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 pockets. 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

❖ AI algorithms

- A Greedy best-first search to find locally optimum solution for computer player

❖ "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)

The following are key locations if you want to go through the code

External Github Repository: *https://github.com/Viveckh/ThreeStones*

Flash Drive

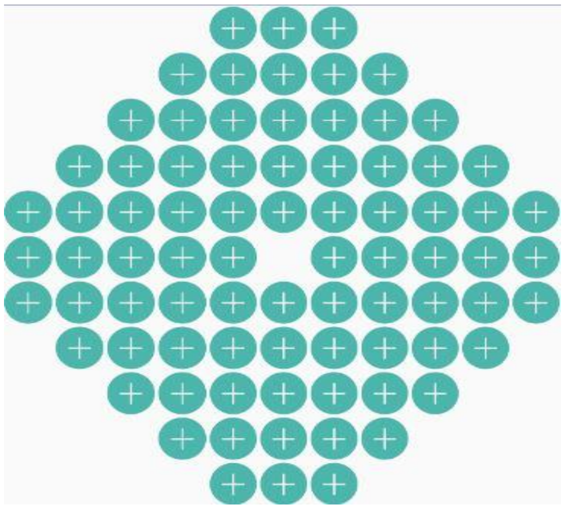
Project Directory: *Senior Project – Vivek Pandey/Project Folder/ThreeStones*

Source Code Folder: *Senior Project – Vivek Pandey/Project Folder/ThreeStones/app/src/main*

3. Game Play 101

3.1 - Board Setup

- ❖ An empty 11x11 octagonal board



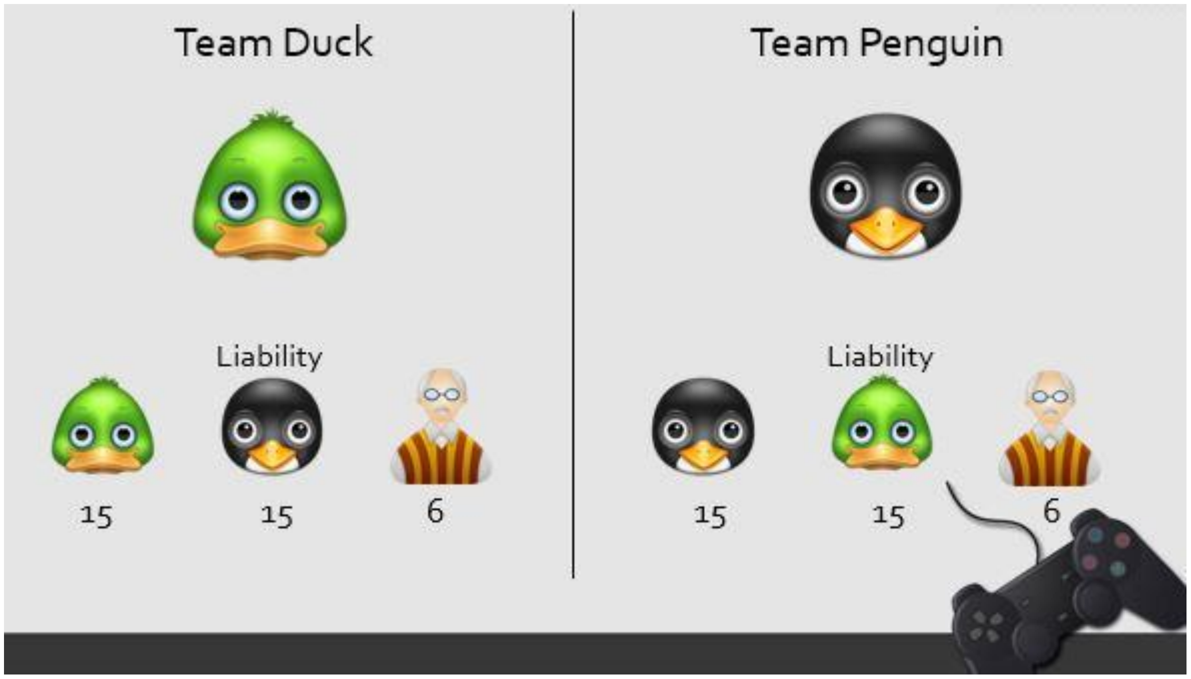
- ❖ Two players, each assigned with their team mascot during toss.
    - *IMPORTANT:* *In the code, the mascots are named as "white" and "black" to avoid their association with particular objects. BUT the view in the game and this documentation refers to the two primary mascots as "duck" and "penguin" 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*.

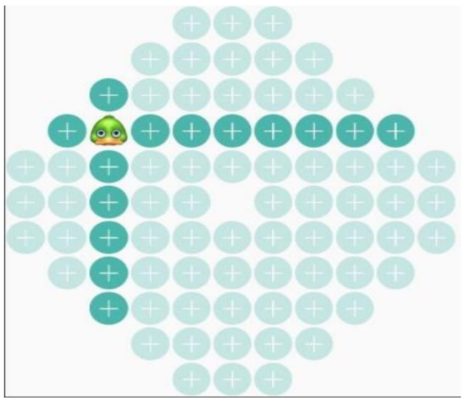
- AND both teams get 6 additional stones each stamped with Miller mascot.

In short,



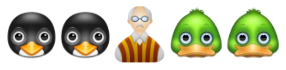
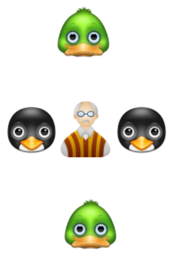
3.2 - How to Play

- **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).

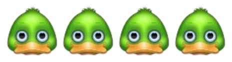
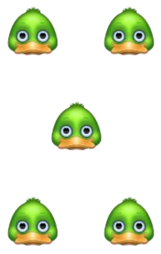


      - If no pocket is available on the row and column of the opponent's last move, the stone can be placed into any empty pocket on the board.
  - **Objective:** The object of each move is to earn points.
    - The player earns point by placing 3 stones in adjacent pockets.
      - The stones may be in a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

