Game Center

Group 0704

Shang Liu, Qi Zou, Runqi Bi, Quan Xu, Zijin Zhang

Game Show Time

Welcome to Shang Liu !

Design Patterns: Factory

Related Classes:

- BoardManager
  - FlipToWinBoardManager
  - ColorMatchingBoardManager
  - ManagerFactory
  - DataManager

- Obscure the creation process for different board managers.
- High cohesion, low coupling
- More extendable.

```java
/**
 * A Factory to generate Manager based on the given information.
 */
class ManagerFactory {


    /**
     * Return the Manager that is required.
     *
     * @return the the Manager that is required.
     */
    SuperManager getManager(String managerType, int complexity) {
        switch (managerType) {
            case "ST":
                return new BoardManager(complexity);
            case "CM":
                return new ColorBoardManager(complexity);
            case "FTW":
                return new FlipToWinBoardManager(complexity);
        }
        return null;
    }
}
```

```java
    /**
     * This will create a new boardManager according to the currentGameName
     */
    public void startNewGame(int complexity) {
        ManagerFactory managerFactory = new ManagerFactory();
        this.boardManager = managerFactory.getManager(currentGameName, complexity);
    }
}
```

Design Pattern: Dependency Injection

Related Classes:
most of classes

- Decouple the usage of an object from its creation
- Helps to follow single responsibility principles
- SOLID's dependency inversion priciple

```
/**
 * Manage a board that has been pre-populated.
 *
 * @param board      the board
 * @param complexity the complexity of game
 */
FlipToWinBoardManager(FlipToWinBoard board, int complexity) {
    super(complexity);
    this.board = board;
}
```

```
/**
 * Manage a board that has been pre-populated.
 *
 * @param board      the board
 * @param complexity the complexity of the game
 */
public BoardManager(Board board, int complexity) {
    super(complexity);
    this.board = board;
}
```

Design Patterns: Iterator Pattern

Related Classes:

ColorBoard
Board
FlipToWinBoard

- Easy Access to the elements of board
- No need to know its underlying representation

```
/**
 * Iterate over color tiles in a range of total number of tiles.
 */
private class ColorBoardIterator implements Iterator<ColorTile> {

    /**
     * The row number of the color tile.
     */
    private int row;
    /**
     * The column number of the color tile.
     */
    private int col;

    @Override
    public boolean hasNext() {
        return row < ColorBoard.this.rowNum &&
                col < ColorBoard.this.colNum;
    }

    @Override
    public ColorTile next() {
        if (hasNext()) {
            if (ColorBoard.this.colNum - 1 == col) {
                ColorTile temp = getGrid(row, col);
                row++;
                col = 0;
                return temp;
            }
            return getGrid(row, col++);
        }
        throw new NoSuchElementException();
    }
}
```

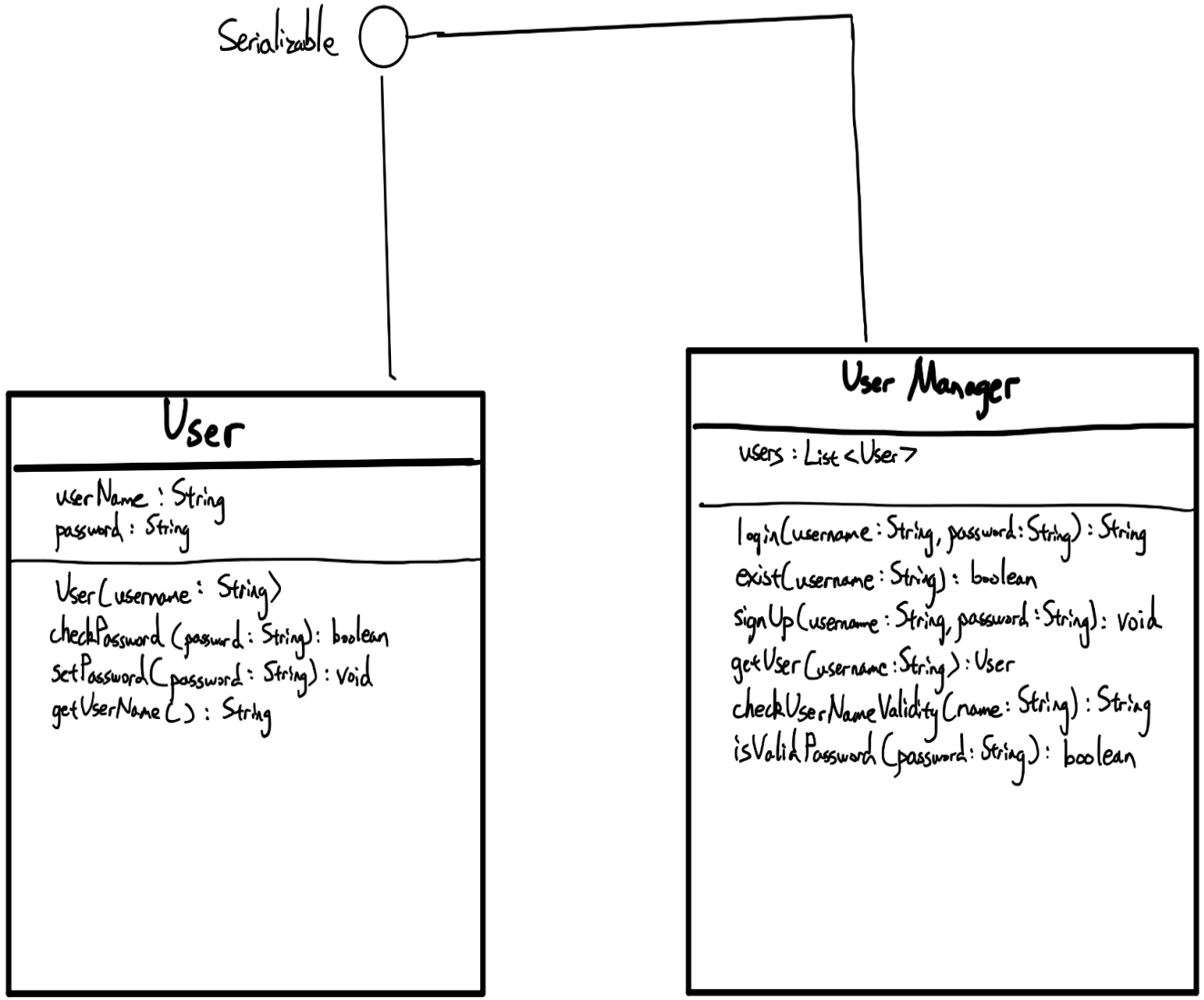
Interface: Undoable

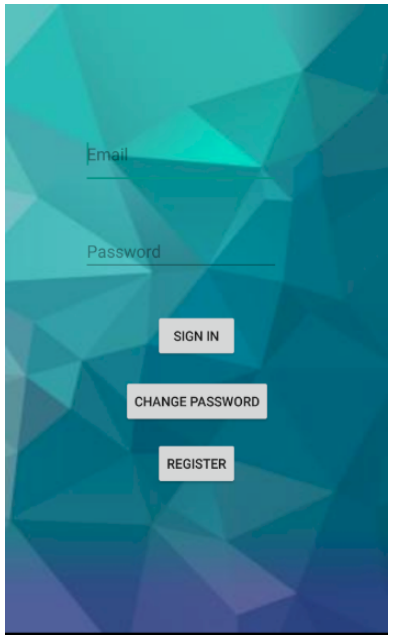
Related Classes:

BoardManager

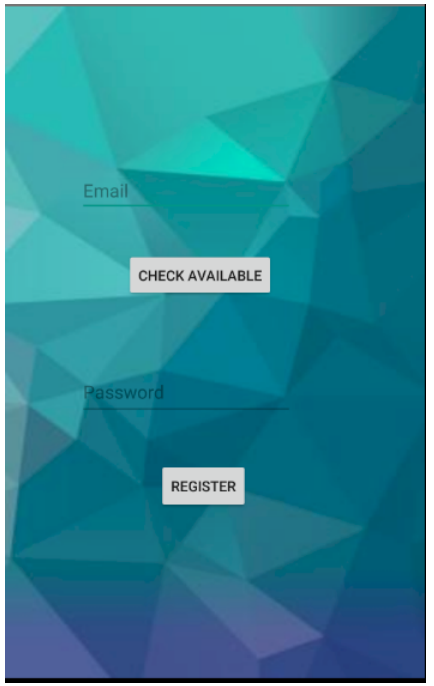
ColorMatchingManager

```java
/**
 * Implementing this interface allows the game to have undo feature.
 */
public interface Undoable {


    /**
     * Undo the previous move
     */
    void undo();


    /**
     * Return true if the undo function is available, false otherwise.
     *
     * @return true if the undo function is available
     */
    boolean undoAvailable();
}
```





