Chromecast Checkers

CS 499 Senior Design Project
University of Kentucky
Spring 2016

Chromecast Checkers

Table of Contents

I. Personnel

Authors and Team Members:

- Zach Almon
- Matt Dunbar
- Omid Omidi

II. Disclaimer

This project has been designed and implemented as a part of the requirements for CS-499 Senior Design Project for Spring 2016 semester. While the authors make every effort to deliver a high quality product, we do not guarantee that our products are free from defects. Our software is provided "as is," and you use the software at your own risk.

We make no warranties as to performance, merchantability, fitness for a particular purpose, or any other warranties whether expressed or implied.

No oral or written communication from or information provided by the authors or the University of Kentucky shall create a warranty.

Under no circumstances shall the authors or the University of Kentucky be liable for direct, indirect, special, incidental, or consequential damages resulting from the use, misuse, or inability to use this software, even if the authors or the University of Kentucky have been advised of the possibility of such damages

III. Abstract

Chromecast Checkers and the Android Checkers companion application are simple game applications designed for Chad McQuillan, of Lexmark. It is intended to allow users to simply play checkers on their Chromecast using Android devices as controllers. The program runs a HTML/Javascript server program that processes everything. This program casts to the Chromecast screen. The Android companion applications connect to the server file and tell it to cast and when to tear down. The Android application also handles the button input, but validation of the input is handled by the server program.

IV. Introduction

The Chromecast is a versatile piece of technology that allows users to turn regular TV's into smart TVs. The Google Chromecast allows a connection to the internet via WiFi, so that application may be downloaded and view. These application can vary wildly from developer to developer, from applications like Netflix to Amazon to applications that let you cast your laptop screen to the TV. Google has been very generous to provide developers with kits to help them start developing applications for whatever needs they may have.

Using AngularJS, JavaScript, Java, and HTML as necessary, our project was to develop a Chromecast app that will connect up to two Android devices to the Chromecast device and allow 2 people to play checkers using the Android devices as controllers and the Chromecast/TV as the screen for the checkerboard. AngularJS is Google's framework for development and the testing of web applications.

Our customer was Chad McQuillan from Lexmark. He is the one that provided the task and some help and insight into the Chromecast API.

V. Project Specifications

The Android devices will act as controllers allowing players to select any of their own checkers and see available moves for them. The Chromecast will act as a central server point and display all checkers play.

Android Controller App Specifications

- Must have center select button
    - Must be able to select and deselect
- Must have four directional arrows surrounding the center select button
    - Must be able to go to all places, even in a multiple jump move
- Must have a back button to go back a screen to be able to switch which checker is selected
- Must be color coded to which player is red or black
- Must show when controller is disconnected from the Chromecast and show a reconnecting icon
- Must show what turn it is and whose turn it is on top
- Must be greyed out while other player is deciding their move

Chromecast Specifications

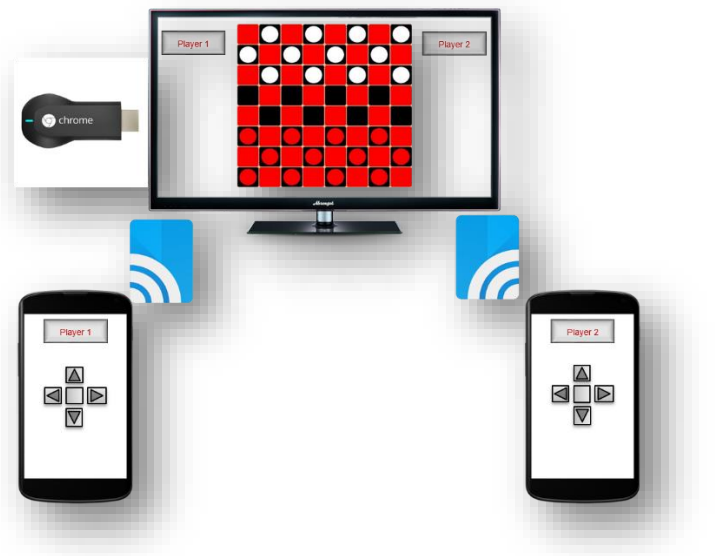
- Must have the checker board center screen big enough for all to see
- Must have player graveyards on either side of the board
- Must show what turn it is and whose turn it is on top
- Must be color coded to match controllers to which player is red or black
- Must show when a controller is disconnected from the Chromecast and show a reconnecting iron
- Must show all possible moves available, including placement after jumps
- Must show selected checker as a highlighted piece

