Logical Operators: Chinese Checkers

Iteration One

February 5th, 2014

Curtis Smith

Peter Pobojewski

Saajid Mohammed

Chris Kellendonk

James Kostiuk

Kuba Subczynski

Ben Stitt

Taylor Whited

1 Table of Contents

2 Requirements

2.1 Use Cases

Use case templates ("Use Case Templates", 2014) provided by TechnoSolutions.

2.11 Create New Game

Revision History

Date	Author	Description of change
2014/01/18	Ben Stitt	Authored use case
2014/01/25	Ben Stitt	Added ID number

Use Case: Create a Game

Id: IT-1-UC-1

Description: The user wishes to start a new game of Chinese checkers. Display the necessary game options and allow the user to interact with them.

Level: User Goal

Primary Actor: Player User

Stakeholders and Interests: Other players may be seeking opponents.

Pre-Conditions: The user has started the game application, and wishes to play a game.

Post Conditions

<u>Success end condition</u>

The game creation screen is displayed, where the user is able to adjust specific game settings, such as number of users, and user names.

<u>Failure end condition</u>:

Game creation screen is not loaded, is not displayed correctly, or the user cannot interact with it properly.

<u>Minimal Guarantee</u>

In the event that any of the above failure conditions occur, the system should remain responsive, and the user should be able to exit the game, without affecting the operating system or files on the device.

Trigger: The user clicks the “New Game” button on the main menu screen of the game application.

Main Success Scenario

1. The user clicks “New Game”.
2. Game option controls are loaded.
3. Defaults are selected.
4. The user interacts with option controls to select their preferences.
5. The user confirms their options by clicking “Start Game”.

Extensions

This use case can be launched for both local and networked games. In the case of a networked game, the name entry fields need not be displayed.

Variations

In the case of a networked game, the user’s name can be retrieved from a stored profile.

Frequency: This use case should occur at the start of any game. Depending on how often the user plays, frequency may range from weekly to hourly.

Assumptions: The user can read English, and can interact through the use of a touch screen.

Special Requirements

The device must run Android and have the necessary space to install the game. If this use case is launched in relation to a networked game, the device must have access to the Internet.

To do

1. Proceed to display game board.

2.12 Display Game Board

Revision History

Date	Author	Description of change
2014/01/18	Ben Stitt	Authored use case
2014/01/25	Ben Stitt	Added ID number

Use Case: Display game board

Id: IT-1-UC-2

Description: A new game has been created, and the user would like to see the game board.

Level: User Goal

Primary Actor: Player User

Stakeholders and Interests: The player's opponents need the player to be able to interact with the game.

Pre-Conditions: The user has already used the game creation screen, and has confirmed their options.

Post Conditions:

Success end condition

The game board graphics are being displayed to the screen.

Failure end condition:

Game board is not displayed, game board is displayed improperly, or game crashes.

Minimal Guarantee

In the event that the game board is not displayed, or is displayed improperly, the system should remain responsive, and the user should be able to exit the game, without affecting the operating system or files on the device.

Trigger: The user clicks the "Start Game" button on the game creation screen.

Main Success Scenario

1. The user clicks “Start Game”.
2. The game board module is loaded.
3. The game grid is generated, and the pegs are placed on the grid.
4. The graphics for the game board, the pegs, and the necessary user controls are rendered and displayed.

Extensions

As different numbers of players may be selected, the pegs generated must have the correct number, colours, and positions.

In addition, this use case may also be triggered by loading a local or online game, or joining an online game. As such, the colours and positions of pegs may be recovered from existing records.

Frequency: This use case should occur at the start or resumption of any game. As such, it may occur once after loading the game, or many times. Depending on how often the user plays, and how many games the user plays at once, frequency may range from weekly to every few minutes.

Assumptions: The device is capable of computing the game grid and displaying the game board.

Special Requirements

The device must run Android and have the necessary space to install the game. If this use case is launched in relation to a networked game, the device must have access to the Internet.

Issues

1. How is game grid represented?
2. How are graphics generated?

To do

1. Also allow user to take their turn.
2. Pass play control to the next player.

3 Planning

3.1 Team Capacity Calculations

Release Level Plan

Start	9-Jan
Finish	27-Mar
Last Day of Class	4-Apr
Work Days	77
Work Weeks	11
# of People	8
Hours/Week/Person	6
Total Hours	528

Iteration Level Plan

Iteration #	**# Weeks**	**Hours**
1st iteration	4	192
2nd iteration	3	144
3rd iteration	3	144
4th iteration	1	48
	Total Hours	528

4 Design

4.1 Flow Control Diagrams

4.11 Hot Seat

- See Appendix A

4.12 Game Event Process

- See Appendix B

4.2 Class Diagrams

4.21 Class Method Description

Class	Function	Description
MainActivity	# startHotSeatGame()	Fired when the user clicks the startHotSeat button. This will launch the HotSeatSetupActivity intent.
HotSeatSetupActivity	- numPlayersChanged()	This occurs when the user selects a different option from the radio group to select players. This will update the number of text input boxes that are visible for players to enter their names.
	# getNumPlayers() : int	Returns the current number of players that will be playing the game as selected by the user.
	# startGame()	Fired when the user clicks the startHotSeat button. This will launch the GameBoardActivity as an intent.
GameBoardActivity	# acceptMove()	Fired when the user clicks the acceptMoveButton. This will send a message to the actual game board logic indicating the players has decided and made there move.
	# resetMove()	Fired when the user clicks the resetMoveButton. This will undo any moves the user has made on the board. Putting them back in the start state for their turn.

4.22 Game Board UI Engine

- See Appendix C

4.23 Game Board Engine

- See Appendix D

4.24 User Interface Component

- See Appendix E

4.25 Hot Seat Interface

- See Appendix F

4.3 Architecture Design

4.31 Game Drawing Sequence Diagram

- See Appendix G

4.4 User Interface Design

4.41 Initial Sketches

- See Appendix K

4.5 Visual Design

4.51 Introduction

The purpose of this document is to analyse and describe why and how Joint Venture styled the Chinese Checkers interface for iteration one of the application. All visual designs and styles were created after an agreed upon user interface was created for the application. The iteration one visual design process will be the most intensive out of all of the iterations of the project because future iterations will be designed based upon the core specifications described in this document.

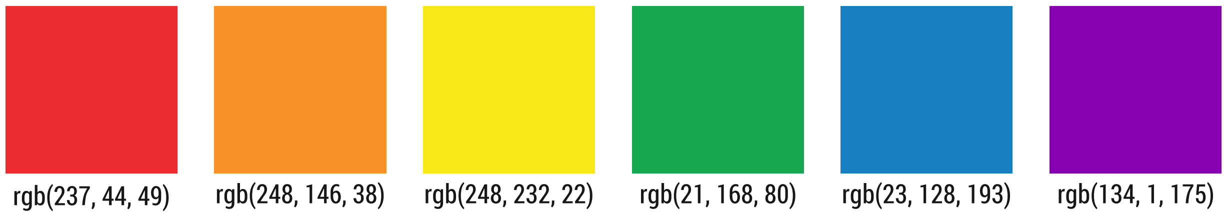
4.52 Textures

Less is More

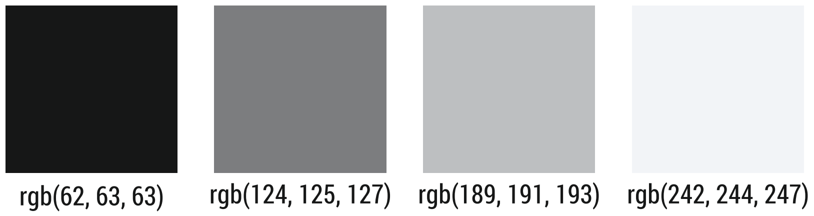
Trends observed in both mobile and web-based applications in 2013 include: simplicity, minimalism, and less is more ("10 Web Design Trends for 2013", 2013). The Chinese Checkers application was designed with minimalistic and flat textures to ensure that the screen was not overwhelmed while being targeted for smaller devices. From the modern fonts to simplistic images – all have been chosen and designed to be fit with restrictive colour palette to achieve our flat design.

4.53 Colour Swatches

Primary Palette



Monochrome Palette



Chosen Swatches

Multiple swatches were produced for each palette: Primary and Monochrome. The current palettes chosen are a combination of the most attractive swatches from the original batch. The Primary Palette is used on game tokens such as players and pegs. The rainbow scheme should exist in every activity either in elements of the layout or in a graphic appropriately placed in the activity. The Monochrome Palette is used for the core application colours; from left to right: background, empty peg slot, extra, and text.

Rejected Swatches

As described above, multiple swatches were rejected to come up with the current palettes. Rejected swatches may be revisited in future iterations, but are not significant enough to this iteration to include in this design document.

4.54 Logo Design

Name

The name of the application is “Chinese Checkers” and, as such, it is the main text in the logo design for the application. The text is in the font Roboto Condensed (see Assets: Typography section for more information on fonts) and is left aligned.

Image

Included in the Chinese Checkers branding is a graphic that represents the six potential players in the six primary colours from the Primary Palette. The squares are aligned in decreasing, diagonal order because a North American user would read from left to right and top to bottom. This image will appear in presentations and other documents where the full brand logo is an inappropriate size.

CHINESE
CHECKERS

On a White Background

CHINESE
CHECKERS

On Application Background

4.55 Assets

Typography

In choosing a font, the team agreed to stay true to a flat and minimalistic style that encompassed the application. The font family Roboto by Android was introduced in Ice Cream Sandwich ("Typography", 2014) and had all of the characteristics of a modern font we required. Roboto comes in two flavours: Normal and Condensed; as well as multiple font styles such as bold, italic and medium. The font in the logo is Roboto Condensed, while the rest of the application is in Roboto Normal.

Iconography

An icon as defined in this document is a graphical representation of an action, status or application. Initially the choice of icon pack to use was the default Android Action Bar Icon Pack, but, after a team discussion, it was decided to use the Font Awesome, MIT licensed icon pack instead. Attribution to Font Awesome will be given in an About section.

The icons chosen are minimalistic, monochromatic images that can scale smoothly from mdpi to xxhdpi resolutions. The colour of the icons is Light Grayish Blue rgb(242, 244, 247) which is consistent with our chosen colour palette. (See the Colour Swatches section for more information on colours.)

