawCenteredString(Graphics, String, Rectangle, Font) void
setCount(int) void
Game
Game()
player1 Player
player2 Player
boardView BoardView
currentPlayer Player
strategy MancalaStrategy
board Board
start() void
nextPlayer() Player
Viewable
addViewer(Viewer) void
getPitCount(Player, int) int
getMancalaCount(Player) int
getPlayer(int) Player
Player
Player(String, int)
name String
number int
getName() String
getNumber() int
setNumber(int) void
setName(String) void
Board
Board(Player, Player)
player2 Player
PITCOUNT int
viewers ArrayList<Viewer>
p1Pits Pit[]
winner Player
p2Pits Pit[]
p1Mancala Mancala
player1 Player
p2Mancala Mancala
addViewer(Viewer) void
getMancalaCount(Player) int
move(Player, int) void
getPlayer(int) Player
updateViews() void
getPitCount(Player, int) int
getWinner() Player
getMove(Player) int
isWinner() boolean
settings
settings()
setProperty(String, Object) void
invokeMethod(String, Object) Object
main(String[]) void
run() Object
setMetaClass(MetaClass) void
getMetaClass() MetaClass
getProperty(String) Object

Plan for the Next Iteration

What needs to be done?

- Finish implementing Mancala rules for stone distribution. Specifically, add the stealing rule and the turn repeat rule.
- Use a factory pattern to create a game with a particular strategy.
- Use the command pattern as a way to encapsulate GUI actions such as clicking on a pit or selecting a difficulty level.
- Get the user to be able to play a game by clicking on the GUI
- Finish the strategy pattern by implementing the Minimax strategy.
- If time permits, change the numbers on the screens to actual stones used in a real Mancala game.

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.

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.

Recorded Demonstration

Video