VI. Planning

A. Program / Platform Environments:
   i. Host / Server JavaScript that handles communications and Checkers Logic
      i. Chromecast Application / Google API for the Chromecast / AngularJS
      ii. Our server file will use JavaScript and HTML; AngularJS is written in JavaScript as well
   ii. Android Controller Application / Google API for Android
      i. Android source code is written in Java
      ii. Android Studio for Mac was used to write and test in, using Android Emulation

B. Program Use Scenarios:
   i. User starts Chromecast
   ii. User starts Checkers for Chromecast App
   iii. User(s) starts Android controller companion app
   iv. User(s) connects Android controller to Chromecast
   v. User(s) start a new game
   vi. User with first move will select and move whichever piece they wish
      i. Second user will be greyed out at this time, unable to do anything
   vii. Second user will gain the ability to move after the first user
      i. First user will be greyed out at this time, unable to do anything
   viii. Repeat 6-7 until someone has no pieces' left, or someone cannot move
   ix. Users, during their turn (#6 and #7), will be able to move any piece (limited by which piece can move) to wherever they want (limited by where specific pieces can move) and jump however many opponent pieces they want by selecting the spaces they wish to move to, however many jumps they want to make (limited by if the piece can jump, and limited to how many jumps the piece can make based on the opponent's pieces).
   x. The Program Detects when someone has no pieces left, and when someone cannot move anymore they are forced to forfeit, since they have lost.

C. Sample Diagram of Working Demonstration:



i. This diagram is to show the components and interfaces. The server side runtime program is not seen by users. The Chromecast application is mainly a display, but will have to be turned on, and the application will have to be run. The Android controllers have the user input to move pieces, as well as maintaining a persistent communicating connection with the Chromecast.

ii. The Chromecast, being connected to the TV, will have to be turned on and the Checkers Application must be run. This will be done manually by the user. After this, the Chromecast will not take any more user input.

iii. The Android controllers will have to be turned on individually and manually connected to the Chromecast. This can take a few minutes. After it is connected the Android applications will be the main user input devices.

D. Effort and Size Estimation:

i. We estimated about 300 lines of code for the Android companion application and around 900+200 lines of code for the Chromecast JavaScript server file and index HTML files respectively.

ii. We ultimately ended with 500 lines of code for the Android Java file, 700 for the JavaScript server file.

iii. We ended up with less code than we had estimated because of AngularJS functions. The functions to communicated between the Android and Server applications, had quite a few lines as well. We ended up with combining the index file and JavaScript code to create the game board, pieces and text and manage all of the backend processing in one file. Altogether the project ended up being a little more complex / requiring more than any of us had anticipated, but we managed very well.

