TMGE Design Document

Lancy Tan, Michael Wijangco, Samhitha Tarra,
Jules-Andrei Labador, Becky Dinh, Aliyah Clayton

University of California, Irvine

INF122: Software Design: Structure & Implementation
Dr. Jae Young Bang
March 14, 2022

Goals

The goal of the TMGE (Tile Matching Game Environment) is to act as a foundation for tile matching games with a board layout. This includes games such as Tetris, Candy Crush, and Bejeweled. All games built on top of the TMGE will follow its implementation.

Timeline

Date	Document	Notes
3/2	Design Document	With complete UML diagram(s)
3/4	Library Research	Research applicable Java libraries for TMGE
3/5 - 3/13	Development	
3/14	Project Package	● Updated design document ● Game documentation PDF ● Interface document (with definitions) ● Finished source code (includes TMGE + 2 games) ● Github repository ● 10 minute video presentation

Revision History

- 3/2 Created preliminary design document
- 3/14 Added revisions for final version of design document
    - Changed games to Connect Four and TicTacToe
    - Updated UML diagram
    - Updated Tile, Board, State, Controller, Player classes
    - Removed Main and GameType classes

Games to Implement

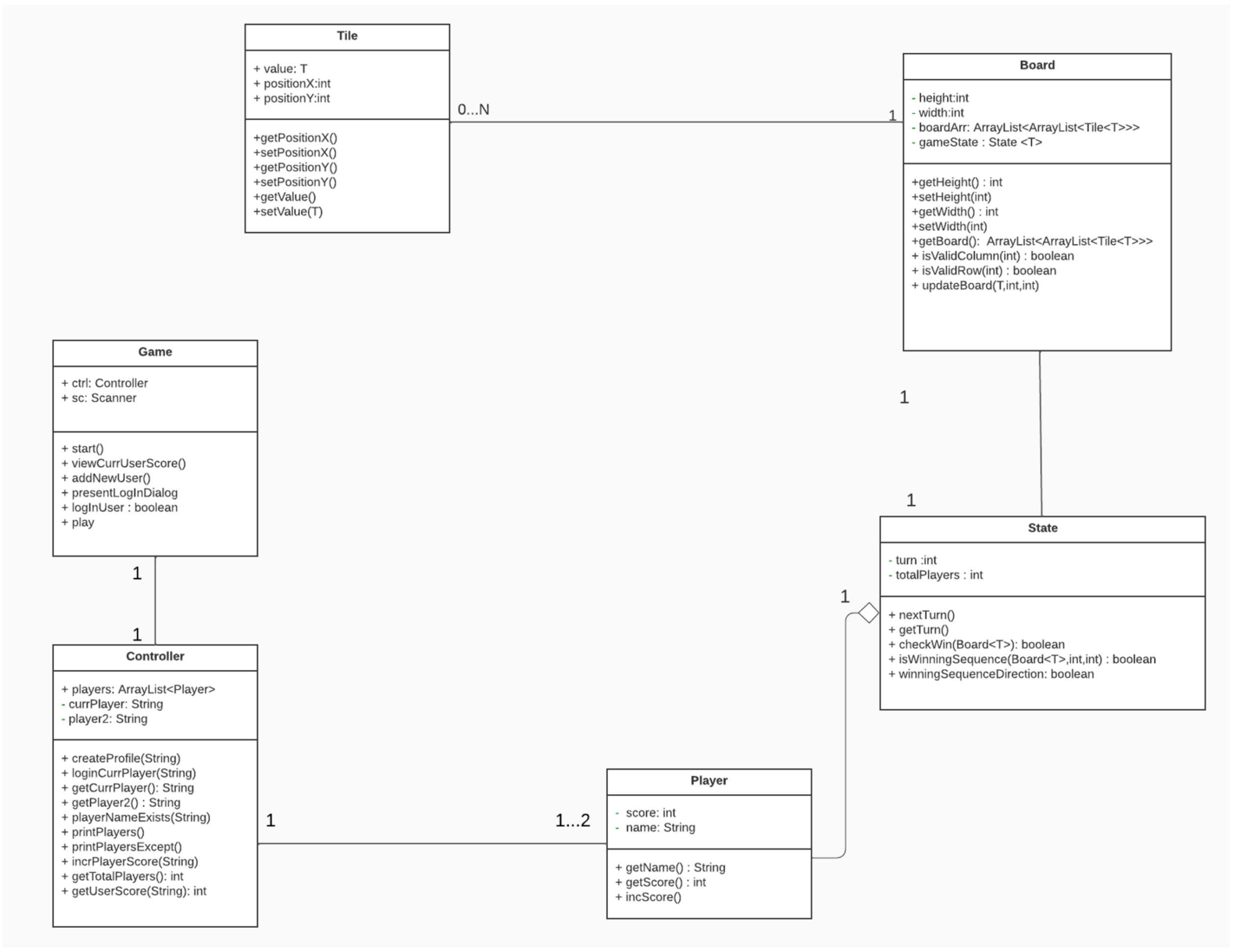
Connect Four:

