Threestones

Documentation

Vivek Pandey
SENIOR PROJECT

Threestones is a Human vs. Computer android board game that draws many similarities with Tic-Tac-Toe and Candy Crush and is played with multi-colored stones on a board of 11x11 octagonal squares. Players take turns building three-stone-arrangements of their team mascot to gain points, while blocking their opponents' attempt at building theirs. A large board, some interesting game rules, and, a tricky distribution of stones among both teams makes the game quickly a challenging one.

Contents

1. Introduction

1.1 - General Overview

Threestones is a Human vs. Computer multiplayer android board game that draws many similarities with Tic-Tac-Toe and Candy Crush. It is played with multi-colored stones on an 11x11 octagonal board with 80 pockets. Players take turns building three-stone-arrangements of their team mascot using the various colored stones they own to gain points, while blocking their opponents' attempt at building theirs. Whoever scores the most points by the end wins the game. A large board, some interesting game rules, and, a tricky distribution of stones among both teams makes the game quickly a challenging one.

1.2 - Key Features

The following are some of the key features implemented in the game which definitely added challenges during development but now, upon completion, facilitate a better user experience. Their design and implementation will be touched upon in other sections later in detail.

❖ Simple, intuitive, and modular design

- Modular application design with strict adherence to Model-View-Controller pattern
- An adaptive and responsive interface for devices of various shapes and sizes
- Animations to highlight previous move, recommended move and valid moves
- Notification panel displays log, warnings and error messages in real time

❖ Computer player algorithm

- A Greedy best-first search to find locally optimum solution

❖ "Help Mode" for human player

- Human player can ask for move recommendations at any point in game

❖ Scoring algorithm

- A complex but efficient scoring algorithm to calculate game scores after each move
- Analyzes all possibilities and is well tested not to miscalculate confusing situations

❖ Serialization

- Save and restore a tournament at any point during the game

2. Installation Instructions

2.1 - Method 1 (Recommended)

1) Locate the apk file located in the following location in project folder.

   Project Folder/ThreeStones/app/app-release.apk

2) Open this file in your android device (by emailing it to yourself)

3) The installation will start automatically.

   Note: Your device needs to have "allow installation of apps from sources other than the Play Store" enabled. You can enable this option by navigating to Settings>Security.

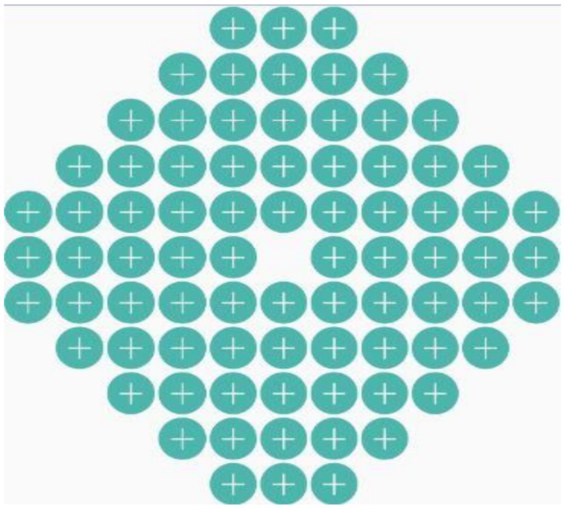
2.2 - Method 2

1) Open Android Studio (or Eclipse equipped with Android Development Tools)
2) Go to File > Open Project and select the main project folder – *Project Folder/ThreeStones*
3) Run using the Emulator (Preferably 6" and over)

3. Game Play 101

3.1 - Board Setup

❖ An empty 11x11 octagonal board



❖ Two players, each assigned with their team mascot during toss.

- *IMPORTANT: In the code, the mascots are named as "white" and "black" to avoid their association with particular objects. BUT the view in the game and this documentation refers to the two primary mascots as "duck" and "penguin" for ease and fun.*

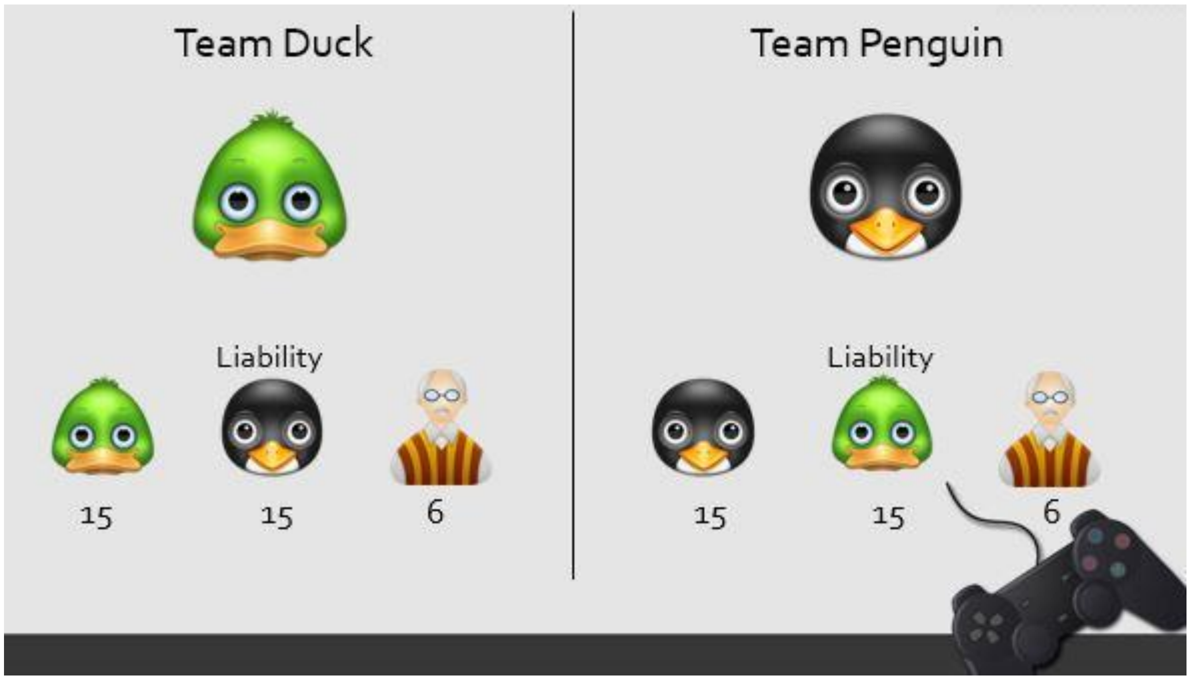
❖ Each player is initially allocated a stone basket with:

- 15 stones stamped with duck mascot
- 15 stones stamped with penguin mascot

- NEWS FLASH: There's a third bonus mascot in the game who works for both the teams. It's the Miller mascot. Everyone loves him.
    - *IMPORTANT: In the code, this mascot is referred to as "clear" to avoid association with objects, but in the view and this documentation, it will be called the "Miller" mascot. This mascot is a common mascot shared by both the teams*.

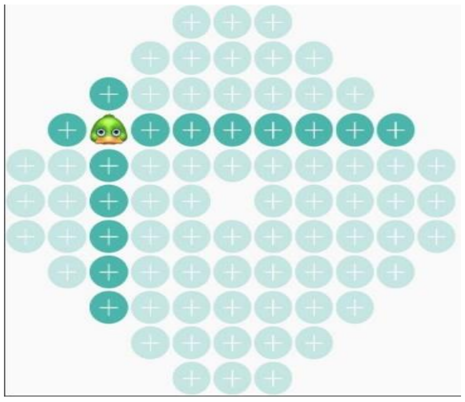
- AND both teams get 6 additional stones each stamped with Miller mascot.

In short,



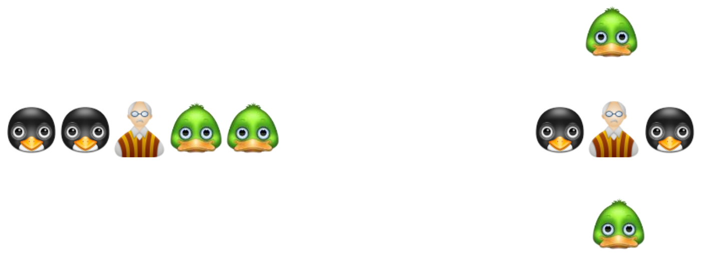
3.2 - How to Play

- **Toss:** Initially, the computer does a coin toss and determines who plays first.
- **Picking mascot:** Upon toss, the human player can pick a preferred mascot.
- **A move:** The player places one of his stones in an empty pocket on the board
    - **Valid move:**
        - If this is the first move, the player can place it anywhere on the board.
        - Otherwise, the player must place the stone into an empty pocket either in the same row or in the same column as the opponent's last move (but not necessarily adjacent to the stone last placed by the opponent).



            - If no pocket is available on the row and column of the opponent's last move, the stone can be placed into any empty pocket on the board.
    - **Objective:** The object of each move is to earn points.
        - The player earns point by placing 3 stones in adjacent pockets.
            - The stones may be in a row, column or diagonal
            - The stones must all be of the player's team mascot or Miller mascot.

        - Miller stones count for either player. As a result, when a clear stone is placed by either player, both players may get a point each, as in the following scenarios:



        - But, a row of 3 Miller stones will not count for either player
    - A stone may be part of more than one scoring arrangement, e.g., each of the following arrangements of duck stones earns two points.

    - After each move, the score card will be updated. Only the new 3-stone arrangements that resulted from the move will be added.
- **Game over:** The game ends when the last available stone is placed.
- **Winning:** When the game ends, the player with the most points wins. If both the players have the same number of points, the game is a draw.