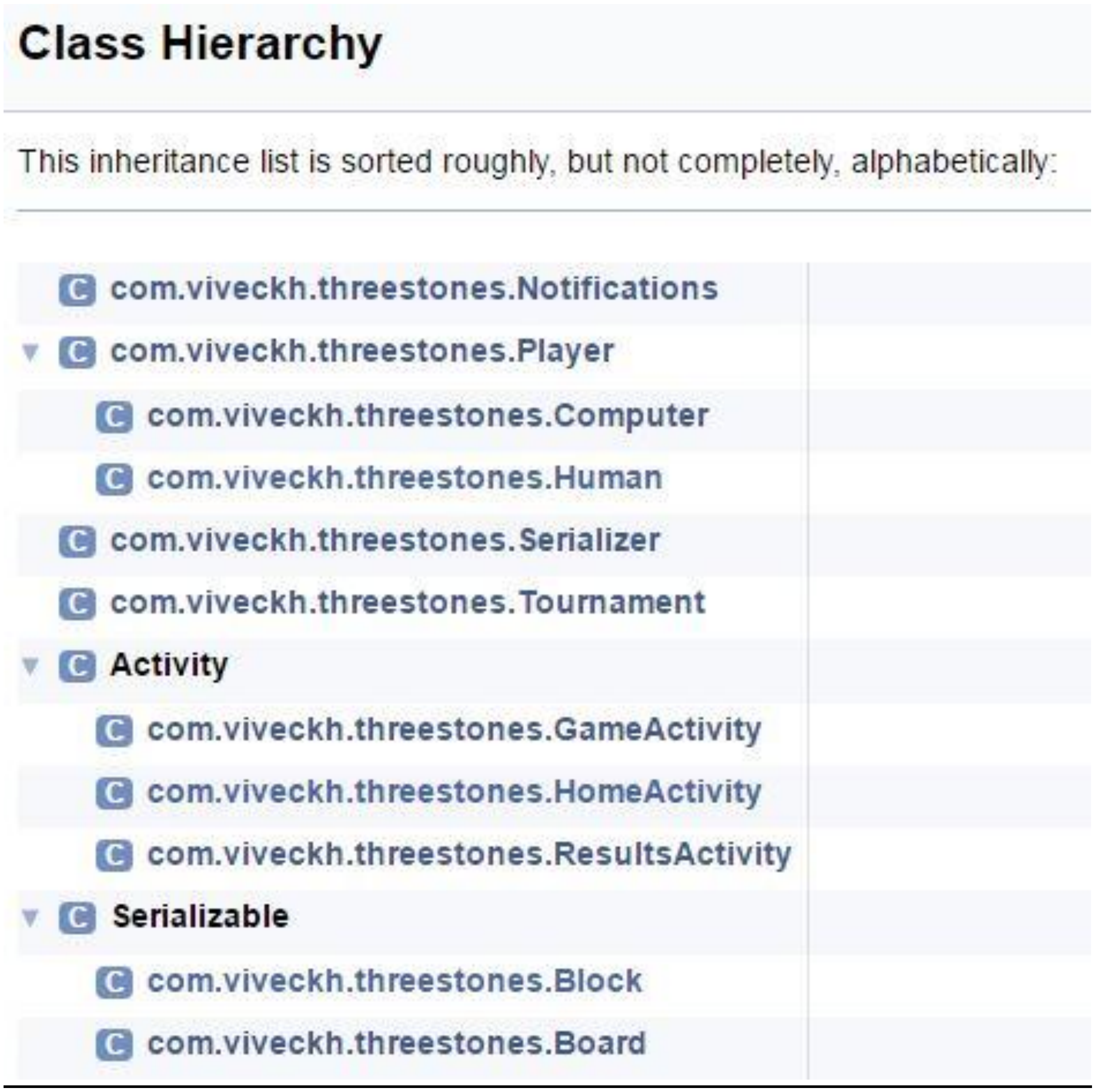
3.3 - Strategy

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

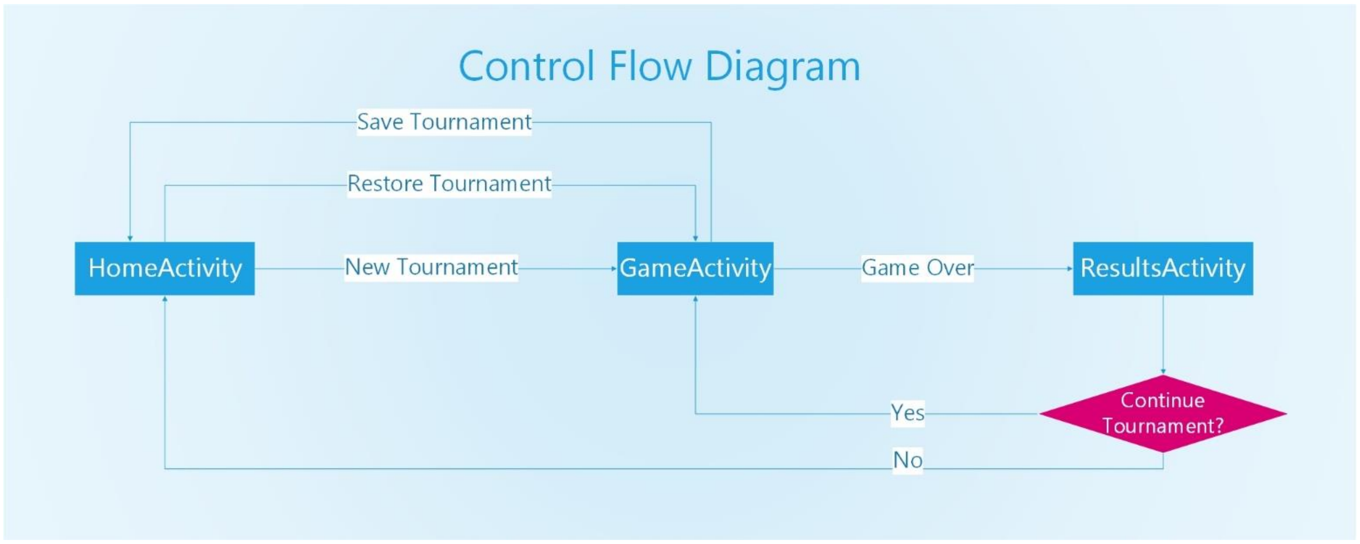


Class Hierarchy

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

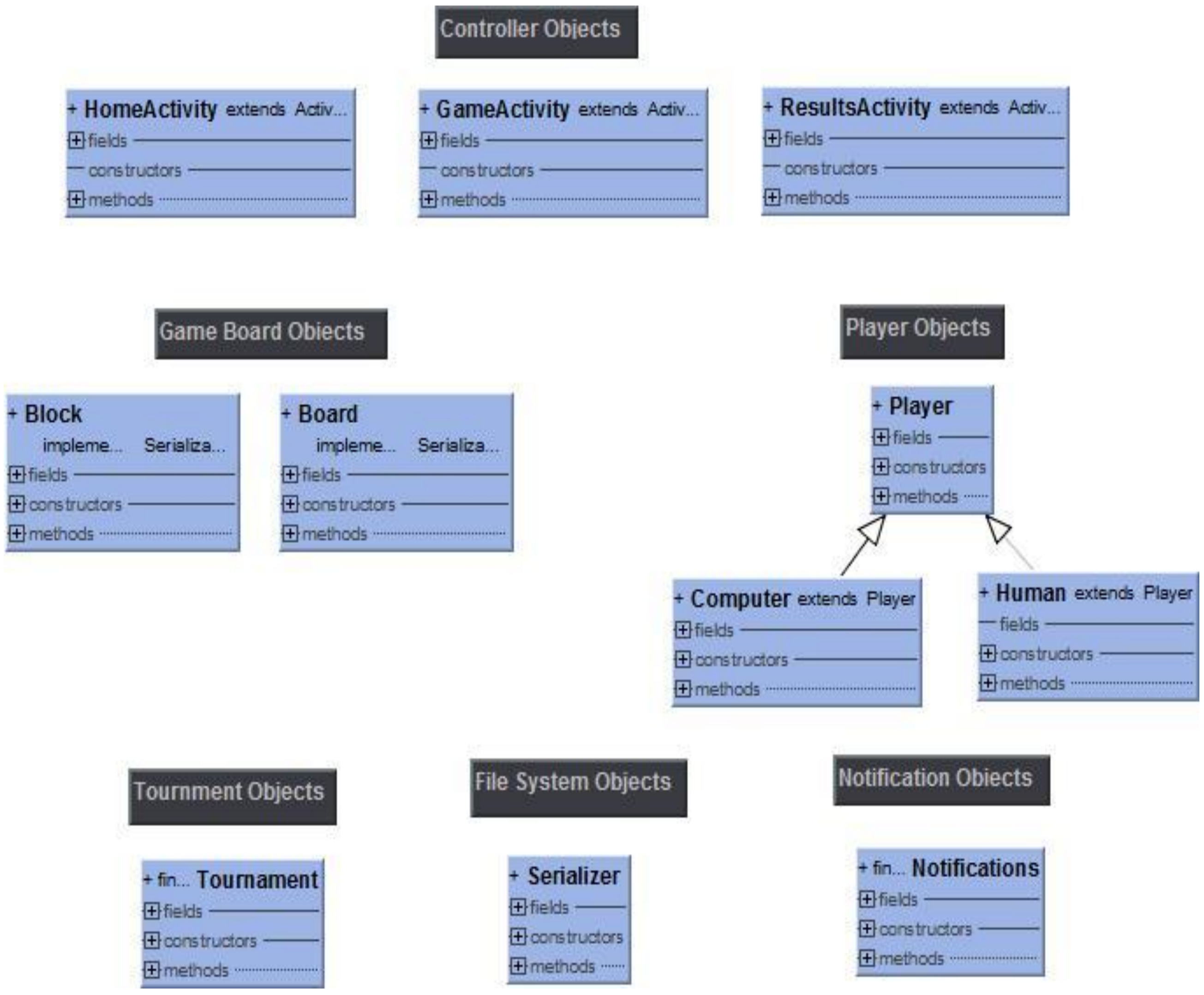
- com.viveckh.threestones.Notifications
- com.viveckh.threestones.Player
  - com.viveckh.threestones.Computer
  - com.viveckh.threestones.Human
- com.viveckh.threestones.Serializer
- com.viveckh.threestones.Tournament
- Activity