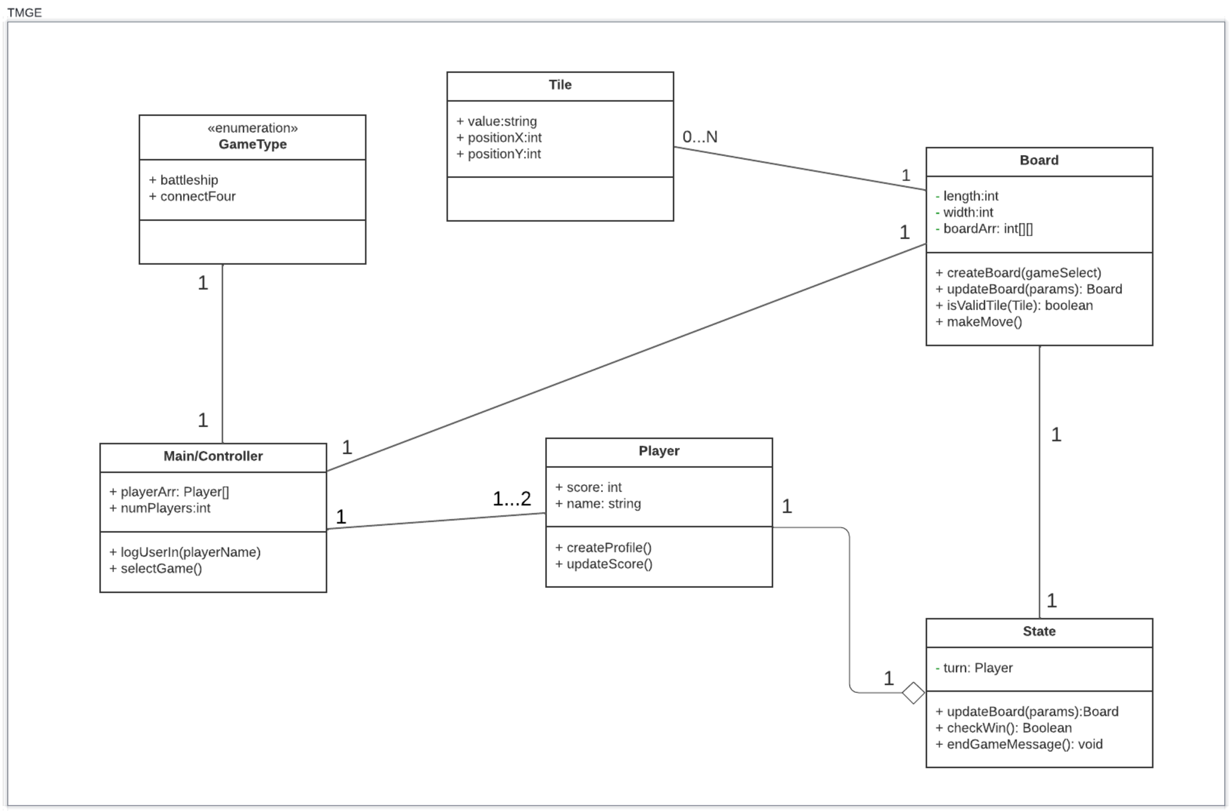
This game involves a 6 by 7 tile board where two players take turns dropping tiles of their colors onto the board. Their tile falls to the bottom-most available position on the column of that board. This process goes on until one player reaches a win. The win condition is when one player is able to achieve four consecutive tiles in a linear pattern, diagonally, vertically, or horizontally.

TicTacToe:

This game involves two players taking turns putting X's and O's on a 3 by 3 board. Each player is restricted to the symbol they are assigned. A player reaches a win when they are able to get their three matching tiles in a row either diagonally, vertically, or horizontally.