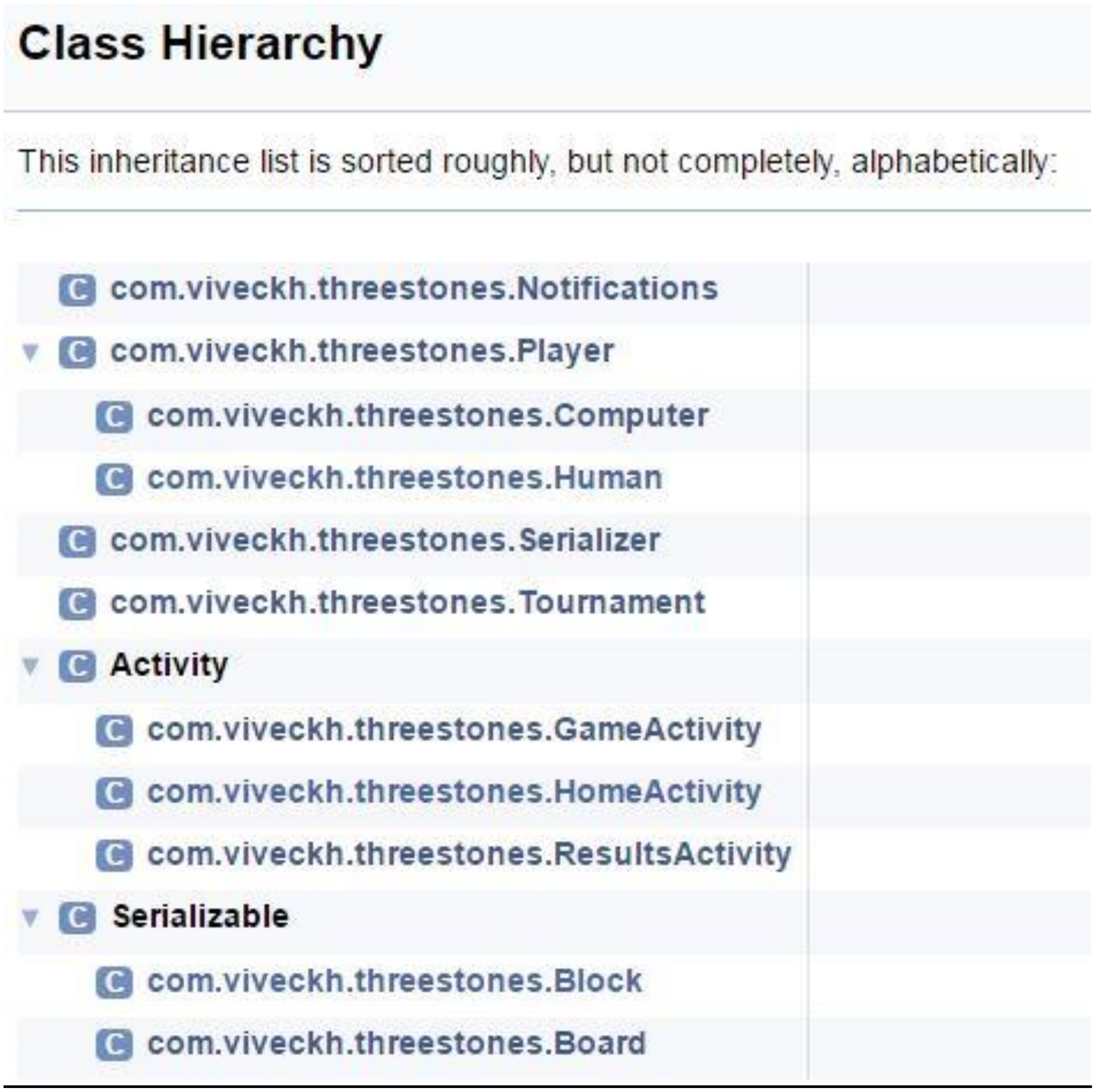
3.3 - Strategy

On every move, the player must determine:

1) **Picking a stone:** Whether to move a stone stamped with own mascot, opponent's mascot, or common Miller mascot.
2) **Placing stone with own mascot:** Player may want to place a stone with own mascot to
    a. Build towards a 3-stone arrangement to gain points.
        i. If the player has more than one incomplete arrangement, complete the one that earn the most points.
    b. Block the opponent from completing a 3-stone arrangement.
3) **Placing stone with opponent mascot:** Player may want to place a stone with opponent's mascot so that the opponent cannot co-opt the stone to earn points, such as by placing the stone in a pocket surrounded by one's own stones. These stones are a burden which the player needs to get rid of carefully so as not to help the opponent towards their 3-stone arrangement.
4) **Playing common Miller stones**
    a. Player may want to hold on to Miller stones and use them only after running out of his own stones.
    b. Player may want to avoid playing a Miller stone when it may also benefit the opponent, as in examples illustrated earlier.

4. Design Diagrams

4.1 - Hierarchy Tree

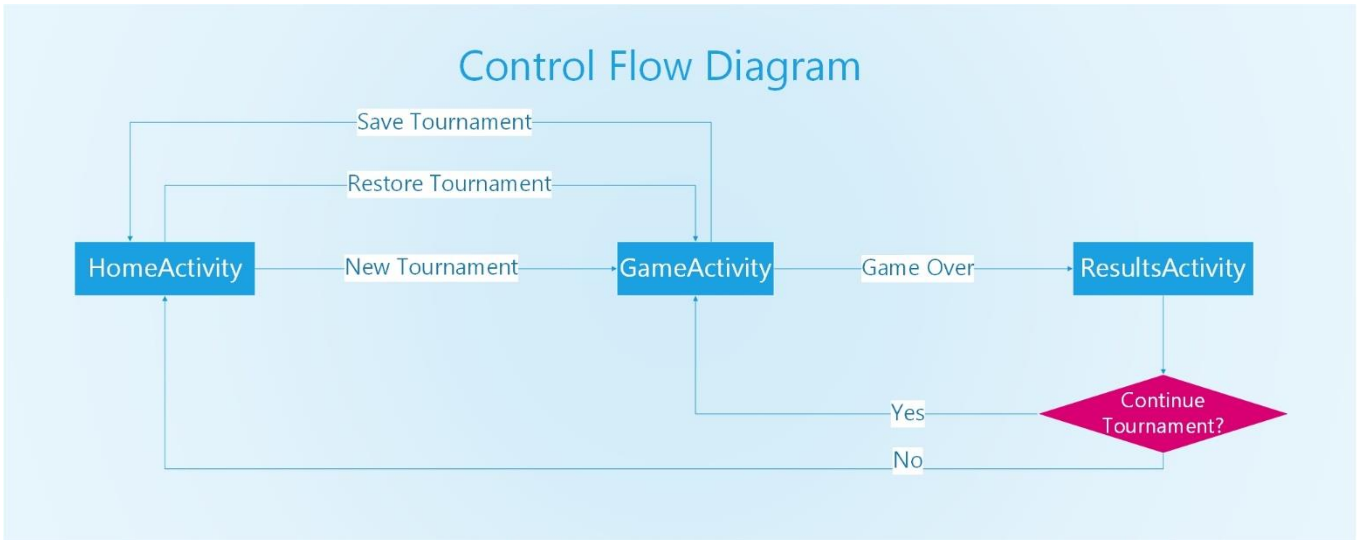


Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

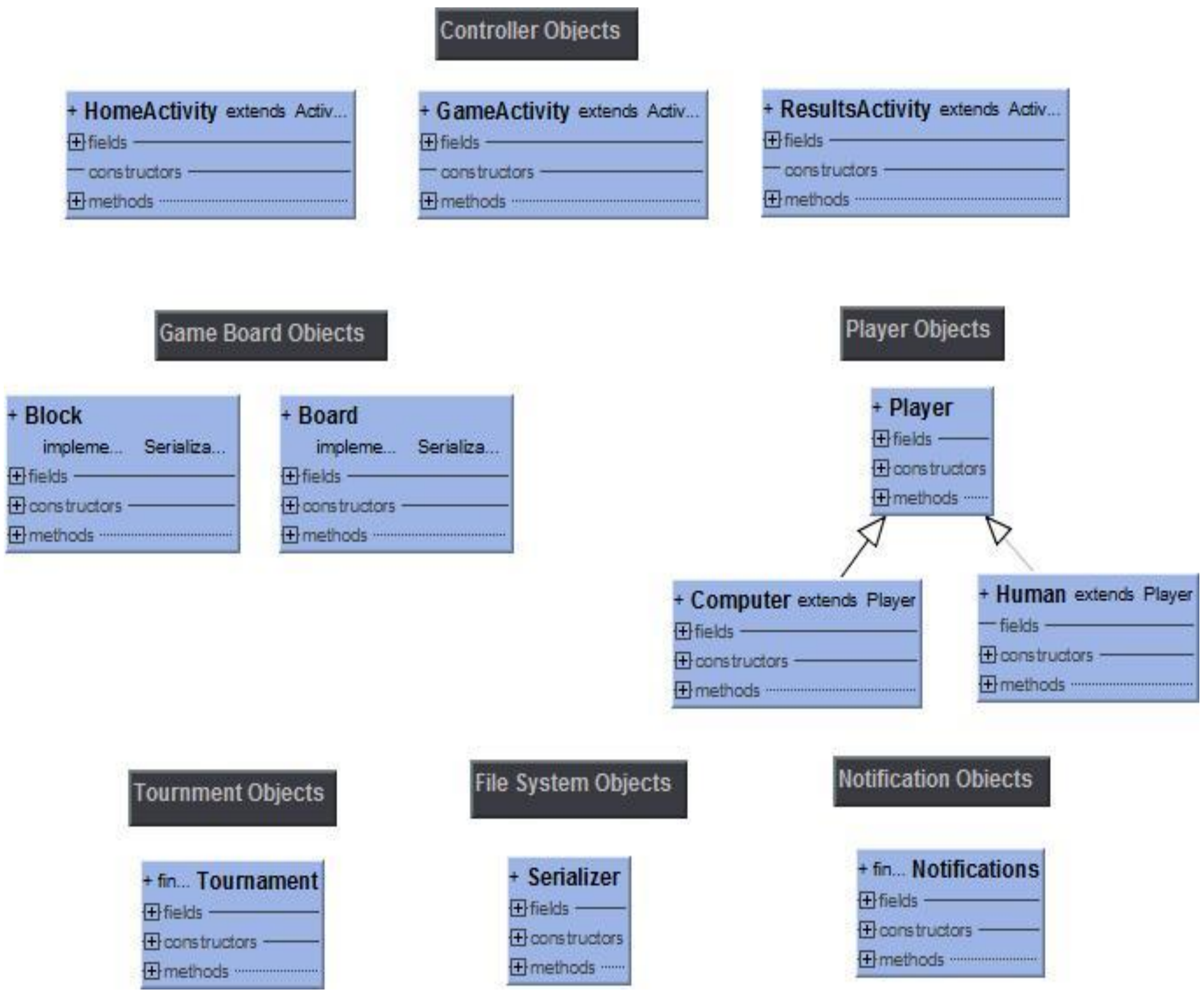
- com.viveckh.threestones.Notifications
- com.viveckh.threestones.Player
  - com.viveckh.threestones.Computer
  - com.viveckh.threestones.Human
- com.viveckh.threestones.Serializer
- com.viveckh.threestones.Tournament
- Activity
  - com.viveckh.threestones.GameActivity
  - com.viveckh.threestones.HomeActivity
  - com.viveckh.threestones.ResultsActivity
- Serializable
  - com.viveckh.threestones.Block
  - com.viveckh.threestones.Board

4.2 - Control Flow Diagram



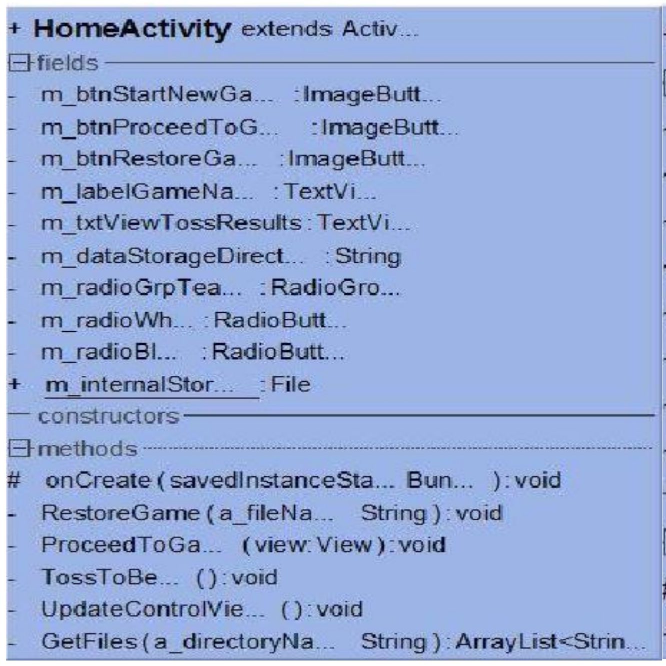
THIS SPACE HAS BEEN INTENTIONALLY LEFT BLANK

