|  |  |
| --- | --- |
| University of Idaho | |
| KickShot Android App | |
| Team Kickin’ It | |
|  | |
| **David Wells** | **Coleman Beasley** |
| **12/10/2012** | |

|  |
| --- |
| **Software Design Document** |

**1. Introduction:**

* 1. **purpose**

This document is to inform continuing developers of this project’s current structure and design.

* 1. **Scope**

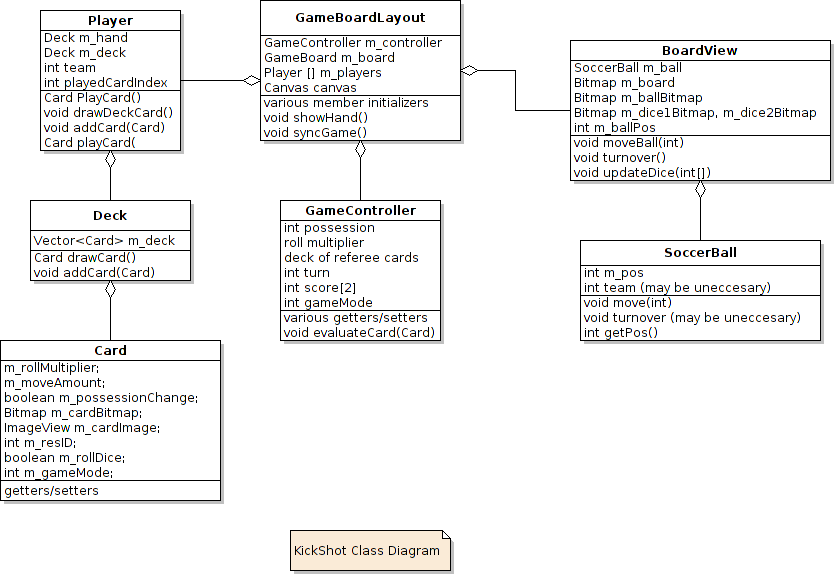
Kickshot Android is an app implementation of the Kickshot board game by Aziz Makhani. It is intended to replicate the board game experience as closely as possible, including the graphics. The game should be visually appealing and have playable performance on devices running Android 2.3 and above.

**2. System Overview:**

The app has two main screen modes: the main menu, and the game board, each have their own activity and associated XML file. Within these modes, interaction can occur on screen. These interactions are detected and applied to game data accordingly. The graphics on screen are loaded from the mode’s XML file. To manipulate graphics, data can make changes to the soccer field (BoardView canvas) and then load the soccer field image into the XML file for viewing.

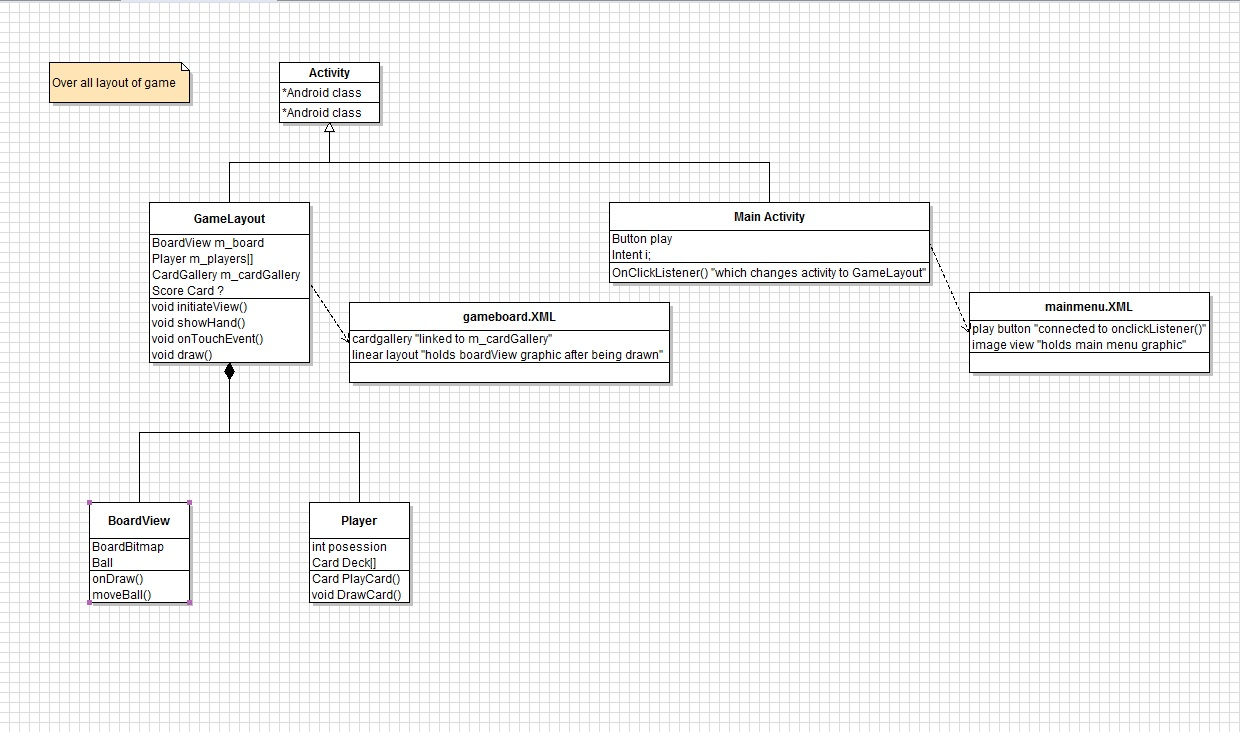
**3. System Architecture**

**3.1 Class Diagram**

****

Figure

**3.2 Class (Activities) Diagram**

****

Figure

**3.2 Design Rationale**

*Figure 1:* GameBoardLayout is considered as the main class in this game. The idea of the game structure is to capture interactions, set/retrieve data, and then manipulate graphics within the GameBoardLayout and BoardView classes. These graphics could be dice, a ball, or cards. The images are then blitted onto the BoardView and then the BoardView is added to the XML layout which can then be seen on the screen.

