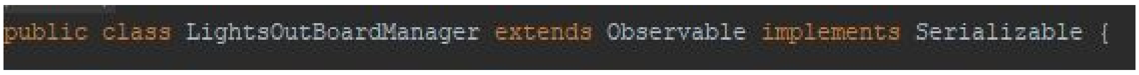
Design patterns

(1) observer

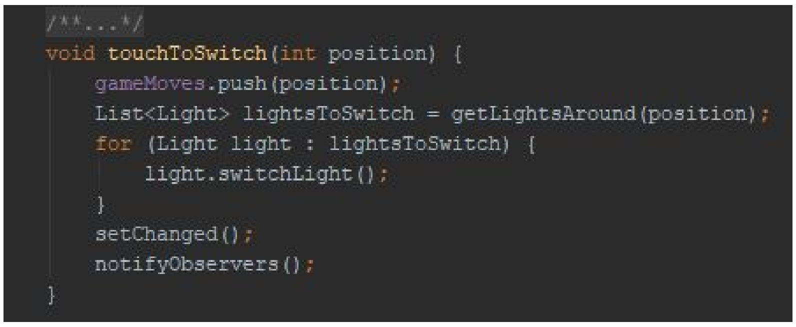
there are 2 part we use observer

(1) we use it in all button click.

(2) we use it to observe the change of the game board, in order to update the view. .

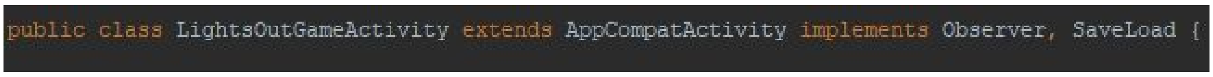


```
public class LightsOutBoardManager extends Observable implements Serializable {
```

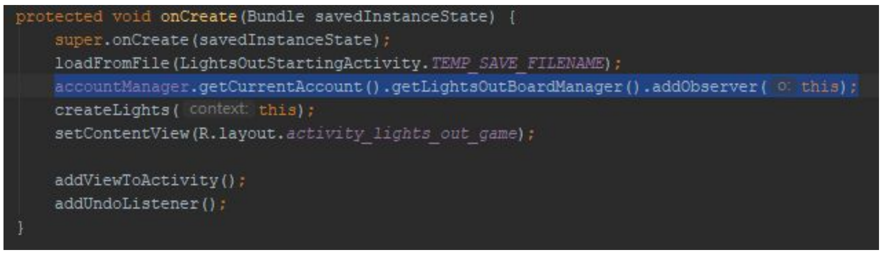


```
/**...*/
void touchToSwitch(int position) {
    gameMoves.push(position);
    List<Light> lightsToSwitch = getLightsAround(position);
    for (Light light : lightsToSwitch) {
        light.switchLight();
    }
    setChanged();
    notifyObservers();
}
```

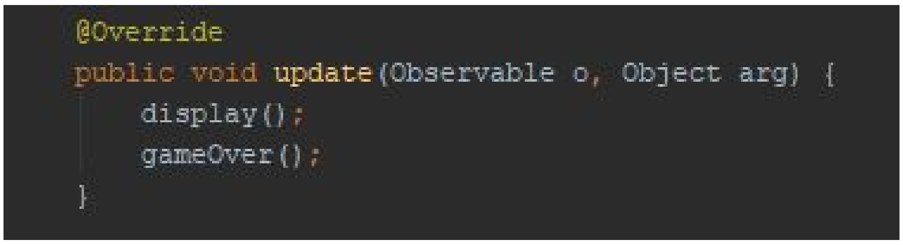
Observer In BoardManager



```
public class LightsOutGameActivity extends AppCompatActivity implements Observer, SaveLoad {
```



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    loadFromFile(LightsOutStartingActivity.TEMP_SAVE_FILENAME);
    accountManager.getCurrentAccount().getLightsOutBoardManager().addObserver( o: this);
    createLights( context: this);
    setContentView(R.layout.activity_lights_out_game);

    addViewToActivity();
    addUndoListener();
}
```



```
@Override
public void update(Observable o, Object arg) {
    display();
    gameOver();
}
```

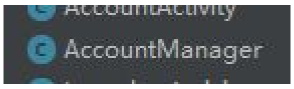
In GameActivity

(2) mvc

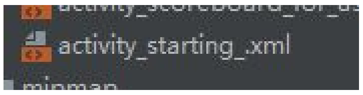
mvc conncet the module with view and controller

.module

Account



AccountManager

.view 

activity_starting_.xml

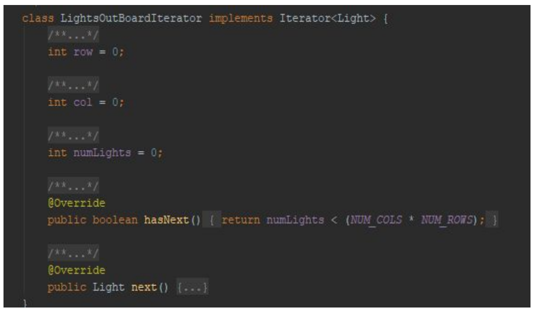
.controller

StartingActivity

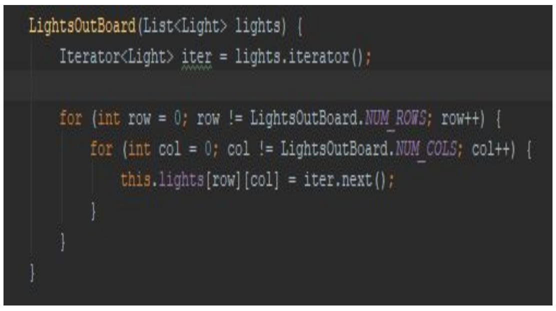
when it click the button on view, controller would receive the click instruction by using observer. then controller would process the information that needed in module and feedback to view

iterator

we use iterator to loop over the game board and put tiles/lights in it.



