Class Overview

**FooGame**

The basic instance of a game, holds the Map representing the state of the board, as well as lists of Players, Rules and WinConditions. Also takes care of turn handling and tracking. It implements the built-in Java Observable interface, and passes itself to observers on updates so observers can pull the information they want. Most observers only pull the boardstate.

Important Fields:

**Map<Pair<Integer,Integer>, Player> boardstate;**

This data structure represents the state of the board using a Map with Pairs (from javafx.util.Pair) as keys representing coordinates of board spaces, and Player values representing which player has a stone in that space. This means that effectively Players place references to themselves in the spaces. Spaces the contain no stones have entry values of null.

**ArrayList<x> playerlist/ruleslist/winlist**

Lists of Players, Rules, and WinConditions. Set in class construction by Mode class.

Important Methods:

**Void startGame()**

Starts the game by making first call to nextTurn()

**Void nextTurn()**

Increments turn count, then prompts the player whose turn it is to make a move, then makes sure move complies to all rules, then allows move to alter the board, then checks for any winners. If no one has won, calls itself recursively to continue turn cycle.

**Void showMove()**

Sends update to all observers. Should be called any time boardstate changes. nextTurn() handles this automatically

**Map<Pair<Integer,Integer>, Player> getBoardState()**

Returns a copy of the current boardstate. Used by observers to pull boardstate. By giving a copy instead of the original, it allows any observer to alter their copy of the Map as needed.

**ArrayList<Player> getPlayers()**

Returns a copy of the list of Players. Used by observers to pull playerlist. WinConditions specifically need this information.

**public static boolean checkAdjacency(Pair<Integer,Integer> c1, Pair<Integer,Integer> c2)**

Useful static method for checking if two Pairs, or “spaces”, are next to each other.

Construction:

**Public FooGame(Mode m)**

The constructor for FooGame takes any object which implements the Mode interfaces, and uses that object’s methods to populate its fields. See Mode for details.

--------------------------------------------------------------------------------------------

**Mode**

This is an interface for a class meant to be used to instantiate the starting conditions of FooGame. The “Default” class implements this interface with the standard form of FooGame as described. We recommend that any minor alterations to types of Players, Rules, or Win Conditions be done by inheriting from Default and overriding the appropriate methods.

Methods:

**public ArrayList<Player> setPlayers(Observable o)**

**public ArrayList<Rule> setRules(Observable o)**

**public Map<Pair<Integer,Integer>,Player> setBoard()**

**public ArrayList<WinCondition> setWinConditions(Observable o)**

**public FooGUI setGUI(Observable o)**

All of these are basically construction methods used by the FooGame constructor. Once implemented, the returns of these methods represent the initial state of the game.

**Player**

To create a new type of Player, simply inherit from the abstract Player class. The constructor of this superclass automatically hooks the Player up as an observer of FooGame, but if your new player type needs information from FooGame other than the boardstate, you will need to override the update() method. Otherwise, the only thing a Player class needs to implement is the makeMove() method.

**protected Map<Pair<Integer,Integer>,Player> boardstate**

Like all other observers of FooGame, Player has its own field to refer to a copy of the current boardstate, which it can reference directly as needed.

**private int stones**

Integer representing the number of stones the Player holds in their hand. Defaults to 21.

**public void useStone()**

Decrements stones. Should be called whenever the Player places a stone from their hand onto the board. (Tip: This means that this method should be called in the alterBoard() method of an appropriate Move)

**public void setStones(int i)**

Sets stones to new value.

**public void getStones()**

Returns stones.

**public abstract Move makeMove()**

Essentially this method is where the Player class decides what kind of move it wants to make, whether this involves an AI running an algorithm or simply asking a user for input. Returns a constructed Move.

**Move**

Abstract class representing an interface for a class which can alter the boardstate of FooGame. Should be constructed by Players, and therefore holds a reference to the Player which constructed them. To make a new type of Move, simply inherit this abstract class and implement the alterBoard() method. Additionally, the constructor of any Move type class should take at least a Player to pass to the super constructor.

**protected Player player**

This field holds a reference to the Player who made this Move.

**public Player getPlayer()**

Returns player field.

**public abstract Map<Pair<Integer,Integer>,Player> alterBoard(Map<Pair<Integer,Integer>,Player> boardstate)**

Method which, when implemented, will directly alter FooGame’s boardstate. This is done usually by placing or moving the Move’s Player in certain spaces.

**Rule**

Abstract class. Observes FooGame. Purpose is to check a Move to make sure it follows set rules of the game.

**public abstract boolean check(Move m)**

Method to be implemented. Takes a Move object. Returns True if move is legal, and False if it is not.

**WinCondition**

Abstract class. Observes FooGame. Purpose is to check if anyone has won. Every WinCondition in winlist is checked at the end of every turn.

**public abstract Player checkForWinner()**

Method to be implemented. Checks for a winner in a certain way using both the current boardstate and information from Players. Should return the Player who has won, or null if no player has won.

**FooGUI**

Abstract class. Observes FooGame. Purpose is to represent boardstate graphically.

**public abstract void draw()**

Method to be implemented. Called at the end of FooGui’s update() method automatically. Should somehow represent boardstate to user graphically.