Roboto Normal

1 2 3 4 5 6 7 8 9 0
A B C D E F G H I
J K L M N O P Q R
S T U V W X Y Z , .
a b c d e f g h i j k l
m n o p q r s t u v w
x y z ! @ # $ % & ( )

Android Action Bar Icon Pack

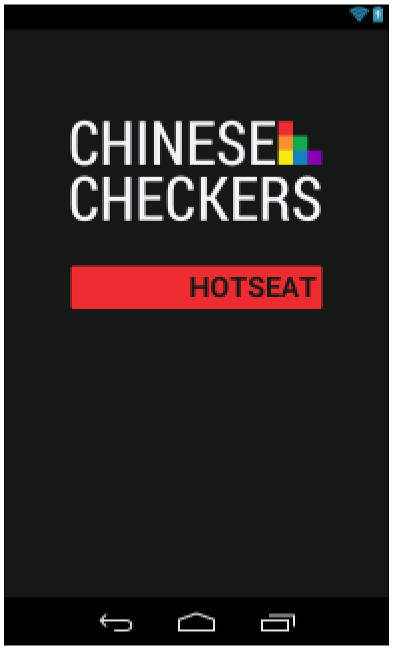
⚠ ✓ ✗

Font Awesome

⚠ ✔ ✖

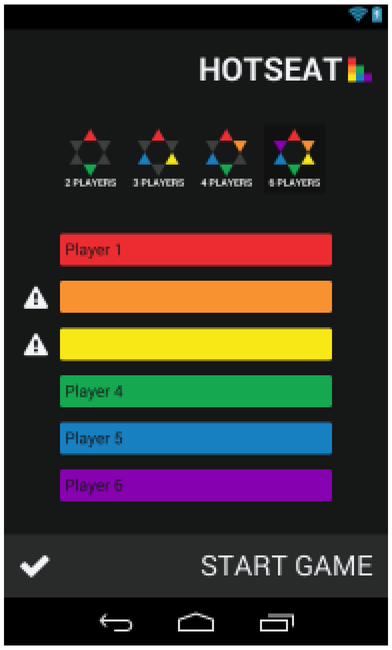
4.56 Mock-ups

Main



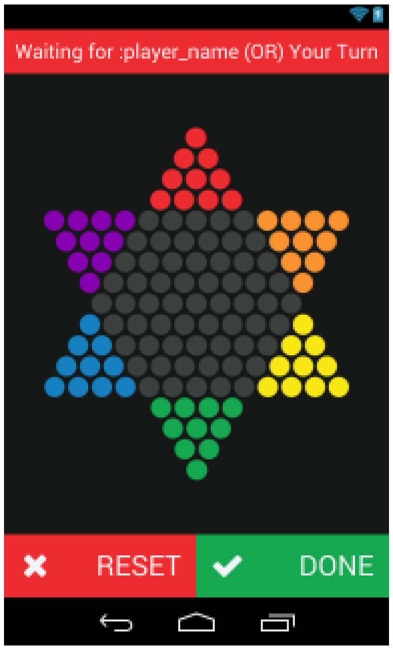
See Appendix H for full-scale mock-up image

Hot Seat Configuration



See Appendix I for full-scale mock-up image

Hot Seat Game



See Appendix J for full-scale mock-up image

Taking into account the design requirements specified in this document, three mock-up images have been produced that display the layouts of the application for iteration one as accurately as possible:

- The Main mock-up portrays the first screen that a user will see when the application is launched.
- The Hot Seat Configuration page will be accessed from the Hot Seat button in the Main mock-up.
- After Start Game is pressed in the Hot Seat Configuration mock-up, the user is displayed the game activity which is portrayed in the Hot Seat Game mock-up.

4.6 AI Design

Details

Levels of AI to Be Created: 3

Naming Convention of AIs:

- Easy, Medium, Hard
    - Aliases: Sentinel, Skynet, HAL9000

Summary of AI Strategies

Easy: A greedy algorithm based AI purely using vertical displacement from the goal state as its sole heuristic.

Medium: Based on the easy AI, the medium difficulty will add minimax algorithms to the heuristic, along with new heuristics such as horizontal displacement from the center of the board, back piece weight, and split.

Hard: Using the heuristics from the medium AI but extending them to thinking a move in advance and taking the best move this turn for the most optimal overall board state next turn. The hard AI will use a depth first search and alpha-beta pruning to reduce response time

Heuristics

Vertical Displacement: The main objective is to win in a game by getting all pieces from one side of the board to the other. This heuristic prevents the AI of using the strategy of keeping at least one piece in an opponent's goal zone to prevent them from winning. It also will drive the pieces to the other side of the board to attempt to beat the opponent(s).

Minimax Algorithm: There will be an algorithm adding to the heuristic value of each game state by attempting to maximize the board state for the AI itself while giving the least optimal potential moves to the opposing player(s).

Horizontal displacement: Keeping pieces in the center of the board minimizes the amount of moves required and increases the opportunities to jump opponent's pieces.

Back piece weight: If all except one piece is in the goal state, that piece will take significantly longer to get there by sliding across the board each turn. By adding this heuristic no pieces will be left behind to cause increased completion times.

Split: Pieces move faster while jumping each other. If there is a large gap in a player’s pieces, it is commonly more beneficial to close that gap first so that all pieces may reach the goal state faster.

Optimization

Advance only: With the split heuristic in place it becomes important to add this restriction to prevent pieces from moving toward the initial state just to ‘save’ a friendly piece. Only moves which advance towards the goal state will be measured, others will be rejected.

Take shortest path to win: If a move this turn has the same result as the optimal move next turn, use the shortest depth path to it.

More optimization will come during implementation and testing.

5 Testing

5.1 Iteration One Testing Plan

Revision History

Date	Author	Description of change
2014/01/20	Saajid Mohammed	Authored testing plan
2014/01/22	Ben Stitt	General revision and editing

5.11 Introduction

The program to be tested is a framed out version of a Chinese checkers game for android. It contains three activities, the first with a button leading to the second, the second activity is a configuration screen for the game you wish to play it contains the options to set the number of players and their names it then launches the game activity which for the current phase contains a canvas with the game board drawn on it and buttons to confirm a players move and undo a move.

5.12 Objectives

Verify the current phase programming is functional and works consistently.

5.13 Testing Phases

Unit Testing

Each activity shall be tested separately

- MainActivity
  - hotseatConfigurationActivityButton
    - Verify isClickable
    - Verify proper methods are being called
    -
- HotSeatSetupActivity
  - hotseatTwoPlayerButton
    - Verify isClickable

        - Verify proper methods are being called
    - hotseatThreePlayerButton
        - Verify isClickable
        - Verify proper methods are being called
        - Verify 3 input fields are presented
    - hotseatFourPlayerButton
        - Verify isClickable
        - Verify proper methods are being called
        - Verify 4 input fields are presented
    - hotseatSixPlayerButton
        - Verify isClickable
        - Verify proper methods are being called
        - Verify 6 input fields are presented
    - hotseatRedPlayerNameEditText
        - Verify accepts text
        -
    - hotseatGreenPlayerNameEditText
        - Verify accepts text
        -
    - hotseatPurplePlayerNameEditText
        - Verify accepts text
        -
    - hotseatBluePlayerNameEditText
        - Verify accepts text
        -
    - hotseatYellowPlayerNameEditText
        - Verify accepts text
        -
    - hotseatOrangePlayerNameEditText
        - Verify accepts text
    - hotseatGameActivityButton
        - Verify isClickable
        - Verify proper methods are being called
        - Verify player name variables are being bundled to be passed
        - Verify intent is being generated and intent is being destroyed
- GameBoardActivity
    - Verify variables are being unwrapped from bundle
    - hotseatMoveResetButton
        - Verify isClickable
        - Verify proper methods are being called
    - hotseatMoveDoneButton
        - Verify isClickable
        - Verify proper methods are being called

5.14 Integration Testing

- Home Activity and Hot Seat Configuration Activity Joined
  - Unit Tests are redone
  - Verify transition between activities are functional
- Home Activity and Hot Seat Configuration Activity and Hot Seat Game Joined
  - Unit Tests are redone on each activity
  - Verify transitions between activities are functional

5.15 Performance and Stress Testing

Performance

N/A

Stress Test

N/A

Regression Testing

N/A

Ease of Use Testing

A focus group shall be assembled and given access to the functioning application they will also be given a set of tasks to accomplish. Each member of the focus group shall then fill out an Ease of Use Testing Form (Appendix M). The forms will then be tallied and actions shall be taken in accordance with the results.

Acceptance Testing

Criteria are to be determined by team leader.

5.16 Testing Feedback Procedure

At the end of each test phase or immediately following a failure of an in phase test, a Test Feedback Form (Appendix L) shall be generated. The feedback form will be generated as an issue on GitHub with the contents of the Appendix L. Further action shall be determined on case bases. However failures for unit tests at the developer level are not required to generate a test feedback form, only passes specified in this document at the unit test phase are required to be documented with a test feedback form.

5.17 Features To Be Tested

- MainActivity
  - Button Functionality
  - Transition To Hot Seat Configuration Activity
- HotSeatSetupActivity
  - Buttons Functionality
  - EditText Functionality
  - Transition To Hot Seat Game Activity
- GameBoardActivity
  - Buttons Functionality

5.18 Features Not To Be Tested

N/A

5.19 Dependencies

- Availability of classes and modules for current phase.

5.110 Tools

- Eclipse
- JUnit API
- Android Testing API

5.111 Approvals

Name	**Project Role**	**Signature**	**Date**
1. Curtis Smith	Project Lead		
2. Peter Pobojewski	Deputy Lead		
3. Ben Stitt	Documentation Lead		
4. Saajid Mohammed	Test Lead		

5.2 Integration Testing

See Appendix N

5.3 Acceptance Testing

Revision History

Date	**Author**	**Description of change**
2014/02/05	Curtis Smith	Initial version

5.31 Introduction

This document outlines the methods and testing procedures for all acceptance testing for the Chinese Checkers application designed and developed by the COSC 3F00 team known as Logical operators. Each test has two possible outcomes: pass or fail. Any failed test is reason for the entire test sequence to be failed.

5.32 Project Description

We are creating a version of Chinese Checkers. Our target audience will be users that are 6 years of age and older. The game will be played with both human and computer players. Our target platform will be Android phones (original launch on OS version 4.0, API version level 14). If there is time we will also consider an implementation that can be used on suitable tablet platforms.

We will be developing the solution using Android Studio using GitHub as our source code repository.

Graphics will be created using OpenGL.

Networking will be accomplished using the internet via HTTP and centralized on a web server using Node.js. Heavy processing for the AI will be executed on the web server so as to limit the processing power needed on the user's device. JSON will be the preferred method to exchange data between the Node server and Android clients.

The database solution will be MySQL.

We will implement the game as a "push game" -- where users are notified when it is their turn. This will add the option of playing a game over multiple sittings, while maintaining the ability to a play fast-paced game.

All technical documents and project tracking will be available from the GitHub wiki and service hooks available from this tool.

5.33 Test Team Personnel

The test team consists of one VENDOR NAME tester and one primary customer witness who have the authority to sign off tests. Optionally, a small, agreed number of additional customer observers can observe the tests and input their observations to the primary witnesses.

Name	**Role**	**Team**
Curtis Smith	Team Leader/Tester	COSC 3F00 Logical Operators
Saajid Mohammed	Test Lead	COSC 3F00 Logical Operators

5.34 Glossary

Term	Definition

5.35 Deliverables

Hardware

The following hardware items must be delivered fully inspected and functional.

Quantity	Deliverable	Pass / Fail	Model/Part #
n/a	n/a	n/a	n/a

Software

The following software items must be delivered fully inspected and functional.

Quantity	Deliverable	Pass / Fail	Product/Part #
n/a	n/a	n/a	n/a

5.36 Acceptance Test Plan

Structure

Each test process involves COSC 3F00 Logical Operator personnel running the test procedures described in this document to the satisfaction of the team.

Order of Tests

The tests in the following section are listed in the same order as they should be performed during acceptance testing. This is to help give a logical flow of work on the system. The order of the tests is designed to minimize the use of the same elements of the system at the same time.