```java
class LightsOutBoardIterator implements Iterator<Light> {
    /**...*/
    int row = 0;

    /**...*/
    int col = 0;

    /**...*/
    int numLights = 0;

    /**...*/
    @Override
    public boolean hasNext() { return numLights < (NUM_COLS * NUM_ROWS); }

    /**...*/
    @Override
    public Light next() {...}
}
```



```java
LightsOutBoard(List<Light> lights) {
    Iterator<Light> iter = lights.iterator();


    for (int row = 0; row != LightsOutBoard.NUM_ROWS; row++) {
        for (int col = 0; col != LightsOutBoard.NUM_COLS; col++) {
            this.lights[row][col] = iter.next();
        }
    }
}
```

important classes

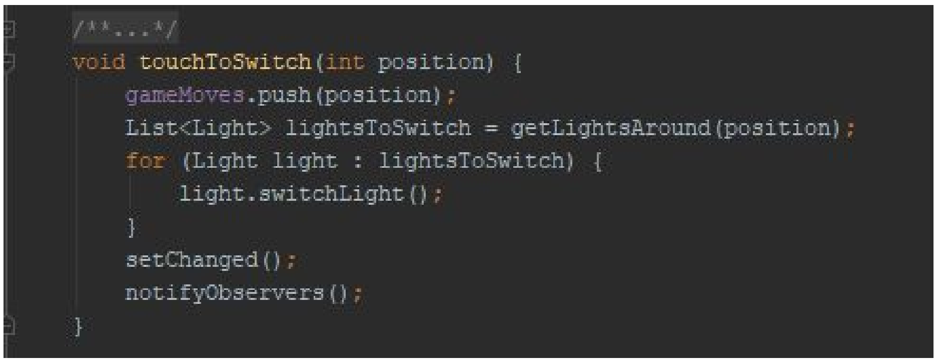
(1) accountManager

this class contains all the information that we need for game (some in account class but also store in accountManager) . we save it in order to save game states. and set the currentUser who are using the game centre by login. the functionality of save and account system is base on the class accountManager

(2) BoardManager/LightsOutBoardManager

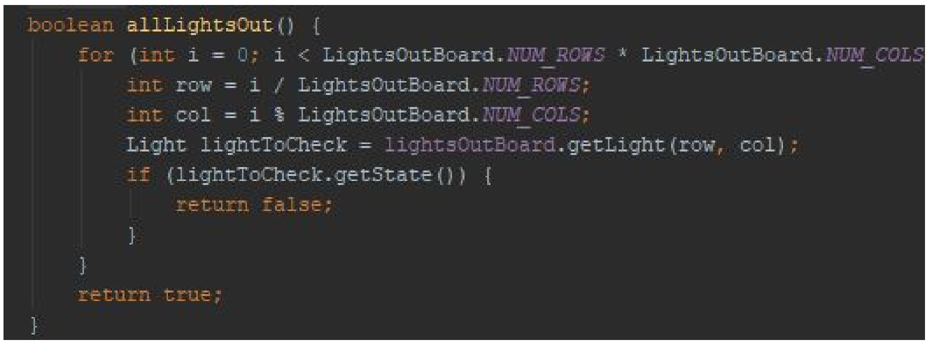
this class contains methods and java codes for actual game.It also contains method to count the score and initializer for Board, which is called when we start playing the game. Here are some really important methods.

the methods used for game play: ie isValidTap, touchMove, touchToSwitch



```java
    /**...*/
    void touchToSwitch(int position) {
        gameMoves.push(position);
        List<Light> lightsToSwitch = getLightsAround(position);
        for (Light light : lightsToSwitch) {
            light.switchLight();
        }
        setChanged();
        notifyObservers();
    }
```

the methods determine the statement of game: allLightsOut, PuzzleSoveld

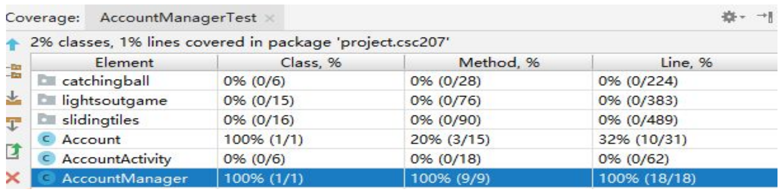


