**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.

Threestones

Documentation

Vivek Pandey

SENIOR PROJECT

**Introduction**

**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.



**Key Features**

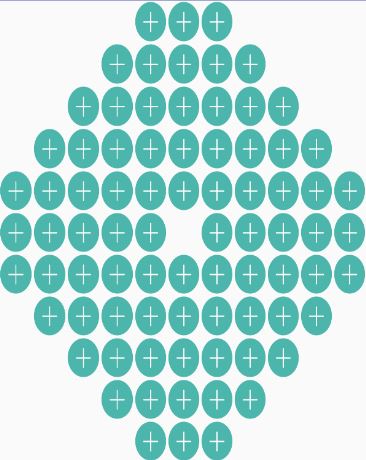
The following are some of the key features implemented in the game which definitely added challenges during development but now, upon completion, facilitate a better user experience. Their design and implementation will be touched upon in other sections later in detail.

* 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

**Game Play 101**

**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***.**

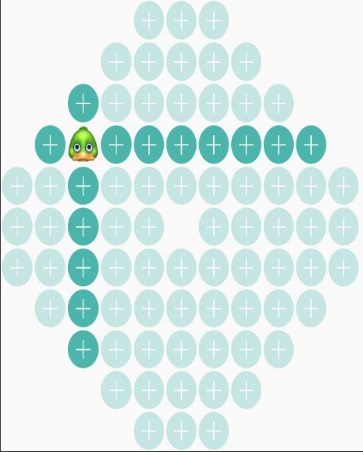


* **AND** both teams get 6 additional stones each stamped with Miller mascot.

**In short,**

**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.

**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
   1. Build towards a 3-stone arrangement to gain points.
      1. If the player has more than one incomplete arrangement, complete the one that earn the most points.
   2. 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**
   1. Player may want to hold on to Miller stones and use them only after running out of his own stones.
   2. Player may want to avoid playing a Miller stone when it may also benefit the opponent, as in examples illustrated earlier.

**Class Descriptions**

**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.

**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**

**XML Layout Files**

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.

**Computer’s Play Algorithm**

The computer uses the following algorithm to make a move.

1. Check if any Computer dice can capture opponent’s king or if computer king can get to opponent’s key square. If yes, use that dice and end the game.
2. Check if any opponent’s dice can get to Computer King or Computer Key Square. If yes,
3. Check if any of Computer’s dice can capture that hostile die. If yes, use it.
4. Check if any Computer dice can block that hostile dice. If yes, use it
5. Try to move the king if it is safe around
6. Check if any Computer’s dice can capture any of the other opponent’s dice. If yes, use it.
7. Check if any of the Computer’s dices other than the king are in threat of being captured. If yes, move that dice under threat to a safe location.
8. If none of the above finds a suitable move, calculate which Computer dice can safely get closest to the king in the next move and move it. This is helpful because the dice will keep safely revolving around the opponent king until a proper top value will come up for capture.

**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.

**Feature Report**

**Additional Features**

1. To make the process of restoring a saved game easier and to make this application completely free from text inputs, I have implemented a ListView rather than an input box to select restoration files. All the available files in the “Duell Data” folder are listed in the game interface and the user just has to tap on the proper name to restore the game. I found this to be way more user-friendly than asking the user to type the file name manually.
2. Another additional feature I have implemented is not necessarily a separate feature but rather a helpful add-on. To help the human player better understand the computer’s thought process, I have a “Bots Mumbling” series of messages that get displayed during computer’s turn. These short messages display the human player with the steps that the computer algorithm took in order to arrive at that move selection.
3. Also, to make the game more fun, I have added a sarcastic and scornful tone in the error messages that get displayed in the game. The notification messages, instead of being encouraging, are moderately derogatory intended to show the Computer’s contempt to the human species. Some of my favorites are:

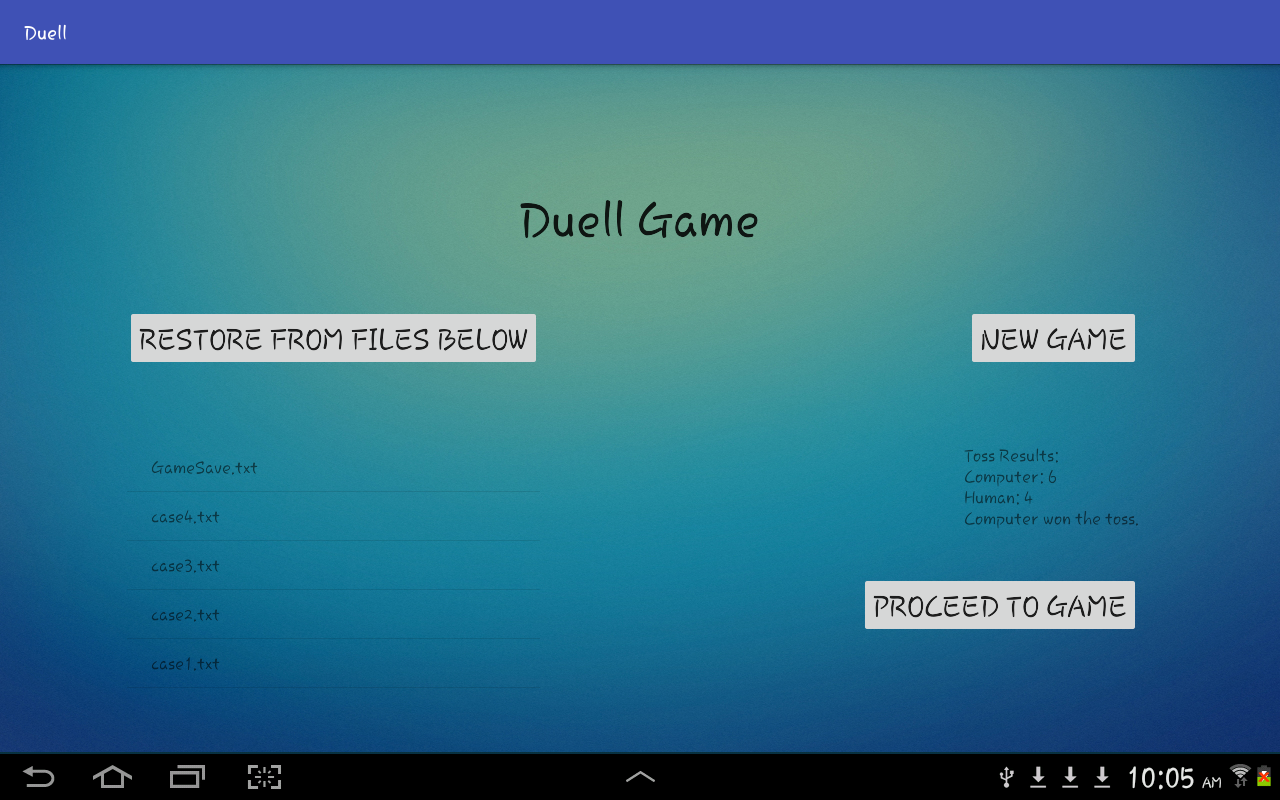
*“Congratulations! You won. Our programmer must’ve done a terrible job on algorithms for someone like you to win.”*

*“You just captured an opponent dice. Impressive for a Knucklehead? Eh!”*

**Missing Features**

None. However, the display of the Computer strategy is slightly different than the given specification. Instead of a single “Computer-chose-this-move-because” explanation, the game displays a series of steps that the computer’s algorithm took in order to choose its move.