Testing

All tests taking place during acceptance testing will be outlined in this section. The following table is an example. Columns that are colored grey mean that that particular test will not occur during that test phase.

#	Test Description	Expected Result	Phase 1	Phase 2	Phase 3	Phase 4
1	Action 1 performed...	Expected Result...				
2	Action 2 performed...	Expected Result...				

Software Testing

Each software component provided by Logical Operators must be tested and perform to the expected standards outlined in the table below. It is expected that several bugs or unexpected behaviors may occur during testing. Any bug, unexpected behavior, or missing functionality will be documented as a deficiency. Each software deficiency must be resolved and tested prior to sign-off unless agreed upon in writing by both parties.

#	Test Description	Expected Result	Pass	Fail
1	Touching the game logo icon from the main apps screen with the device oriented right side up.	The application will launch and come to rest on the HOTSEAT screen and will match the design mock-up “main.png”	X	
2	Touching the game logo icon from the main apps screen with the device oriented upside down.	The application will launch and come to rest on the HOTSEAT screen and will match the design mock-up “main.png”		X

3	On the HOTSEAT screen, touching the HOTSEAT button	The HOTSEAT game configuration screen will be displayed and will match the design mock-up “hotseat_config.png” but will be defaulted to a 2 player game with the input text field for player 1 in focus and the 2 players game selection icon will appear in “selected” mode.	X	
4	On the HOTSEAT game configuration screen, touching the 3 players game option icon	A total of 3 Player input text fields will be displayed and the 3 player game selection icon will appear in “selected” mode.	X	
5	On the HOTSEAT game configuration screen, touching the 4 players game option icon	A total of 4 Player input text fields will be displayed and the 4 players game selection icon will appear in “selected” mode	X	
6	On the HOTSEAT game configuration screen, touching the 6 players game option icon	A total of 6 Player input text fields will be displayed and the 6 player game selection icon will display in “selected” mode.	X	
7	Touch the “START GAME” button on the HOTSEAT game configuration screen with at least one of the player name input fields left blank.	Validation of the player name input fields should prevent the application from navigating to the HOTSEAT game screen. All empty player name input fields should be marked with an alert icon to indicate missing data	X	

8	Touch the “START GAME” button on the HOTSEAT game configuration screen with all the enabled player name input fields populated with data.	The application should navigate to the HOTSEAT game screen which will match the design mock-up hotseat_game.png. The name and color of the first player will be shown at the top of the screen. The RESET and DONE buttons will be enabled	X	

5.37 Deficiencies

Issues

#	Test # / Sequence	Description	Comments
1	2	The HOTSEAT screen did not reorient itself and was displayed upside down	Ideally the application will reorient itself to display right side up at all times.

Action Plan

Deficiency	Action Plan
1	A defect will be opened in GitHub and the feasibility of allowing reorientation of the application will be discussed by the team and a decision will be made on how best to proceed.

5.38 Final sign-off

Acceptance Testing

The Acceptance Testing for Chinese Checkers prepared by Logical Operators for COSC 3F00 was

ACCEPTED / ACCEPTED WITH DEFICIENCIES / REJECTED.

Tested by:	**Date:**
Curtis Smith	2/5/14

5.4 Ease of Use Testing

5.41 Tasks

Task 1: Launch game

Task 2: Get to a new game setup screen

Task 3: Set up a game for two players using random names and start it

Task 4: Set up a game for six players

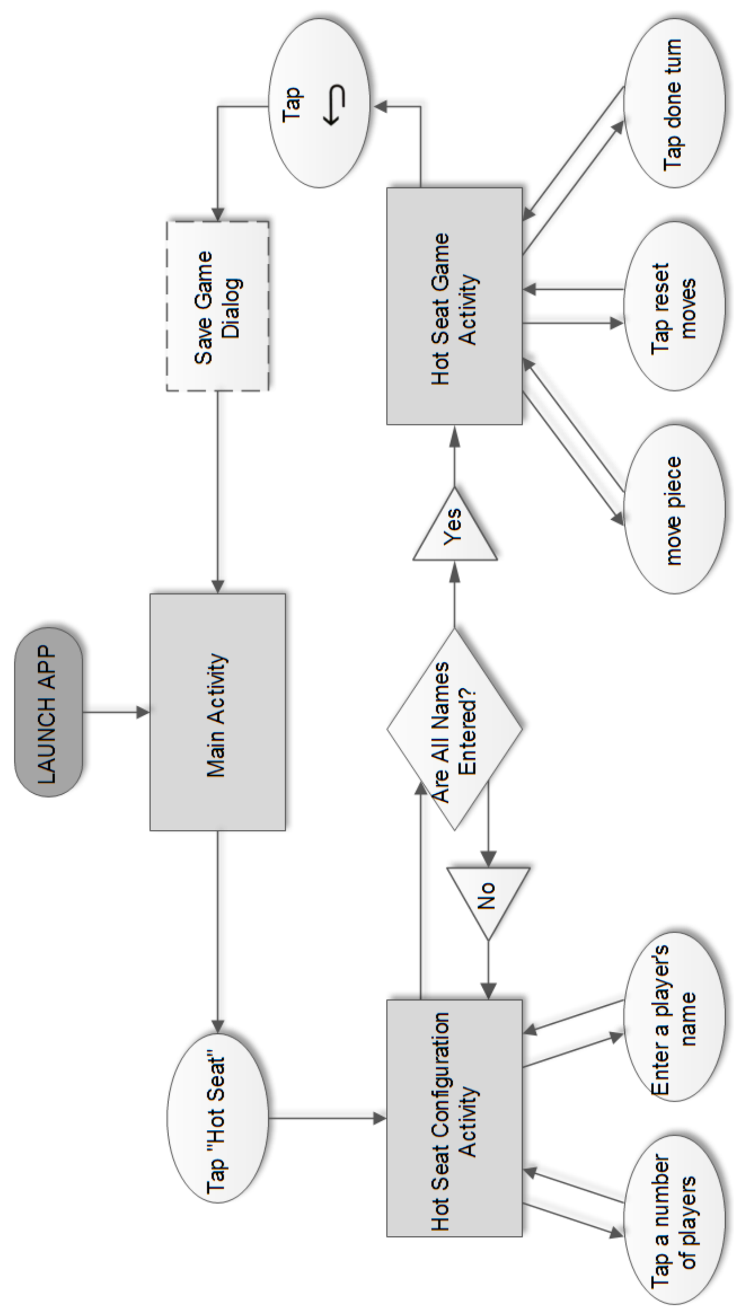
1. Put your name for Red
2. Put Curtis for Blue
3. Put Ben for Purple
4. Put Kuba for Yellow
5. Put Chris for Orange
6. Put Taylor for Green
7. Start the game

5.42 Ease of Use Task Results

See Appendix 0

6 Appendices

6.1 Appendix A



Hot Seat Flow Control

6.2 Appendix B

Game Board Engine

The Game Board Engine runs as an event driven system. The engine is made of three main components: the Game Activity, the Game State Manager, and the Game Board UI Drawing Engine. When the local user makes a move the game state is updated and then the game state notifies the drawing handler callback (implemented in the Game Activity) to redraw the board. The handler will then call the Drawing Engine to redraw the board. In this sense the drawing engine is a “dumb/slave” system. It doesn’t actually know about the state of the game, it is just given command to move pieces and do other visual activities.

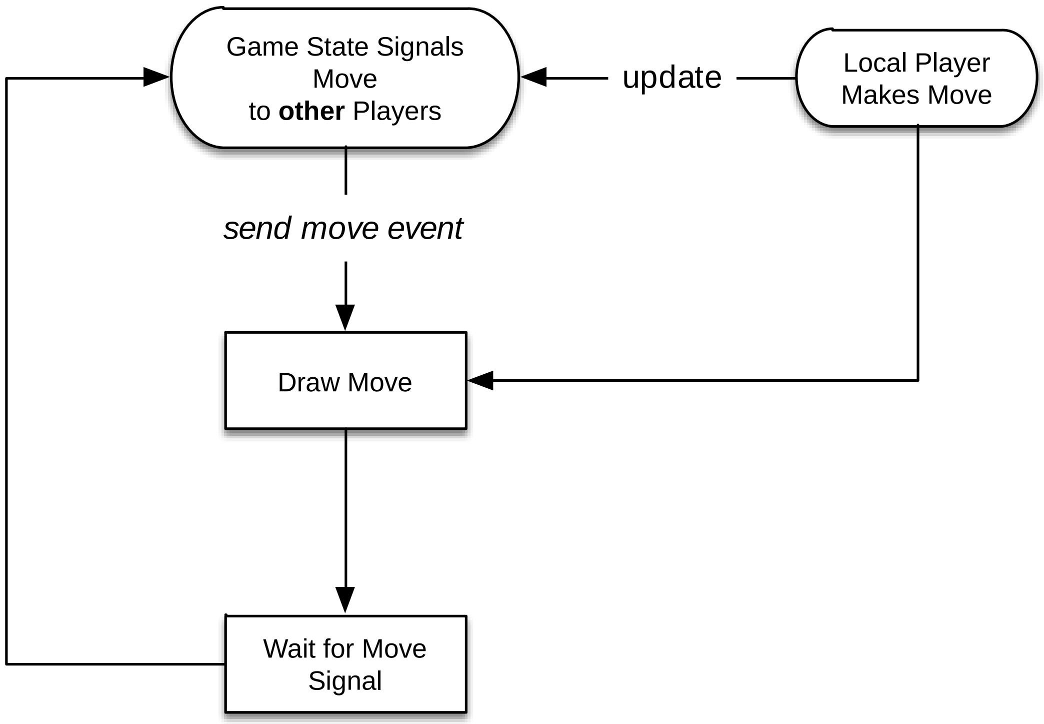


Figure 5.21 – Event flow when making a move.

Game State Manager

The game state manager manages making sure the state of the game board stays in sync with all other players playing the game. For network-based games this system will involve communicating with the server to send and receive moves to other players. The Hot Seat game play mode which works completely locally works by pretending that it is communicating with the server and handling the move events the same way. Thus those events can be forwarded to the Game

Activity without the Game Activity needing much modification to work for both local and network based games.

Game Board UI Drawing Engine

The drawing engine receives commands telling it where to move pieces or what to draw from the Game Activity. It then executes those visual commands and redraws the board to reflect the changes.

This engine is also able to detect touch events on any visual element on the game board. Thus the Game Activity can register to be notified when a piece is touched.

The drawing engine expects that all position related information be given to it in row-index form starting at 0. There is an interface called “Position” that should be implemented and follow these constraints.

