Final Project Report

Multiplayer Scrabble Game
Alex Walker & Jack McKinstry
Repository: https://github.com/alwa2786/OOAD_Project_5

State of System Statement
All of our originally planned features were implemented, all use cases function as intended. In functionality and appearance, the state of the system is essentially as was planned during project 5. Our original planned functional characteristics that were completed and implemented include:

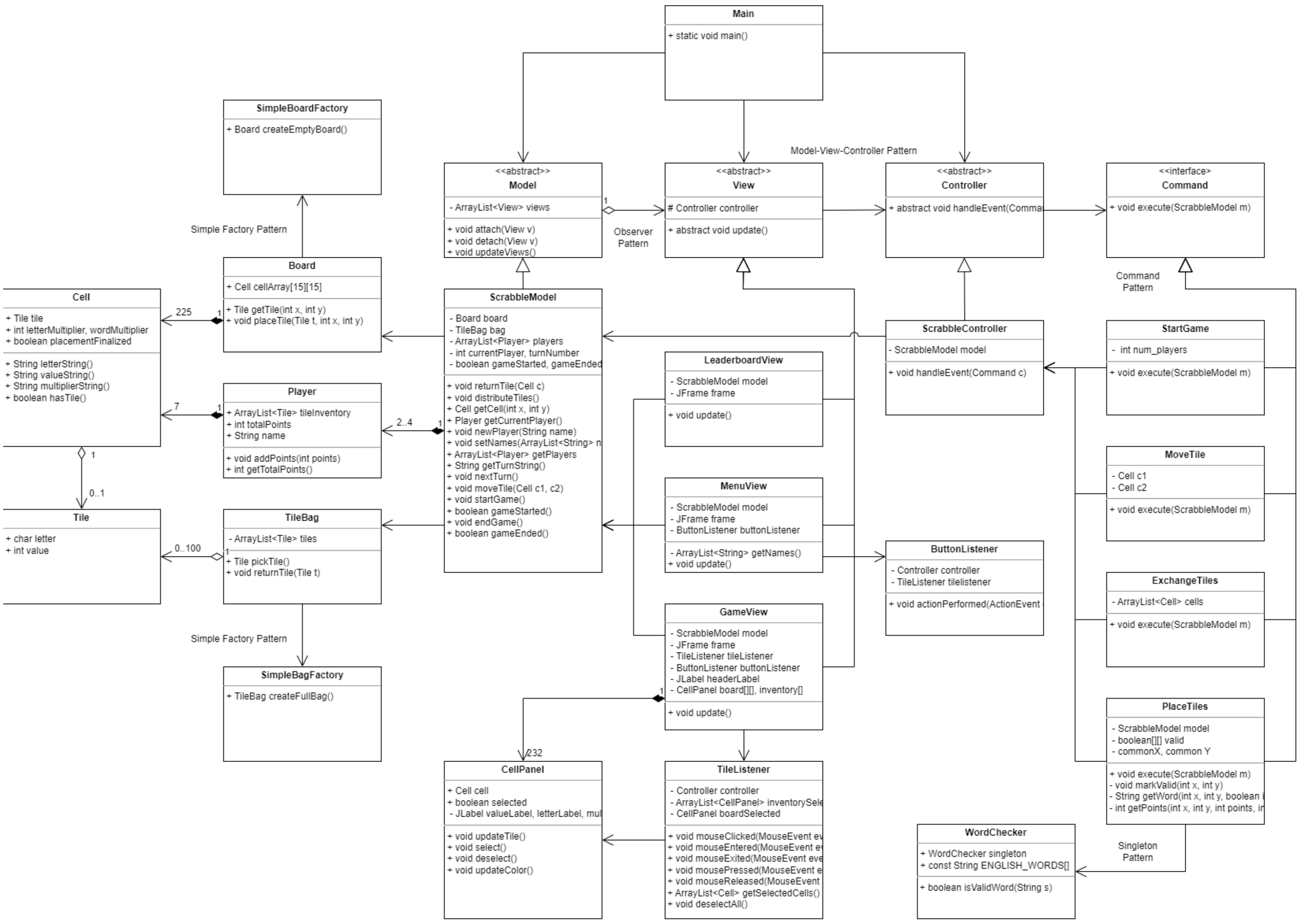
- Random distribution of tiles to players (7 at a time from a fixed set)
- GUI of board & letter tiles where users can place tiles to form words
- Local multiplayer functionality (2-4 unique players in a game, each with independent inventory, custom name, and score)
- Turn-based system to keep track of which player is currently active.
- Validity verification of words played from a dictionary of English words and are held within a single row or column
- Calculation of word point totals based on letter values and letter/word multipliers on cells on the board, 50 point bonus for using all 7 tiles
- Ability for players to skip a turn by exchanging 0-7 letter blocks in their “inventory” with others
- Ability to end game automatically once a player has no more tiles, then announce winner