E. Schedule and Milestones:
    i. Some of the professor set milestones were:
        i. Friday February 12th: Webpage Due
        ii. Monday March 7th: Design Document Due
        iii. Monday April 4th: Test Plan Due
        iv. Friday April 8th: Test Plan Review
        v. Friday April 15th: Code Review
        vi. Wednesday April 27th: Project Presentation
        vii. Thursday May 5th: Product delivery to professor and customer must be completed by the end of the semester.
        viii. Some of these dictated when we had to be done with certain parts, as in the testing plan, we had to have some code done to know what to test and how to test it specifically. The design dictated we would have to have met together and formed a plan on how to commence with the code writing.
    ii. Some of our own module milestones were:
        i. Get familiar with Google Cast API by February 26th
        ii. Get familiar with Google Android API by February 26th
        iii. Get checkerboard JavaScript and index.html file done by March 13th
        iv. Get Android Application done by April 7th
        v. Get Android Application and JavaScript file communicating by April 7th
        vi. Debug entire project individually and together and get as many test cases done and completed as we possibly can by April 15th
        vii. Note: When we state 'done by' we mean a working program. By no means does it have to be free of bugs or be complete. We just wanted something tangible to work off of from the 'due date'.
        viii. We unfortunately had quite a few bugs in our program with the communication and checkers game play, and it was not completed (working, debugging complete) until April 22nd.
    iii. We now only have report, presentation, and deliverables' milestones left

VII. Design

Program Components:
1. Chromecast display Application
2. Server side runtime program
3. Android Controller Application

Program Interfaces:
1. Chromecast Application Interface
2. Server / Host Program for Chromecast

3. Android Controller Interface
4. Android-Chromecast Connection/communication

Figure 1:

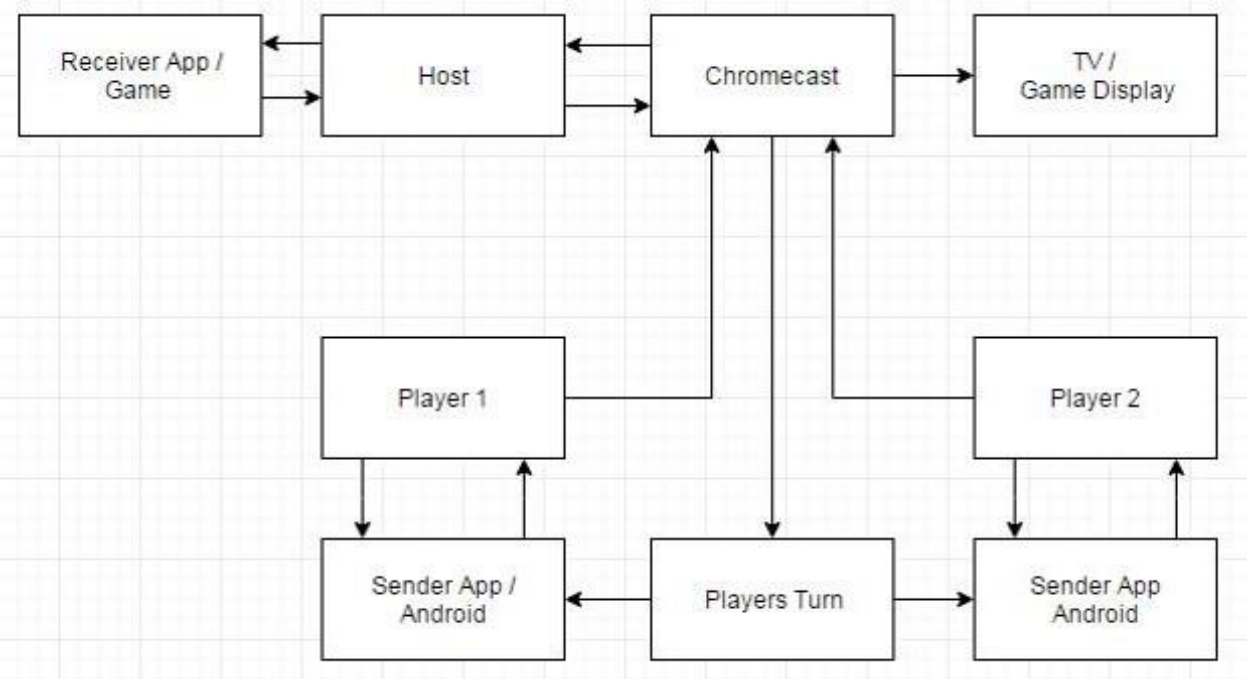


Figure 1 is more technical in showing the actual communication flow between the programs and applications, than the sample working demonstration diagram shown earlier. Starting with the Chromecast, it displays to the TV/game display. It, the Chromecast, also communicates to the hosting server and receiver app. Both of these programs process the messages and does some processing and sends back responses based on what happened/communicated. The Chromecast/receiver application also controls the players turns which disables the controller which is not active at the moment. The sender apps decipher these messages and when the player inputs an action, the sender app encodes this and sends it to the Chromecast/receiver app.