```java
boolean allLightsOut() {
    for (int i = 0; i < LightsOutBoard.NUM_ROWS * LightsOutBoard.NUM_COLS
        int row = i / LightsOutBoard.NUM_ROWS;
        int col = i % LightsOutBoard.NUM_COLS;
        Light lightToCheck = lightsOutBoard.getLight(row, col);
        if (lightToCheck.getState()) {
            return false;
        }
    }
    return true;
}
```

Unit Test

classes with 100% code coverage: Lights,AccountManager,lightsOutBoard…..

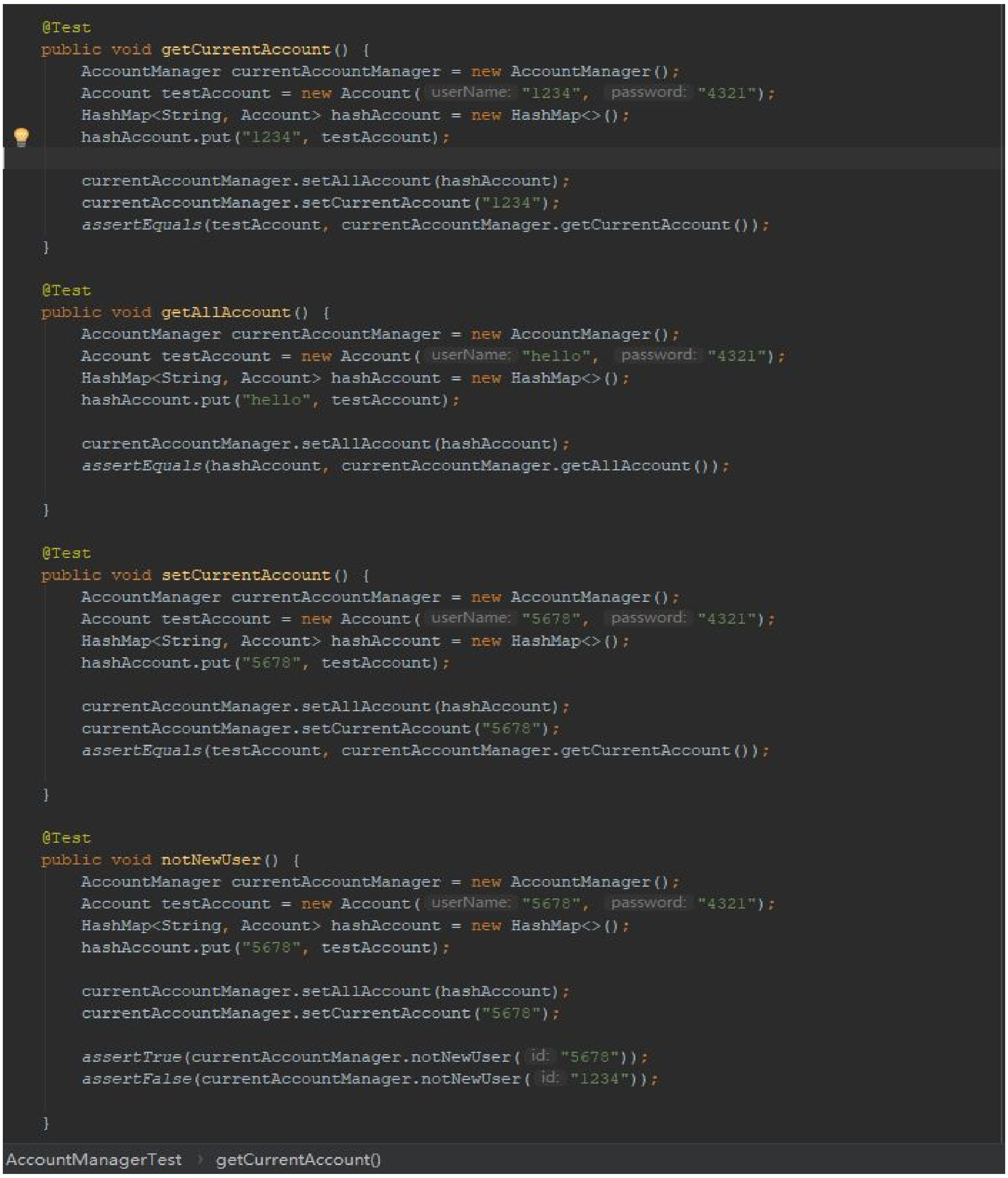
unit test to show: AccountManager:



Coverage: AccountManagerTest

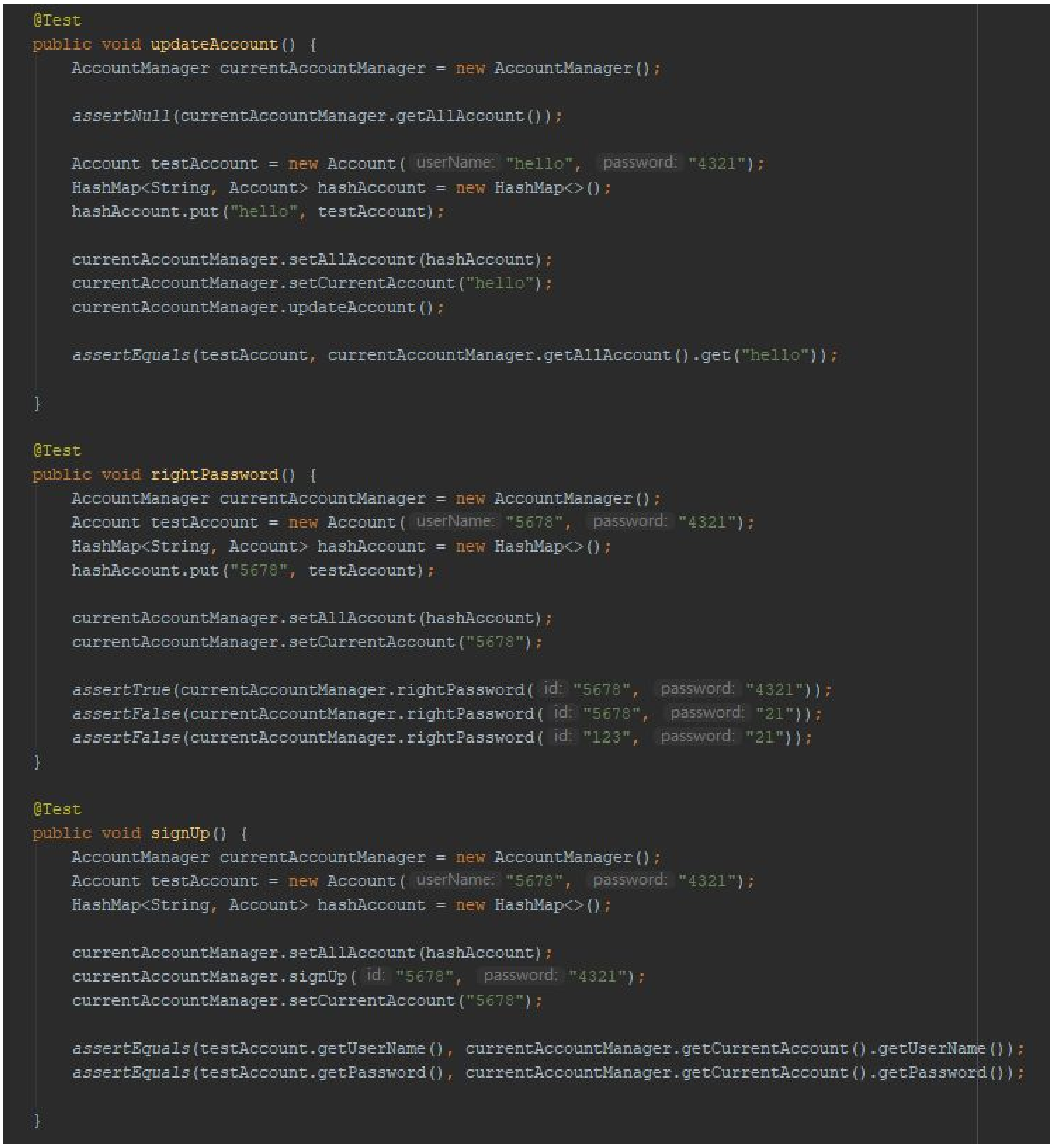
2% classes, 1% lines covered in package 'project.csc207'

Element	Class, %	Method, %	Line, %
catchingball	0% (0/6)	0% (0/28)	0% (0/224)
lightsoutgame	0% (0/15)	0% (0/76)	0% (0/383)
slidingtiles	0% (0/16)	0% (0/90)	0% (0/489)
Account	100% (1/1)	20% (3/15)	32% (10/31)
AccountActivity	0% (0/6)	0% (0/18)	0% (0/62)
AccountManager	100% (1/1)	100% (9/9)	100% (18/18)

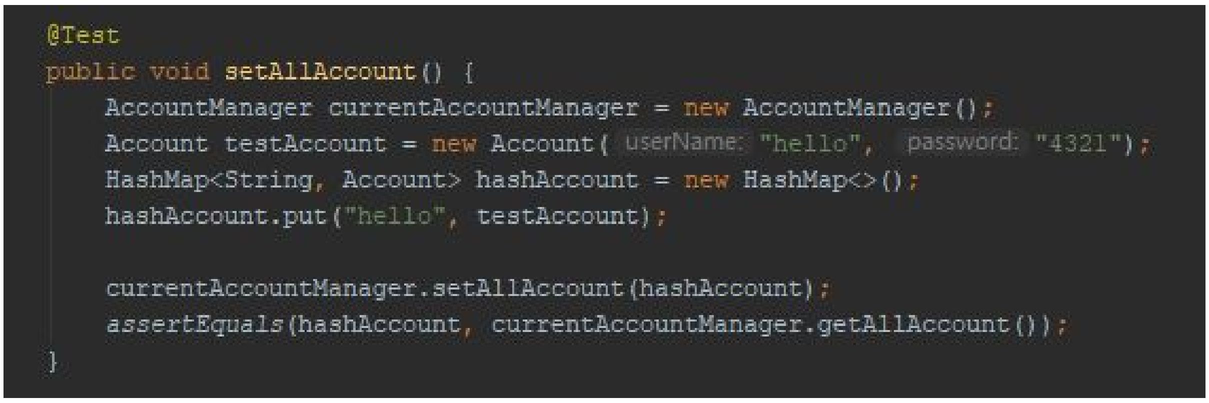