Our planned nonfunctional features including intuitive gameplay to users, the application running smoothly without extensive loading times, and the program being platform-independent (assuming Java is installed) were also all achieved.

Features that were not implemented are primarily aesthetic. More work could be done by fine-tuning the placement of UI elements and adding helpful information like a real-time leaderboard. These changes were not made due to time constraints and prioritization of functional features and gameplay completeness over appearances.

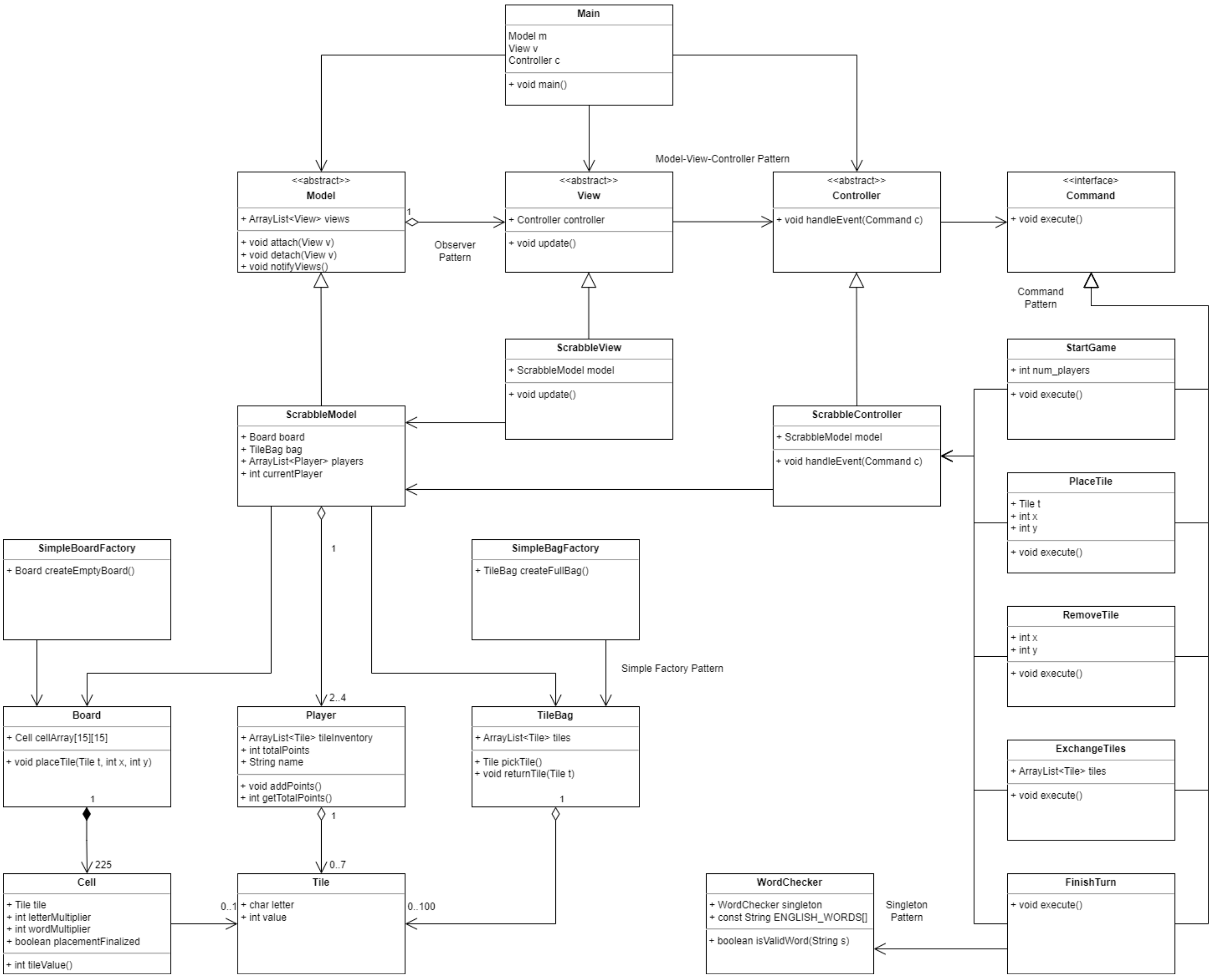
The most notable changes from Project 5 and 6 were additional classes added to support UI elements in the Swing library. Modifications were made as detailed below.