4.3 - UML Class Diagram

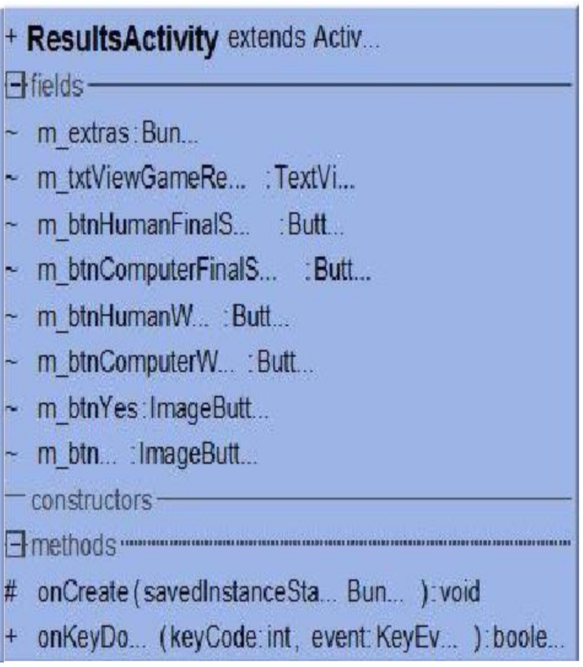


NOTE: Since the original complete UML Class Diagram with all the details was too large to fit in a standard page of this size, the above is a condensed diagram. This individual components of this diagram have been individually expanded over the next few pages.

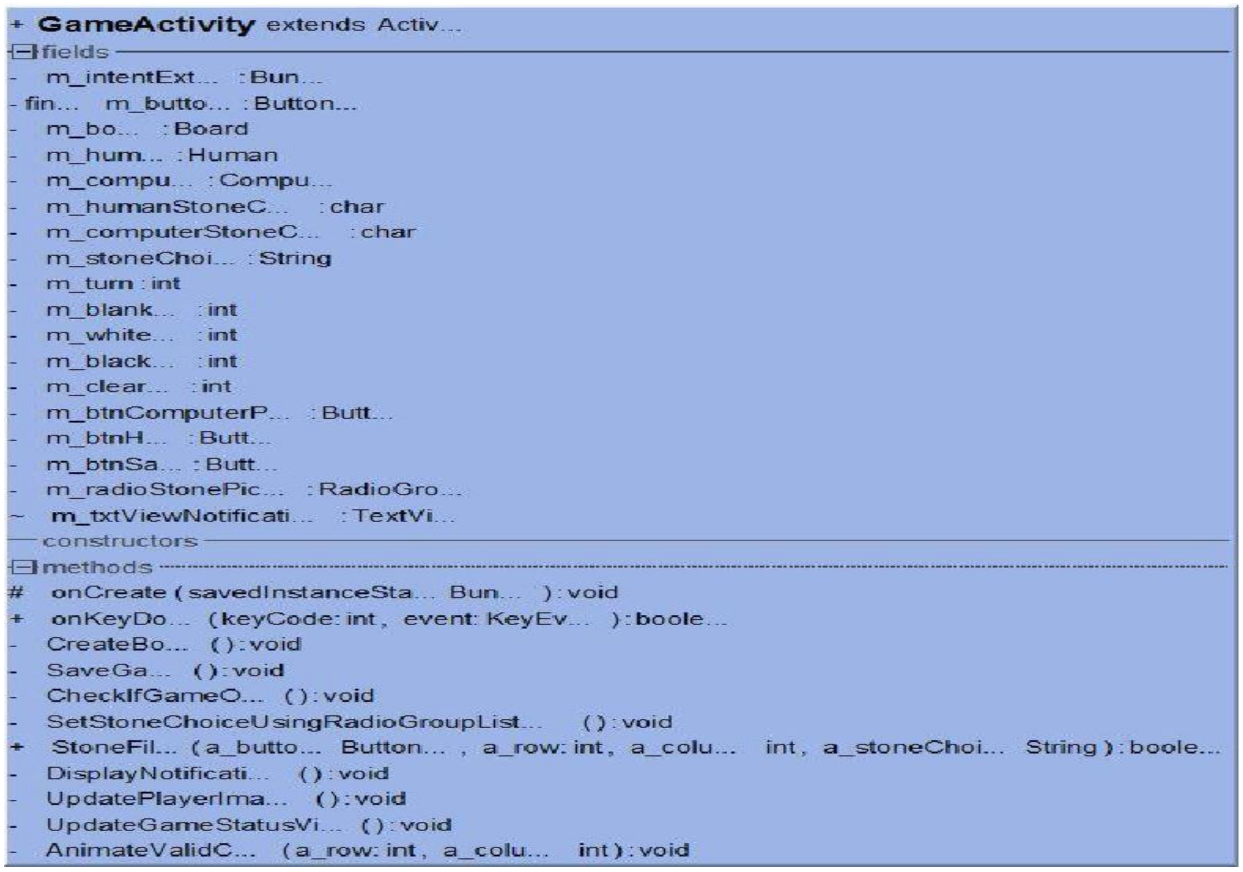
Controller Objects



```
+ HomeActivity extends Activ...
fields
-  m_btnStartNewGa...  :ImageButt...
-  m_btnProceedToG...  :ImageButt...
-  m_btnRestoreGa...  :ImageButt...
-  m_labelGameNa...  :TextVi...
-  m_txtViewTossResults : TextVi...
-  m_dataStorageDirect...  :String
-  m_radioGrpTea...  :RadioGro...
-  m_radioWh...  :RadioButt...
-  m_radioBl...  :RadioButt...
+  m_internalStor...  :File
constructors
methods
#  onCreate ( savedInstanceSta...  Bun...  ) : void
-  RestoreGame ( a_fileNa...  String ) : void
-  ProceedToGa...  ( view: View ) : void
-  TossToBe...  () : void
-  UpdateControlVie...  () : void
-  GetFiles ( a_directoryNa...  String ) : ArrayList<Strin...
```

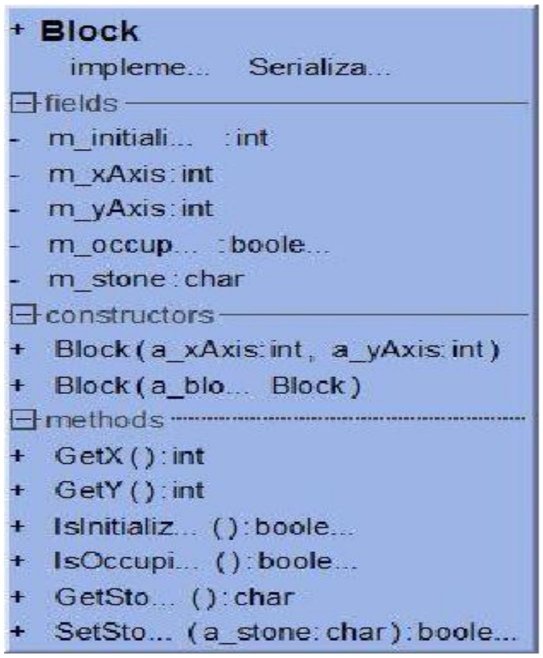


```
+ ResultsActivity extends Activ...
fields
~  m_extras : Bun...
~  m_txtViewGameRe...  :TextVi...
~  m_btnHumanFinalS...  :Butt...
~  m_btnComputerFinalS...  :Butt...
~  m_btnHumanW...  :Butt...
~  m_btnComputerW...  :Butt...
~  m_btnYes : ImageButt...
~  m_btn...  :ImageButt...
constructors
methods
#  onCreate ( savedInstanceSta...  Bun...  ) : void
+  onKeyDo...  ( keyCode: int ,  event: KeyEv...  ) : boole...
```

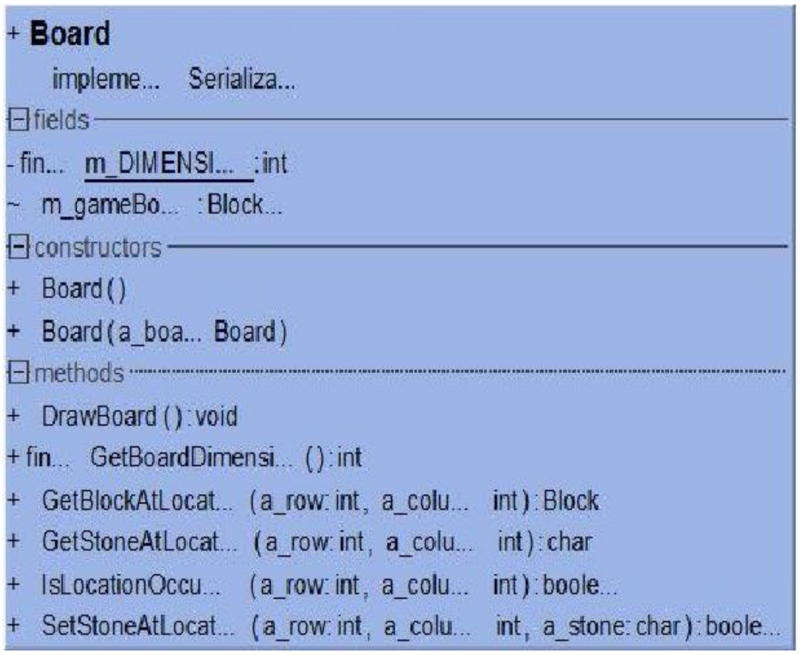


```
+ GameActivity extends Activ...
fields
-  m_intentExt...  :Bun...
- fin...  m_butto...  : Button...
-  m_bo...  :Board
-  m_hum...  :Human
-  m_compu...  :Compu...
-  m_humanStoneC...  :char
-  m_computerStoneC...  :char
-  m_stoneChoi...  :String
-  m_turn : int
-  m_blank...  :int
-  m_white...  :int
-  m_black...  :int
-  m_clear...  :int
-  m_btnComputerP...  :Butt...
-  m_btnH...  :Butt...
-  m_btnSa...  :Butt...
-  m_radioStonePic...  :RadioGro...
~  m_txtViewNotificati...  :TextVi...
constructors
methods
#  onCreate ( savedInstanceSta...  Bun...  ) : void
+  onKeyDo...  ( keyCode: int ,  event: KeyEv...  ) : boole...
-  CreateBo...  () : void
-  SaveGa...  () : void
-  CheckIfGameO...  () : void
-  SetStoneChoiceUsingRadioGroupList...  () : void
+  StoneFil...  ( a_butto...  Button... ,  a_row: int ,  a_colu...  int ,  a_stoneChoi...  String ) : boole...
-  DisplayNotificati...  () : void
-  UpdatePlayerIma...  () : void
-  UpdateGameStatusVi...  () : void
-  AnimateValidC...  ( a_row: int ,  a_colu...  int ) : void
```