**Screenshots**



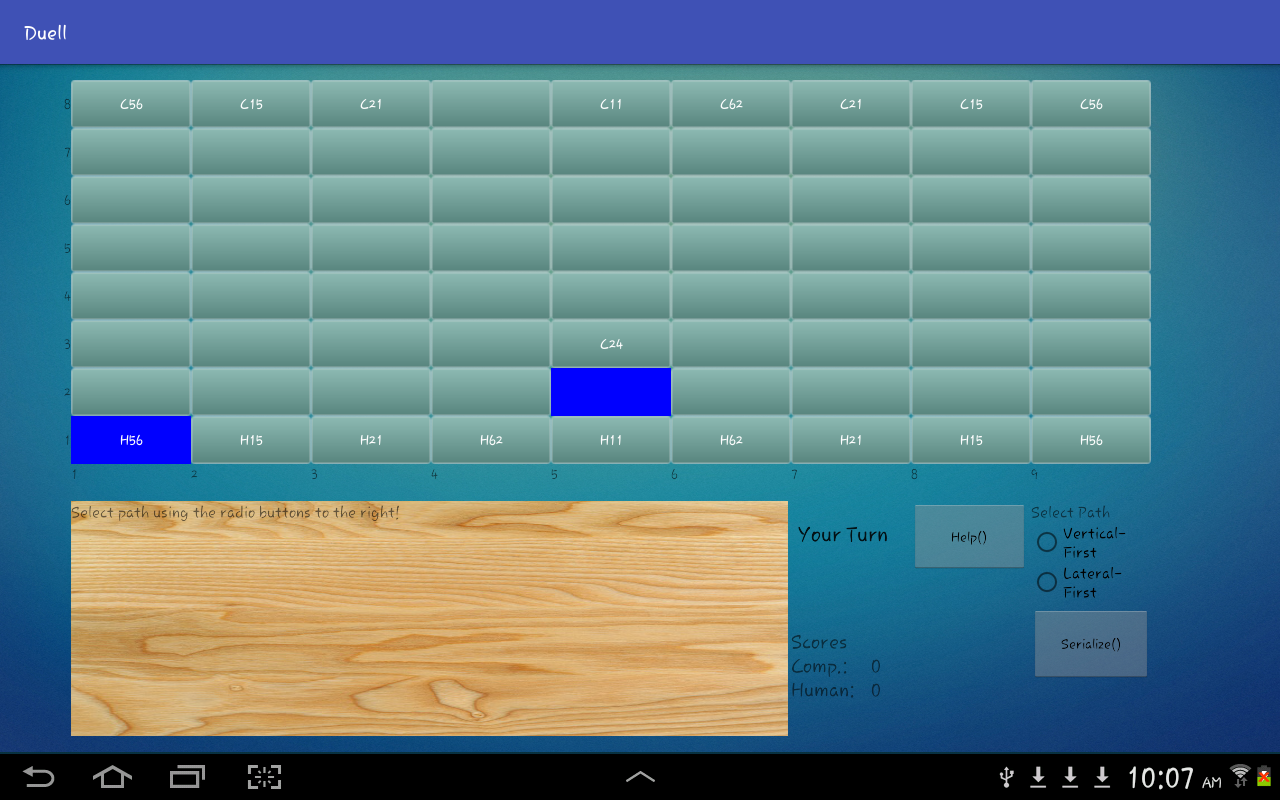
**Fig. Home Activity Fig. New Game**



**Fig. After Computer’s Move**

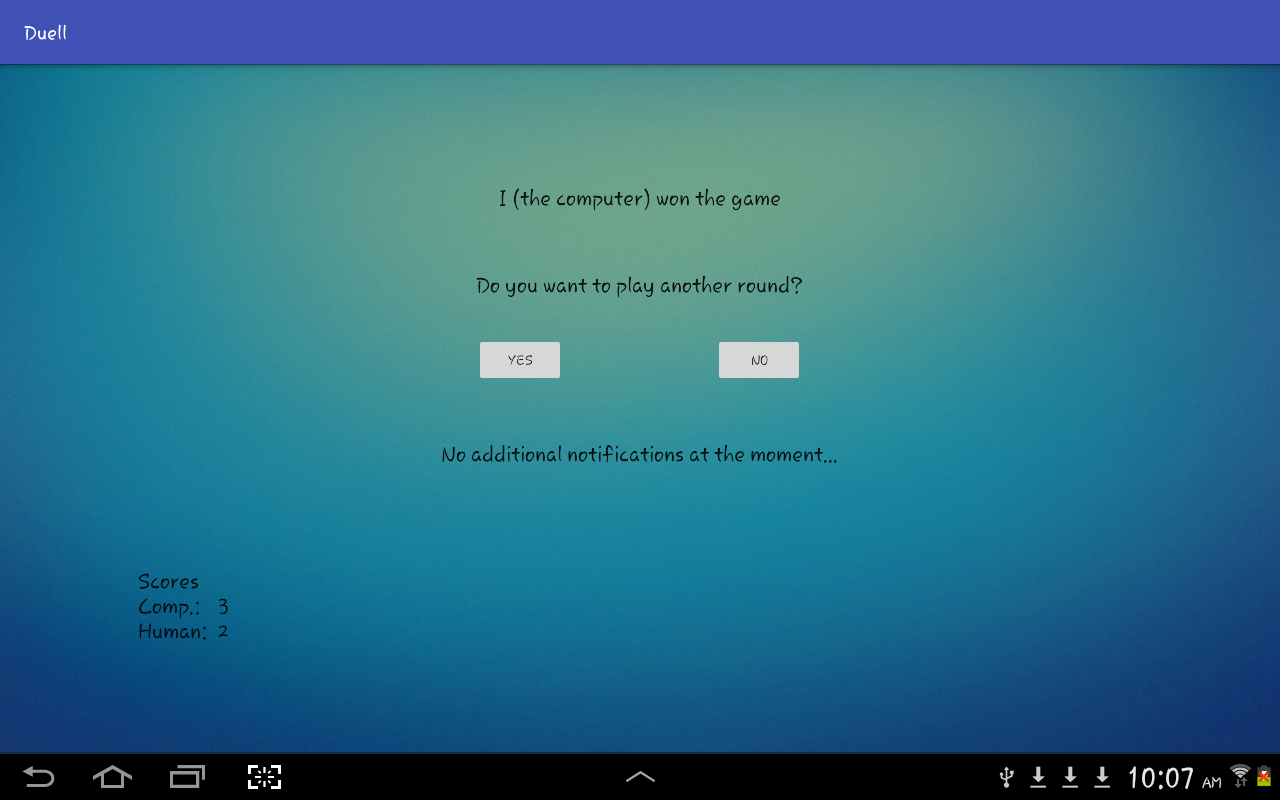


**Fig. Help Mode**



**Fig. Path Selection in 90 degree turns**

**Fig. Game Over**



**Fig. Results Activity**

****

**Fig. Ending Tournament**

**How to Play**

1. Open Android Studio (or Eclipse equipped with Android Development Tools)
2. Go to File > Open Project and select the main project folder.
3. Run using the Emulator or an Android Device (Preferably 10” and over)

**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.