  - com.viveckh.threestones.GameActivity
  - com.viveckh.threestones.HomeActivity
  - 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

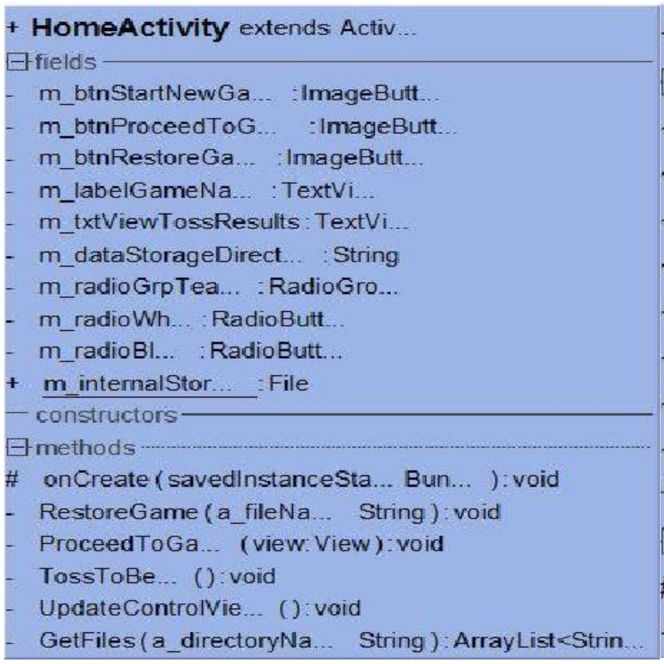
4.3 - UML Class Diagram



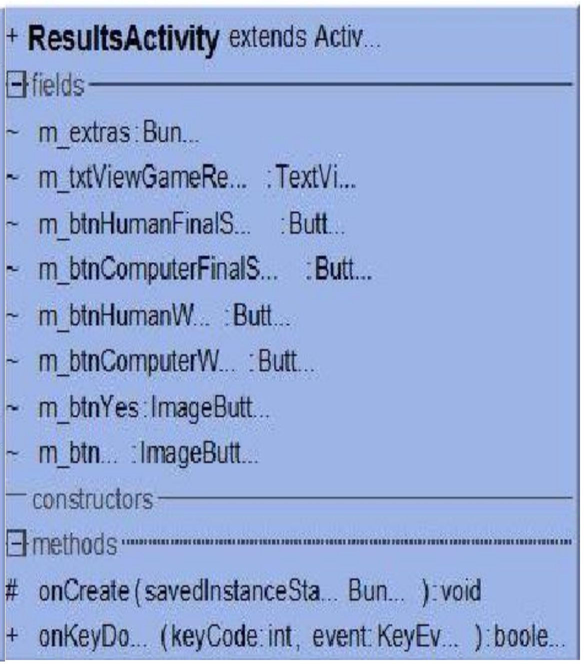
NOTE: Since the original complete UML Class Diagram with all the details was too large to fit in a standard page of this size, the above is a condensed diagram. This individual components of this diagram have been individually expanded over the next few pages.

]