Classes


Tile
+ value: T
+ positionX:int
+ positionY:int
+getPositionX()
+setPositionX()
+getPositionY()
+setPositionY()
+getValue()
+setValue(T)
0...N
1
Board
- height:int
- width:int
- boardArr: ArrayList<ArrayList<Tile<T>>>
- gameState : State <T>
+getHeight() : int
+setHeight(int)
+getWidth() : int
+setWidth(int)
+getBoard(): ArrayList<ArrayList<Tile<T>>>
+ isValidColumn(int) : boolean
+ isValidRow(int) : boolean
+ updateBoard(T,int,int)
1
1
Game
+ ctrl: Controller
+ sc: Scanner
+ start()
+ viewCurrUserScore()
+ addNewUser()
+ presentLogInDialog
+ logInUser : boolean
+ play
1
1
State
- turn :int
- totalPlayers : int
+ nextTurn()
+ getTurn()
+ checkWin(Board<T>): boolean
+ isWinningSequence(Board<T>,int,int) : boolean
+ winningSequenceDirection: boolean
1
Controller
+ players: ArrayList<Player>
- currPlayer: String
- player2: String
+ createProfile(String)
+ loginCurrPlayer(String)
+ getCurrPlayer(): String
+ getPlayer2() : String
+ playerNameExists(String)
+ printPlayers()
+ printPlayersExcept()
+ incrPlayerScore(String)
+ getTotalPlayers(): int
+ getUserScore(String): int
1
1...2
Player
- score: int
- name: String
+ getName() : String
+ getScore() : int
+ incScore()



Tile

The tile class stores information about all the information necessary to display a tile, including the position of the tile on the board. The position will be X and Y integer coordinates. The value takes in a generic value to accommodate the type of value appropriate with the game. Methods of this class allow the access and mutation of its attributes.

Value: any (generic)
positionX: int
positionY: int
----
getPositionX()
setPositionX()
getPositionY()
setPositionY()
getValue()
setValue(T)

Board

The Board class stores information about the board grid, which is a two dimensional array of Tiles. Boards have a fixed length and width, and all tiles existing on the board will stay within the boundaries of the board. Methods in this class include setters and getters for the board, width and height dimensions, validation of boundaries (i.e. isValidRow(...) and isValidColumn(...)), updating the Board, and checking for wins. Checking wins is included in Board due to the unique nature of tile matching games, where the game may conclude after one action on the board. The state of the Board also determines whether there is a win or not.

The Board class should be used as a superclass for future game implementations for special Board actions unique to the game. Certain methods such as checkWin() should be overridden for win conditions applicable to the game.

```
- height:int
- width:int
- boardArr: ArrayList<ArrayList<Tile<T>>>
- gameState : State <T>
--------------
getHeight() : int
setHeight(int)
getWidth() : int
setWidth(int)
getBoard():  ArrayList<ArrayList<Tile<T>>>
isValidColumn(int) : boolean
isValidRow(int) : boolean
updateBoard(T,int,int)
```

State

The State handles switching turns between the players, and checking if the Board has reached a win condition. It does this by checking if there is a winning sequence with the current Tile placements. This is a superclass that is encouraged to be extended to add winning conditions of the game being implemented.

turn :int
totalPlayers : int
----
nextTurn()
getTurn()
checkWin(Board<T>): boolean
isWinningSequence(Board<T>,int,int) : boolean
winningSequenceDirection: boolean

Game

The game class is responsible for implementing functionality for the game that's to be created using the TMGE SDK. It ensures that enough players have been created in order to begin the game and is responsible for receiving user input . To implement game logic, you must override the `play()` function and implement it there.

ctrl: Controller
sc: Scanner
----
start()
viewCurrUserScore()
addNewUser()
presentLogInDialog()
logInUsers()
play()

Controller

The Controller class acts as a menu. Housing access to players and their scores. Functionality for game dialog that includes player information is "controlled" here. Currently our implementation is focused on 2-player functionality. The player's ArrayList acts as the container for all the player profiles that have been created, for most implementations, only 2 players from that list can be logged in at a time.

players: ArrayList<Player>
currPlayer: String
player2: String
----
createProfile(String)
loginCurrPlayer(String)
getCurrPlayer(): String
getPlayer2() : String
playerNameExists(String)
printPlayers()
printPlayersExcept()
incrPlayerScore(String)
getTotalPlayers(): int
getUserScore(String): int

Player

The Player class stores information about the player's login information (username) and the score that they have in the game. Methods of this class include getting their name, getting the score, and incrementing the score.

Score: int
Name: String
----
getName() : String
getScore() : int
incScore()