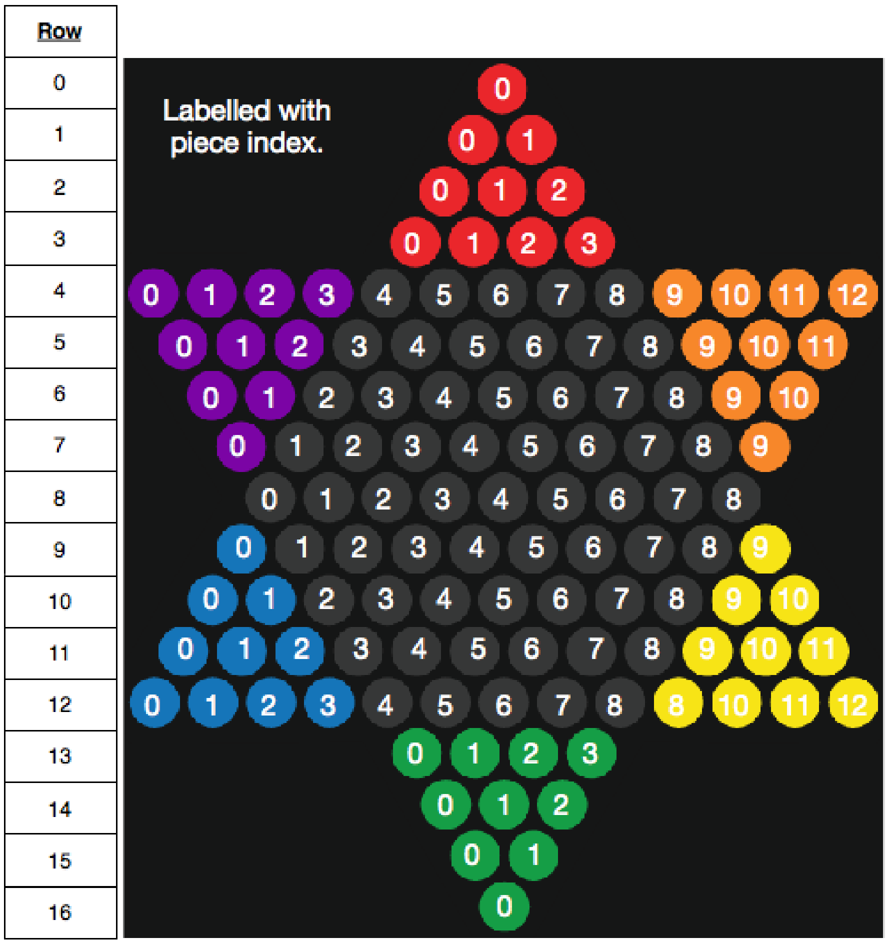


Figure 5.22 – The Row-Index Format

Game Activity

The Game Activity that holds the game board UI is responsible for handling all the intermediary actions to with updating the state of the game board in the Game State Manager and drawing the state onto the screen. This activity knows about the Game State Manger and the Game Board UI Drawing Engine. Using these the activity can receive events from the state manager and then signal to the drawing engine to move or redraw the board based on whatever state change occurred. The game activity can also signal any other drawing event to the drawing engine that it needs to. As well as it will receive touch events on the game board from the drawing engine, which it can then handle appropriately.

The Game Board

The game board is responsible for keeping the state of where all the players pieces are in the game. It also validates potential moves so that the player can check if a move is allowed. Every piece is guaranteedto be in a valid location based on our defined game rules. Currently players only have a name associated with them.

Players

There are three types of players that implement the “Player” interface.

Local Player

This is a player that is in contact with the actual device. A local player in one instance of a game will be a network player for someone else running the application.

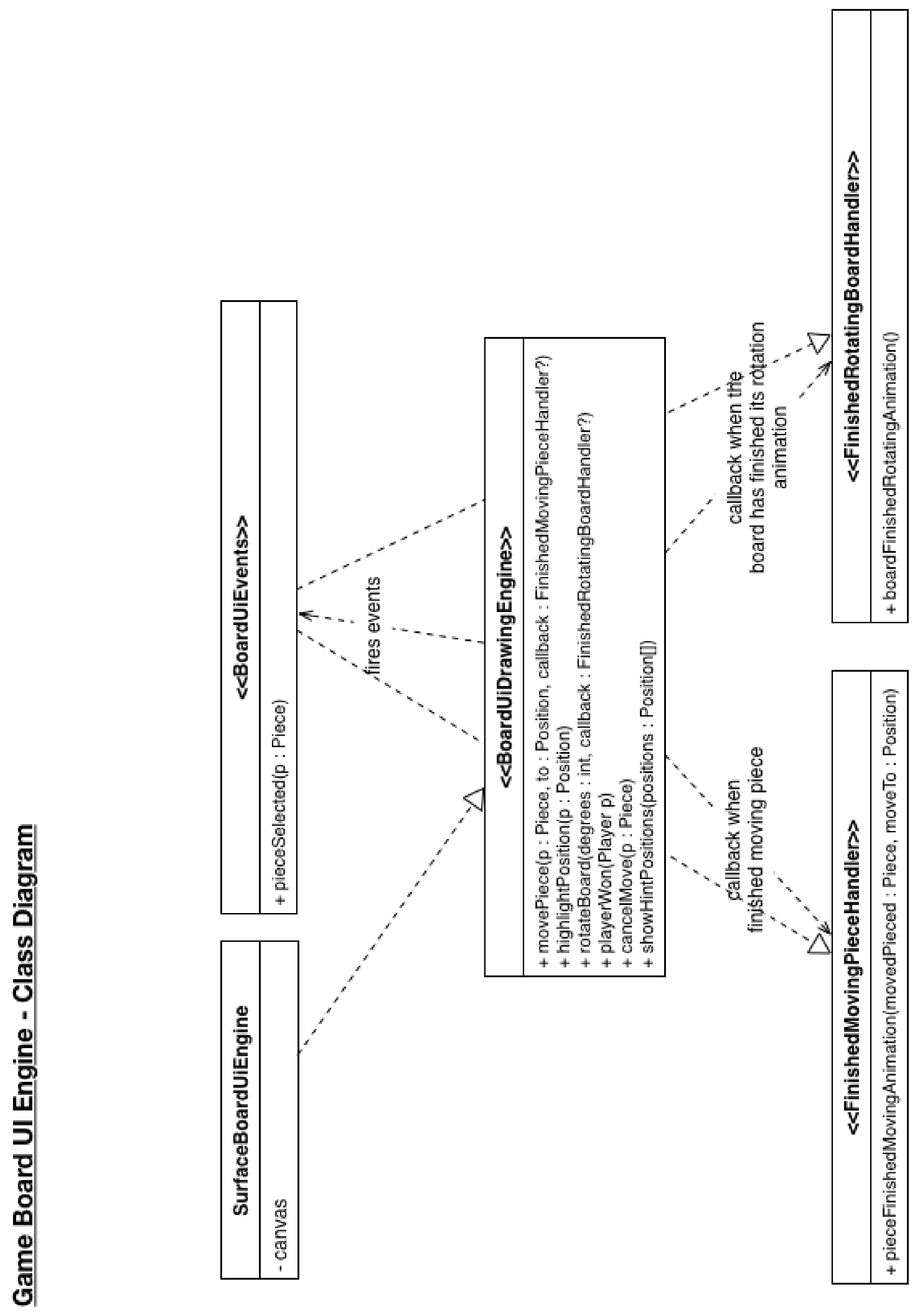
Network Player

This player instance interacts with the server to get details about another player in the game as they are needed.

AI Player (Artificial Intelligence)

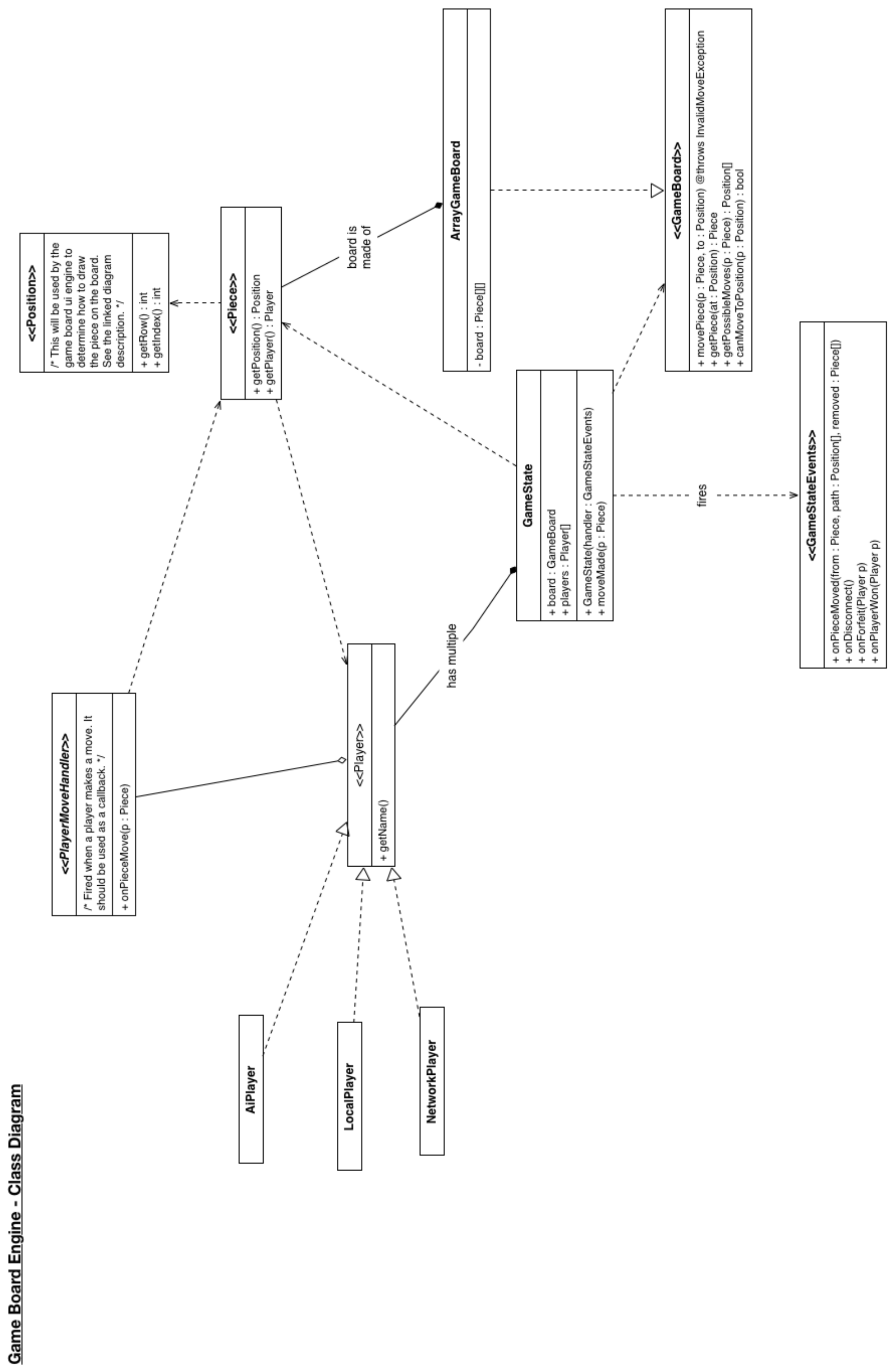
The AI player is just an extension of the local player. It can generate custom names as needed.

