Konane Manual

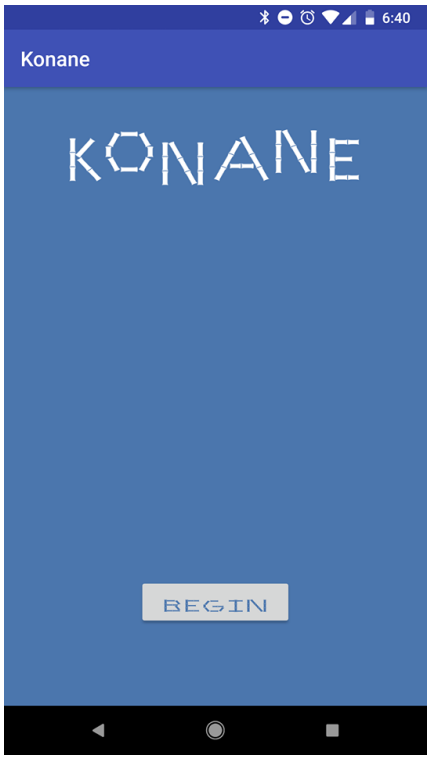
Neel Verma

January 2018

1 Activities

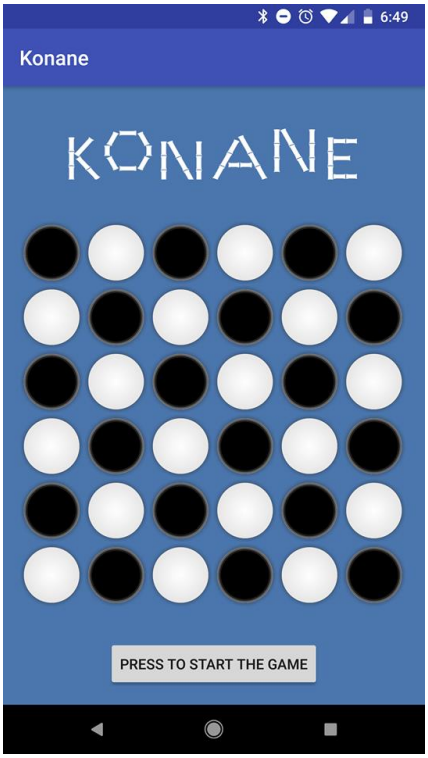
1.1 MainActivity

This activity is the welcome screen for my game. It displays the game's name (Konane) and has one button. This button is the one that must be pressed to begin the game. All of this is contained within a relative layout. After the "BEGIN" button is pressed, this activity will destroy itself and redirect to "BoardActivity". One other thing to mention is that this game must be played in portrait mode. Landscape mode is not available. This is so that the board will fit nicely on the screen.

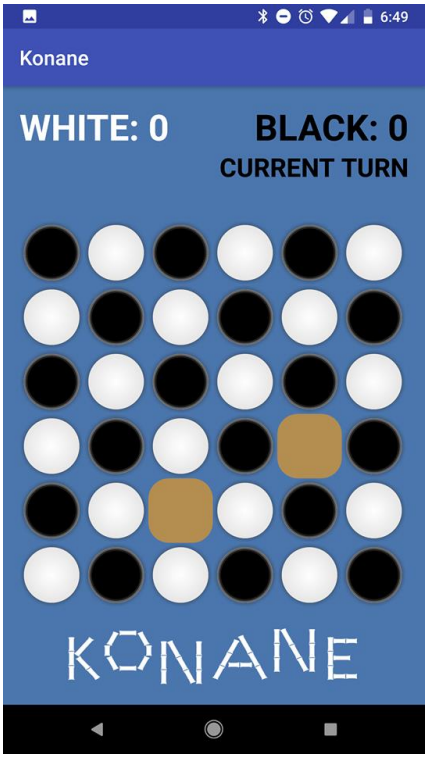


1.2 BoardActivity

This activity is where the game takes place. It handles all game logic (via the model) and applies changes in game state to the GUI. At first, it has a button that says to press it to begin the game.



As you can see, it starts with the initial state of the board, being that all 36 pieces are in order and none are removed. All of this is contained in a relative layout. The board itself is contained in a linear layout. The pieces are implemented as image buttons. In this present state, none of those buttons are enabled, meaning that the state of the board cannot be modified until the “PRESS TO START THE GAME” button is pressed. After it is pressed, the model will remove two random slots from the board, which will be reflected in the GUI. It also makes available the two players’ scores as well as the current turn. The name of the game was moved to the bottom so that the scores and current turn could be most easily read at the top. This can be seen on the next page.



After the game starts, the two users can start alternating turns until the game is over. All invalid moves are handled. The list goes as follows:

1) Trying to move from an empty space.

2) Trying to move a piece of a different color than yours.

3) Trying to move outside of the four given directions (up, down, left, right).

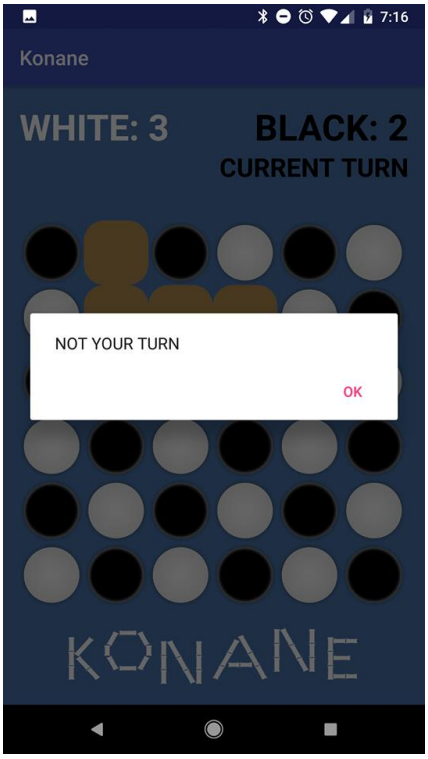
4) Trying to jump over your own piece, or over an empty space.