Game Board Objects

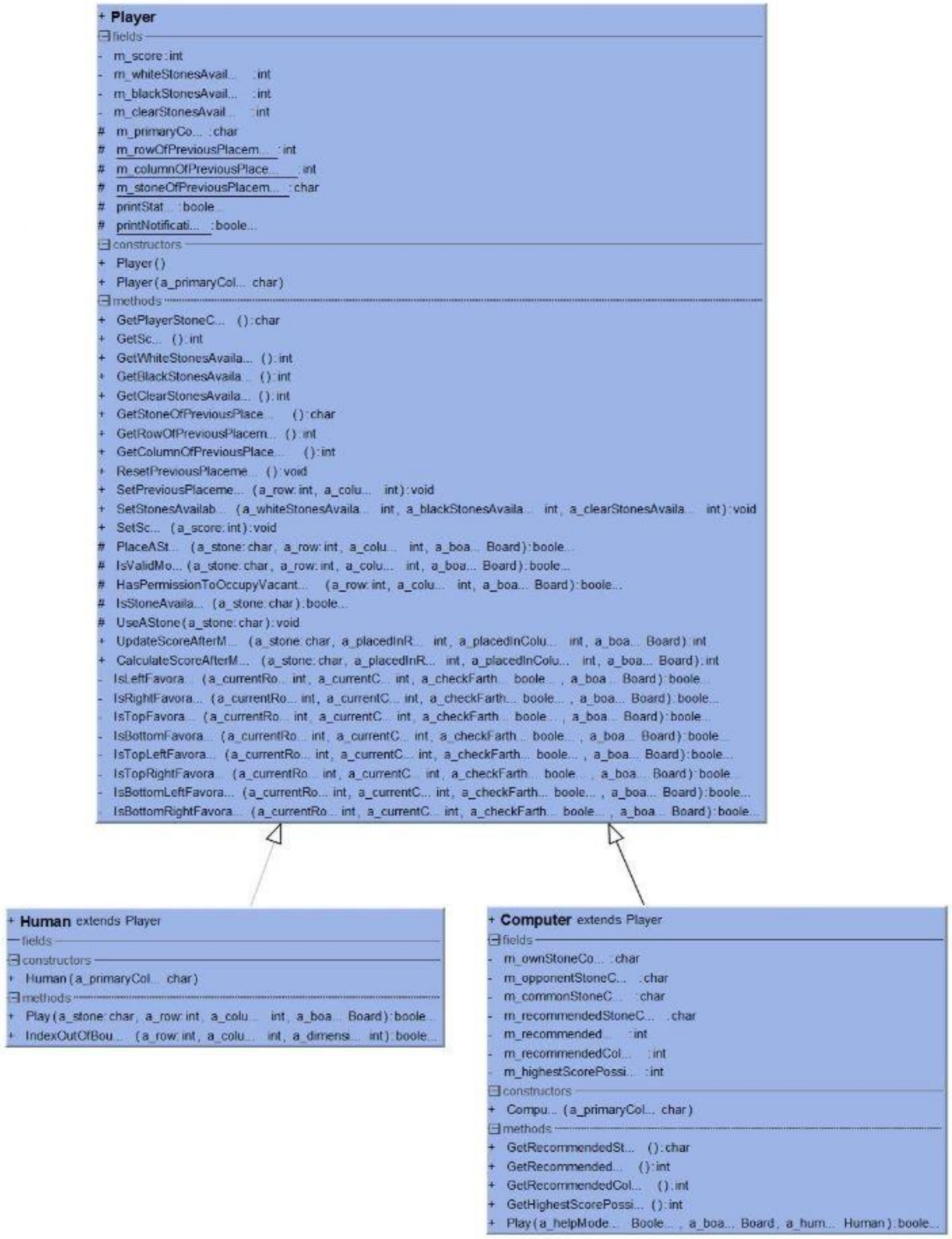


```
+ Block
    impleme...   Serializa...
⊟fields
-  m_initiali...  : int
-  m_xAxis : int
-  m_yAxis : int
-  m_occup...  : boole...
-  m_stone : char
⊟constructors
+  Block ( a_xAxis: int ,  a_yAxis: int )
+  Block ( a_blo...  Block )
⊟methods
+  GetX ( ) : int
+  GetY ( ) : int
+  IsInitializ...  ( ) : boole...
+  IsOccupi...  ( ) : boole...
+  GetSto...  ( ) : char
+  SetSto...  ( a_stone: char ) : boole...
```

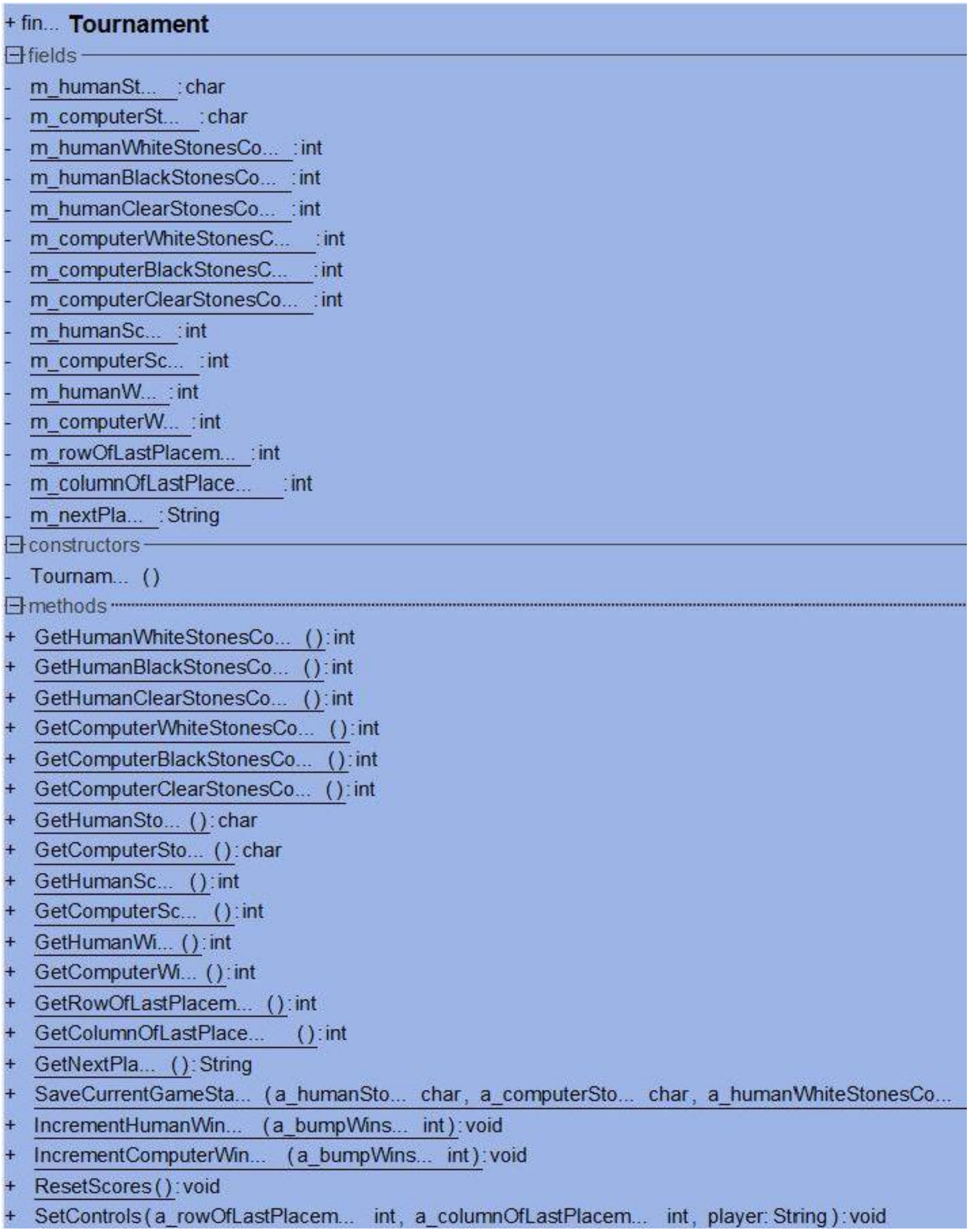


```
+ Board
    impleme...   Serializa...
⊟fields
- fin...  m_DIMENSI...  : int
~  m_gameBo...  : Block...
⊟constructors
+  Board ( )
+  Board ( a_boa...  Board )
⊟methods
+  DrawBoard ( ) : void
+ fin...  GetBoardDimensi...  ( ) : int
+  GetBlockAtLocat...  ( a_row: int ,  a_colu...  int ) : Block
+  GetStoneAtLocat...  ( a_row: int ,  a_colu...  int ) : char
+  IsLocationOccu...  ( a_row: int ,  a_colu...  int ) : boole...
+  SetStoneAtLocat...  ( a_row: int ,  a_colu...  int ,  a_stone: char ) : boole...
```