```
@Test
public void getCurrentAccount() {
    AccountManager currentAccountManager = new AccountManager();
    Account testAccount = new Account( userName: "1234", password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();
    hashAccount.put("1234", testAccount);

    currentAccountManager.setAllAccount(hashAccount);
    currentAccountManager.setCurrentAccount("1234");
    assertEquals(testAccount, currentAccountManager.getCurrentAccount());
}

@Test
public void getAllAccount() {
    AccountManager currentAccountManager = new AccountManager();
    Account testAccount = new Account( userName: "hello", password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();
    hashAccount.put("hello", testAccount);

    currentAccountManager.setAllAccount(hashAccount);
    assertEquals(hashAccount, currentAccountManager.getAllAccount());

}

@Test
public void setCurrentAccount() {
    AccountManager currentAccountManager = new AccountManager();
    Account testAccount = new Account( userName: "5678", password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();
    hashAccount.put("5678", testAccount);

    currentAccountManager.setAllAccount(hashAccount);
    currentAccountManager.setCurrentAccount("5678");
    assertEquals(testAccount, currentAccountManager.getCurrentAccount());

}

@Test
public void notNewUser() {
    AccountManager currentAccountManager = new AccountManager();
    Account testAccount = new Account( userName: "5678", password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();
    hashAccount.put("5678", testAccount);

    currentAccountManager.setAllAccount(hashAccount);
    currentAccountManager.setCurrentAccount("5678");

    assertTrue(currentAccountManager.notNewUser( id: "5678"));
    assertFalse(currentAccountManager.notNewUser( id: "1234"));

}
```

AccountManagerTest › getCurrentAccount()

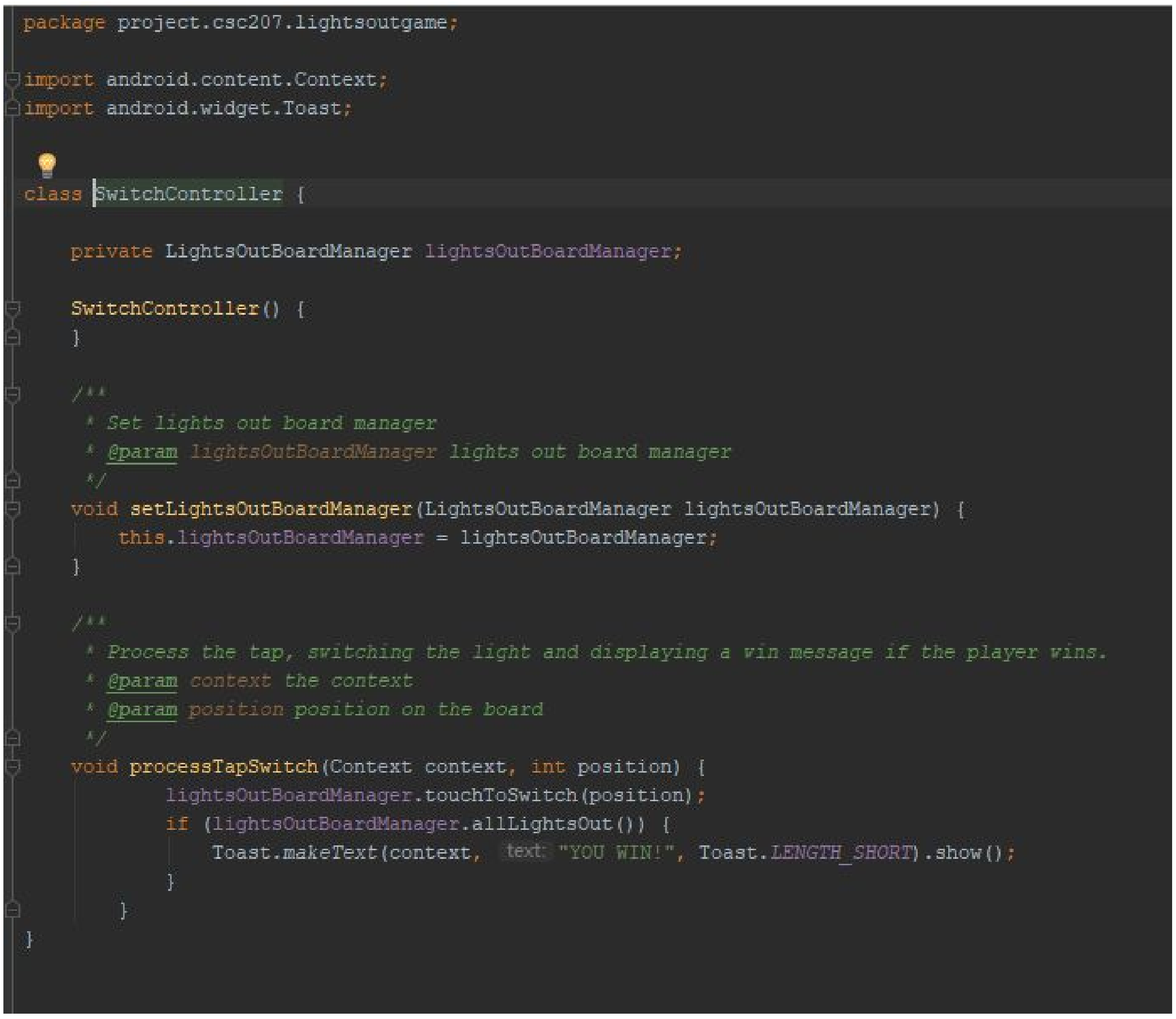