Class Diagram and Comparison Statement


Main
+ static void main()
Model-View-Controller Pattern
<<abstract>>
Model
- ArrayList<View> views
+ void attach(View v)
+ void detach(View v)
+ void updateViews()
<<abstract>>
View
# Controller controller
+ abstract void update()
<<abstract>>
Controller
+ abstract void handleEvent(Comma
<<interface>>
Command
+ void execute(ScrabbleModel m)
Observer Pattern
Command Pattern
SimpleBoardFactory
+ Board createEmptyBoard()
Simple Factory Pattern
Board
+ Cell cellArray[15][15]
+ Tile getTile(int x, int y)
+ void placeTile(Tile t, int x, int y)
Cell
+ Tile tile
+ int letterMultiplier, wordMultiplier
+ boolean placementFinalized
+ String letterString()
+ String valueString()
+ String multiplierString()
+ boolean hasTile()
225
1
7
1
Player
+ ArrayList<Tile> tileInventory
+ int totalPoints
+ String name
+ void addPoints(int points)
+ int getTotalPoints()
2..4
1
0..1
Tile
+ char letter
+ int value
0..100
1
TileBag
- ArrayList<Tile> tiles
+ Tile pickTile()
+ void returnTile(Tile t)
Simple Factory Pattern
SimpleBagFactory
+ TileBag createFullBag()
ScrabbleModel
- Board board
- TileBag bag
- ArrayList<Player> players
- int currentPlayer, turnNumber
- boolean gameStarted, gameEnded
+ void returnTile(Cell c)
+ void distributeTiles()
+ Cell getCell(int x, int y)
+ Player getCurrentPlayer()
+ void newPlayer(String name)
+ void setNames(ArrayList<String> n
+ ArrayList<Player> getPlayers
+ String getTurnString()
+ void nextTurn()
+ void moveTile(Cell c1, c2)
+ void startGame()
+ boolean gameStarted()
+ void endGame()
+ boolean gameEnded()
LeaderboardView
- ScrabbleModel model
- JFrame frame
+ void update()
MenuView
- ScrabbleModel model
- JFrame frame
- ButtonListener buttonListener
- ArrayList<String> getNames()
+ void update()
GameView
- ScrabbleModel model
- JFrame frame
- TileListener tileListener
- ButtonListener buttonListener
- JLabel headerLabel
- CellPanel board[][], inventory[]
+ void update()
1
232
CellPanel
+ Cell cell
+ boolean selected
- JLabel valueLabel, letterLabel, mul
+ void updateTile()
+ void select()
+ void deselect()
+ void updateColor()
TileListener
- Controller controller
- ArrayList<CellPanel> inventorySele
- CellPanel boardSelected
+ void mouseClicked(MouseEvent ev
+ void mouseEntered(MouseEvent e
+ void mouseExited(MouseEvent eve
+ void mousePressed(MouseEvent e
+ void mouseReleased(MouseEvent
+ ArrayList<Cell> getSelectedCells()
+ void deselectAll()
ScrabbleController
- ScrabbleModel model
+ void handleEvent(Command c)
ButtonListener
- Controller controller
- TileListener tilelistener
+ void actionPerformed(ActionEvent
StartGame
- int num_players
+ void execute(ScrabbleModel m)
MoveTile
- Cell c1
- Cell c2
+ void execute(ScrabbleModel m)
ExchangeTiles
- ArrayList<Cell> cells
+ void execute(ScrabbleModel m)
PlaceTiles
- ScrabbleModel model
- boolean[][] valid
- commonX, common Y
+ void execute(ScrabbleModel m)
- void markValid(int x, int y)
- String getWord(int x, int y, boolean i
- int getPoints(int x, int y, int points, in
WordChecker
+ WordChecker singleton
+ const String ENGLISH_WORDS[]
+ boolean isValidWord(String s)
Singleton Pattern