Player Objects

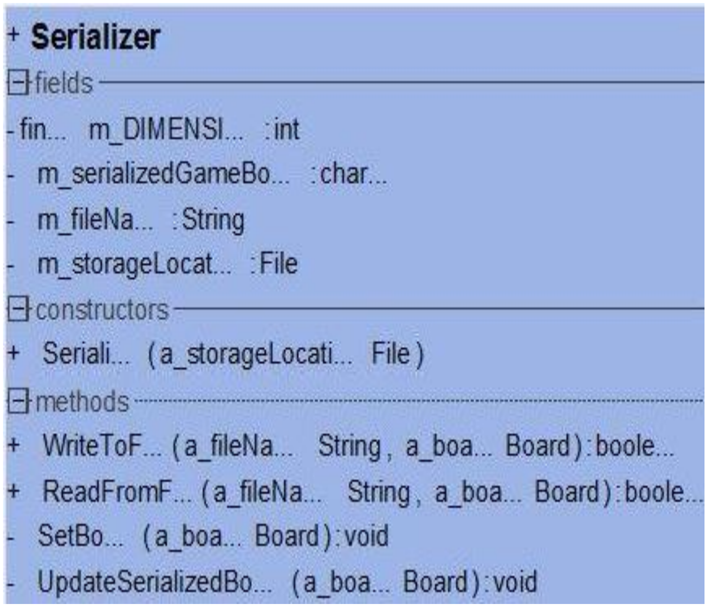

+ Player
fields
- m_score : int
- m_whiteStonesAvail... : int
- m_blackStonesAvail... : int
- m_clearStonesAvail... : int
# m_primaryCo... : char
# m_rowOfPreviousPlacem... : int
# m_columnOfPreviousPlace... : int
# m_stoneOfPreviousPlacem... : char
# printStat... : boole...
# printNotificati... : boole...
constructors
+ Player ( )
+ Player (a_primaryCol... char)
methods
+ GetPlayerStoneC... ( ) : char
+ GetSc... ( ) : int
+ GetWhiteStonesAvaila... ( ) : int
+ GetBlackStonesAvaila... ( ) : int
+ GetClearStonesAvaila... ( ) : int
+ GetStoneOfPreviousPlace... ( ) : char
+ GetRowOfPreviousPlacem... ( ) : int
+ GetColumnOfPreviousPlace... ( ) : int
+ ResetPreviousPlaceme... ( ) : void
+ SetPreviousPlaceme... (a_row: int , a_colu... int) : void
+ SetStonesAvailab... (a_whiteStonesAvaila... int , a_blackStonesAvaila... int , a_clearStonesAvaila... int) : void
+ SetSc... (a_score: int) : void
# PlaceASt... (a_stone: char , a_row: int , a_colu... int , a_boa... Board) : boole...
# IsValidMo... (a_stone: char , a_row: int , a_colu... int , a_boa... Board) : boole...
# HasPermissionToOccupyVacant... (a_row: int , a_colu... int , a_boa... Board) : boole...
# IsStoneAvaila... (a_stone: char) : boole...
# UseAStone (a_stone: char) : void
+ UpdateScoreAfterM... (a_stone: char , a_placedInR... int , a_placedInColu... int , a_boa... Board) : int
+ CalculateScoreAfterM... (a_stone: char , a_placedInR... int , a_placedInColu... int , a_boa... Board) : int
- IsLeftFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
- IsRightFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
- IsTopFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
- IsBottomFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
- IsTopLeftFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
- IsTopRightFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
- IsBottomLeftFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
- IsBottomRightFavora... (a_currentRo... int , a_currentC... int , a_checkFarth... boole... , a_boa... Board) : boole...
+ Human extends Player
fields
constructors
+ Human (a_primaryCol... char)
methods
+ Play (a_stone: char , a_row: int , a_colu... int , a_boa... Board) : boole...
+ IndexOutOfBou... (a_row: int , a_colu... int , a_dimensi... int) : boole...
+ Computer extends Player
fields
- m_ownStoneCo... : char
- m_opponentStoneC... : char
- m_commonStoneC... : char
- m_recommendedStoneC... : char
- m_recommended... : int
- m_recommendedCol... : int
- m_highestScorePossi... : int
constructors
+ Compu... (a_primaryCol... char)
methods
+ GetRecommendedSt... ( ) : char
+ GetRecommended... ( ) : int
+ GetRecommendedCol... ( ) : int
+ GetHighestScorePossi... ( ) : int
+ Play (a_helpMode... Boole... , a_boa... Board , a_hum... Human) : boole...

Tournament Objects



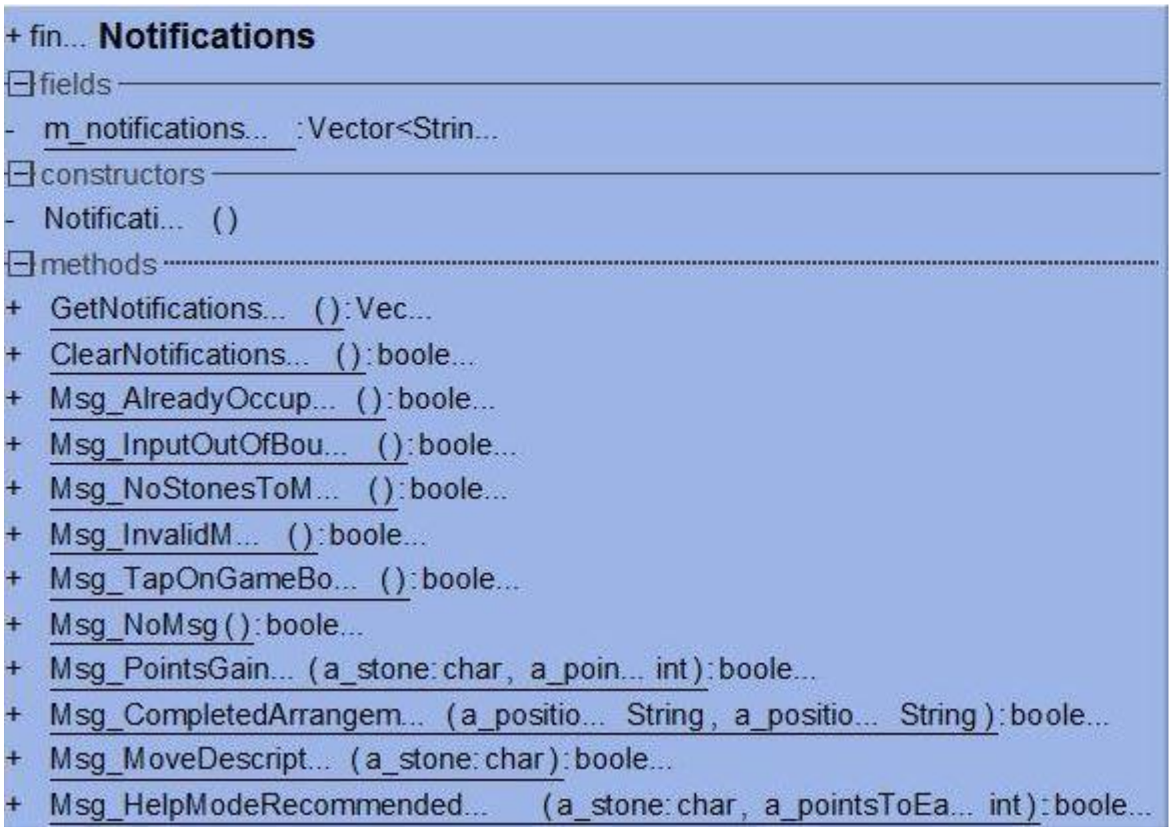
```
+ fin... Tournament
fields
-  m_humanSt... : char
-  m_computerSt... : char
-  m_humanWhiteStonesCo... : int
-  m_humanBlackStonesCo... : int
-  m_humanClearStonesCo... : int
-  m_computerWhiteStonesC... : int
-  m_computerBlackStonesC... : int
-  m_computerClearStonesCo... : int
-  m_humanSc... : int
-  m_computerSc... : int
-  m_humanW... : int
-  m_computerW... : int
-  m_rowOfLastPlacem... : int
-  m_columnOfLastPlace... : int
-  m_nextPla... : String
constructors
-  Tournam... ()
methods
+  GetHumanWhiteStonesCo... () : int
+  GetHumanBlackStonesCo... () : int
+  GetHumanClearStonesCo... () : int
+  GetComputerWhiteStonesCo... () : int
+  GetComputerBlackStonesCo... () : int
+  GetComputerClearStonesCo... () : int
+  GetHumanSto... () : char
+  GetComputerSto... () : char
+  GetHumanSc... () : int
+  GetComputerSc... () : int
+  GetHumanWi... () : int
+  GetComputerWi... () : int
+  GetRowOfLastPlacem... () : int
+  GetColumnOfLastPlace... () : int
+  GetNextPla... () : String
+  SaveCurrentGameSta... (a_humanSto... char, a_computerSto... char, a_humanWhiteStonesCo...
+  IncrementHumanWin... (a_bumpWins... int) : void
+  IncrementComputerWin... (a_bumpWins... int) : void
+  ResetScores () : void
+  SetControls (a_rowOfLastPlacem... int, a_columnOfLastPlacem... int, player: String) : void
```

File System Objects



```
+ Serializer
⊟fields
- fin...  m_DIMENSI...  :int
-  m_serializedGameBo...  :char...
-  m_fileNa...  :String
-  m_storageLocat...  :File
⊟constructors
+  Seriali...  (a_storageLocati...  File )
⊟methods
+  WriteToF... (a_fileNa...  String ,  a_boa...  Board) : boole...
+  ReadFromF... (a_fileNa...  String ,  a_boa...  Board) : boole...
-  SetBo...  (a_boa...  Board) : void
-  UpdateSerializedBo...  (a_boa...  Board) : void
```

Notifications Objects



```
+ fin... Notifications
⊟fields
-  m_notifications...  :Vector<Strin...
⊟constructors
-  Notificati...  ()
⊟methods
+  GetNotifications...  () : Vec...
+  ClearNotifications...  () : boole...
+  Msg_AlreadyOccup...  () : boole...
+  Msg_InputOutOfBou...  () : boole...
+  Msg_NoStonesToM...  () : boole...
+  Msg_InvalidM...  () : boole...
+  Msg_TapOnGameBo...  () : boole...
+  Msg_NoMsg () : boole...
+  Msg_PointsGain...  (a_stone: char,  a_poin...  int) : boole...
+  Msg_CompletedArrangem...  (a_positio...  String ,  a_positio...  String ) : boole...
+  Msg_MoveDescript...  (a_stone: char) : boole...
+  Msg_HelpModeRecommended...  (a_stone: char ,  a_pointsToEa...  int) : boole...
```

5. Class Descriptions

5.1 - Controller Classes

1) **HomeActivity Class –** It serves as the entry point of application and allows the user to start a fresh tournament through toss or restore a previously saved tournament in the state it was left in.
2) **GameActivity Class –** This activity is the most important View-Controller and handles the main game play. It starts by setting the view to the appropriate layout files, and generates the game board. Then, it sets the onClickListener on every pockets (which are Buttons) of game board, control buttons and selection radio buttons. Before every insertion, the user can see whose turn it is and also has the choice to select stones if it is his/her turn. On inserting a stone, it updates the Board, players' scores and stone numbers and the view itself. The background of the pouch is changed depending on which stone is inserted giving the user a feel of an actual stone placement. If the user wants to save the tournament at any point during the game, he/she can tap the "save" button. When all the stones are used by both the players, the controller redirects the player to the next activity.
3) **ResultsActivity Class –** Displays the results of a game and gets user choice on whether to continue or end a tournament to act accordingly.

5.2 - Model Classes

1) **Block Class –** Holds the properties associated with an individual pocket within the bigger game board. Properties include initialization state, X-Y coordinate, occupied/unoccupied state and contained stone.
2) **Board Class –** Initializes a game board and provides helper functions to access/modify the board. The board itself is an 11x11 multidimensional array of 'Block' objects but only those Blocks that fall within the octagonal game board are initialized.
3) **Player Class –** Implements the rules and strategies of the game for a generic player, handles move validations and processing, along with calculation/update of scores for the game play.
4) **Human Class –** Inherits from the 'Player' class and serves the primary purpose of holding properties/attributes associated with the human player, validating and initiating moves on behalf of the human player.
5) **Computer Class –** Inherits from the 'Player' class and implements a greedy best-first-search algorithm to issue moves on behalf of the computer player and to recommend moves to the human player under Help Mode.
6) **Tournament Class –** Singleton class that keeps track of the tournament wins, along with attributes associated with current game like the primary mascot/color of each player, available stones, scores and info about last placement. It comes in handy while saving and restoring a tournament.
7) **Serializer Class –** Contains necessary member functions to serialize or restore a tournament to and from a text file.