```java
@Test
public void updateAccount() {
    AccountManager currentAccountManager = new AccountManager();

    assertNull(currentAccountManager.getAllAccount());

    Account testAccount = new Account( userName: "hello",  password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();
    hashAccount.put("hello", testAccount);

    currentAccountManager.setAllAccount(hashAccount);
    currentAccountManager.setCurrentAccount("hello");
    currentAccountManager.updateAccount();

    assertEquals(testAccount, currentAccountManager.getAllAccount().get("hello"));

}

@Test
public void rightPassword() {
    AccountManager currentAccountManager = new AccountManager();
    Account testAccount = new Account( userName: "5678",  password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();
    hashAccount.put("5678", testAccount);

    currentAccountManager.setAllAccount(hashAccount);
    currentAccountManager.setCurrentAccount("5678");

    assertTrue(currentAccountManager.rightPassword( id: "5678",  password: "4321"));
    assertFalse(currentAccountManager.rightPassword( id: "5678",  password: "21"));
    assertFalse(currentAccountManager.rightPassword( id: "123",  password: "21"));
}

@Test
public void signUp() {
    AccountManager currentAccountManager = new AccountManager();
    Account testAccount = new Account( userName: "5678",  password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();

    currentAccountManager.setAllAccount(hashAccount);
    currentAccountManager.signUp( id: "5678",  password: "4321");
    currentAccountManager.setCurrentAccount("5678");

    assertEquals(testAccount.getUserName(), currentAccountManager.getCurrentAccount().getUserName());
    assertEquals(testAccount.getPassword(), currentAccountManager.getCurrentAccount().getPassword());

}
```

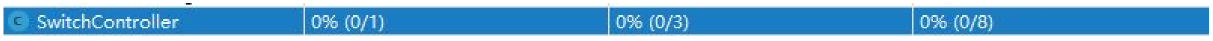


```java
@Test
public void setAllAccount() {
    AccountManager currentAccountManager = new AccountManager();
    Account testAccount = new Account( userName: "hello",  password: "4321");
    HashMap<String, Account> hashAccount = new HashMap<>();
    hashAccount.put("hello", testAccount);

    currentAccountManager.setAllAccount(hashAccount);
    assertEquals(hashAccount, currentAccountManager.getAllAccount());
}
```

Example For 0% Test Coverage:



```java
package project.csc207.lightsoutgame;

import android.content.Context;
import android.widget.Toast;

class SwitchController {

    private LightsOutBoardManager lightsOutBoardManager;

    SwitchController() {
    }

    /**
     * Set lights out board manager
     * @param lightsOutBoardManager lights out board manager
     */
    void setLightsOutBoardManager(LightsOutBoardManager lightsOutBoardManager) {
        this.lightsOutBoardManager = lightsOutBoardManager;
    }

    /**
     * Process the tap, switching the light and displaying a win message if the player wins.
     * @param context the context
     * @param position position on the board
     */
    void processTapSwitch(Context context, int position) {
            lightsOutBoardManager.touchToSwitch(position);
            if (lightsOutBoardManager.allLightsOut()) {
                Toast.makeText(context,  text: "YOU WIN!", Toast.LENGTH_SHORT).show();
            }
        }
}
```



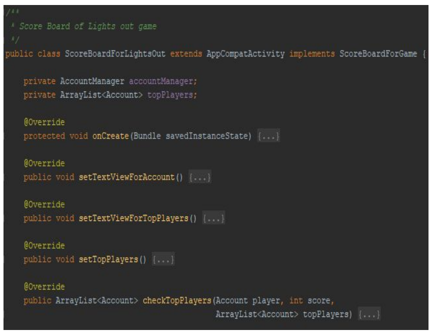
SwitchController	0% (0/1)	0% (0/3)	0% (0/8)

ScoreBoards

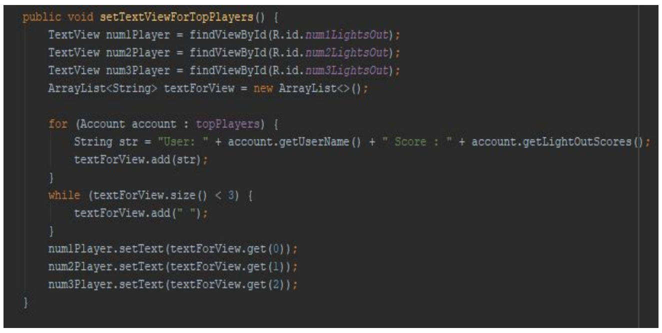
ScoreBoard per Game ScoreBoard per User ScoreResult

Score Board per game

Each Game has its own ScoreBoard, so three ScoreBoard per game in total



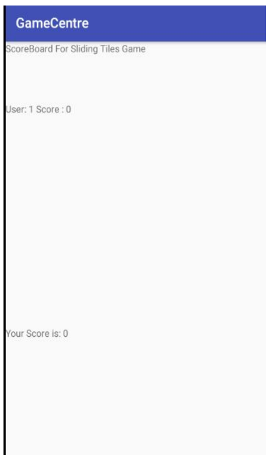
```
/**
 * Score Board of Lights out game
 */
public class ScoreBoardForLightsOut extends AppCompatActivity implements ScoreBoardForGame {

    private AccountManager accountManager;
    private ArrayList<Account> topPlayers;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    public void setTextViewForAccount() {...}