Login



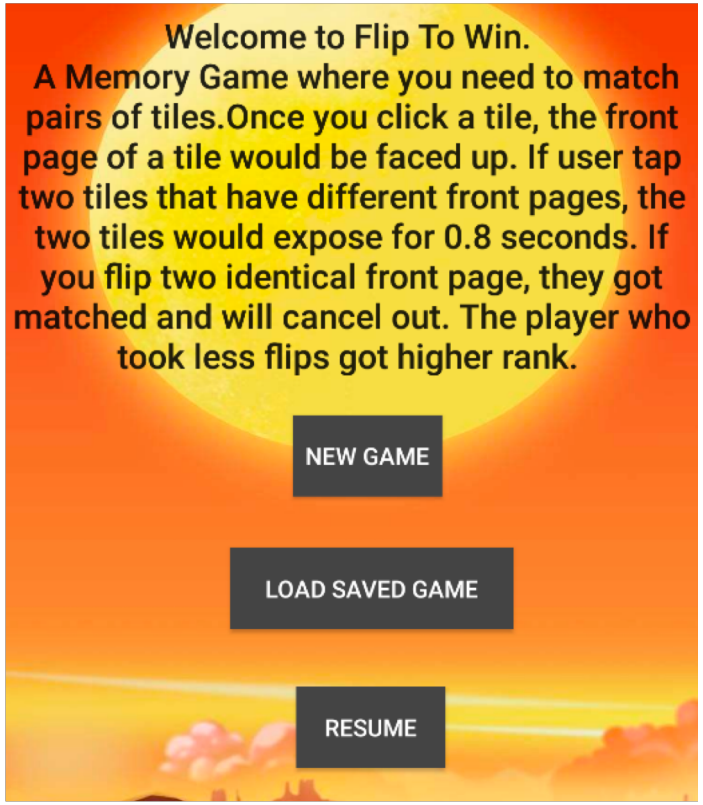
Register

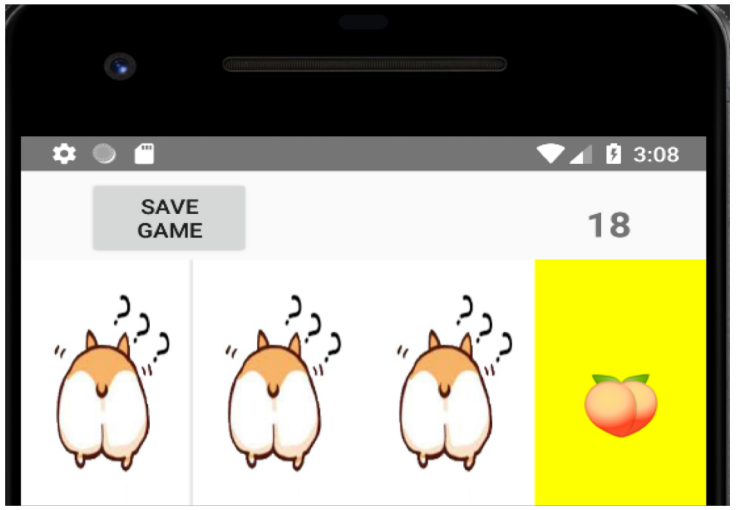
Welcome to Zijin Zhang!

FileManager

FileManager	
Responsibilities	Collaborations
Save to file, Load user Manager, Load Score board, Save/load game	User Manager, Score Board

Save and Load






```java
/**
 * Dispatch onPause() to fragments.
 */
@Override
protected void onPause() {
    super.onPause();
    FileManager.saveGame(this.getApplicationContext(),  operation: "Auto");
}

/**
 * Dispatch onStop() to fragments.
 */
@Override
protected void onStop() {
    super.onStop();
    FileManager.saveGame(this.getApplicationContext(),  operation: "Auto");
}
```

DataManager

DataManager	
Responsibilities	Collaborations
Set/get game name, Set/get user name, set/get boardManager Start new game	Board Manager

Does it important?

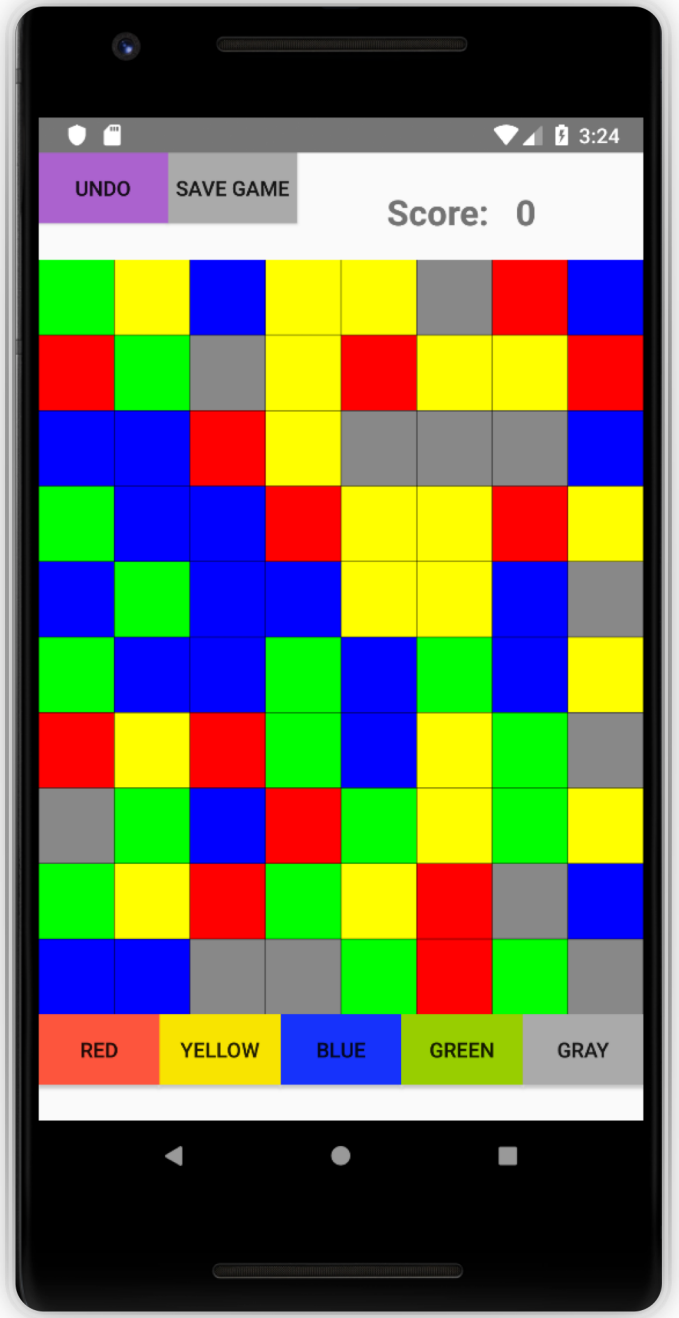
What is it?

Singleton Design pattern

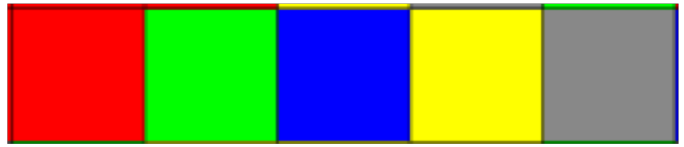
Why we use it?

Welcome to Runqi Bi !

Color Matching

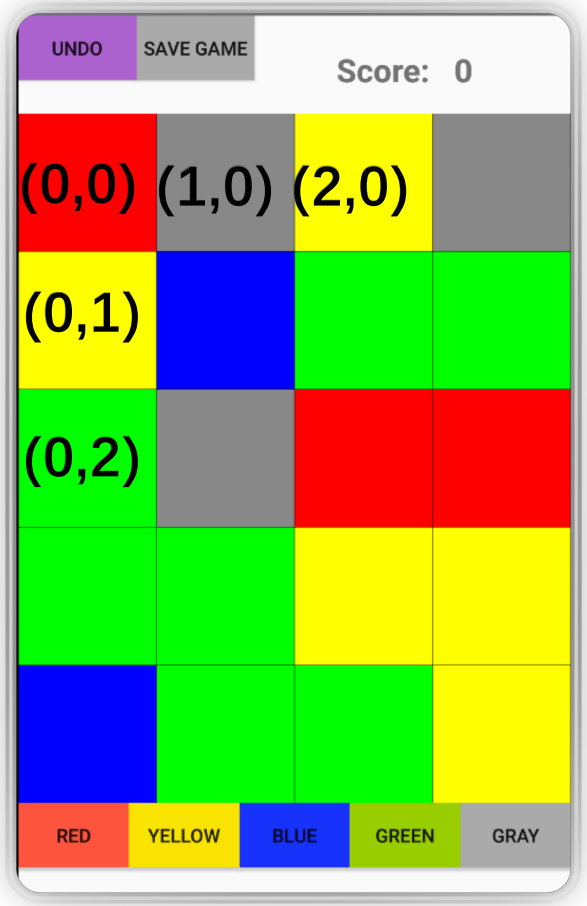


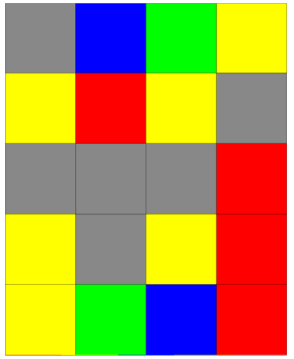
ColorTile



ColorTile	

Responsibilities	Collaborations
X-axis	ColorBoard
Y-axis	
Color	
Generate random Color	


UNDO
SAVE GAME
Score: 0
(0,0)
(1,0)
(2,0)
(0,1)
(0,2)
RED
YELLOW
BLUE
GREEN
GRAY



ColorBoard

ColorBoard (Iterator Design Pattern)	Super: SuperBoard
Responsibilities	Collaborations
Generate ColorBoard	ColorBoardManager
Get neighbour for a ColorTile	
Get a ColorTile on (x,y)	
Iterator	