8) **Notifications Class –** Singleton class that logs the various error messages and notifications that get generated at various points during the game play. Everything is stored in a centralized vector, and can be cleared periodically to start fresh.

9) **No other special data structures were used besides the ones mentioned above**

5. XML Layout Files

All the following layout files make use of GridLayouts to ensure the views are adaptive to devices of different shapes and sizes.

1) **Activity_home.xml –** Consists of the design and layout information for the home page of the application. View elements associated with toss, mascot/color selection, and listing of saved game files are laid out in here.

2) **Activity_game.xml –** Consists of the design and layout information for the game play window. Lays out the view elements that were necessary for the game board, score board, controls, notification panel.

3) **Activity_results.xml –** Layout file associated with displaying the final game results to the user while presenting with options to continue or end the tournament.

6. Computer Algorithm

The computer player uses a greedy best-first search approach using the following heuristic function and algorithm to make a move and to recommend moves to the human player. This algorithm can be seen in effect in the ***Play function in the Computer Class***.

6.1 - Heuristic function

To grade the quality of a move, the computer uses the following heuristic function:

Points gained by self from this move – Points gained by opponent from this move

6.2 - Algorithm

```
1) For all the pockets in the current game board.
    a. If a pocket isn't null and isn't occupied
        i. If placing a stone with own mascot is valid
            1. Calculate the heuristic value to examine the quality of this move
            2. If this is the best heuristic value so far,
                a. Make a note of this stone and location combination
        ii. If placing  a stone with opponent mascot is valid
            1. Calculate the heuristic value to examine the quality of this move
            2. If this is the best heuristic value so far,
                a. Make a note of this stone and location combination
        iii. If placing a stone with common miller mascot is valid
            1. Calculate the heuristic value to examine the quality of this move
```

2. If this is the best heuristic value so far,
    a. Make a note of this stone and location combination

2) Pick the pocket and stone combination that resulted in the best heuristic value so far.

3) Initiate the move.

7. Bug Report

All the bugs that were discovered during the development and testing phase were fixed right then. At the time of submission, none of the bugs have been left unresolved.

8. Serialization

The application creates a private data directory in the internal memory to store the files generated by the application in order to save and restore a tournament. These files cannot be accessed by the user or any other application outside the program directly.

Following is the structure that these files adhere to. The ‘x’s refer to the unallocated indexes within the multidimensional array which serves as the board. ‘n’s are the empty indexes in the board. The ‘w’, ‘b’, and ‘c’ refer to the three mascots – white, black and clear respectively.

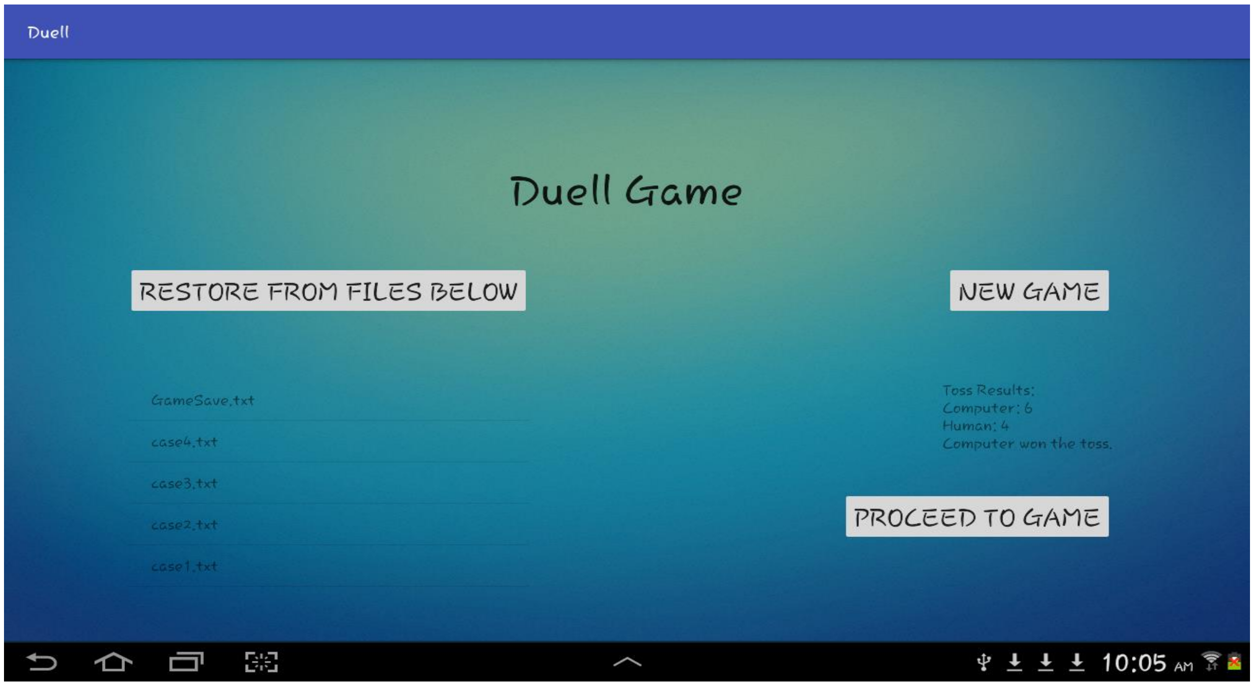
```
Board
x     x     x     x     b     w     b     x     x     x     x
x     x     x     b     b     w     w     w     x     x     x
x     x     b     b     b     b     w     w     w     x     x
x     b     b     w     b     w     w     w     n     n     x
b     b     b     n     w     b     w     w     n     n     n
n     n     n     n     n     x     n     n     n     n     n
n     n     n     n     n     n     n     n     n     n     n
x     n     n     n     n     n     n     n     n     n     x
x     x     n     n     n     n     n     n     n     x     x
x     x     x     n     n     n     n     n     x     x     x
x     x     x     x     n     n     n     x     x     x     x

computer stone: b
computer score: 12
computer wins: 0
computer white stones: 15
computer black stones: 0
computer clear stones: 6

human stone: w
human score: 12
human wins: 0
human white stones: 1
human black stones: 15
human clear stones: 6

last placement row: 4
last placement column: 2
next player: human
```

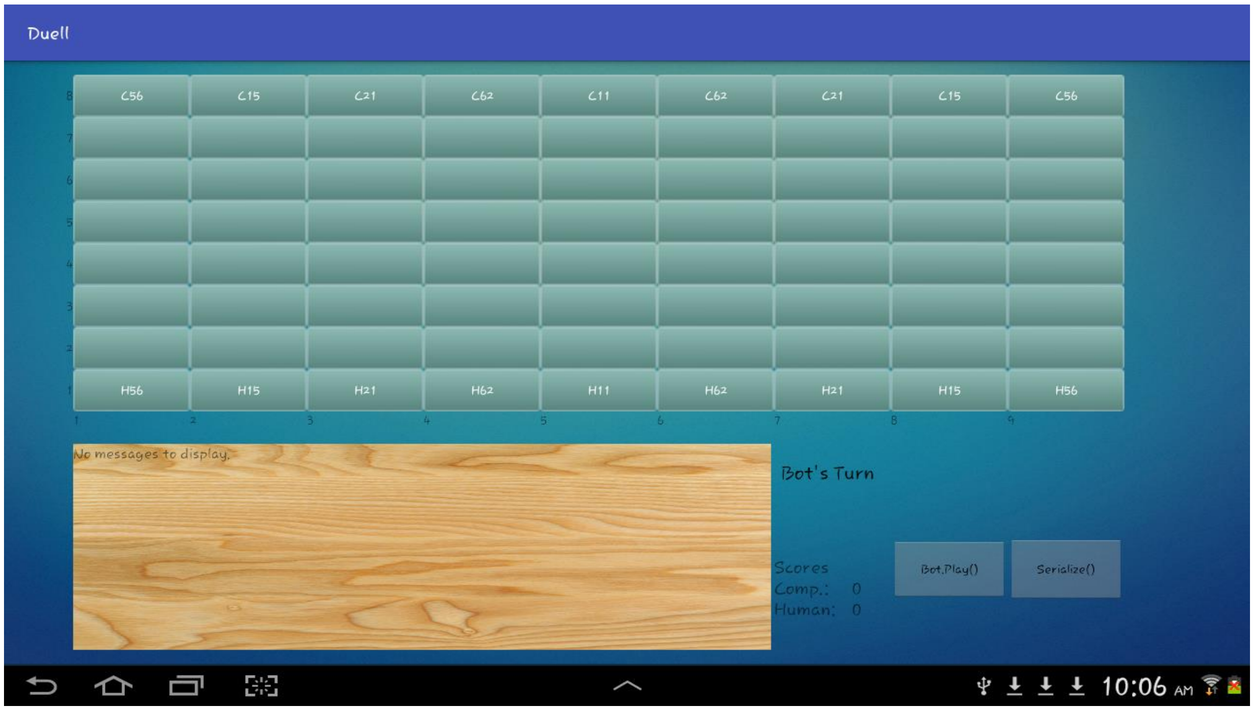
Screenshots



GameSave.txt
case4.txt
case3.txt
case2.txt
case1.txt

Toss Results:
Computer: 6
Human: 4
Computer won the toss.