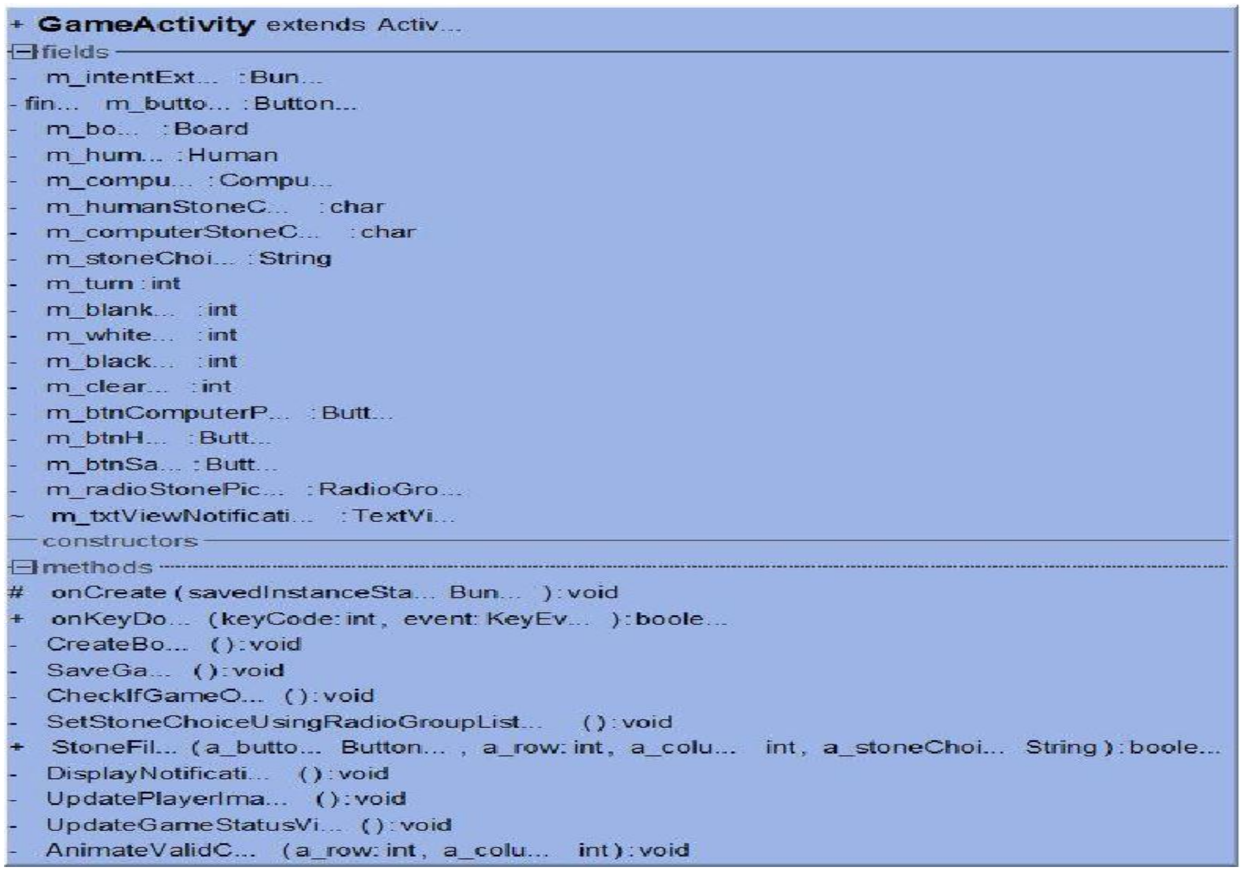
Controller Objects



```
+ 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...
```

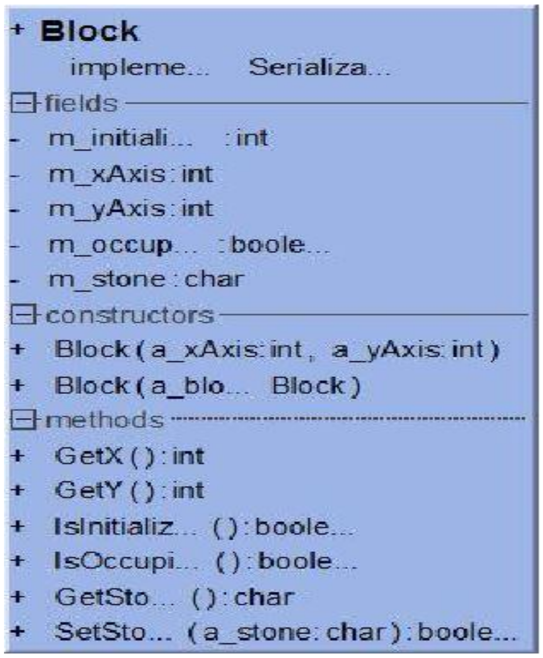


```
+ ResultsActivity extends Activ...
fields
~   m_extras : Bun...
~   m_txtViewGameRe...  :TextVi...
~   m_btnHumanFinalS...  :Butt...
~   m_btnComputerFinalS...  :Butt...
~   m_btnHumanW...  :Butt...
~   m_btnComputerW...  :Butt...
~   m_btnYes : ImageButt...
~   m_btn...  :ImageButt...
constructors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

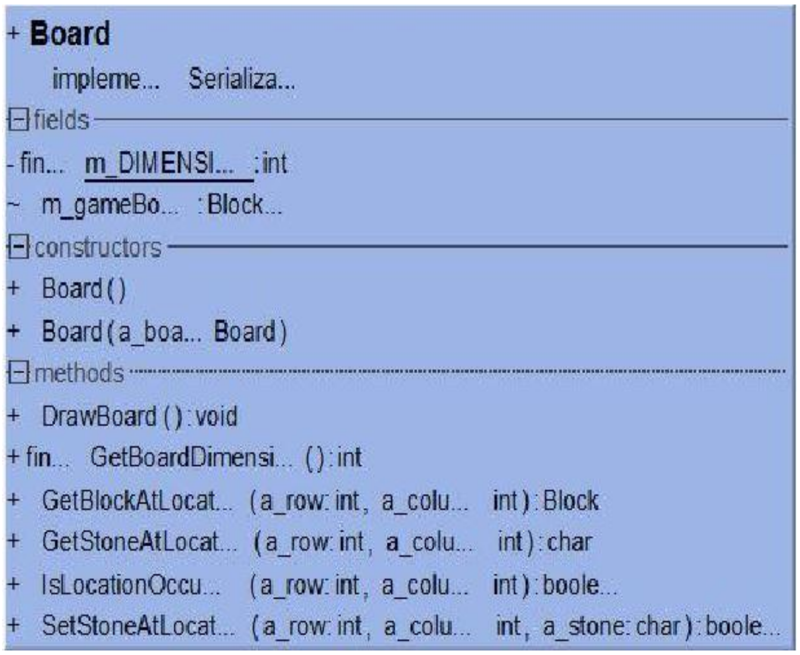


```
+ GameActivity extends Activ...
fields
-   m_intentExt...  :Bun...
- fin...   m_butto...  :Button...
-   m_bo...  :Board
-   m_hum...  :Human
-   m_compu...  :Compu...
-   m_humanStoneC...  :char
-   m_computerStoneC...  :char
-   m_stoneChoi...  :String
-   m_turn : int
-   m_blank...  :int
-   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