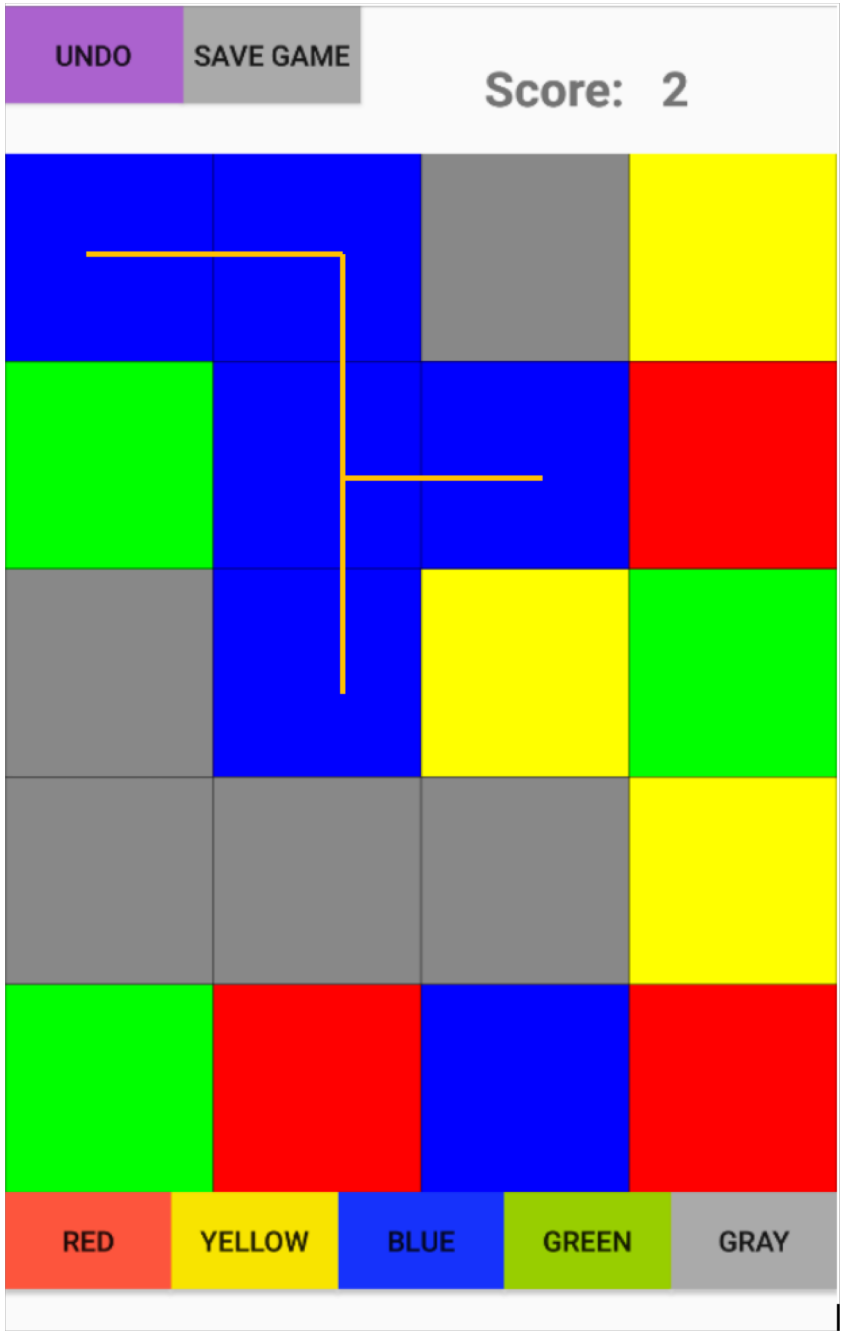
ColorBoardManager

ColorBoardManager	Super: SuperManager
Responsibilities	Collaborations
set ColorBoard according to complexity	Extends SuperManager
get all ColorTiles connect with ColorTile(0,0)	ColorBoard
Change Color	
undo	

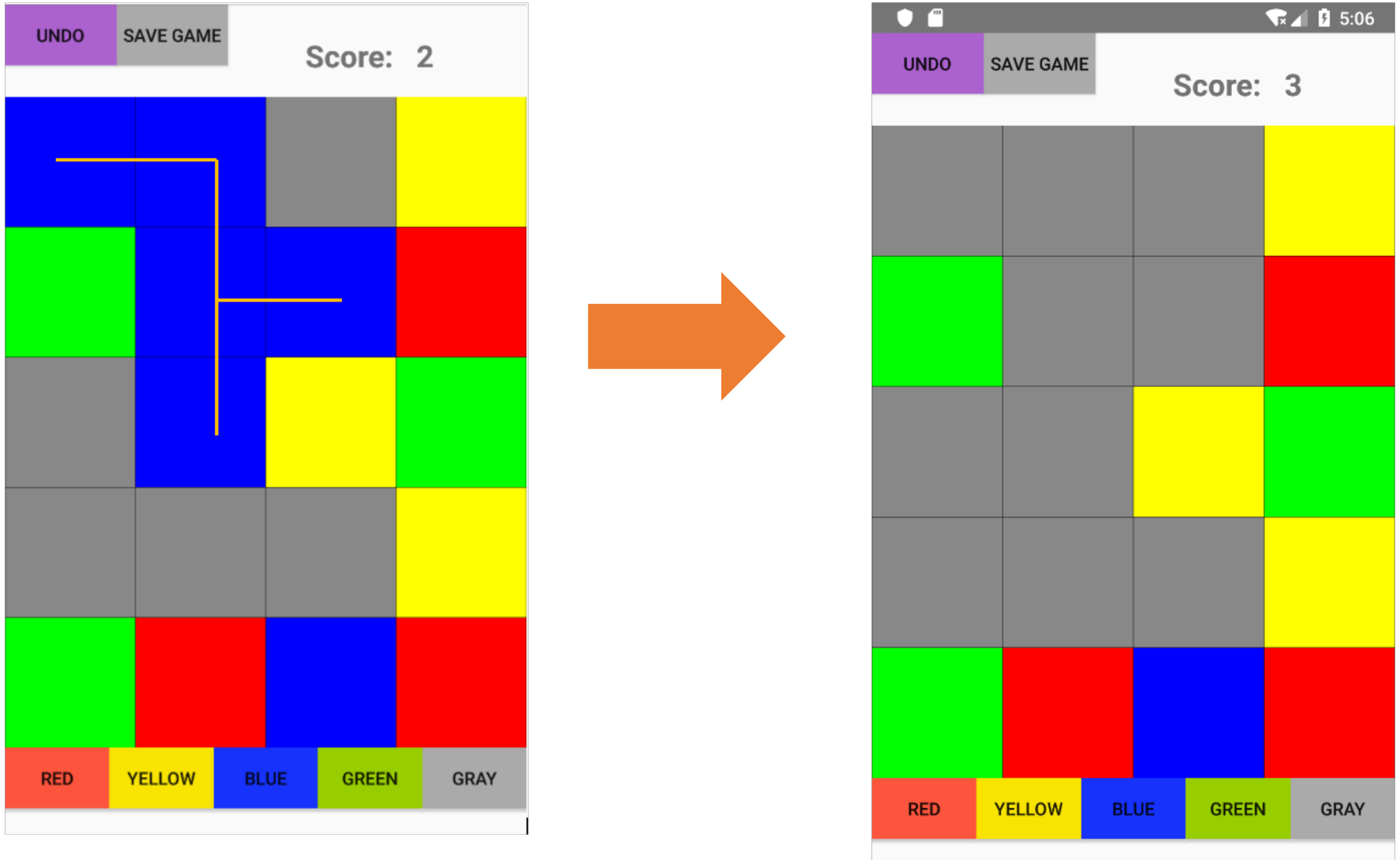
makeChange(**int** newColor)

- Find all Color Tiles connected with ColorTile(0,0) with the same color as ColorTile(0,0)
- Change those Color Tiles's color to newColor

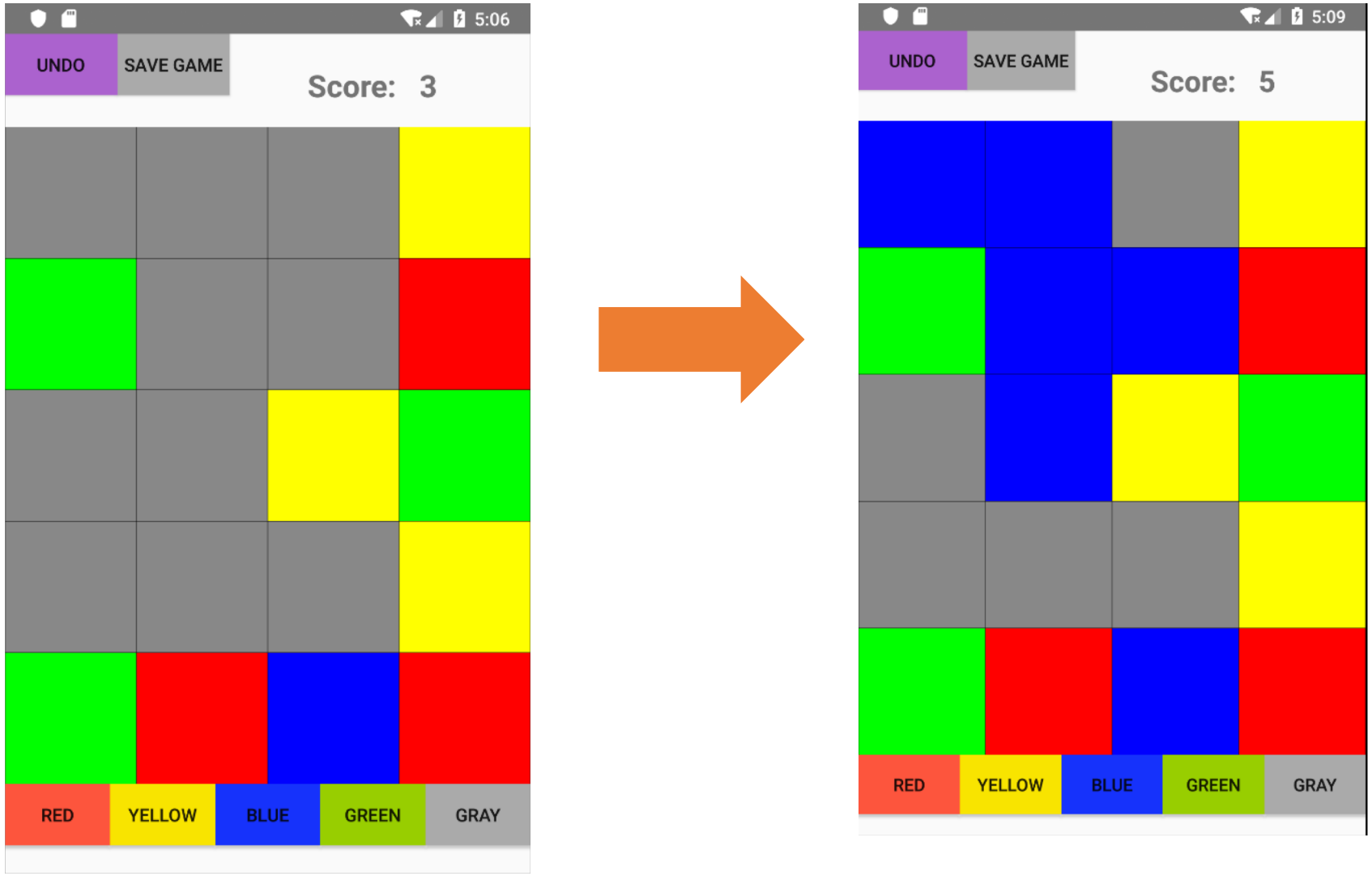
Find all Color Tiles connected with ColorTile(0,0) with the same color as ColorTile(0,0)