5) Trying to jump to a filled space.

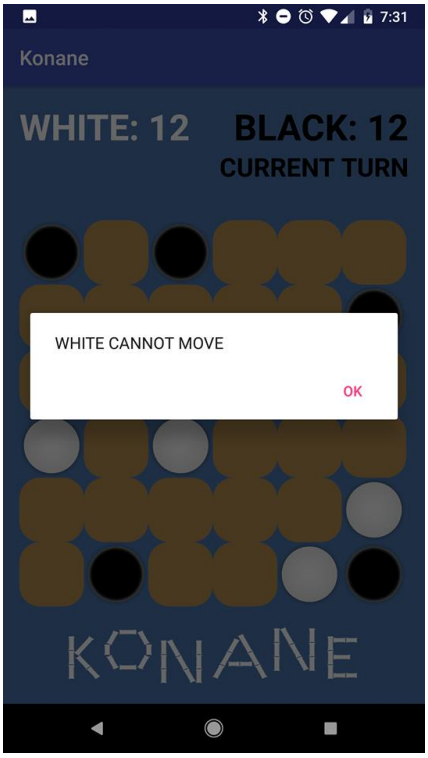
6) Trying to jump too far.

7) When in a multi-jump move, trying to move a piece that you were not already moving.

These are all communicated to the player(s) via dialog boxes. See below for an example.

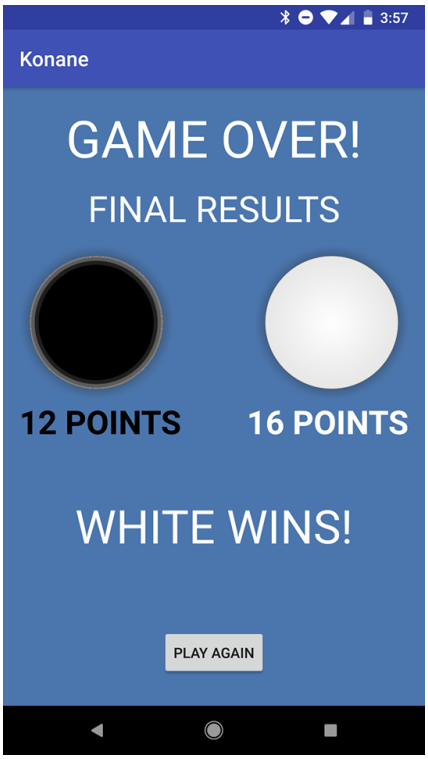


This shows that a player cannot move when it is not their turn. If one player cannot move but the other player still can, a dialog box will pop up telling that that player cannot move. In this instance, the turns are not switched. An example is shown below.



Now, when all moves have been exhausted and when both players cannot move, a dialog box pops up telling the user(s) to press “OK” to see the game results. This redirects to a new activity called “EndActivity”.

1.3 EndActivity



As you can see, the ending screen shows the scores of both players and tells who wins or if there was a draw. It then gives the option to play again if the user(s) wants to.

2 Model Classes and AI

2.1 Model Classes

- Slot: This class represents a single slot in the game board. It has a row, column, and a color. It provides methods to get and set the row, column, and color so that this information may be used to modify the game board.
- Player: This class represents one player of the game. A player has a color, a way of knowing whether it’s their turn, and a score. It provides getter and setter methods, as well as a method to add to the existing score.
- Board: This class represents the game board. The board is implemented as a 6x6 2D array. It has methods to set a slot (only used in the constructor), set a slot color, and get a slot.

- Game: This class enforces all game logic. It has 2 players, one playing black pieces and one playing white pieces, and a board object. It has a method to remove two slots, verify the validity of a move, make a move, determine whether a player can move, determine whether a player can make a successive move, and verify the validity of the possible successive move.

2.2 AI

- Breadth First Search: Used to find the next n available moves for a given player and board state.
- Depth First Search: Used to find the next n available moves for a given player and board state.
- Best First Search: Used to find the next n available moves for a given player and board state, while suggesting best moves first.
- Branch and Bound: Used to find the most optimal move for the given board and board state. It also shows the steps taken to make that move.

3 Bug Report

- I thought that all searches worked properly, but I had never tested it with the edge cases that were given, so all searches don't show the correct moves in the correct order for the given test cases.

4 Feature Report

4.1 Implemented

- Implemented all features as part of the project requirements.
- Allowed the user to save AND load the game from the saved file.

4.1 Not Implemented

- Implemented all features as part of the project requirements.

5 Log

- 2/12/2018 (4 hours): Implemented saving and loading to and from a text file.
- 2/13/2018 (12 hours): Implemented DFS, BFS, Best FS, and Branch and Bound, but they actually made the move instead of suggesting.
- 2/26/2018 (8 hours): Implemented DFS to suggest moves instead of making them and bug squashed.

- 2/27/2018 (8 hours): Implemented BFS to suggest moves instead of making them and bug squashed.
- 2/28/2018 (8 hours): Implemented Best FS to suggest moves instead of making them and bug squashed.
- 3/1/2018 (8 hours): Implemented Branch and Bound to suggest moves instead of making them and bug squashed.
- 3/2/2018 (2 hours): Implemented loading from a provided text file.

Total: 50 hours