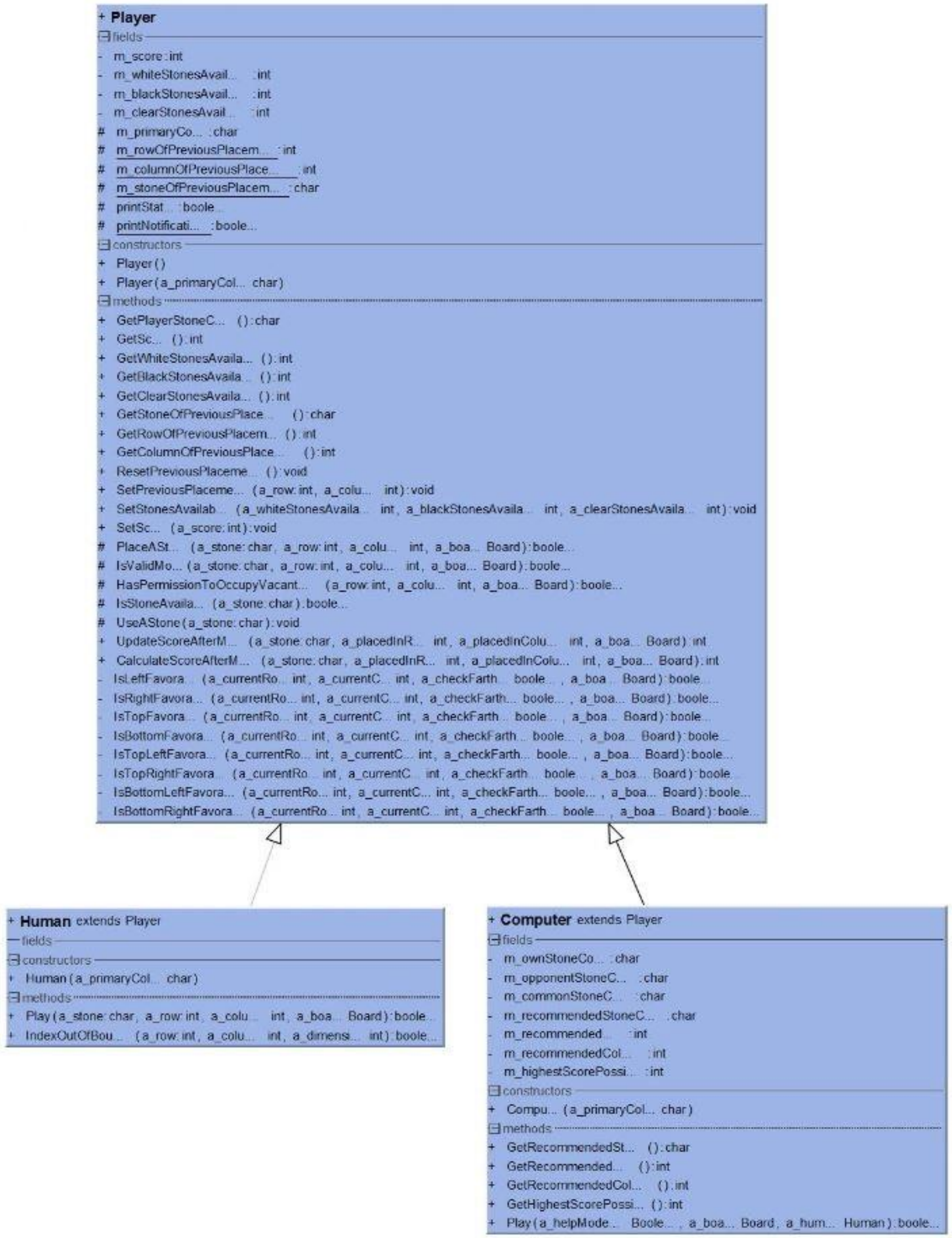


```
+ Block
    impleme...   Serializa...
⊟ fields
-   m_initiali...   : int
-   m_xAxis : int
-   m_yAxis : int
-   m_occup...   : boole...
-   m_stone : char
⊟ constructors
+   Block ( a_xAxis: int ,  a_yAxis: int )
+   Block ( a_blo...   Block )
⊟ methods
+   GetX ( ) : int
+   GetY ( ) : int
+  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

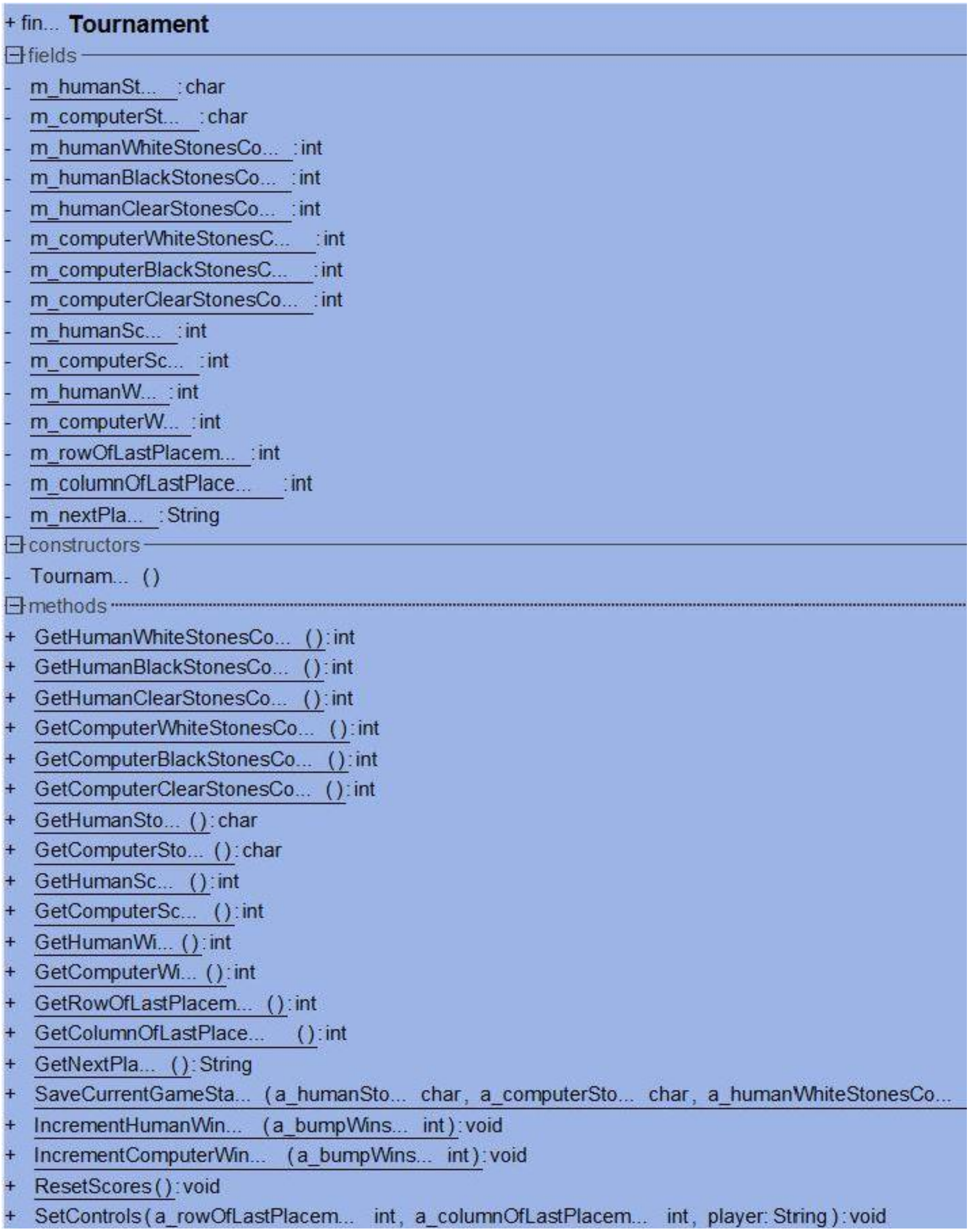


