Reversi Design Document

# Introduction

In this document I will show the design of my Reversi app. I will show how I have designed the UI elements and the structure of the application. I have tried to conform to the developer UI guidelines as closely as possible. I have changed timed mode slightly to the spec to be more like speed chess, in the sense that the timer stop and starts each time you take a move.

# The Look

Below is a diagram of how I would like the final product to look. The diagram shows how a player would get into a game, either Normal mode or Timed. If the player plays in normal mode the timers to the right of the player names in screen 3 just show “∞”. I added a “Show moves” button in case the user gets stuck, this can be turned off in the options menu (Screen 5).

## Following the guidelines

### Enchant Me

#### Let me make it mine

Allow the user to import names and pictures from contacts.

### Simplify my life

#### Keep it brief

Any modal views are kept short and simple, giving the user a choice of two options.

There are no long winded sentences in my app design.

#### Only show what I need when I need it

There is no “New Game” button on screen in screen 3; it is in the options menu.

#### I should always know where I am

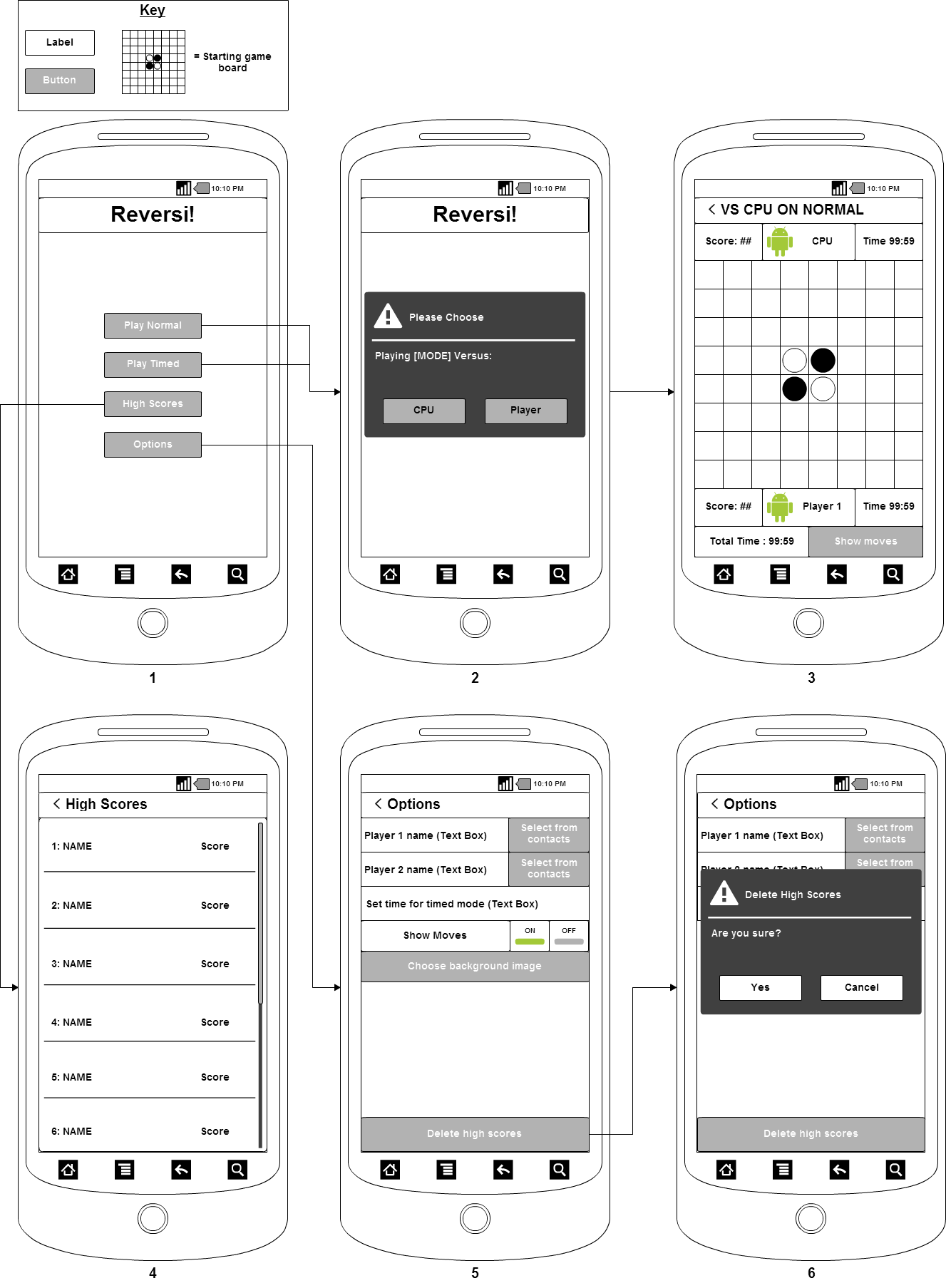
There is always a way to get back to the main menu in my app. When you move to another activity a bar along the top will hold a back button.