Fig. Home Activity

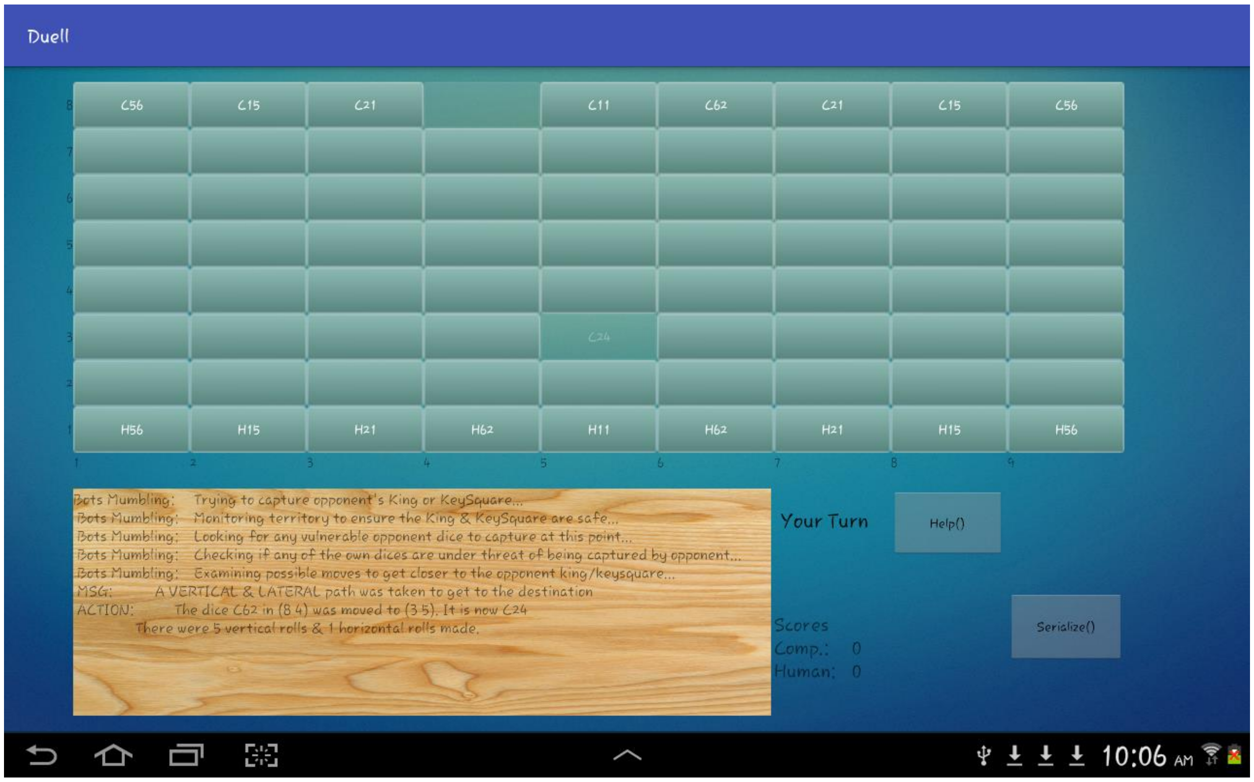

Duell
C56 C15 C21 C62 C11 C62 C21 C15 C56
H56 H15 H21 H62 H11 H62 H21 H15 H56
Bot.Play()
Serialize()
10:06 AM

No messages to display.

Bot's Turn

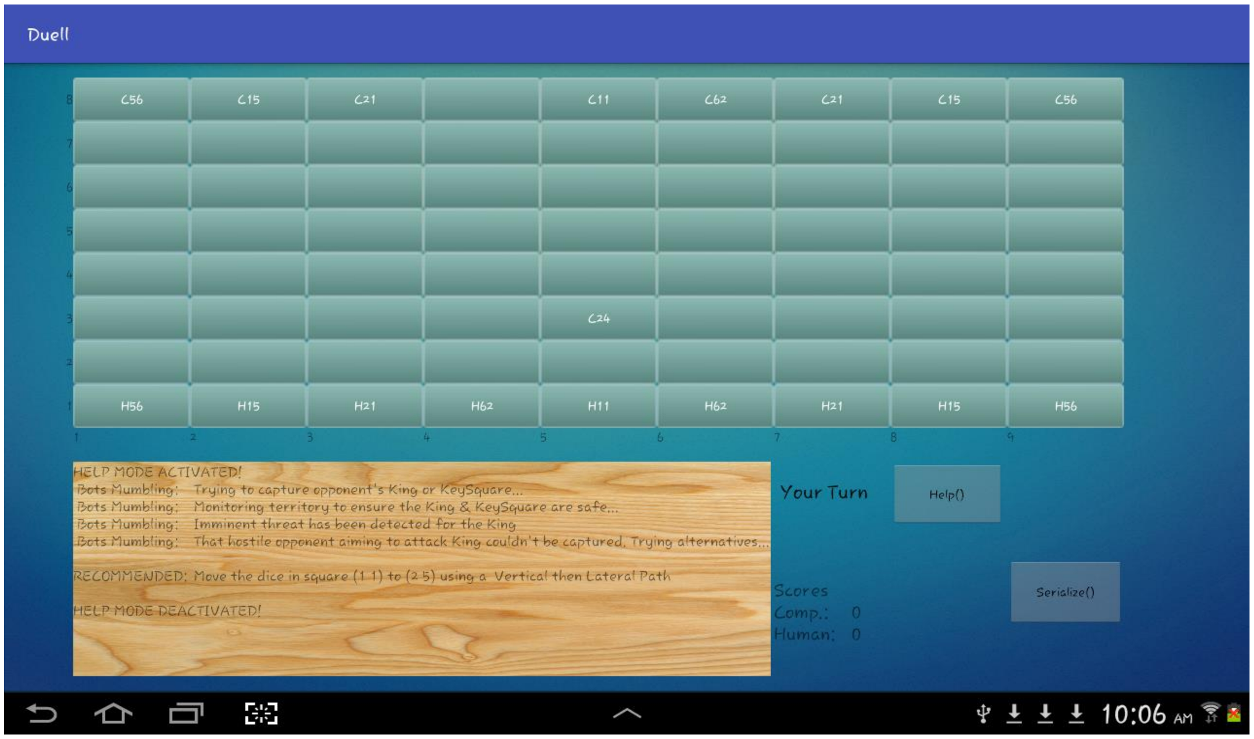
Scores
Comp.: 0
Human: 0

Fig. New Game



```
Bots Mumbling:   Trying to capture opponent's King or KeySquare...
Bots Mumbling:   Monitoring territory to ensure the King & KeySquare are safe...
Bots Mumbling:   Looking for any vulnerable opponent dice to capture at this point...
Bots Mumbling:   Checking if any of the own dices are under threat of being captured by opponent...
Bots Mumbling:   Examining possible moves to get closer to the opponent king/keysquare...
MSG:          A VERTICAL & LATERAL path was taken to get to the destination
ACTION:          The dice C62 in (8 4) was moved to (3 5). It is now C24
              There were 5 vertical rolls & 1 horizontal rolls made.
```

Fig. After Computer's Move

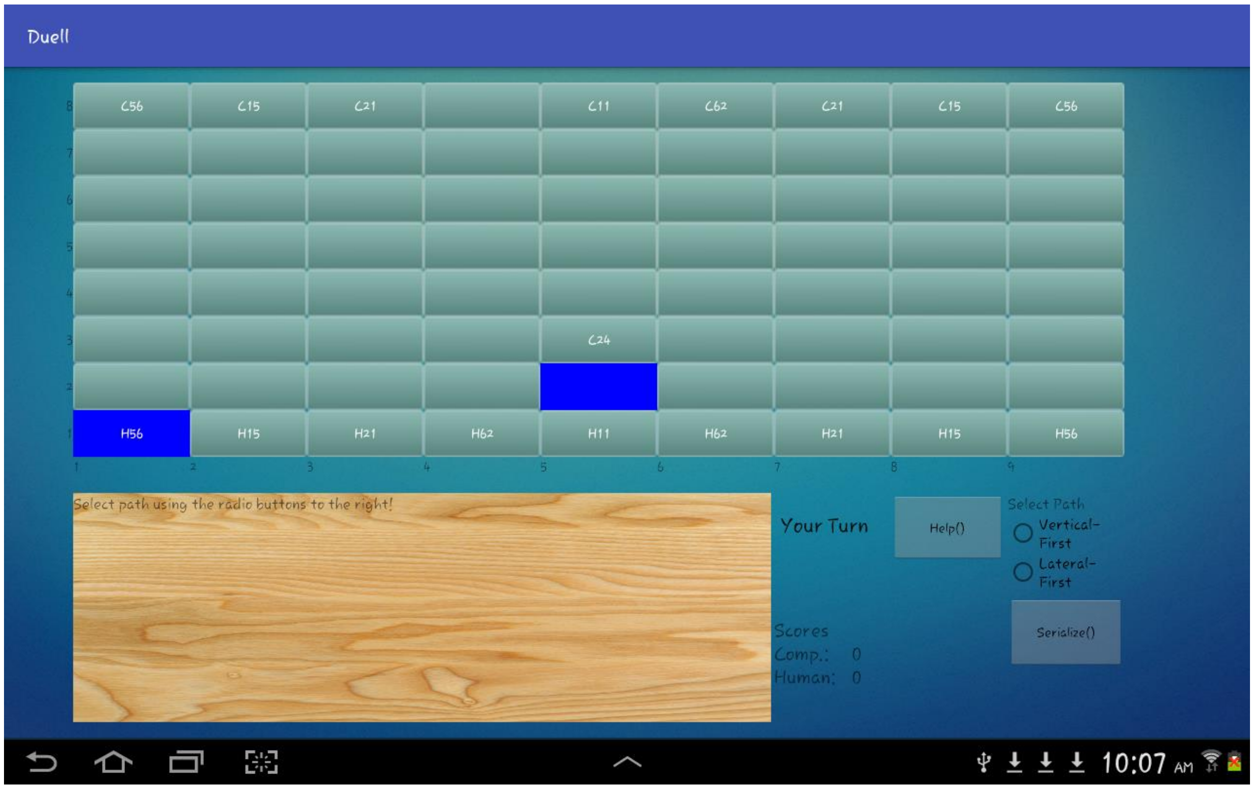


```
HELP MODE ACTIVATED!
Bots Mumbling:   Trying to capture opponent's King or KeySquare...
Bots Mumbling:   Monitoring territory to ensure the King & KeySquare are safe...
Bots Mumbling:   Imminent threat has been detected for the King
Bots Mumbling:   That hostile opponent aiming to attack King couldn't be captured. Trying alternatives...

RECOMMENDED: Move the dice in square (1 1) to (2 5) using a  Vertical then Lateral Path

HELP MODE DEACTIVATED!
```

Fig. Help Mode


Duell

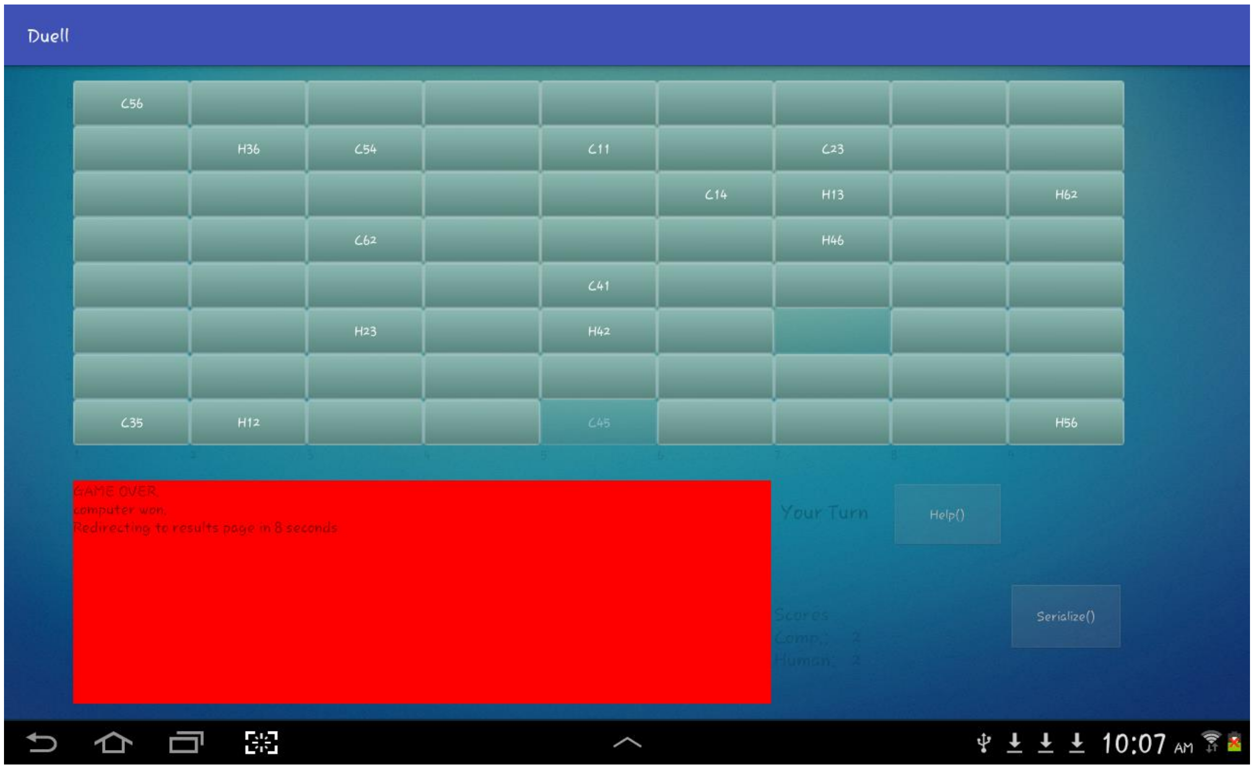
Select path using the radio buttons to the right!