    @Override
    public void setTextViewForTopPlayers() {...}

    @Override
    public void setTopPlayers() {...}

    @Override
    public ArrayList<Account> checkTopPlayers(Account player, int score,
                                              ArrayList<Account> topPlayers) {...}
```



```
public void setTextViewForTopPlayers() {
    TextView num1Player = findViewById(R.id.num1LightsOut);
    TextView num2Player = findViewById(R.id.num2LightsOut);
    TextView num3Player = findViewById(R.id.num3LightsOut);
    ArrayList<String> textForView = new ArrayList<>();

    for (Account account : topPlayers) {
        String str = "User: " + account.getUserName() + " Score : " + account.getLightOutScores();
        textForView.add(str);
    }
    while (textForView.size() < 3) {
        textForView.add(" ");
    }
    num1Player.setText(textForView.get(0));
    num2Player.setText(textForView.get(1));
    num3Player.setText(textForView.get(2));
}
```

For Score Board per game

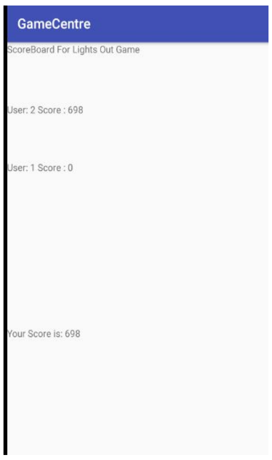
It will Show the Top 3 Players With highest Score of this kind game and the highest score of the current Account, access in Starting activity



ScoreBoard For Sliding Tiles Game

User: 1 Score : 0

Your Score is: 0

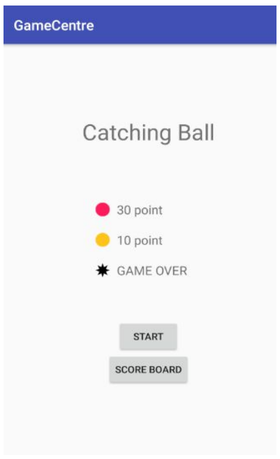


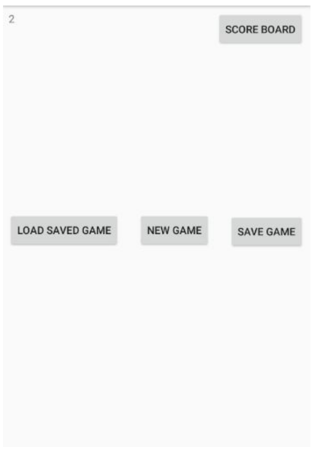
ScoreBoard For Lights Out Game

User: 2 Score : 698

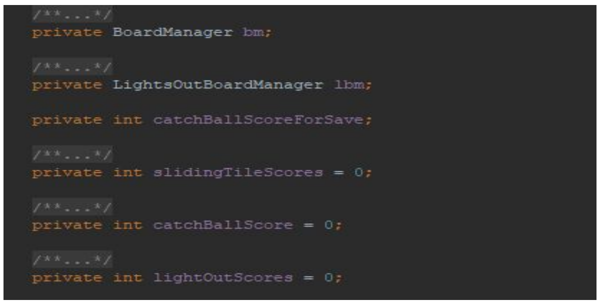
User: 1 Score : 0

Your Score is: 698

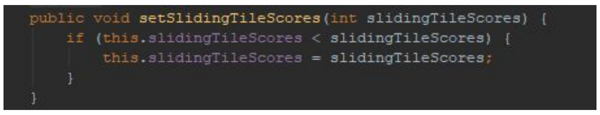




The Scores are stored in Account with private type and getter and setter,set 0 by default, the setter set the score as the record if the given score is higher than the record. (Code in Account.class)

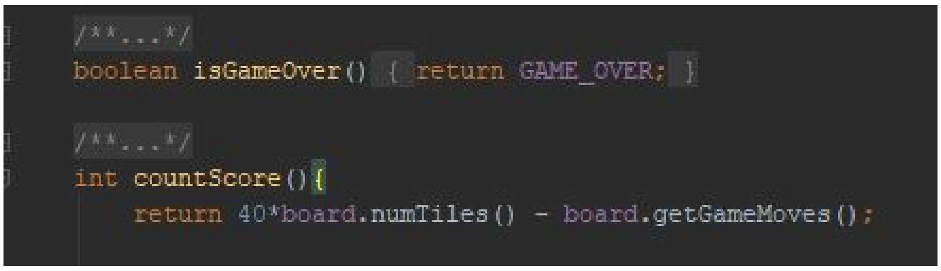


```
/**...*/
private BoardManager bm;

/**...*/
private LightsOutBoardManager lbm;

private int catchBallScoreForSave;

/**...*/
private int slidingTileScores = 0;

/**...*/
private int catchBallScore = 0;