Change those Color Tiles's color to new Color



Undo


5:06
UNDO
SAVE GAME
Score: 3
RED
YELLOW
BLUE
GREEN
GRAY
5:09
UNDO
SAVE GAME
Score: 5
RED
YELLOW
BLUE
GREEN
GRAY

ColorView

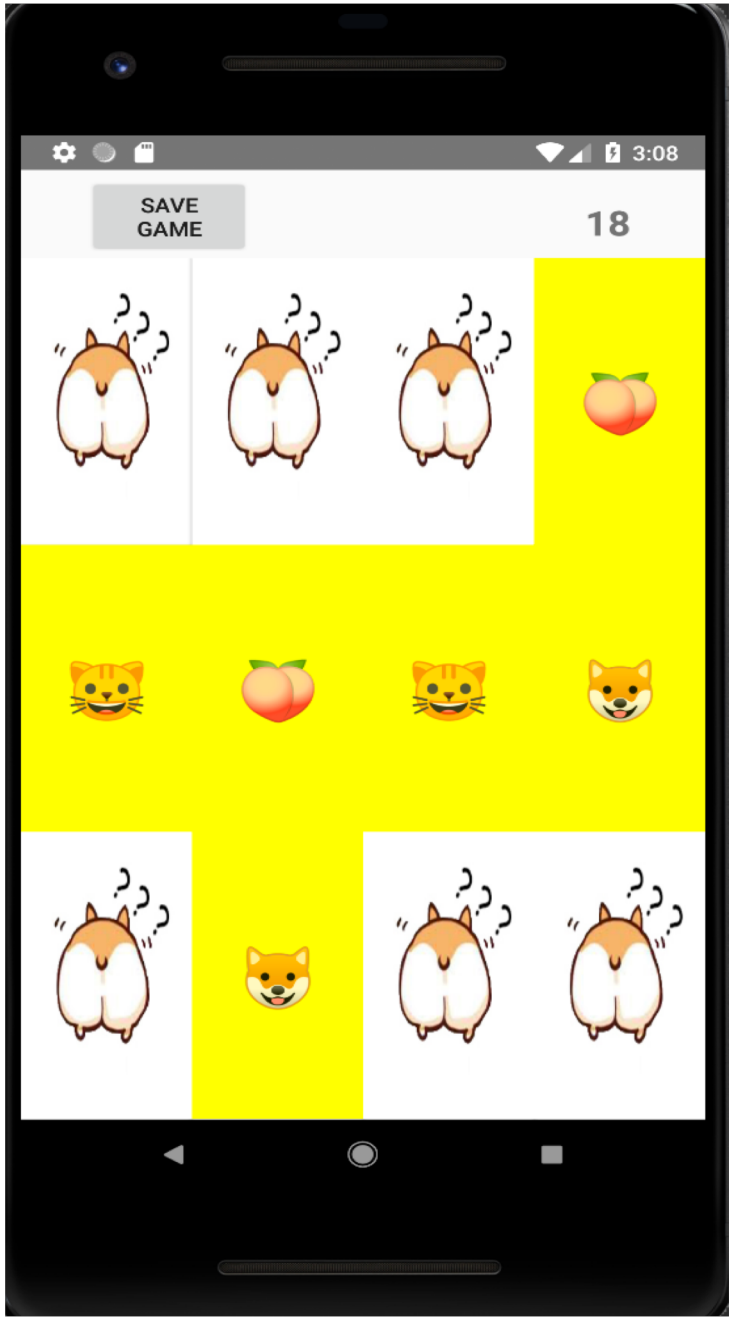
ColorView	
Responsibilities	Collaborations
view	ColorMatchingGameActivity
draw lines	
draw ColorTiles	ColorMatchingGameActivity

ColorMatchingActivity

ColorMatchingActivity	Super: AppCompatActivity
Responsibilities	Collaborations
Initialize Data	ColorBoardManager
Initialize View	ColorView
Draw color tiles and lines on Canvas	
Create 5 Buttons for changing colors	
Create Undo, Save Buttons	
Check Win	

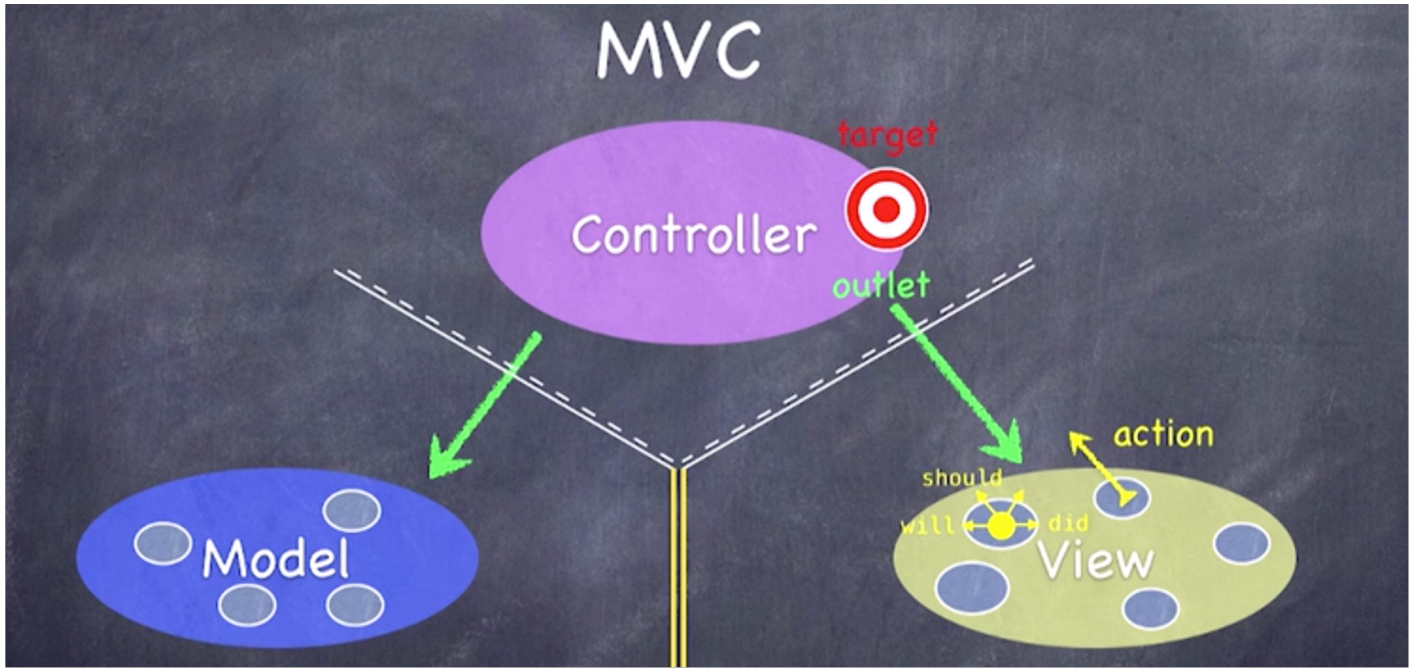
Welcome to Quan Xu !