Your Turn

Select Path
Vertical-First
Lateral-First

Scores
Comp.: 0
Human: 0

Fig. Path Selection in 90 degree turns

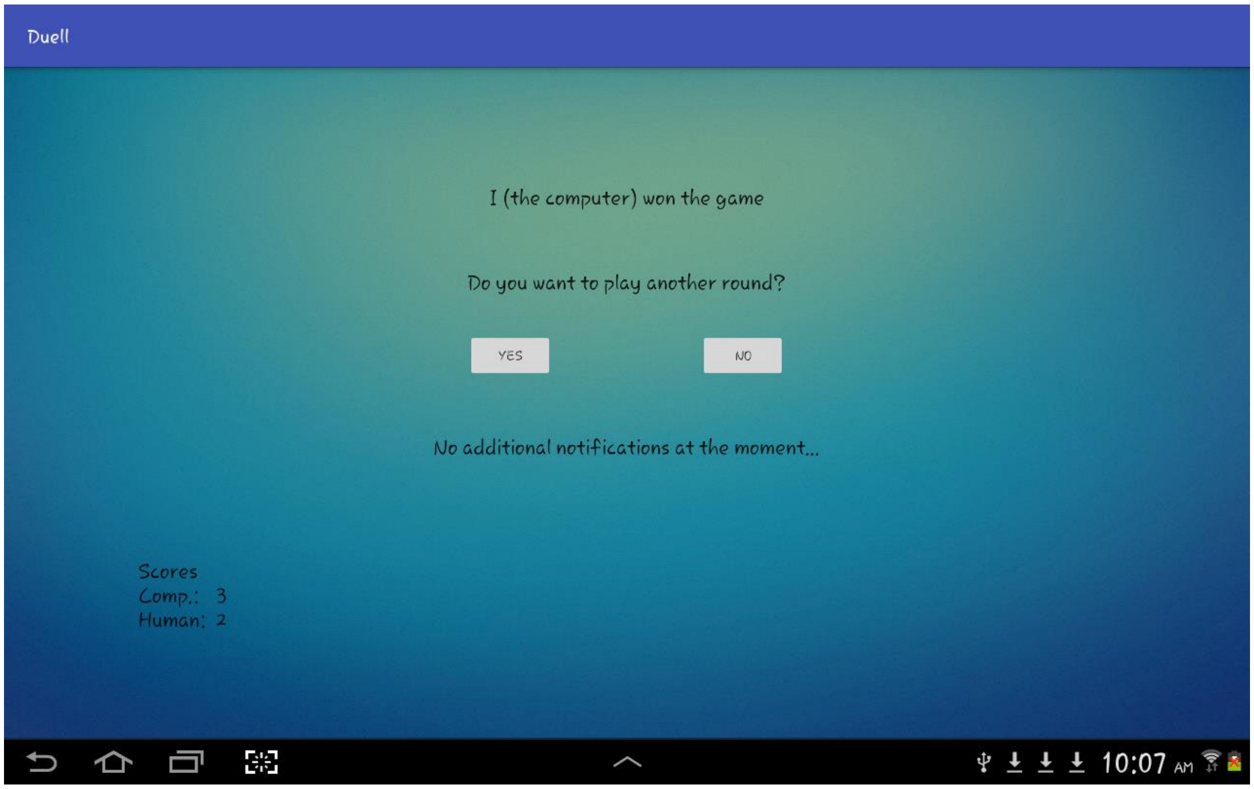

Duell

GAME OVER
computer won.
Redirecting to results page in 8 seconds

Your Turn

Scores
Comp.: 2
Human: 2

Fig. Game Over


Duell

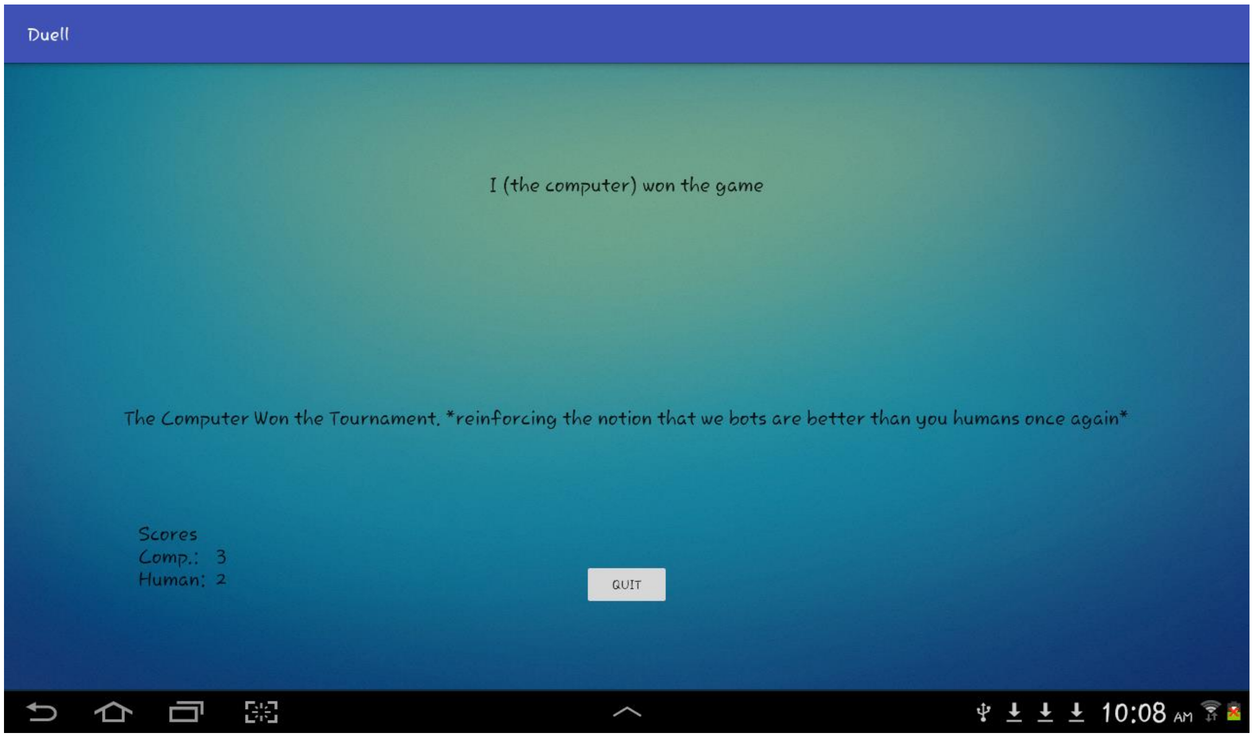
I (the computer) won the game

Do you want to play another round?

No additional notifications at the moment...

Scores
Comp.: 3
Human: 2

Fig. Results Activity


Duell

I (the computer) won the game

The Computer Won the Tournament. *reinforcing the notion that we bots are better than you humans once again*

Scores
Comp.: 3
Human: 2

Fig. Ending Tournament

Log

10/30/2016

Gameboard buttons added to the interface.

Buttons layout customized with background wallpaper.

Migrated following classes from C++ Project: Dice, Square and Board. And made necessary changes to match the specifications of java.

Implemented DrawBoard function in the GameActivity to set text in buttons.

10/31/2016

Migrated the C++ Player class and made necessary changes to accommodate the functions that previously used pass by value.

11/1/2016

Added listeners and implemented functions in GameActivity to handle user input on the board.

Player strategies are fully functional at this point.

11/2/2016

Implemented animations to highlight the start and end coordinates of a move.

11/3/2016

Implemented a static notifications class to keep track of errors and notifications during game play.

Implemented GameOverConditionMet() function in Board class

Added visible labels on the side of the gameboard through xml.

Implemented ResetButtonAvailability() function and added it throughout the GameActivity to ensure only relevant buttons are visible depending on whose turn it is.

11/4/2016

Fixed a small build error associated with const variable not being static in Notifications class

11/6/2016

Implemented HomeActivity and passed new game intents.

Need to handle serialization, ResultsActivity and hightlighting Computer Moves

GameActivity functional besides highlighting computer moves

Implemented notifications display pretty much completely besides new additions, if necessary for android.

11/7/2016

Migrated and made necessary changes to the Serializer class.

11/8/2016

Implemented Serialization class and added dialog to confirm user wants to serialize.

Major components completed, now just need to make the entire game flow properly.

11/14/2016

Implemented File selection ListView in HomeActivity.

Fixed file read issue due to initial spaces by using str.trim()

Need to implement Tournament wrapper in the game now, and file name selection for saving game.

11/15/2016

Implemented ResultsActivity and made arrangments for passing intents and necessary values.

New Issue: Found out that the set caputured is not doing its job in RollUp, RollDown functions because were were returning reference to dice by pass by reference, as a pointer. Can't do that anymore, so it's not working. Need to fix that.

Issue lies somewhere in passing dice objects in functions. It has to do with the pass by reference – pass by value changes that were done while migrating from C++.

11/16/2016

Still struggling in the problem caused by serialization; works when a fresh game is started, but issues persist during a serialized game on the computer’s turn.

11/17/2016

Fixed the bug associated with game restore. The problem existed with making a copy of the board while passing from HomeActivity to GameActivity.

Solved by adding additional lines in Board copy constructor to make sure the square residents are pointing to their counterparts in the dice array.

Gmae is fully functional at this point. Needs a proper way to go to ResultsActivity after game is over because right now I just have a timer to redirect automatically which restricts the user from looking at the board after the game is over. Also maybe, highlight computer moves as well to make the game more efficient.

11/25/2016

Implemented functionality to highlight the computer move

11/27/2016

All Model Classes documented

11/28/2016

Code fully documented, functional and ready for submission

Rubric completed

11/29/2016

Technical Manual Ready.

Project ready for demo and submission.