```
+ Board
    impleme...   Serializa...
⊟ fields
- fin...   m_DIMENSI...   : int
~   m_gameBo...   : Block...
⊟ constructors
+   Board ( )
+   Board ( a_boa...   Board )
⊟ methods
+   DrawBoard ( ) : void
+ fin...   GetBoardDimensi...   ( ) : int
+   GetBlockAtLocat...   ( a_row: int ,  a_colu... 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

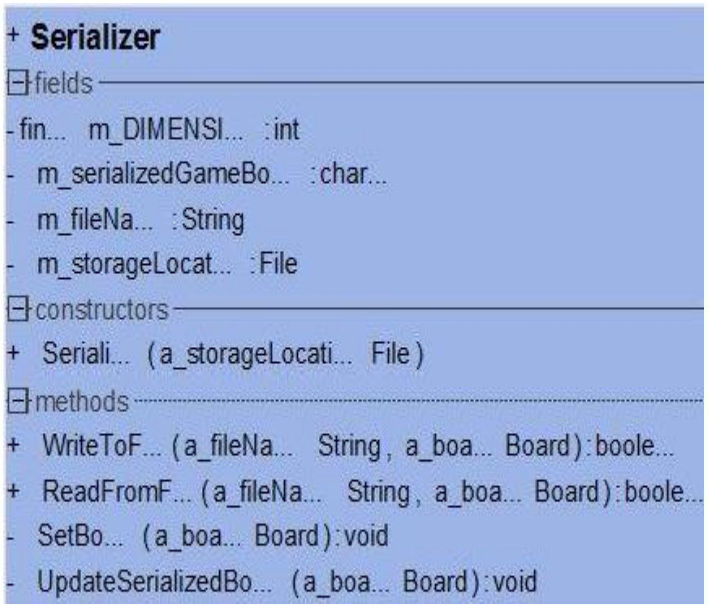

+ Player
fields
- m_score : int
- m_whiteStonesAvail... : int
- m_blackStonesAvail... : int
- m_clearStonesAvail... : int
# m_primaryCo... : char
# m_rowOfPreviousPlacem... : int
# m_columnOfPreviousPlace... : int
# m_stoneOfPreviousPlacem... : char
# printStat... : boole...
# printNotificati... : boole...
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