6.3 Appendix C


Game Board UI Engine - Class Diagram
SurfaceBoardUiEngine
- canvas
<<BoardUiEvents>>
+ pieceSelected(p : Piece)
fires events
<<BoardUiDrawingEngine>>
+ movePiece(p : Piece, to : Position, callback : FinishedMovingPieceHandler?)
+ highlightPosition(p : Position)
+ rotateBoard(degrees : int, callback : FinishedRotatingBoardHandler?)
+ playerWon(Player p)
+ cancelMove(p : Piece)
+ showHintPositions(positions : Position[])
callback when the board has finished its rotation animation
<<FinishedRotatingBoardHandler>>
+ boardFinishedRotatingAnimation()
callback when finished moving piece
<<FinishedMovingPieceHandler>>
+ pieceFinishedMovingAnimation(movedPieced : Piece, moveTo : Position)

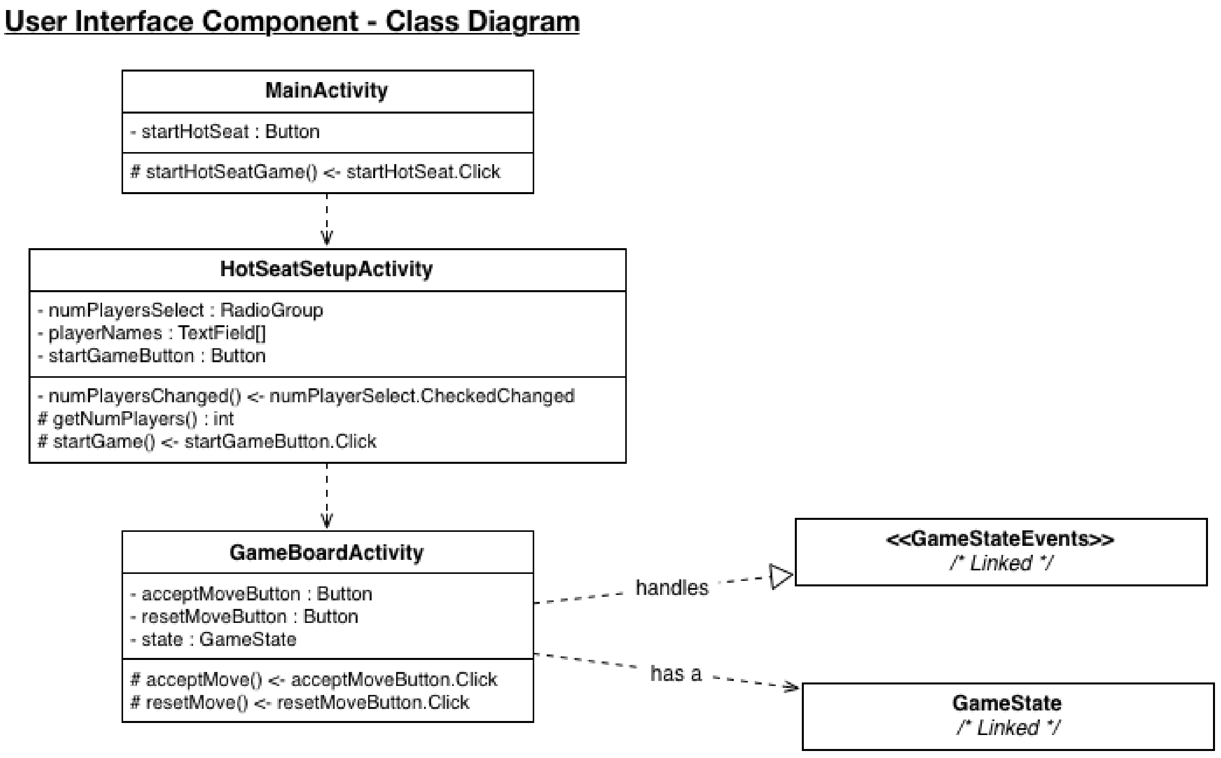
Game Board UI Engine Class Diagram

6.4 Appendix D


Game Board Engine - Class Diagram
<<Position>>
/* This will be used by the game board ui engine to determine how to draw the piece on the board. See the linked diagram description. */
+ getRow() : int
+ getIndex() : int
<<Piece>>
+ getPosition() : Position
+ getPlayer() : Player
board is made of
ArrayGameBoard
- board : Piece[][]
<<GameBoard>>
+ movePiece(p : Piece, to : Position) @throws InvalidMoveException
+ getPiece(at : Position) : Piece
+ getPossibleMoves(p : Piece) : Position[]
+ canMoveToPosition(p : Position) : bool
GameState
+ board : GameBoard
+ players : Player[]
+ GameState(handler : GameStateEvents)
+ moveMade(p : Piece)
fires
<<GameStateEvents>>
+ onPieceMoved(from : Piece, path : Position[], removed : Piece[])
+ onDisconnect()
+ onForfeit(Player p)
+ onPlayerWon(Player p)
has multiple
<<PlayerMoveHandler>>
/* Fired when a player makes a move. It should be used as a callback. */
+ onPieceMove(p : Piece)
<<Player>>
+ getName()
AIPlayer
LocalPlayer
NetworkPlayer

Game Board Engine Class Diagram

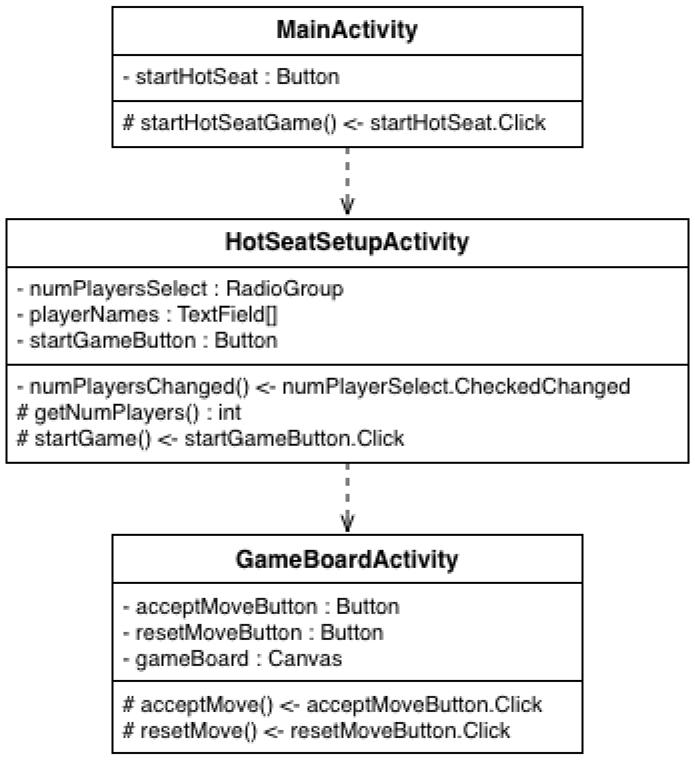
6.5 Appendix E


User Interface Component - Class Diagram
MainActivity
- startHotSeat : Button
# startHotSeatGame() <- startHotSeat.Click
HotSeatSetupActivity
- numPlayersSelect : RadioGroup
- playerNames : TextField[]
- startGameButton : Button
- numPlayersChanged() <- numPlayerSelect.CheckedChanged
# getNumPlayers() : int
# startGame() <- startGameButton.Click
GameBoardActivity
- acceptMoveButton : Button
- resetMoveButton : Button
- state : GameState
# acceptMove() <- acceptMoveButton.Click
# resetMove() <- resetMoveButton.Click
handles
<<GameStateEvents>>
/* Linked */
has a
GameState
/* Linked */

User Interface Component Class Diagram

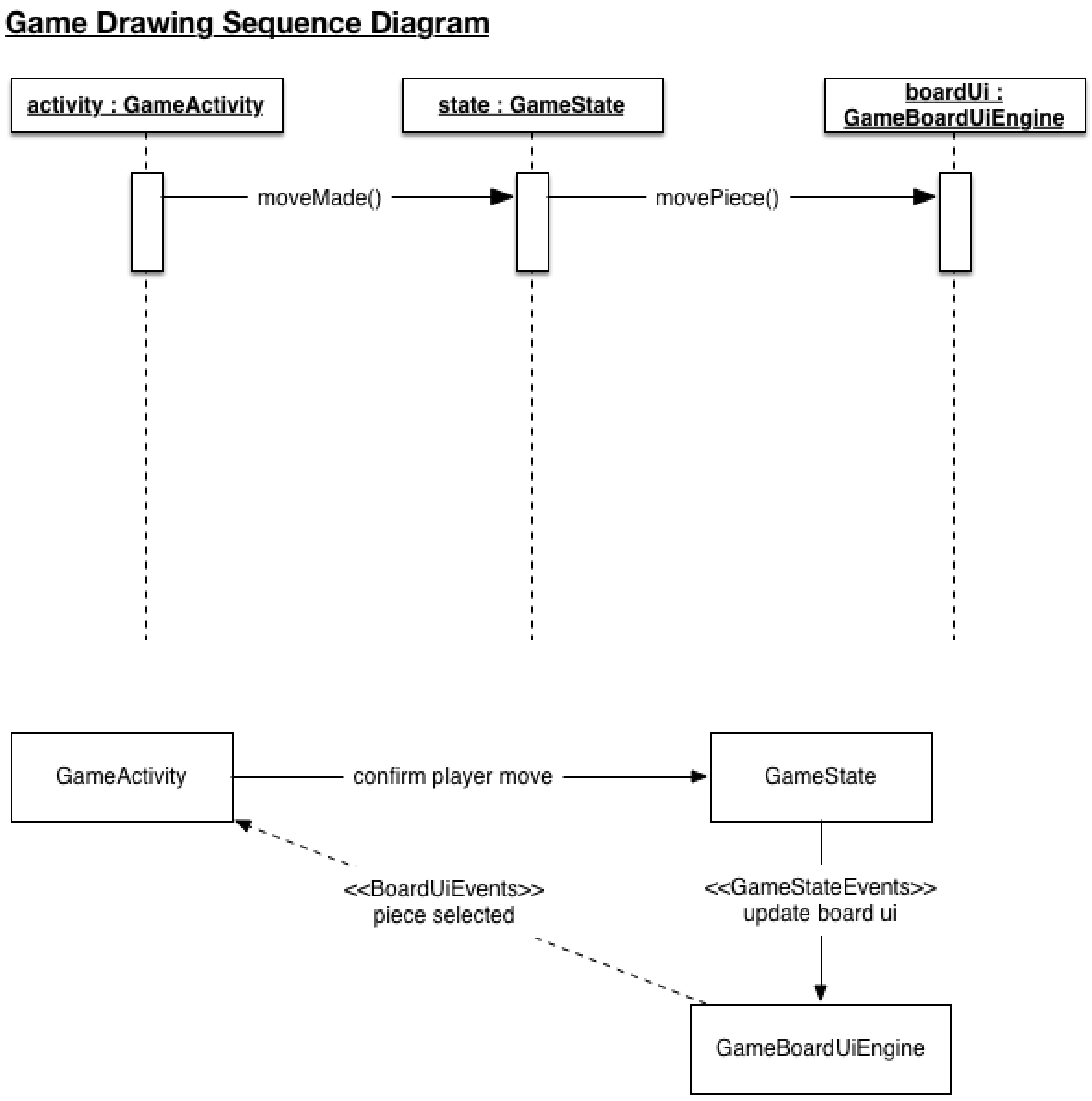
6.6 Appendix F

Hot Seat Interface Class Diagrams


MainActivity
- startHotSeat : Button
# startHotSeatGame() <- startHotSeat.Click
HotSeatSetupActivity
- numPlayersSelect : RadioGroup
- playerNames : TextField[]
- startGameButton : Button
- numPlayersChanged() <- numPlayerSelect.CheckedChanged
# getNumPlayers() : int
# startGame() <- startGameButton.Click
GameBoardActivity
- acceptMoveButton : Button
- resetMoveButton : Button
- gameBoard : Canvas
# acceptMove() <- acceptMoveButton.Click
# resetMove() <- resetMoveButton.Click