/**...*/
private int lightOutScores = 0;
```



```
public void setSlidingTileScores(int slidingTileScores) {
    if (this.slidingTileScores < slidingTileScores) {
        this.slidingTileScores = slidingTileScores;
    }
}
```

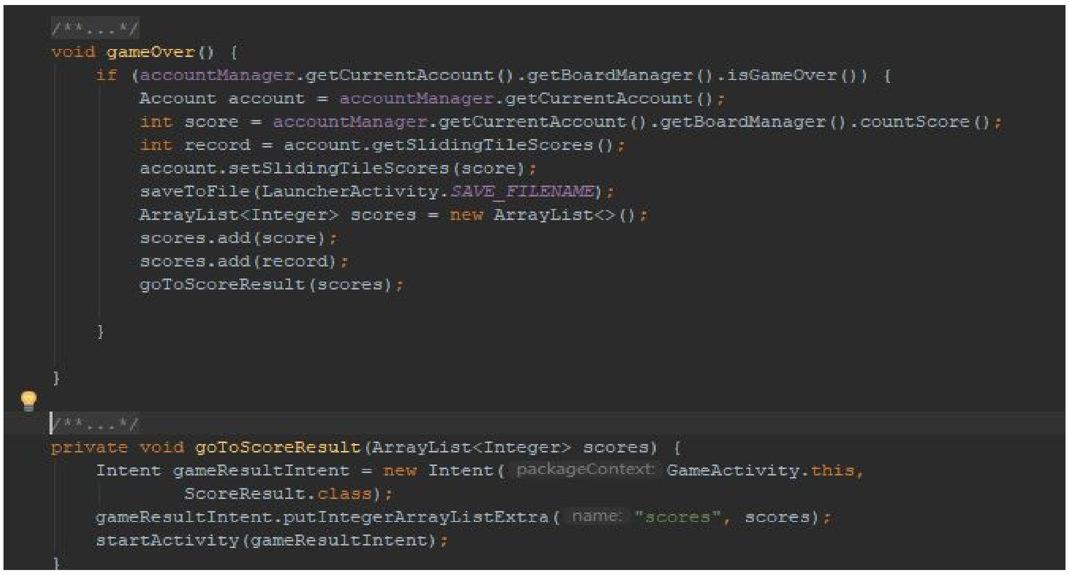
The score counter is located in BoardManager for Sliding Tile and Lights Out, and will be called when the game is over. (Code in LightsOutBoardManager)



```java
/**...*/
boolean isGameOver() { return GAME_OVER; }

/**...*/
int countScore(){
    return 40*board.numTiles() - board.getGameMoves();
```

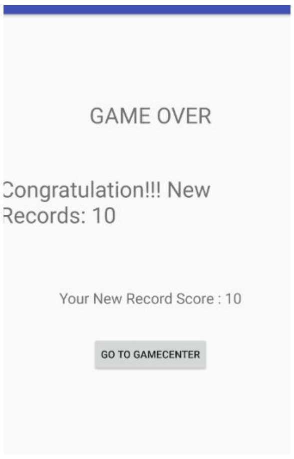
GameActivity will check if the game is over for every move, and if the game is over, it will call the countScore method in corresponding boardmanager to calculate score and then call the setter of this type game's score.

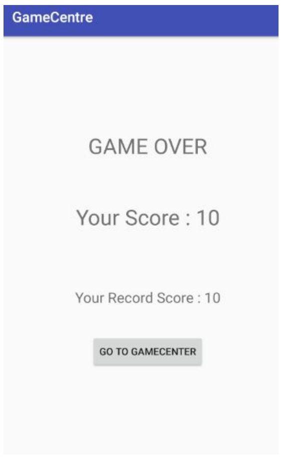
Then go To ScoreResult with score and previous records in an arraylist by goToScoreResult method.

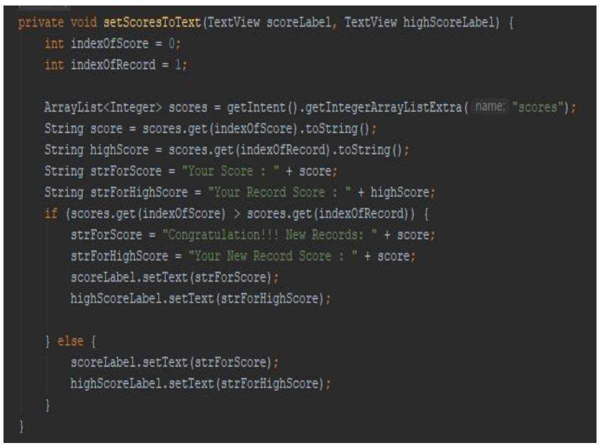


```java
/**...*/
void gameOver() {
    if (accountManager.getCurrentAccount().getBoardManager().isGameOver()) {
        Account account = accountManager.getCurrentAccount();
        int score = accountManager.getCurrentAccount().getBoardManager().countScore();
        int record = account.getSlidingTileScores();
        account.setSlidingTileScores(score);
        saveToFile(LauncherActivity.SAVE_FILENAME);
        ArrayList<Integer> scores = new ArrayList<>();
        scores.add(score);
        scores.add(record);
        goToScoreResult(scores);

    }

}
/**...*/
private void goToScoreResult(ArrayList<Integer> scores) {
    Intent gameResultIntent = new Intent( packageContext: GameActivity.this,
            ScoreResult.class);
    gameResultIntent.putIntegerArrayListExtra( name: "scores", scores);
    startActivity(gameResultIntent);
}
```

ScoreResult
setScoreTotext method takes extra scores and determine what test view to show based on the score and record,if the score is higher than previous record, it will as left, otherwise right. Go to Gamecenter to start new game after game is over



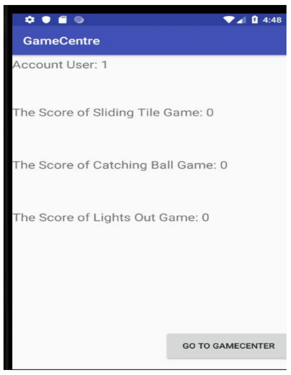




```java
private void setScoresToText(TextView scoreLabel, TextView highScoreLabel) {
    int indexOfScore = 0;
    int indexOfRecord = 1;

    ArrayList<Integer> scores = getIntent().getIntegerArrayListExtra( name: "scores");
    String score = scores.get(indexOfScore).toString();
    String highScore = scores.get(indexOfRecord).toString();
    String strForScore = "Your Score : " + score;
    String strForHighScore = "Your Record Score : " + highScore;
    if (scores.get(indexOfScore) > scores.get(indexOfRecord)) {
        strForScore = "Congratulation!!! New Records: " + score;
        strForHighScore = "Your New Record Score : " + score;
        scoreLabel.setText(strForScore);
        highScoreLabel.setText(strForHighScore);