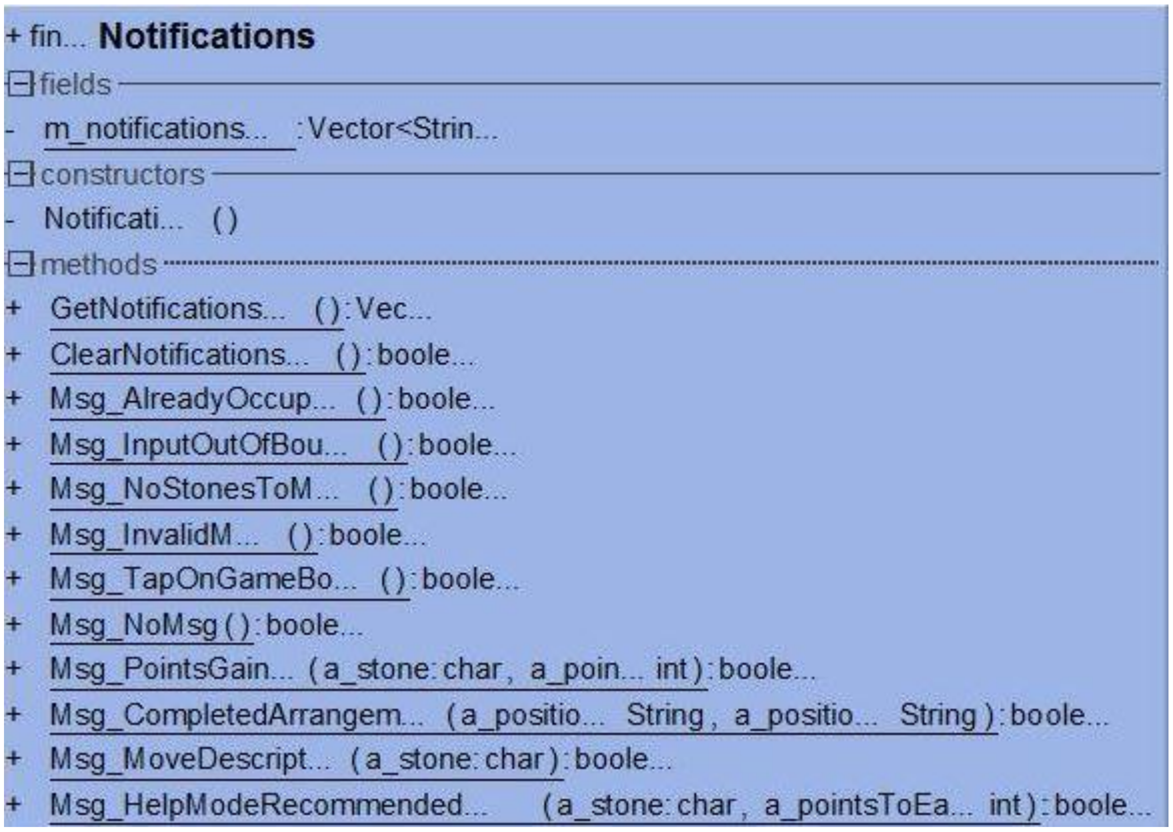
```
+ fin... Tournament
⊟ fields
-  m_humanSt... : char
-  m_computerSt... : char
-  m_humanWhiteStonesCo... : int
-  m_humanBlackStonesCo... : int
-  m_humanClearStonesCo... : int
-  m_computerWhiteStonesC... : int
-  m_computerBlackStonesC... : int
-  m_computerClearStonesCo... : int
-  m_humanSc... : 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

6. 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.

7. AI / Algorithms

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***.

7.1 - 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

7.2 - 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.

THIS SPACE HAS BEEN INTENTIONALLY LEFT BLANK

8. User Interface

8.1 – Overview

The user interface has been designed and developed by ensuring that it is adaptive and responsive for devices of various shapes and sizes. This was essential in order to guarantee that users will have a consistently pleasant experience no matter whichever device they choose to use. The application has been thoroughly tested for its adaptability in devices ranging from a rectangular phone with 5” display to a relatively square tablet with 11” display and there no difference was noticed in the view and application speed. The game play window (which is the most important window of all) has been horizontally divided into two equal sections with one section consisting of the game board whereas the other half consisting of other important elements of game play like the scoreboard, controls to select stones, help button, and the notification panel.

8.2 - Layouts

In order to achieve the above, the application uses a combination of RelativeLayout and GridLayout view groups to encompass all the view objects. RelativeLayout allows the position of each view to be specified as relative to sibling elements or parent elements which maintains the a uniform positioning of overall view elements with respect to each other, no matter which device it is rendered in. On the other hand, GridLayout lays out view components in a rectangular grid. The combination of these two layouts made it possible to design a view that could adjust on multiple devices.

8.3 - Controls

In addition, the application makes use of buttons, radio buttons and clickable list view items to completely avoid text inputs making the application use much simpler and faster for the user. The only text input that the game uses is when the user decides to save the game. At that point, user is presented with a text box to type the filename. Apart from that, there is no use to any other text inputs in the application.

8.4 - Animations

Furthermore, the game playing experience is made as intuitive as possible through the use of various animations. A player's turn is highlighted through an animation on the player's scoreboard section. Invalid moves are blurred and updated after every move, the last move is highlighted through a blinking animation, and computer recommendations are identified by rotating locations within the game board.

8.5 - Sounds

As long as the user have the sounds enabled on their device, the game uses a default sound whenever they successfully place a stone on the game board. A successful move is always followed with a blinking animation, like mentioned in above, but the sound adds to it.

8.6 – Notifications Panel

Although, most of the notifications are handled through animations, a Notifications panel on the bottom right corner during the game play is continuously updated after every move during the play. It displays warnings, error messages, and resulting effect of moves.

9. Screenshots

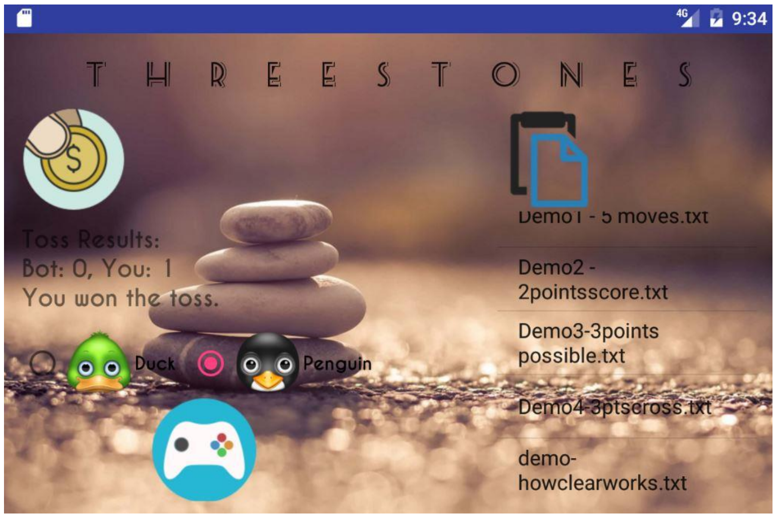
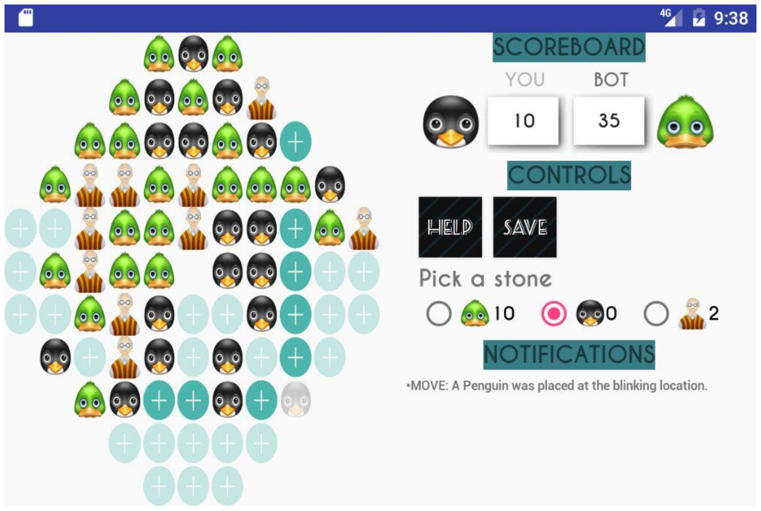


Fig. Home Page

This is the view of the home page of the application. The left half of the view allows the user to conduct a toss, select a stone and start a fresh tournament. The right half of the view displays all the previously saved tournament/games from which the user can instantly restore any by just tapping on the name.



•MOVE: A Penguin was placed at the blinking location.

Fig. Game View – Human Turn

This is the view of the actual game window where the game is played. This particular screenshot reflects a human's turn. You can see the user's option to save the game or ask for help under the controls section. You can also see the description of the last move under the Notifications Panel. Unfortunately, the animations cannot be seen in these static images, but there are blinking effects going on to reflect player's turn and previous move.

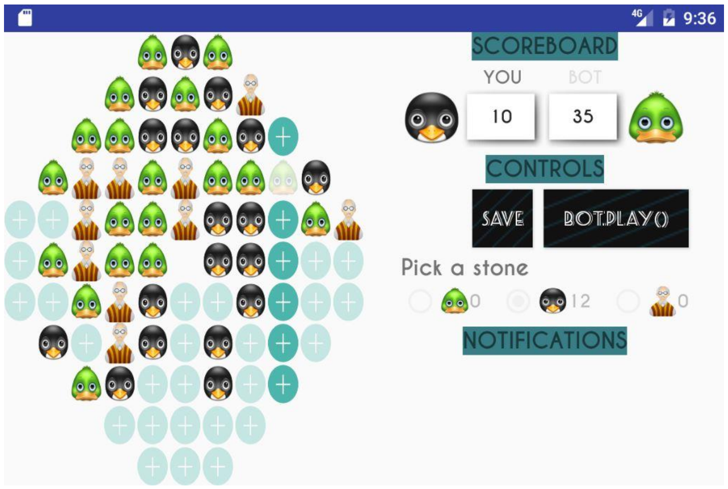


Fig. Game View - Computer Turn

Very similar to the Human's turn described above. The only difference being the invisibility of help mode button (since it's computer's turn) and the visibility of Bot.Play() button. The reasoning behind having a button to initiate the computer's move instead of making it automatic is that to provide the user with the flexibility to continue or save the game at any point.

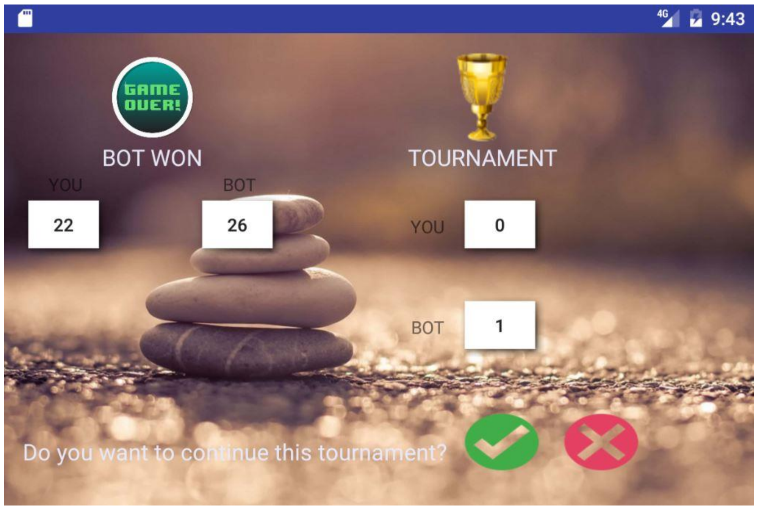


Fig. Results View

The user is auto-redirected to this view when the game is over. The final game scores are mentioned on the left along with the winning team's name. On the right, you can see the tournament stats so far. User has the option to quit or continue the tournament. If the user chooses to continue, a fresh game is started again. Otherwise, he/she will be redirected to the home activity.

10. 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
```