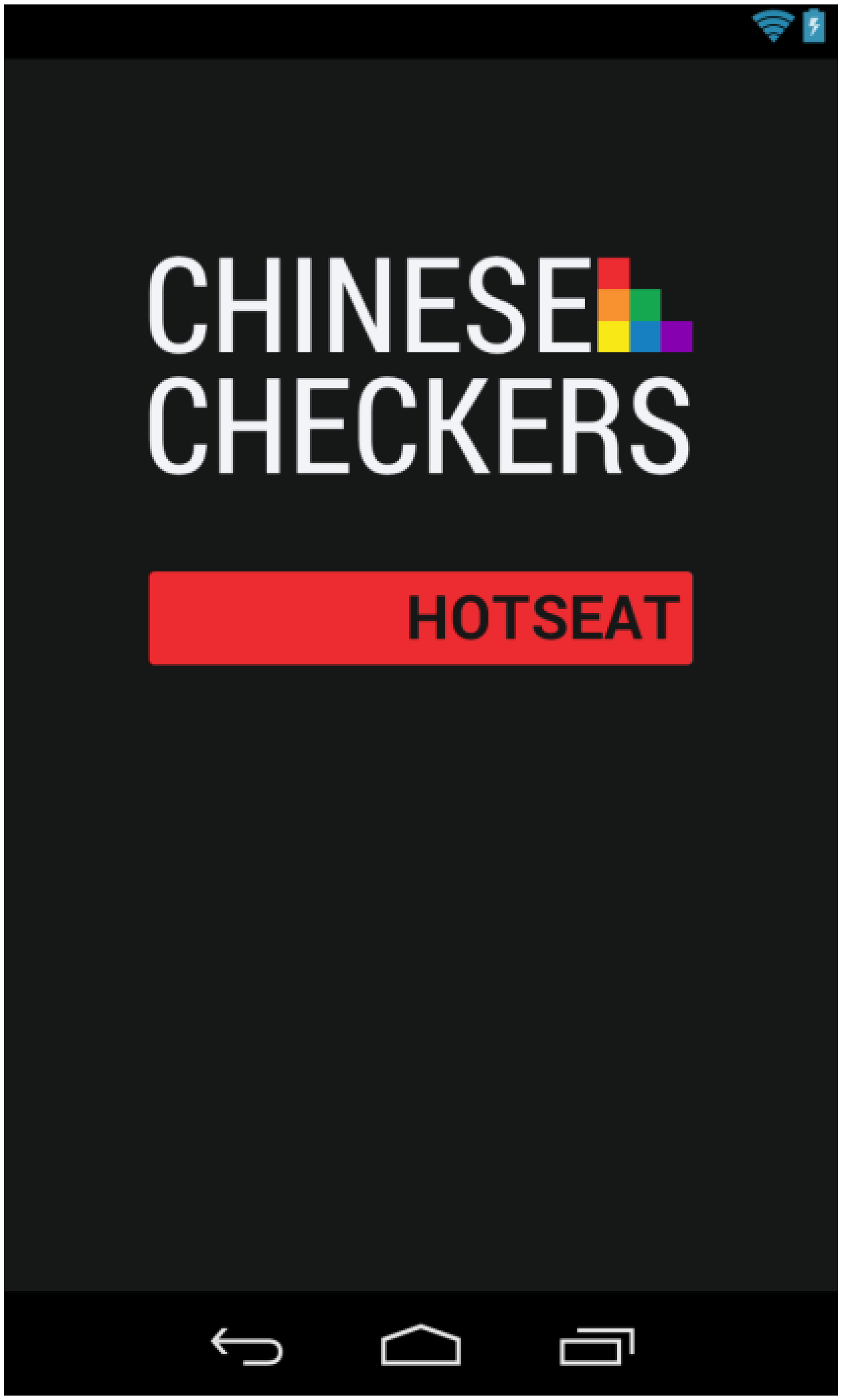
Hot Seat Interface Diagram

6.7 Appendix G


Game Drawing Sequence Diagram
activity : GameActivity
state : GameState
boardUi : GameBoardUiEngine
moveMade()
movePiece()
GameActivity
confirm player move
GameState
<<BoardUiEvents>> piece selected
<<GameStateEvents>> update board ui
GameBoardUiEngine