*Figure 2:*Because this app simultaneously makes use of buttons and canvas, this design seemed to be the optimal solution. Adding the canvas to the XML allows for Android buttons to be added on top of the game board. Without using the canvas in a layout, Android buttons cannot be utilized. A work around for this design would be to create custom buttons with custom button graphics; however this may add more complexity to the app.

**4. Component Design:**

**4.1 Activities:**

KickShot (Activity) contains: KickShot.java, activity\_main.xml

GameBoardLayout (Activity) contains: GameboardLayout.java, GameController.java, Player.java, Dice.java, Card.java, SoccerBall.java, Deck.java, BoardView.java, game\_layout.xml

**4.2 Classes:**

Card:

* parent class for each individual card type.
* contains mutators for each attribute that the card can modify
* contains the reference ID for its image
* its child classes use setters to set up their individual attributes
* each child's attributes are set according to the card descriptions on the rules sheet
* has an attribute that determines whether card can be played on offense or defense

Important Members:

* double m\_rollMultiplier: each individual card type sets its roll multiplier. The multiplier defaults to 1.
* int m\_moveAmount: the amount that each card should move the ball. If the ball requires a dice roll or a defensive card, the amount should be set to zero
* int m\_gameMode: sets the mode for the game. Cards that move the card up and down the field and change possession will have a game mode of 1. Shots on goal, and ref cards should change the game mode
* int m\_resID: holds the integer value of the card's image.

Important Methods:

Player:

* contains 2 Decks: the hand and the full deck of cards
* knows current team
* knows the card the user selected to play

GameController:

* controls the logic side of the game
* tracks game state
* score, possession, game mode, etc.
* evaluates the card the player plays, sends updated information to the BoardView
* contains Deck of referee cards
* all players draw from it

Important Members:

* int: m\_possession, m\_moveAmount, m\_gameMode, m\_numRefCards

These members are set in order to allow outside classes (i.e. the GameBoardLayout) to access them in order to synchronize them with the board.

* int[]: m\_roll, m\_score

These arrays are returned in order to synchronize the dice rolls on the game board, and to synchronize the game score.

Important Methods:

* void evaluateCard(Card)

takes a card and sets the controller's members based off of the card's properties. If the card calls for a dice roll, the controller will roll them and apply whatever card multipliers are necessary.

BoardView:

* is the layout of the game board
* contains the SoccerBall, and updates its position and possessing team

Important Members:

* SoccerBall m\_ball: SoccerBall object that is displayed on top of the board
* int m\_ballPos: current position of the ball. Synced with the SoccerBall member
* int m\_currentTeam: current team in posession of the ball (-1 for home, 1 for away). Used to set the SoccerBall to the correct color
* int m\_diceImages []: array of resIDs for the dice images.

Important Methods:

* void onDraw(Canvas): draws the board, ball, and dice images. This is called by the GameBoardLayout's onDraw(Canvas) method.
* void moveBall(int): increments the balls position (in pixels) by a number of slots\*(pixel differential). The pixel differential may need to be tweaked in future versions.
* void turnover(): switches possession of the ball, and reloads the ball's image with the new resID
* void updateDice(int []): uses the array parameter to update the dice images at the top of the screen. The value of the roll is used to index into the m\_diceImages array.

GameBoardLayout:

* main graphical class
* contains BoardView and GameController
  + syncs them so that the board matches the GameController's state
* contains the Players
* displays BoardView, buttons and the current Player's cards
* passes the Player's chosen Card to the GameController for evaluation

Important Members:

BoardView m\_board: Holds gameboard graphic. This is added to XML in onDraw().

GameController m\_controller: Synced with boardView

ImageView[] m\_cardImage: Holds cards for card gallery

Canvas m\_canvas: what we draw m\_board to

Important Methods:

Public void initializeCardGallery(): sets up all data necessary for card gallery to operate. Including onClickListeners for buttons. This syncs button click to an action.

Private void updateHand(): Sets on click listeners for each card in card gallery to make cards clickable.

Public void onDraw(): updates gameboard graphics to screen

Public void showHand(): displays player’s hand through the card gallery.

Private void syncGame(): gets dice and ball data from m\_controller.

**5. Interface and Graphics:**

Android KickShot puts much emphasis into interface and graphics. Below are interfaces that have been implemented in KickShot for Android.

**5.1 Screen Shots:**

**  **

**5.2 Interactions:**

**5.2.1 Buttons:**

We currently have 3 clickable buttons in this app:

* Play
  + In Main Menu XML
  + When clicked, switches to GameBoardLayout activity
* Don’t Play Card
  + In GameBoardLayout
  + When clicked, exits enlarged card and places focus back on card gallery.
* Play Card
  + In GameBoardLayout
  + When clicked, sets current selected card to selected card, exits card gallery

**5.3.2 Card Gallery:**

Another interface that has been implemented is the *card gallery.* The card gallery appears when the game board is touched. The card gallery is an Android Horizontal Scroll View that allows multiple images to be scrolled through and selected. The selection of a card triggers that card to stretch full screen using an animation.