FlipToWin

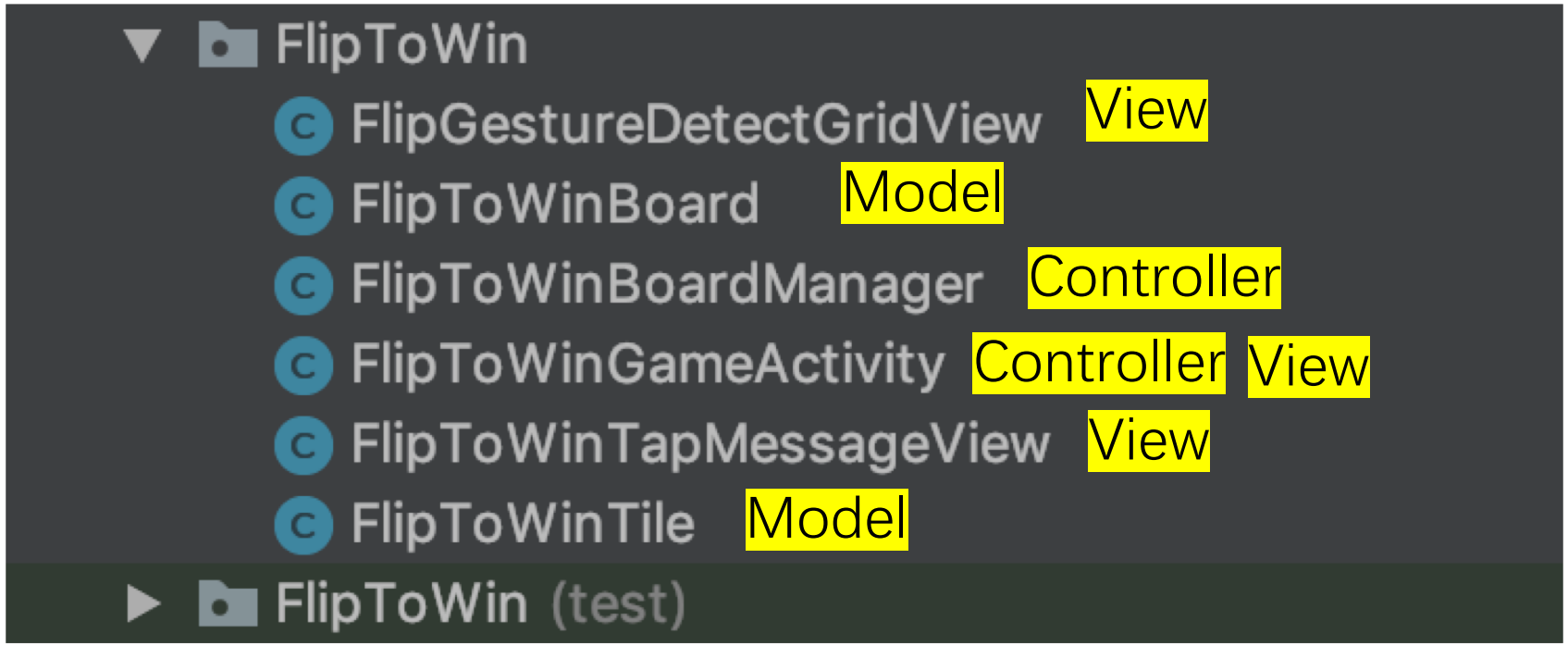


Design Patterns

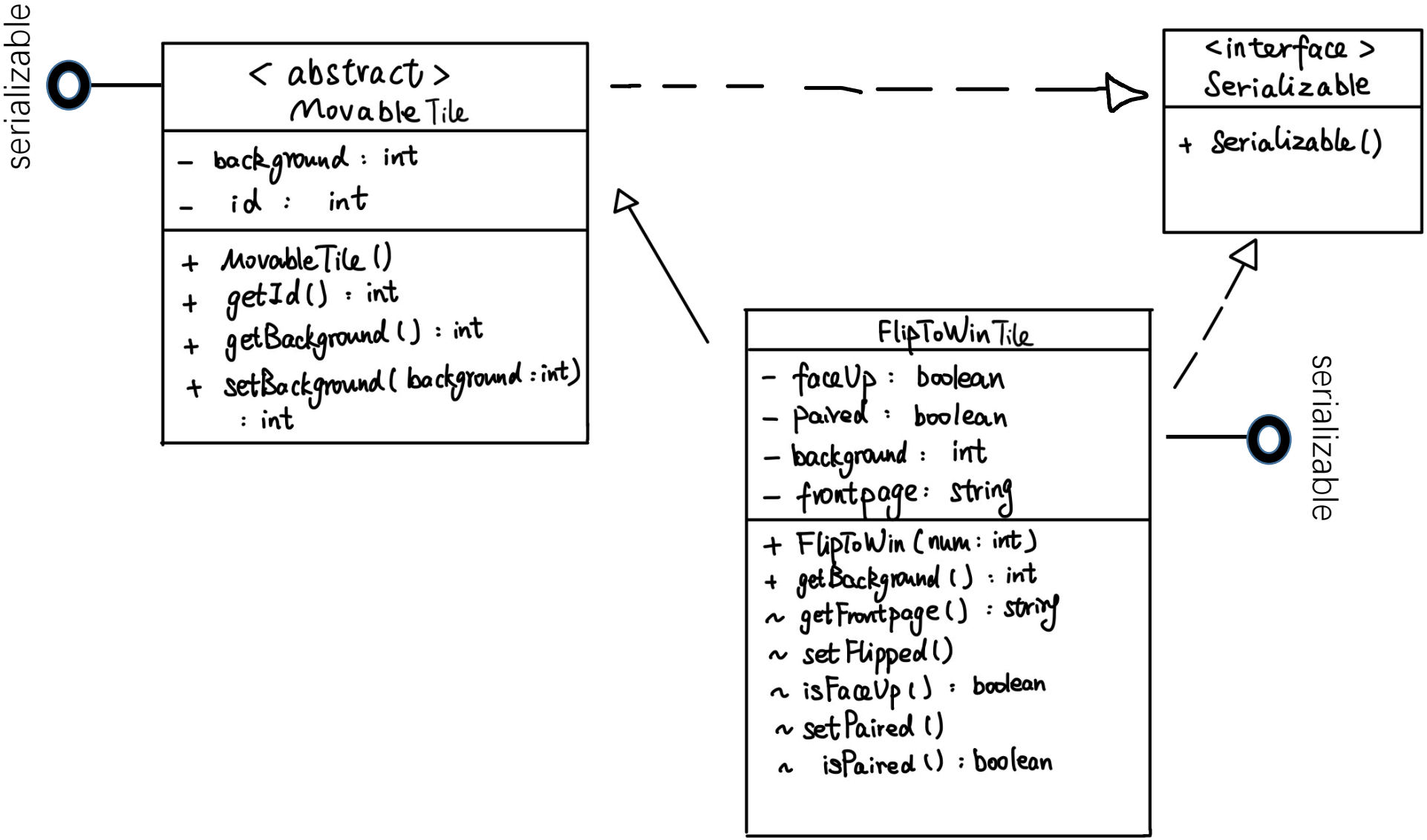
Model View Controller



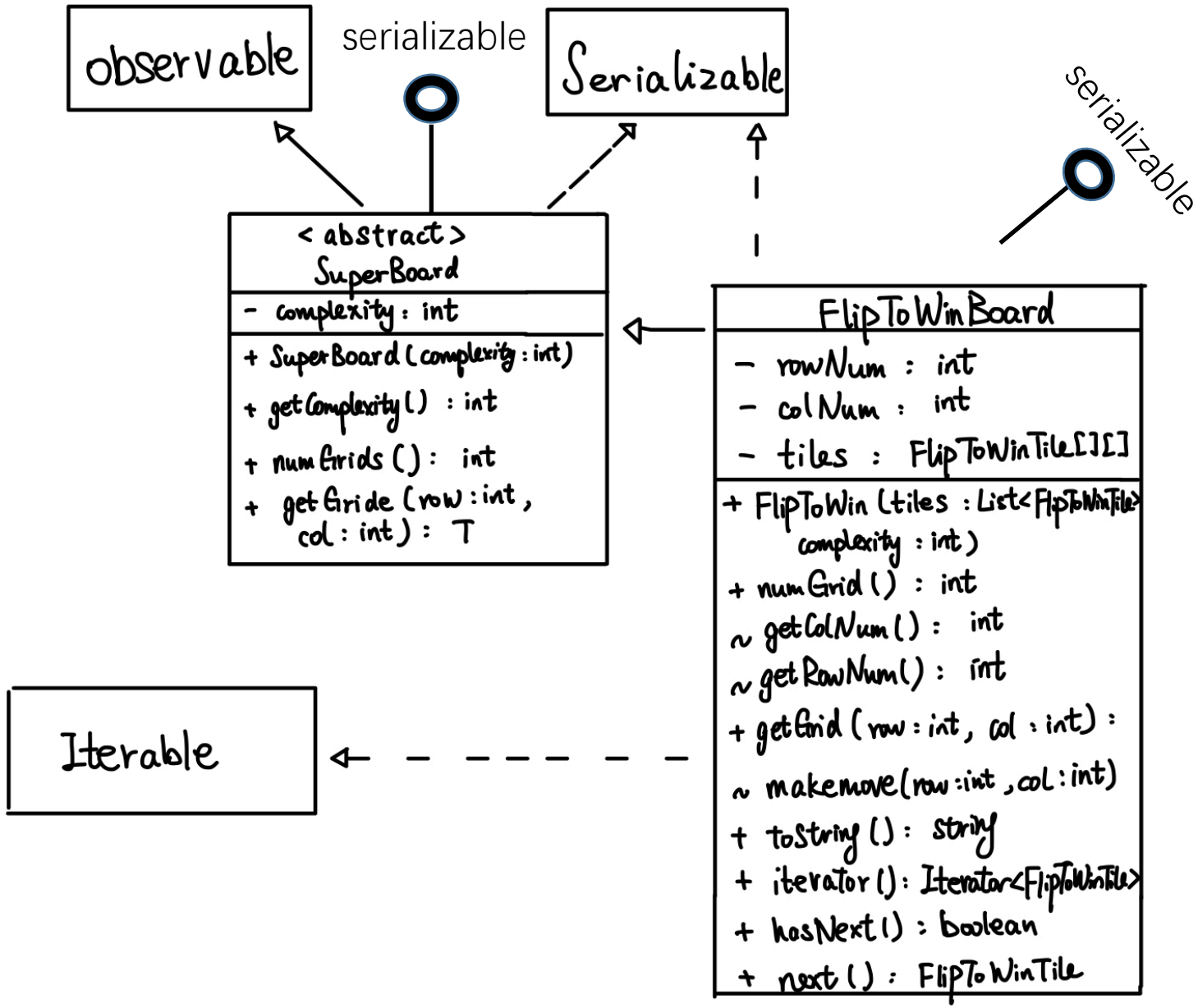
MVC design pattern



FlipToWinTile (model)