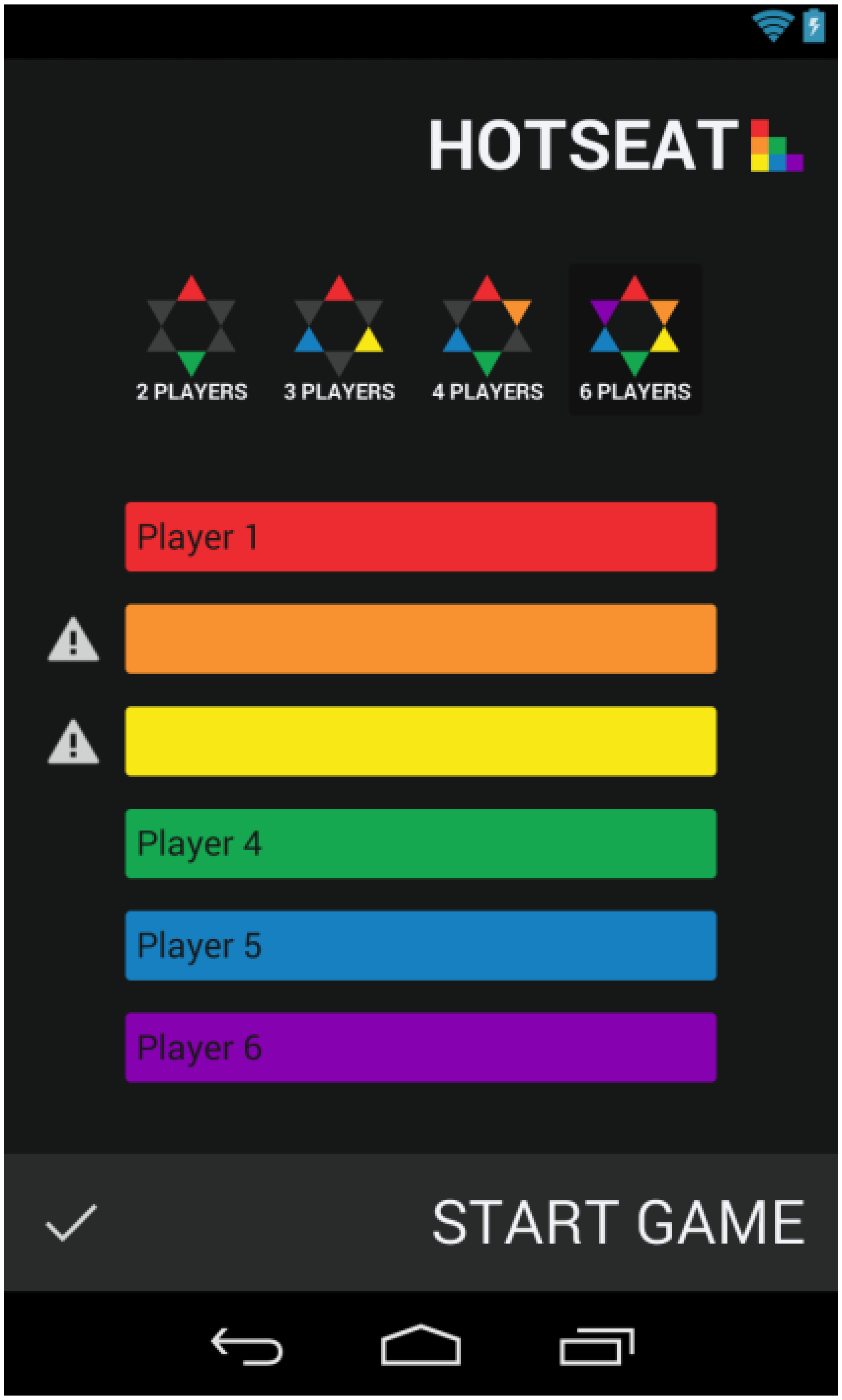
Game Drawing Sequence Diagram

6.8 Appendix H



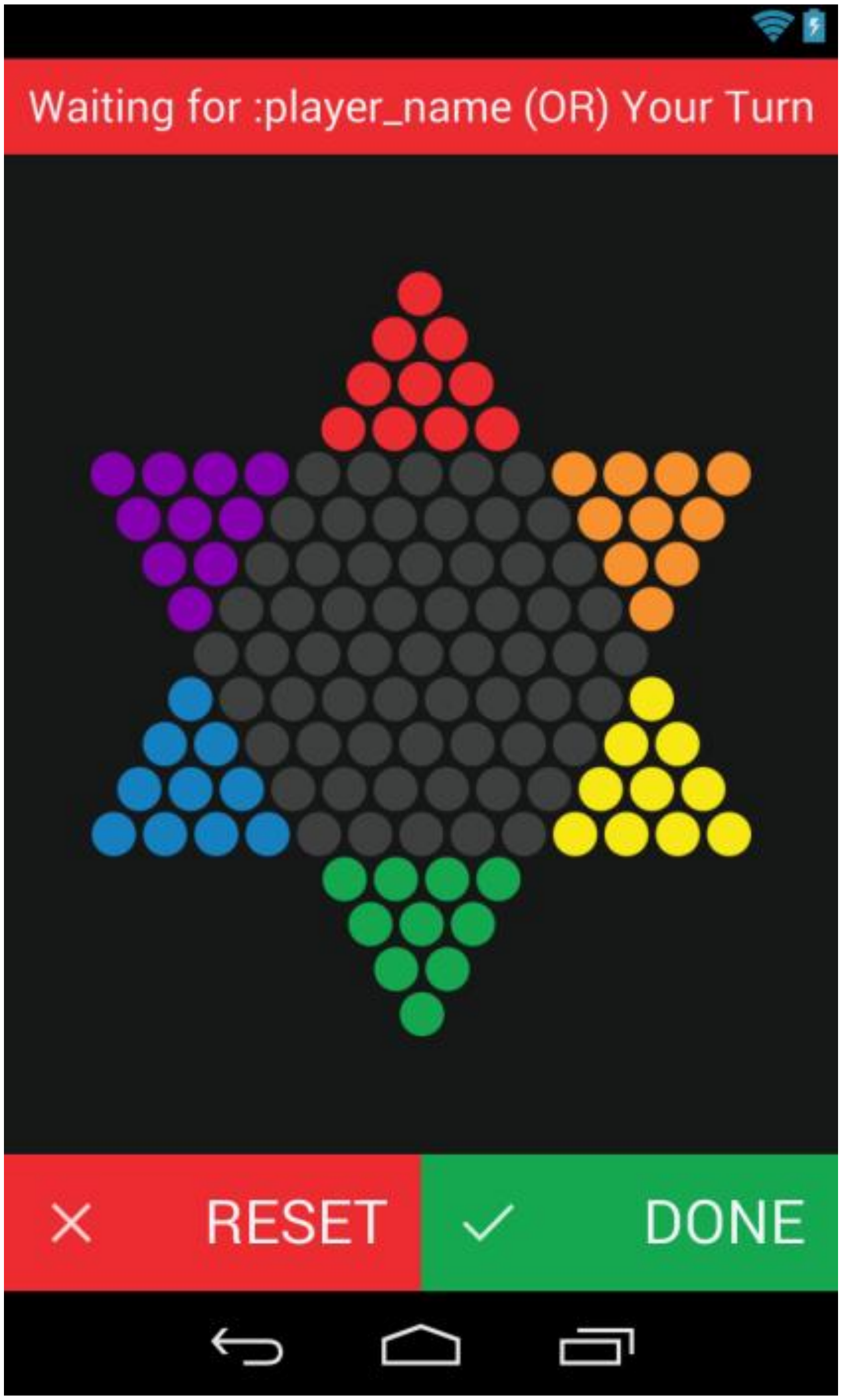
Main Activity Mock-up

6.9 Appendix I



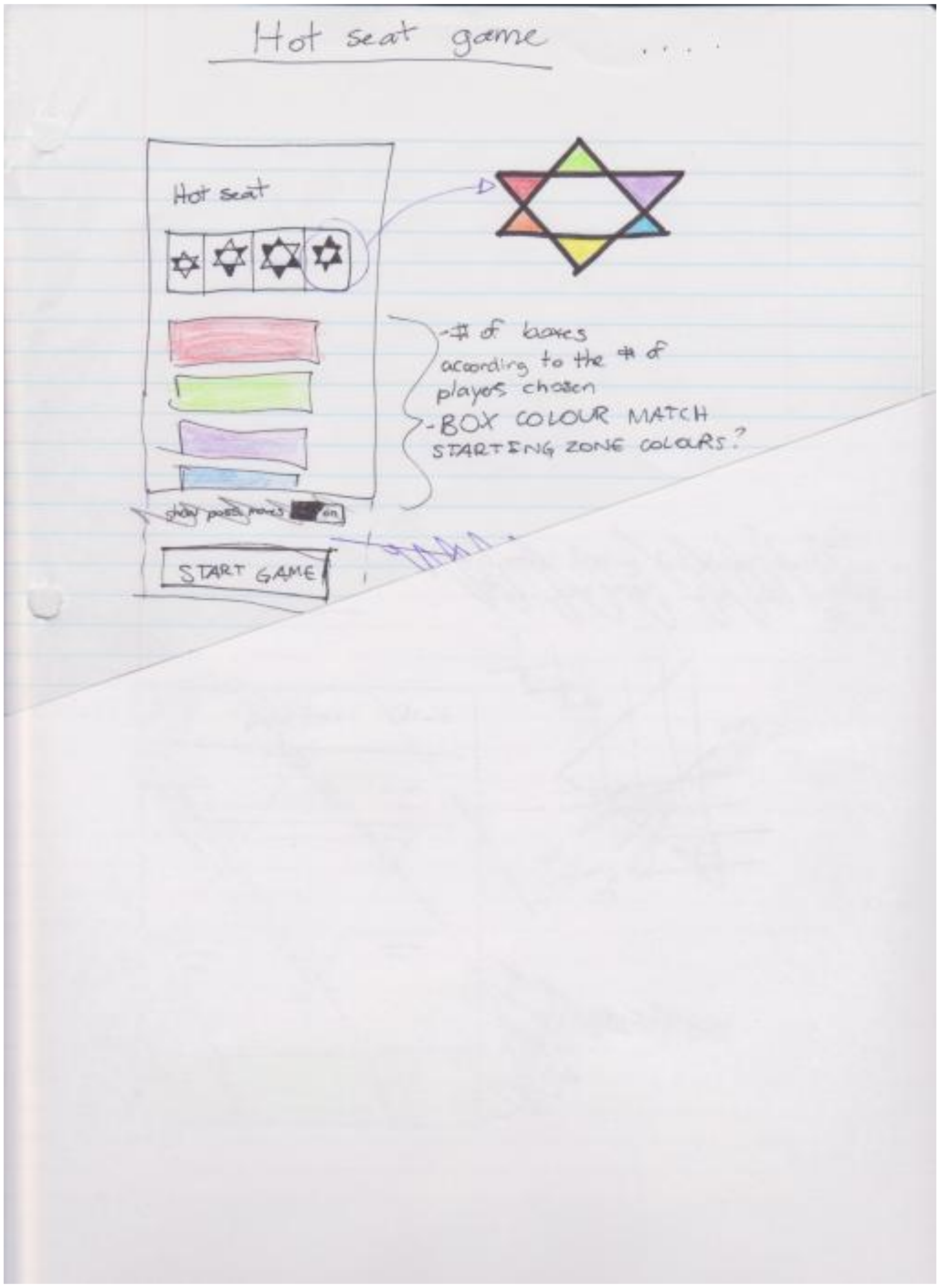
Hot Seat Configuration Activity Mock-up

6.10 Appendix J

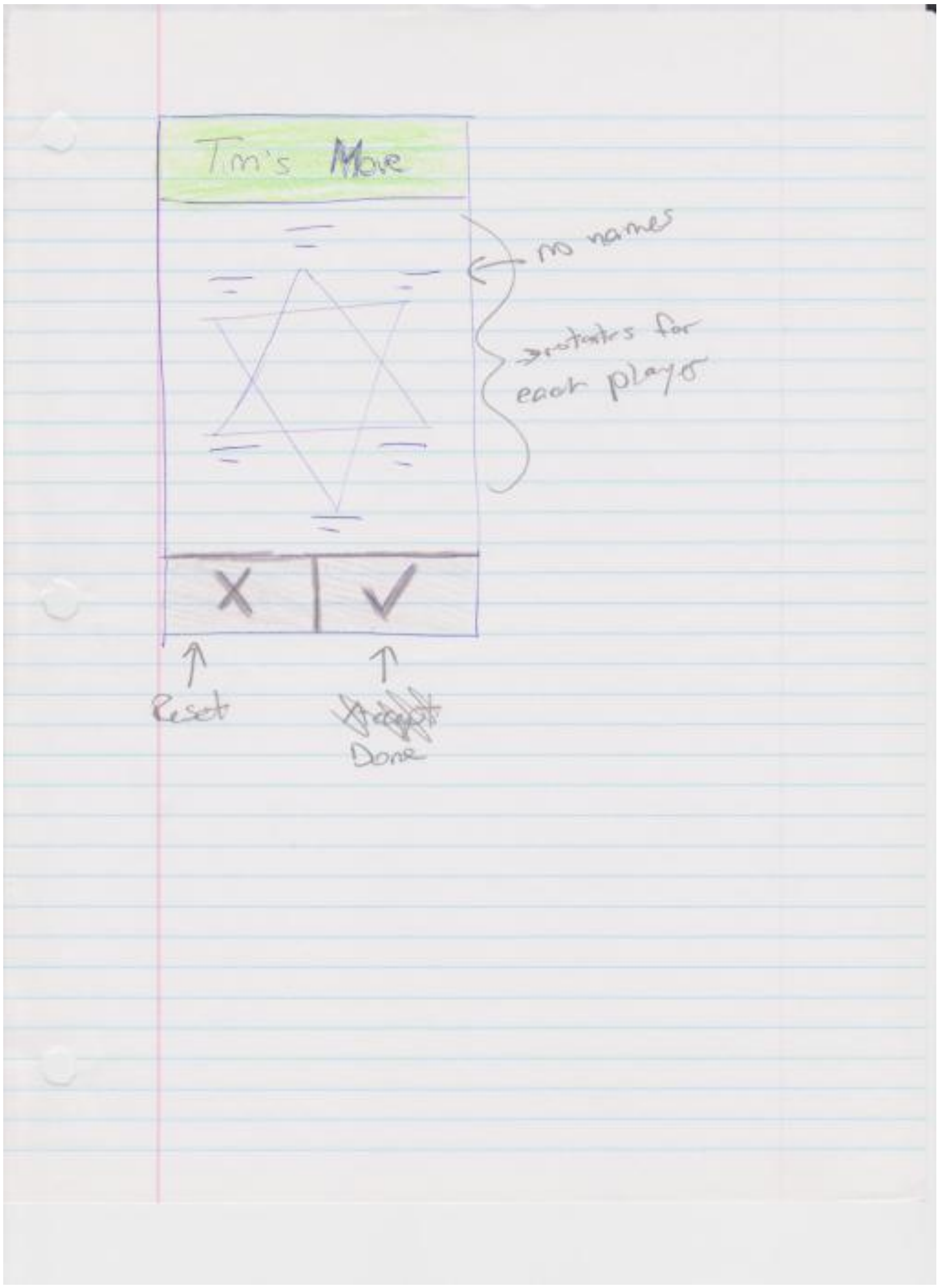


Hot Seat Game Mock-up

6.11 Appendix K


Hot seat game
Hot seat
-# of boxes according to the # of players chosen
-BOX COLOUR MATCH STARTING ZONE COLOURS?
show poss. moves on
START GAME

Hot Seat Configuration User Interface Sketch



Hot Seat Game User Interface Sketch

6.12 Appendix L

Test Feedback Form

Project: ____

Project Phase: _____________ Test Phase: _______________ Date: ______

Tester: ___________________

Pass\Fail\Extra Consideration Required: ________________________________

Remarks (If fail what caused failure)**:**
____
____
____
____
____
____
____
____
_____

Test Output (If Applicable)**: ____**

Action Taken:

____

____

Tester Signature: _____________ **Project Lead Signature:**_______________

Test Lead Signature: _______________ **Project Lead Name:**_____________

Test Lead Name: __________________

6.13 Appendix M

Ease of Use Testing Feedback Form

Give each of the tasks a rating from 0 to 10 based on the difficulty of accomplishing the relevant tasks.

Task 1 Score: _____

Task Remarks:

____

____

Task 2: _____

Task Remarks:

____

____

Task 3: _____

Task Remarks:

____

____

Task 4: _____

Task Remarks:

____

____

Task 5: _____

Task Remarks:

____

____

Task 6: _____

Task Remarks:

____

____

Task 7:_____

Task Remarks:

____

____

General Remarks:

____

____

____

____

6.14 Appendix N

Test Feedback Form

Project: Chinese Checkers

Project Phase: 1 **Test Phase:** Integration Testing **Date:** 02/05/14

Tester: Saajid Mohammed

Pass\Fail\Extra Consideration Required: Pass

Remarks (If fail what caused failure)**:** None

Test Output (If Applicable)**:** See Integration Test Results

Action Taken: No Action Need

Tester Signature: ______________ **Project Lead Signature:** ________________

Test Lead Signature: _______________ **Project Lead Name:** Curtis Smith

Test Lead Name: Saajid Mohammed

Integration Test Results

21.38 s

All Tests: 9 total, 9 passed

4.15 s

ca.brocku.chinesecheckers.tests.HomeAndSeatTest

4.15 s

passedtestTransition

9.50 s

ca.brocku.chinesecheckers.tests.HomeSeatAndGameTest

9.50 s

passedtestTransition

6.73 s

ca.brocku.chinesecheckers.tests.HotseatConfigurationActivityUnitTest

2.58 s

passedtestActivity

951 ms

passedtestFourPlayerConfig

1.08 s

passedtestSixPlayerConfig

1.02 s

passedtestThreePlayerConfig

1.10 s

passedtestTwoPlayerConfig

500 ms

ca.brocku.chinesecheckers.tests.HotseatGameActivityUnitTest

500 ms

passedtestActivity

500 ms

ca.brocku.chinesecheckers.tests.MainActivityUnitTest

500 ms

passedtestActivity

Generated by Android Studio on 2/5/14 7:13 PM

6.15 Appendix O

Ease of Use Testing Feedback Form

Give each of the tasks a rating from 0 to 10 based on the difficulty of accomplishing the relevant tasks.