    } else {
        scoreLabel.setText(strForScore);
        highScoreLabel.setText(strForHighScore);
    }
}
```

ScoreBoard per User

Shows the name of Account user and the score of three different games

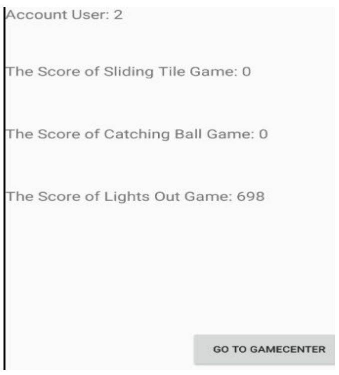


Account User: 1

The Score of Sliding Tile Game: 0

The Score of Catching Ball Game: 0

The Score of Lights Out Game: 0



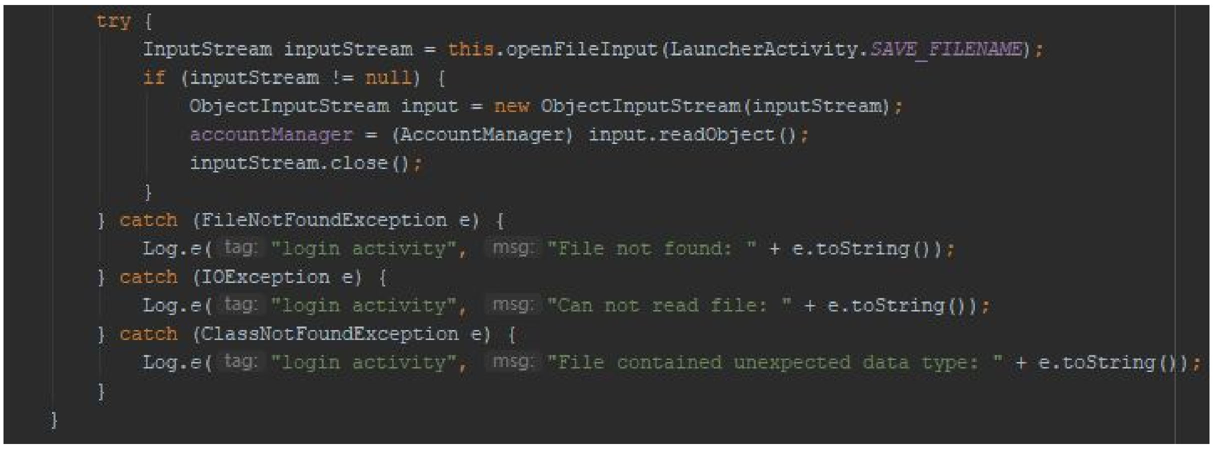
Account User: 2

The Score of Sliding Tile Game: 0

The Score of Catching Ball Game: 0

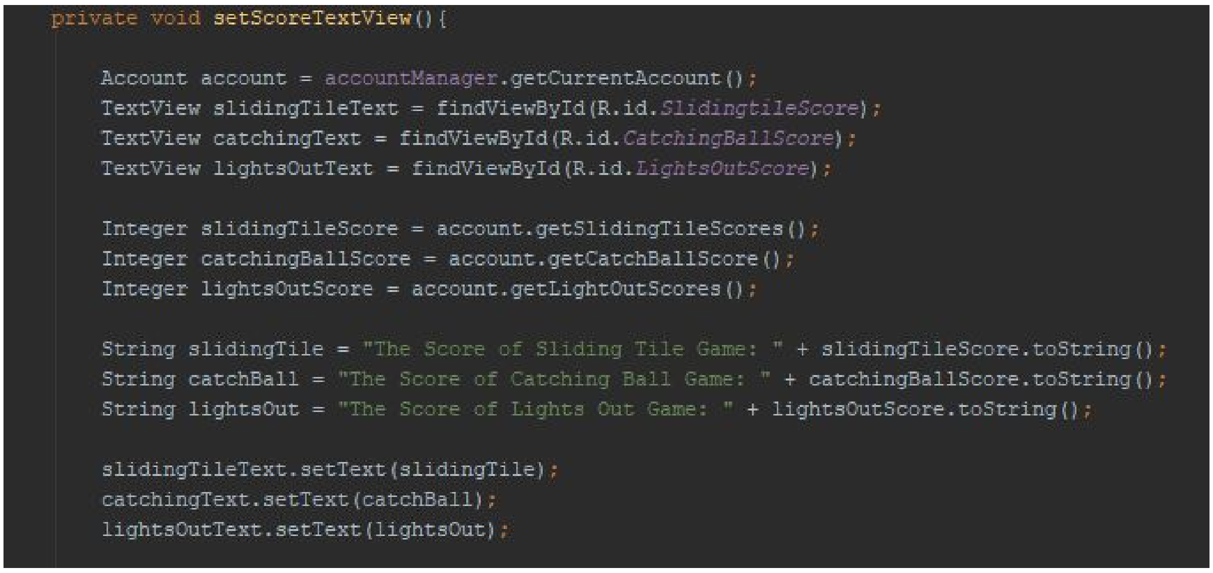
The Score of Lights Out Game: 698

load from file to set accountManager to be the current Account Manager



```
    try {
        InputStream inputStream = this.openFileInput(LauncherActivity.SAVE_FILENAME);
        if (inputStream != null) {
            ObjectInputStream input = new ObjectInputStream(inputStream);
            accountManager = (AccountManager) input.readObject();
            inputStream.close();
        }
    } catch (FileNotFoundException e) {
        Log.e( tag: "login activity",  msg: "File not found: " + e.toString());
    } catch (IOException e) {
        Log.e( tag: "login activity",  msg: "Can not read file: " + e.toString());
    } catch (ClassNotFoundException e) {
        Log.e( tag: "login activity",  msg: "File contained unexpected data type: " + e.toString());
    }
}
```

Get the score of Account by setScoreTextView method, throught getCurrentAccount method of accountManager and getter of scores



```
private void setScoreTextView(){

    Account account = accountManager.getCurrentAccount();
    TextView slidingTileText = findViewById(R.id.SlidingtileScore);
    TextView catchingText = findViewById(R.id.CatchingBallScore);
    TextView lightsOutText = findViewById(R.id.LightsOutScore);

    Integer slidingTileScore = account.getSlidingTileScores();
    Integer catchingBallScore = account.getCatchBallScore();
    Integer lightsOutScore = account.getLightOutScores();

    String slidingTile = "The Score of Sliding Tile Game: " + slidingTileScore.toString();
    String catchBall = "The Score of Catching Ball Game: " + catchingBallScore.toString();
    String lightsOut = "The Score of Lights Out Game: " + lightsOutScore.toString();

    slidingTileText.setText(slidingTile);
    catchingText.setText(catchBall);
    lightsOutText.setText(lightsOut);
```