### Make Me Amazing

#### It's not my fault

If the user presses an invalid square the game will provide a simple Toast to say “Invalid Move” and they’ll be able to choose again.

## User interface diagram



# Structure

## Activities

There will be four activities in my game.

### MainScreen (UI Screen 1/2)

This Controls the main menu/title screen.

It is the default activity.

It has a premade XML file.

#### Methods

##### onCreate(Bundle)

This runs when the activity is created.

It creates an onClickListener for each button.

The “Normal mode” and “Timed mode” buttons are similar in that they produce a modal view in which the player chooses to play CPU or another player. They only differ in that they set a different game mode. Here is some pseudo code for the “Normal mode” button:



### GameScreen (UI Screen 3)

This controls the actual gameplay.

It has some premade XML, but the board is created when the activity opens.

Here is the pseudo code for this class:





### HighScores (UI Screen 4)

This will simply show a list of high scores using content providers.

For this I will make a content provider that stores the contact ID, name and score of the player. They will be ranked from highest score to lowest. I could also implement other high score tables such as ranked from amount of time they have left in timed mode.

I will use a database and content provider to create this. The screen will be a simple list that fetches and in order list of scores and names using SQL.

### Options (UI Screen 5/6)

The user can edit gameplay elements and UI elements from here.

Uses Shared Preferences and a simple list to save and retrieve data.

The preferences I need to track and there storage types are:

* Player 1 name - String
* Player 2 name - String
* Player 1 Contact Photo ID - String
* Player 2 Contact Photo ID - String
* Time each player is allowed in timed mode - String
* Whether they are allowed hints or not – Boolean

It is easier to store the ID number of the Contact and then the name separately as the user may want to change the name after they have imported the photo (e.g. the have a really long name).

The Activity will use a pre made XML file as there is no need for anything to be generated.

The delete data button will have a modal view popup to make sure the button wasn’t click accidently.

Everything other than the save button will be in a scroll view, this means the save button will stay glued to the bottom of the screen.

## Classes

### Cell

This is what the board[] contains.

It has a state of 0, 1 or 2. 0 is empty and 1 and 2 are player tokens.

It also has an isHinting Boolean. This is set by the show moves button.

#### Variables

Integer:

* State =0

Boolean:

* Hinting = false

#### Methods

##### isHinting()

Returns hinting

##### getState()

returns state

##### setState()

Sets State

### ContactPhotoLoader

This class simply helps loading in images so that code is not repeated 3 times across the high scores, game screen and options activities.

#### Variables

Bitmap:

* photo

#### Methods

##### Load(String id, ContentResolver cr)

Takes a contact ID and returns a photo as a bitmap.

### ImageAdapter

This converts the board[] into the gridView.

If a cell is hinting it overrides the current image and places a transparent glowing piece as the view to indicate that the player is allowed to move there.

#### Variables

Context:

* Context

Cell[]:

* Board

Integer:

* boardSize

#### Methods

##### ImageAdapter(Context c, Cell[] board, int boardSize)

Sets variables

##### getView(int position, View convertView, ViewGroup parent)



### Queue

A very simple Integer ArrayList that takes the first int added and removes it when .remove() is called.

#### Variables

ArrayList of integers called queue

#### Methods

##### add(int position)

Adds position to end of ArrayList

##### size()

Returns size of queue

##### Get(int index)

Returns value at index

##### Remove()

Removes the first element of the ArrayList

##### isEmpty()

Returns true if list has no elements