Above is the final UML class diagram for the state of the system in Project 7. It shows all classes and their relationships, along with the patterns that were implemented in the design.

Compared to the initial UML diagram submitted in Project 5 (pictured below), the fundamental structure of the code has remained largely unchanged. The only major changes in the Model and Controller subsystems were various new helper methods to improve communication between classes. The majority of changes took place in the View subsystem as we discovered the intricacies of creating UI elements with the Swing library. A handful of classes were added to support different windows (menu, game, and leaderboard), as well as to encapsulate repetitive details like displaying tiles and handling button presses.



Third-Party code vs. Original code Statement

The Java Swing framework was utilized in order to create a GUI for the application. The Java Tutorials (https://docs.oracle.com/javase/tutorial/) were referenced in the creation of the visual interface elements of the application within the several concrete View objects. Besides the use of standard libraries, all code was written from scratch in accordance with our system requirements & design.

Statement on the OOAD process for your overall Semester Project

1. Utilizing the Model-View-Controller pattern as the main basis of the design of the system led to questions about the functional responsibilities of the three major components. The Model subsystem holds all information on the state of the game, requiring a large number of functions for reading and modifying the state within the concrete ScrabbleModel class. The View subsystem is responsible for displaying the game as it exists in the Model, while the Controller subsystem is dedicated to execution of the various Commands which encapsulate game logic. Though it was sometimes difficult to separate the concerns of each subsystem, it ultimately resulted in more cohesive code within each one.
2. We decided to utilize multiple concrete implementations of Views in order to delegate which should be used during different states of the game (Menu, Gameplay, and Leaderboard), rather than a less-cohesive single catch-all concrete view implementation. The views are changed based on the current game state in an observer-type relation through the invocation of the update method. While code repetition was increased in some ways (due to necessary boiler-plate code for a view with the Swing library) by this decision, it improved the cohesiveness of each of the concrete Views, as they were focused on displaying a single dynamic frame rather than a multitude of varying displays.
3. The separation between the Model and View subsystems seemed to create an especially large amount of increased complexity and code repetition. An additional CellPanel object had to be created and managed for each cell on the board and player inventory, which frequently had to call back to get information from the associated Cell object in the Model. It is not entirely clear whether the increase in cohesion (separating the state of each cell from the on-screen element that represents it) is worth the overall increase in system complexity and increased coupling of objects.