FlipToWinBoard (model)



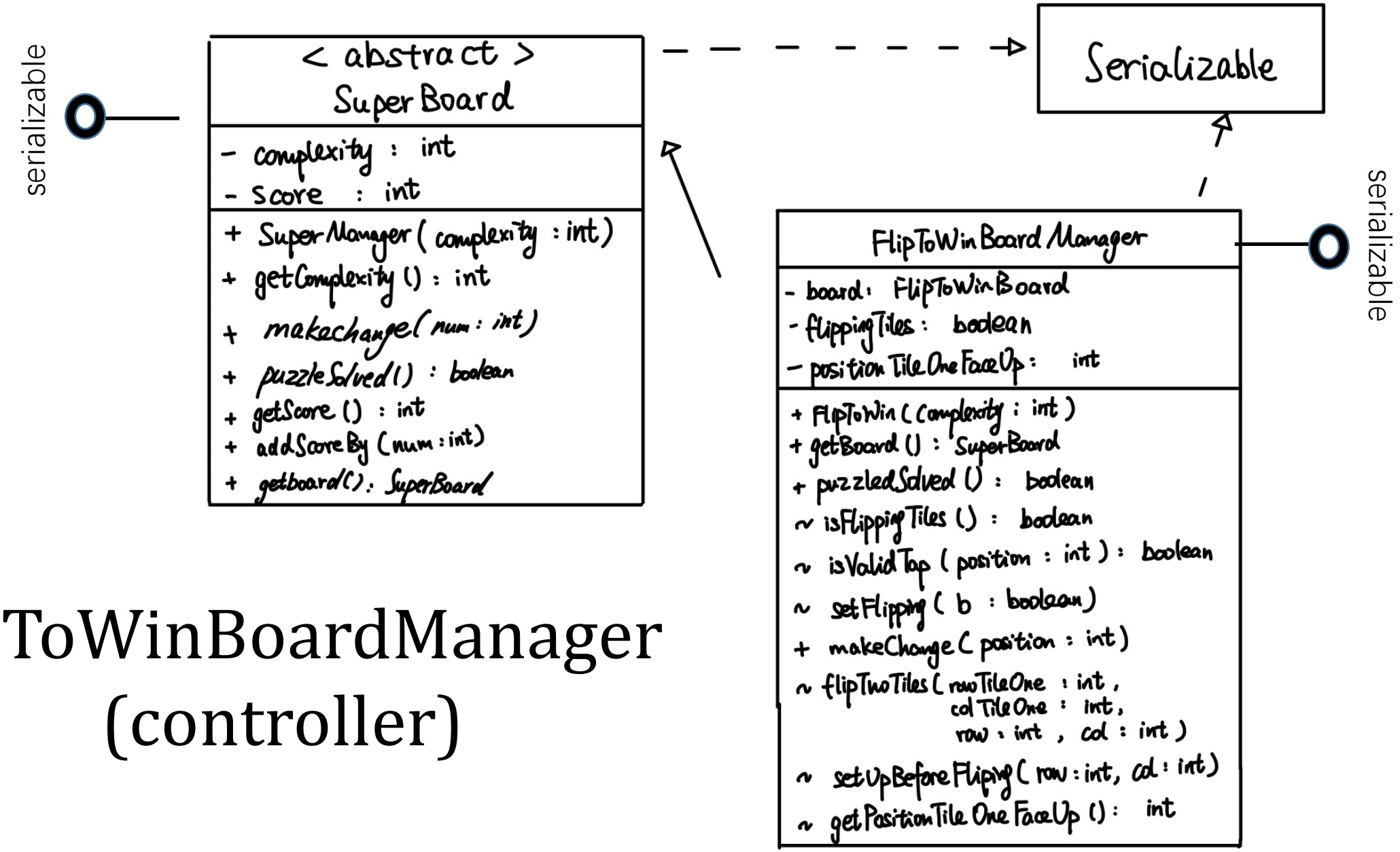
Design pattern: Observer

In FlipToWinGameActivity we use addObserver(), and In Board Class we setChanged() and notifyObservers() to update changes.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    createTileButtons( context: this);
    setContentView(R.layout.activity_flip_to_win_game);
    // Add View to activity
    flipGridView = findViewById(R.id.FTW_GridView);
    flipGridView.setNumColumns(((FlipToWinBoard) DataManager.INSTANCE.getBoardManager().getBoard()).getColNum());
    DataManager.INSTANCE.getBoardManager().getBoard().addObserver( o: this);
    // Observer sets up desired dimensions as well as calls our di  ay function
    flipGridView.getViewTreeObserver().addOnGlobalLayoutListener(
            new ViewTreeObserver.OnGlobalLayoutListener() {
                @Override
                public void onGlobalLayout() {
```

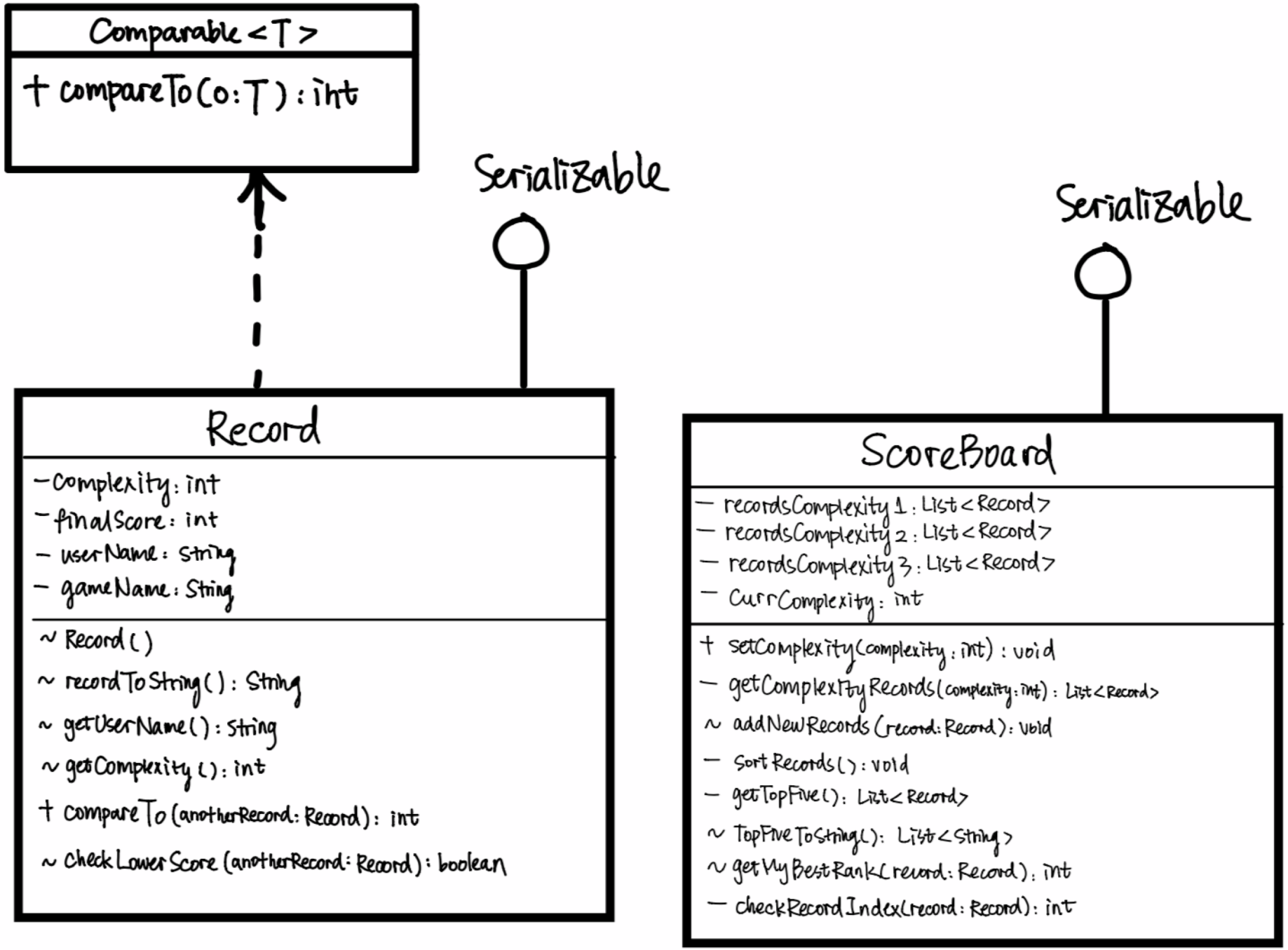
- Class:
  - Board in SlidingTile file
  - FlipToWinBoard in FlipToWin file