Task 1 Score: 0

Task Remarks: Icon was visually attractive, it also worked when I clicked it which was nice

Task 2: 2

Task Remarks: I had to think about where on the screen to touch to begin the process.

Task 3: 0

Task Remarks: This task was not difficult at all, very organized and user friendly.

Task 4: 1

Task Remarks: N/A

General Remarks: Overall the program was visually appealing and user friendly. Good work.

Ease of Use Testing Feedback Form

Give each of the tasks a rating from 0 to 10 based on the difficulty of accomplishing the relevant tasks.

Task 1 Score: 0

Task Remarks: Vibrant app logo made it easy to

Task 2: 2

Task Remarks: Was unsure of where to click

Task 3: 0

Task Remarks: N/A

Task 4: 0

Task Remarks: N/A

Task 5: 0

Task Remarks: N/A

General Remarks: Very pretty app. Buttons on the setup screen were not centered.

Ease of Use Testing Feedback Form

Give each of the tasks a rating from 0 to 10 based on the difficulty of accomplishing the relevant tasks.

Task 1 Score: 0

Task Remarks: like the icon for the game.

Task 2: 2

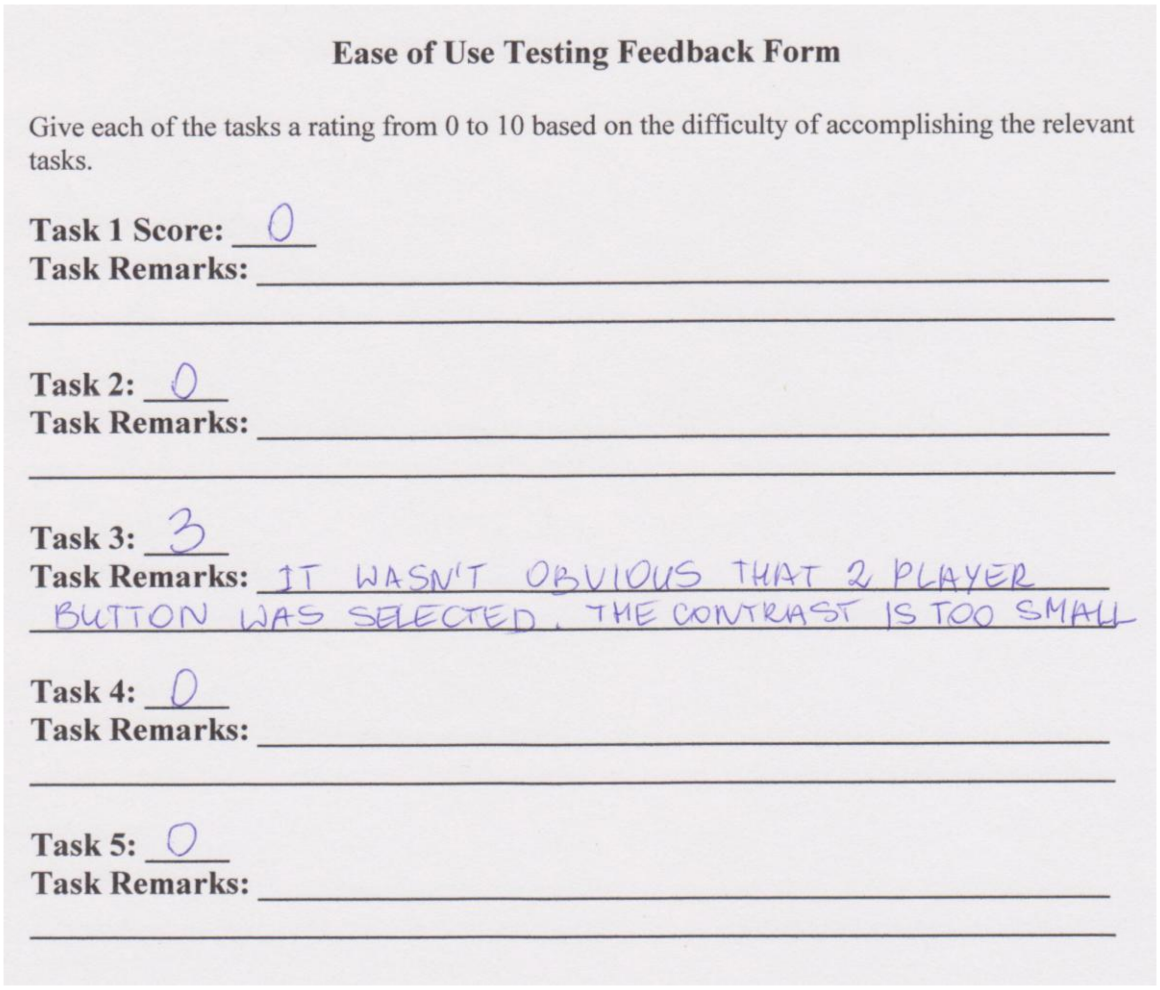
Task Remarks: it would look better if the button and the logo was in the middle of the screen. easy to read.

Task 3: 7

Task Remarks: can’t see the word in the red box. _like the designs of the number of player selection buttons but would make them as big as a normal fingernail. Like the position of the selection input at the top.

Task 4: 9

Task Remarks: Harder than task 3 because there were more player name to enter. liked the colors. Liked the position of the input fields. Liked the game board size and position. Size of the name text could be bigger. Buttons at the bottom are good size.



Ease of Use Testing Feedback Form

Give each of the tasks a rating from 0 to 10 based on the difficulty of accomplishing the relevant tasks.

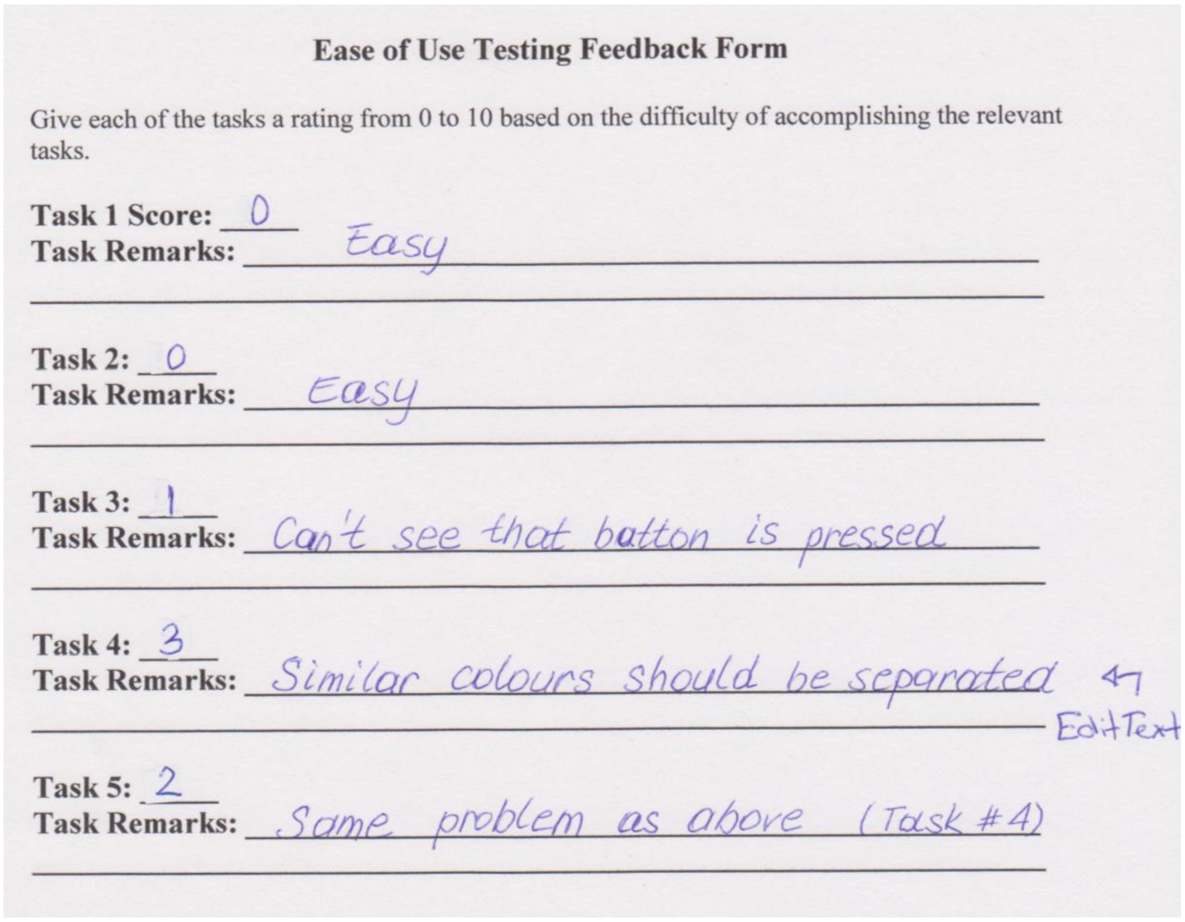
Task 1 Score: 0
Task Remarks:

Task 2: 0
Task Remarks:

Task 3: 3
Task Remarks: IT WASN'T OBVIOUS THAT 2 PLAYER BUTTON WAS SELECTED. THE CONTRAST IS TOO SMALL

Task 4: 0
Task Remarks:

Task 5: 0
Task Remarks:



Ease of Use Testing Feedback Form

Give each of the tasks a rating from 0 to 10 based on the difficulty of accomplishing the relevant tasks.

Task 1 Score: 0
Task Remarks: Easy

Task 2: 0
Task Remarks: Easy

Task 3: 1
Task Remarks: Can't see that button is pressed

Task 4: 3
Task Remarks: Similar colours should be separated ← EditText

Task 5: 2
Task Remarks: Same problem as above (Task #4)

7 References

10 Web Design Trends for 2013. (2013, February 27). *Awwwards*. Retrieved January 13, 2014, from http://www.awwwards.com/10-web-design-trends-for-2013.html

Typography. (n.d.). *Android Developers*. Retrieved February 5, 2014, from http://developer.android.com/design/style/typography.html

Use Case Templates. (n.d.). *TechnoSolutions*. Retrieved January 18, 2014, from http://www.technosolutions.com/use_case_template.html