11. Testing

11.1 - Automated Agile Testing

Since the application was developed following an agile methodology, most of the testing was done during the development process. For each of the model and controller classes, the testing was done by initiating a few instances of the class within the class's main function and calling various internal functions to ensure the results were as expected. For the view classes, which were basically xml layout files, the testing was done by re-rendering the view using the android emulator in multiple devices of various shapes and sizes. This helped me discover some of the critical bugs early on during development and I was able to deal with them in time.

11.2 - Manual Testing

After the application was fully developed, it was tested through a few hours of manual testing. Thanks to the Notifications Panel I add implemented in the game view, I was able to customize and use it to exactly track what my actions during the game play were doing on the underlying model. I also used the android debugger to step through the code to see if things were running as expected. Carefully placed breakpoints throughout the code also helped me catch any unexpected behavior during the play. Since most of the testing was done during the agile development phase, there were not any critical issues that were left undiscovered by the time the application was complete.

Therefore, 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.

12. Log

The following log was maintained during the development process and contains important details about how the implementation process was carried out. It will serve as an important reference point for a future developer who might be working on the project.

02/13/2017

- Initial project setup, created an android project and created corresponding github repository for version control

03/21/2017

- Implemented the Block, Board and Player class on a local eclipse project.
- Migrated the code later to an Android Studio Project.
- Tested and ensured that the code was working on the new platform.

03/25/2017

- Refactored the Block, Board and Player class to fit the MVC model.

03/26/2017

- Bugs were found on the scoring and stone update functionality.
- Debug next time to carefully see where specifically the problem is.

03/27/2017

- Fixed the bugs associated with scoring and stones update.
- Had to make the 'primarycolor' a private variable in player class initializing it through super in derived class.
- Fixed scoring by separating the Score Update function from the Stone Placement function and updating it for each player on every move.

03/29/2017

- Implemented Play function in computer class to make a move if it can earn a score.
- Found issue while trying to check if the block object is null, because once an object is returned null, it will still contain elements. So, adding an additional class member to identify a block's initialized status helped.
- Fixed the issue with null indexes passing as valid indexes during scoring by adding IsInitialized() function to Block class.

04/11/2017

- Adding animations for highlighting moves and making valid moves more clear.
- Tried implementing Minimax algorithm, but stumbled across few issues so reverted back to old strategy of greedy best-first algorithm which seems to be working well now.
- A move is always guaranteed from the computer at this point which is good.
- Need to make the GUI look better at this point and notifications.

04/12/2017

- Made the game status section look aesthetically pleasing.
- Added scoreboard with buttons, and finally was able to arrange the grid with colspans to fit properly.
- Need to work on that more next time so that stone selection and number of stones available can be shown in a single list to save space.

04/13/2017

- Added gdrive/dropbox/github icons
- Added typeface to use fonts in order to make the view look better
- Added radio buttons in order to select stones.

04/15/2017

- Added Notifications Panel which now prints the log that is stored in Notifications class.
- Found an issue with scoring the way left-farLeft or right-farRight is setup.

04/17/2017

- Implemented Tournament Class
- Fixed the bug associated with scoring in previous commit using a Boolean variable while checking which sides are favorable.

04/18/2017

- Implemented the home screen of the application so it has everything we might essentially need.

04/22/2017

- Implemented Help Mode to recommend moves to the Human Player
- Add save button and implemented the WriteToFile() function is Serializer class
- Folder creation successful in internal storage.
- No need to use external SD Card since the game creates text files that require very low storage space.

04/23/2017

- Implemented ReadFromFile() function to read contents and restore the game board
- Highlighting last moves and blurring invalid moves after board restoration
- Implemented the ResultsActivity partially.

04/24/2017

- Displays scores in results window
- All features implemented for submission and wiped out clutter of unused code.
- Duck and Penguin image used as two teams to make the game more easy to interpret.

04/26/2017 – 05/10/2017

- Code Documentation, User Manual, Presentation and Preparation for Submission.

13. Conclusion

Developing a large project definitely has taught me to put a significant amount of thought into the design process. A well thought out design really simplified my implementation process in the Threestones project. However, it is also important for me to mention that I have found it to be equally important to be ready to make minor adjustments in the design along the implementation process. Halfway during my implementation, I began to see how I could refactor everything I had coded so far to really cut down the number of lines of code while also increasing the modularity of the various classes. So, I went back and readjusted my design and coded a few sections of the project again. This design process was definitely the part of the project that I enjoyed the most.

Another important skill I have learned in this project is also the user interface design. I have never been a front-end person, and the most important challenge I have always faced in my past applications is making the view adaptable to adjust to various devices. So, I did spend a considerable amount of time and effort in this project to ensure that the view will maintain its uniformity across various devices and I am happy that I was able to achieve that in this project. Also, making the view aesthetically pleasing was another challenge and I had to experiment with various designs to finally settle with the one that seemed to be the best fit.

The most challenging part I would say was to think from a user's perspective. As a developer, I often had to sit on the users' side and constantly ask myself the question – "What would a user think about this?" And since the game rules of Threestones is considerably complicated, I wanted to free the user from all the confusions he/she will encounter during the play. Hence, I found myself adding and implementing new features just to simplify the life of the player. Features like the animations to highlight moves, computer recommendations, simplified notifications panel, enlarged score board were all a result of putting this thinking hat on.

Furthermore, the application increased my respect for the agile development methodology once again. Implementing one functionality at a time, and completely testing it before moving to the next one helped me discover bugs early on in the development saving me a lot of problems that I would have to deal with later when the project got long and complicated.

Overall, this project has taught me to see the various facets of developing a full scale application. It has made me confident about my ability to go out there in the real world and build applications for a larger user base.

Thank you Professor Miller for making this fun!

14. Bibliography

Google, Inc. *Android Developers Site.* Google, 2017. Web. 15 Mar. 2017.

Kumar, Amruth. *Course Website.* Ramapo College, 2017. Web. 3 Mar. 2017.

Miller, Victor. *Course Website.* Ramapo College, 2017. Web. 22 Apr. 2017.

Stack Exchange, Inc. *Stack Overflow*. SE, 2017. Web. 18 Apr. 2017.