```
/**
 * Flip the tile
 */
void makeMove(int row, int col) {
    tiles[row][col].setFlipped();
    setChanged();
    notifyObservers();
}
```



FlipToWinBoardManager (controller)

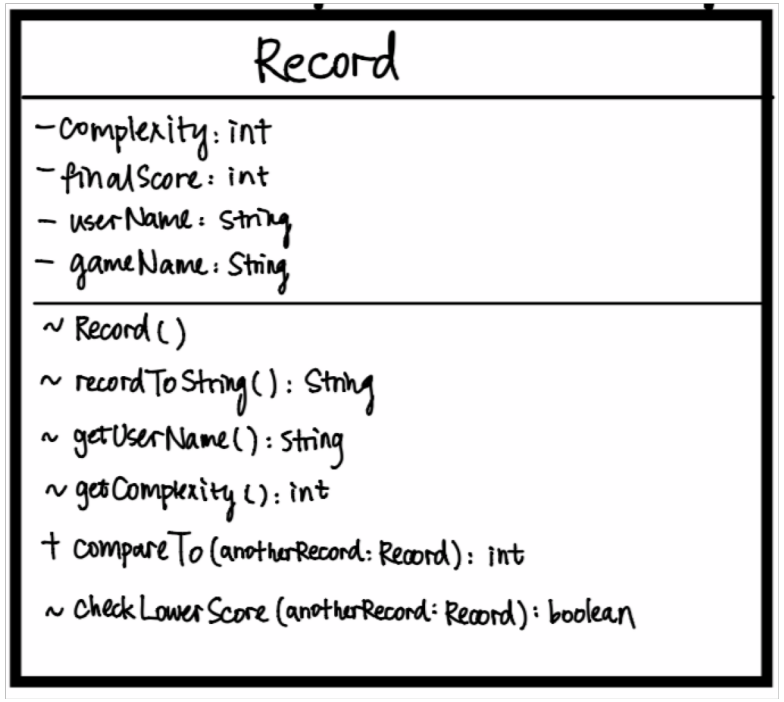
Welcome to Zou Qi !


Comparable<T>
+ compareTo(o:T):int
Serializable
Record
-complexity:int
-finalScore:int
-userName:String
-gameName:String
~Record()
~recordToString():String
~getUserName():String
~getComplexity():int
+compareTo(anotherRecord:Record):int
~checkLowerScore(anotherRecord:Record):boolean
Serializable
ScoreBoard
-recordsComplexity1:List<Record>
-recordsComplexity2:List<Record>
-recordsComplexity3:List<Record>
-currComplexity:int
+setComplexity(complexity:int):void
-getComplexityRecords(complexity:int):List<Record>
~addNewRecords(record:Record):void
-sortRecords():void
-getTopFive():List<Record>
~TopFiveToString():List<String>
~getMyBestRank(record:Record):int
-checkRecordIndex(record:Record):int

Record Class

```
/**
 * Create an instance of Record
 */
Record() {
    this.complexity = DataManager.INSTANCE.getBoardManager().getComplexity();
    this.finalScore = DataManager.INSTANCE.getBoardManager().getScore();
    this.userName = DataManager.INSTANCE.getCurrentUserName();
    this.gameName = DataManager.INSTANCE.getCurrentGameName();
}
```

- Every Record contains information of :
  - which complexity
  - which user
  - which game
  - the final score.



ScoreBoard

```java
/**
 * this is a List of all records of Complexity 1
 */
private List<Record> recordsComplexity1 = new ArrayList<>();

/**
 * this is a List of all records of Complexity 2
 */
private List<Record> recordsComplexity2 = new ArrayList<>();

/**
 * this is a List of all records of Complexity 3
 */
private List<Record> recordsComplexity3 = new ArrayList<>();

/**
 * get the current complexity  the game.
 */
private int currComplexity;
```

- **How are the scores stored?**

- In Scoreboard class, we have 3 List of Record which store Record in 3 ArrayList responding to 3 different complexity.

- We have the currComplexity which is the current complexity.

```java
/**
 * Return return a List records which corresponding to the current game complexity.
 *
 * @param complexity the complexity the game
 * @return return a List which corresponding to the current game complexity.
 */
private List<Record> getComplexityRecords(int complexity) {
    if (complexity == 3) {
        return recordsComplexity1;
    }
    if (complexity == 4) {
        return recordsComplexity2;
    } else {
        return recordsComplexity3;
    }
}
```

ScoreBoard

- ~addNewRecords(Record record): void

- In this method, when we want to add *record* to our List<Record>:

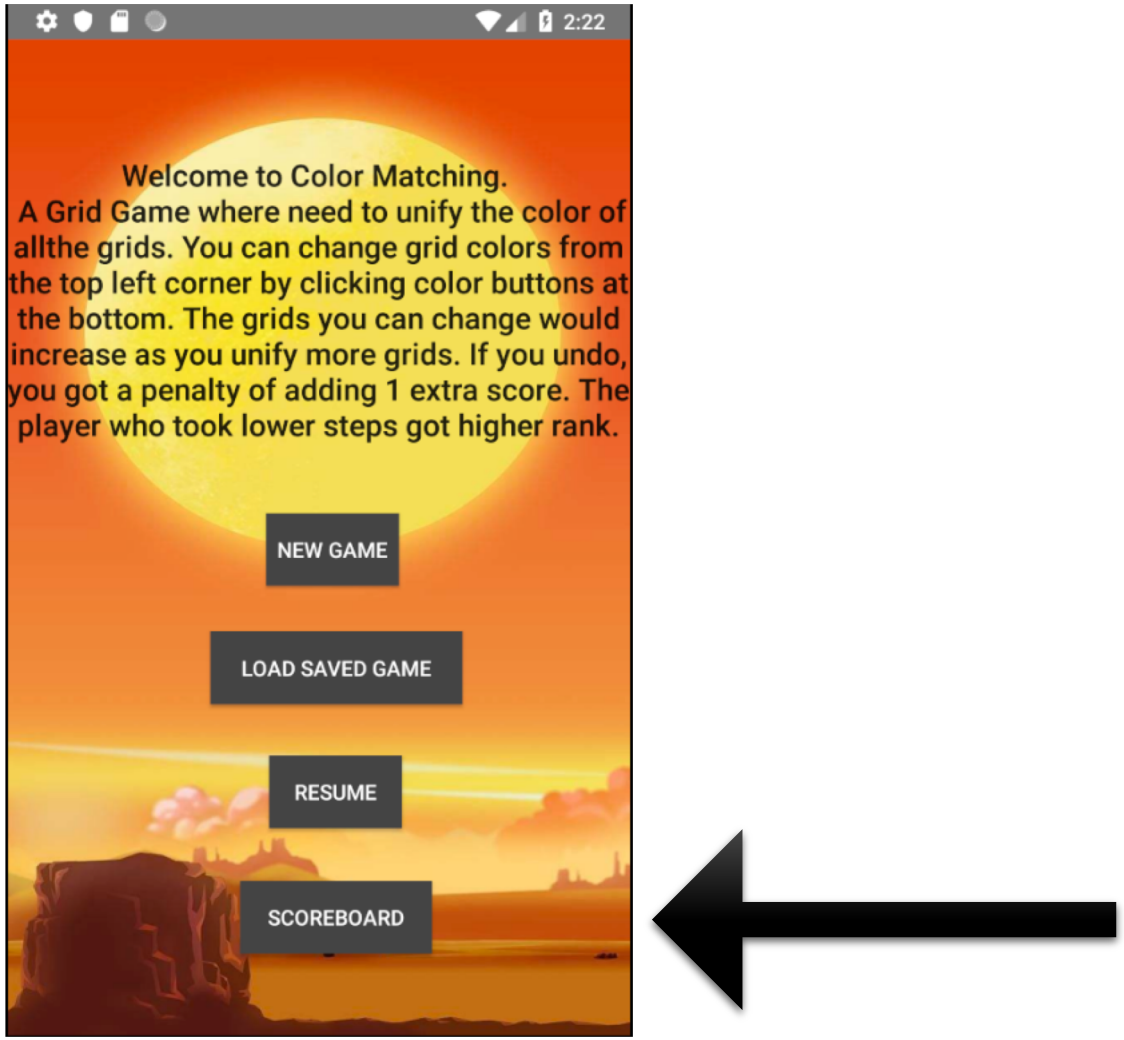
- we first get the complexity from *record* to know which of the 3 Arraylist this *record* should be added to.

- Then we check whether the *record* already exist in our Arraylist or not.
  - ➢ If not, just simply add it;
  - ➢ if already exist, then we need to check whether the score of *record* is lower, if lower then replace the old *record* with the new one. (In our games, one movement responds to 1 score, so the user who took less movement got less score, and will get higher rank. So we check which one is lower)

```java
/**
 * Modify List records which corresponding to the current game complexity(add a record
 * in it)
 *
 * @param record a record
 */
void addNewRecords(Record record) {
    int complexity = record.getComplexity();
    setComplexity(complexity);

    // get the List records which corresponding to the current game
    // complexity(add a record)
    List<Record> currentRecords = getComplexityRecords(currComplexity);

    // add or replace the record
    int temp = checkRecordIndex(record);
    if (temp == -1) {
        currentRecords.add(record);
    } else if (currentRecords.get(temp).checkLowerScore(record)) {
        currentRecords.remove(temp);
        currentRecords.add(record);
    }
    sortRecords();
}
```

ScoreBoard

- There are two ways to access Score Board page:
- one is after the user won the game, it would automatically jump to the Score Board page so you can check your current score and the rank.
- The other is after you choose a game, you will get to this page, and you can press the Score Board button to check the scoreboard.

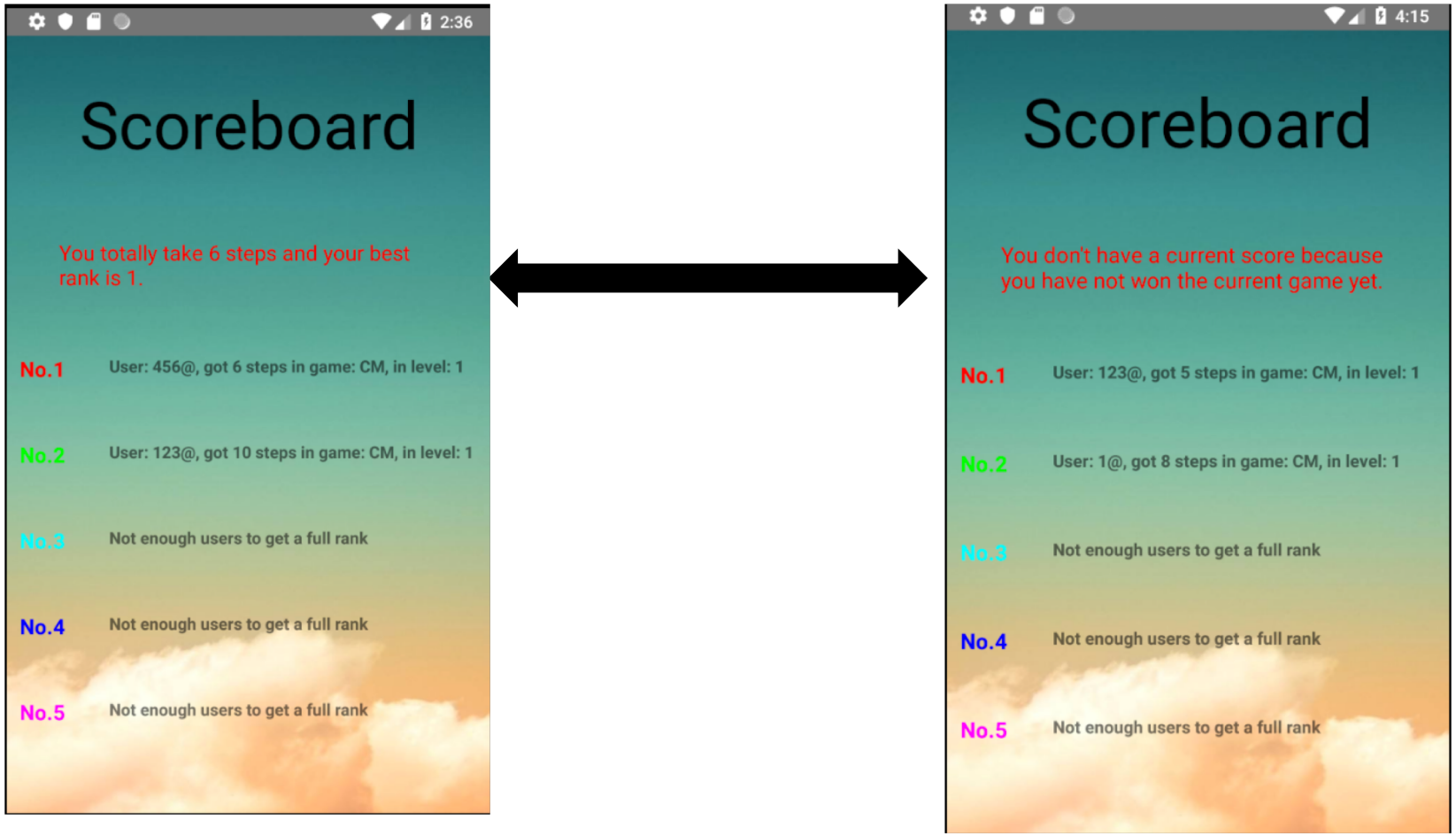


ScoreBoard

```
Intent preIntent = getIntent();
Bundle bundle = preIntent.getExtras();

// if there is Extra message passed.
if (bundle != null) {
    int complexity = bundle.getInt( key: "complexity");

    String noScore = "You don't have a current score because you have not won the current "
            "game yet.";

    myTextView.setText(noScore);
    scoreBoard.setComplexity(complexity);
    this.isOnlyViewScoreBoard = true;
}

// when user won the game.
else {
    Record myRecord = new Record();
    scoreBoard.addNewRecords(myRecord);
    FileManager.saveToFile(this.getApplicationContext(), scoreBoard, type: "SB");

    String myScore = Integer.toString(DataManager.INSTANCE.getBoardManager().getScore());
    int myRank = scoreBoard.getMyBestRank(myRecord);
    String myScoreToString = "You totally take " + myScore + " steps and your best rank is "
            + myRank + ".";

    myTextView.setText(myScoreToString);
    this.isOnlyViewScoreBoard = false;
}
```

- Each complexity of a game have their own scoreboard.
- Thus if the user access Score Board before he/she plays a game, he/she would need to choose which complexity want to look at. To get information of game and complexity, we use putExtras and getExtras, and get information from the bundle, then we can display the corresponding Score Board.
- If bundle == null, then this means the user wins the game, and it is automatically jumped to the score board page, so we display the user’ s current score, as well as the top 5 ranks.

ScoreBoard


2:36
Scoreboard
You totally take 6 steps and your best rank is 1.
No.1 User: 456@, got 6 steps in game: CM, in level: 1
No.2 User: 123@, got 10 steps in game: CM, in level: 1
No.3 Not enough users to get a full rank
No.4 Not enough users to get a full rank
No.5 Not enough users to get a full rank
4:15
Scoreboard
You don't have a current score because you have not won the current game yet.
No.1 User: 123@, got 5 steps in game: CM, in level: 1
No.2 User: 1@, got 8 steps in game: CM, in level: 1
No.3 Not enough users to get a full rank
No.4 Not enough users to get a full rank
No.5 Not enough users to get a full rank

ScoreBoard

```java
/**
 * Return a List that contains the top five record if possible
 *
 * @return a List that contains the top five record if possible
 */
private List<Record> getTopFive() {

    List<Record> result = new ArrayList<>();

    int i = 1;
    while (i <= getComplexityRecords(currComplexity).size() && i <= 5) {
        result.add(getComplexityRecords(currComplexity).get(i - 1));
        i++;
    }
    return result;
}
```

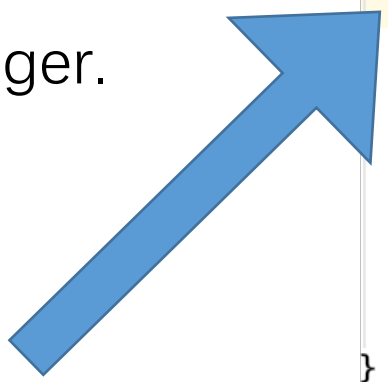
- **Where are the high scores stored?**

➢ Every time we add a new record to ArrayList, we sort the ArrayList.

➢ Then if we want to get our top 5 scores of the current game in current complexity, we just need to get the first 5 index of that list.

ScoreBoard

- **How to store ScoreBoard ?**
- Each game has a scoreboard file.
- ’ Save’ action is taken in FileManager.



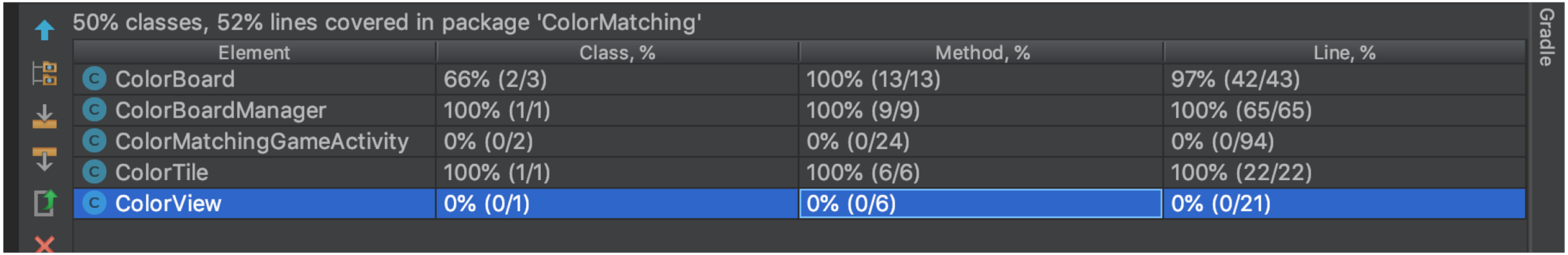
```java
/**
 * Save the given object to file.
 * <p>
 * Precondition: the type is either "UM" or "SB"
 *
 * @param fileContext this.getApplicationContext()
 * @param object      the object that will be saved
 * @param type        the type of the object.
 */
public static void saveToFile(Context fileContext, Object object, String type) {
    try {
        String fileName;
        if ("UM".equals(type)) {
            fileName = "UserManager.ser";
        } else {
            fileName = DataManager.INSTANCE.getCurrentGameName() + "_ScoreBoard.ser";
        }
        ObjectOutputStream outputStream = new ObjectOutputStream(
                fileContext.openFileOutput(fileName, Context.MODE_PRIVATE));
        outputStream.writeObject(object);
        outputStream.close();
    } catch (IOException e) {
        Log.e( tag: "Exception",  msg: "File write failed: " + e.toString());
    }
}
```

ScoreBoard

- **How are they get displayed?**
- TopFiveToString() method returns a List of String, which is an ArrayList of the message going to be displayed.
- If there are less than 5 players who have played the current game in the current complexity, then the message would be: “Not enough users to get a full rank.”
- In ScoreBoardActivity, we get the String from TopFiveToString() and display the messages in order.

```
/**
 * Return a List that contains the String type of the top five record if possible.
 *
 * @return Return a List that contains the String type of the top five record if possible,
 * if not possible then show "Not enough users to get a fully rank" instead.
 */
List TopFiveToString() {
    List<Record> topFive = getTopFive();
    List<String> result = new ArrayList<>();

    for (Record r : topFive) {
        result.add(r.recordToString());
    }

    if (result.size() < 5) {
        int diff = 5 - result.size();
        while (diff > 0) {
            String temp = "Not enough users to get a full rank";
            result.add(temp);
            diff--;
        }
    }
    return result;
}
```

Unit Test Time



The ColorBoardManager unit test got 100% line coverage, since every method in ColorBoardManage focuses on managing the ColorBoard and it is a controller.

This is an activity class with view elements in it, so it's hard or impossible to test.
This class contains update-buttons methods and non-logical methods.
So it has 0% line coverage.


FreeWallpapersHd.net
Thank
you!