Figure 2:

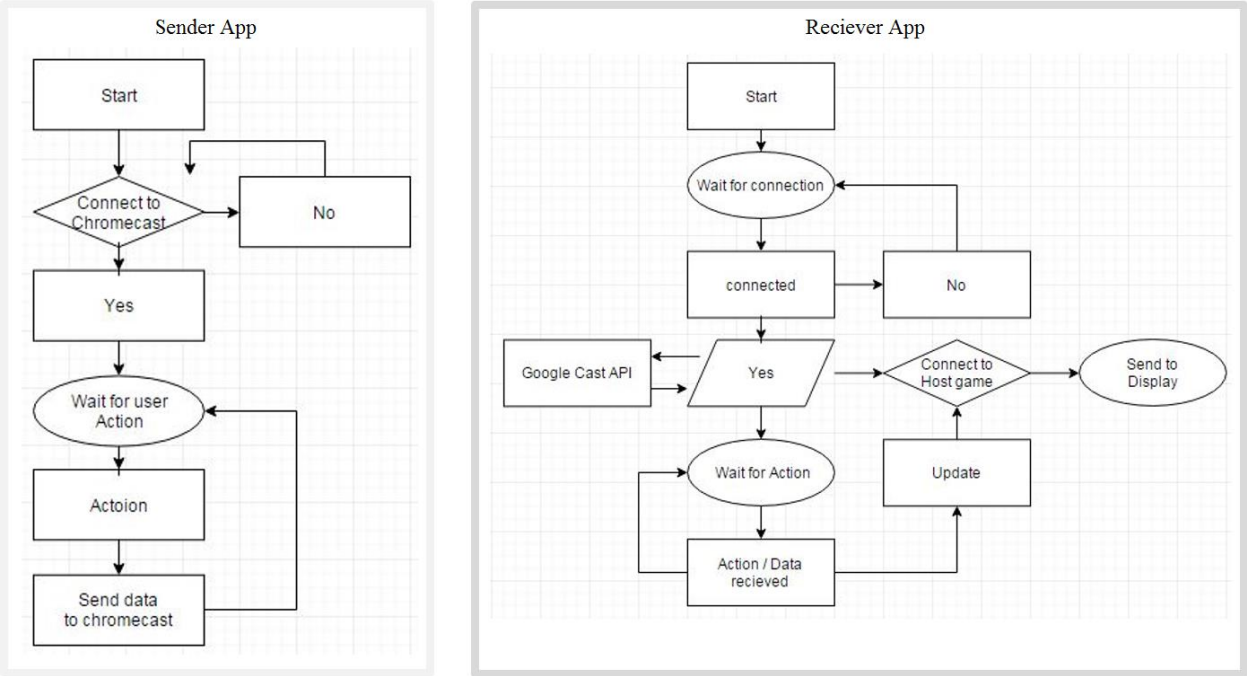


Figure 2 has two parts. Starting with the sender application, the Android controller, the app starts up and attempts to connect to the Chromecast. From here it will take user input and send any input to the Chromecast. It is also getting data from the Chromecast, in which could disable the controller if it is not the users turn. The receiver application, the host/server, also starts and waits until it connects to the Chromecast. When connected it will wait until action or data is received from the controllers. It will process the data and send the update information to the Chromecast to be displayed on the TV/game device.

Figure 3:

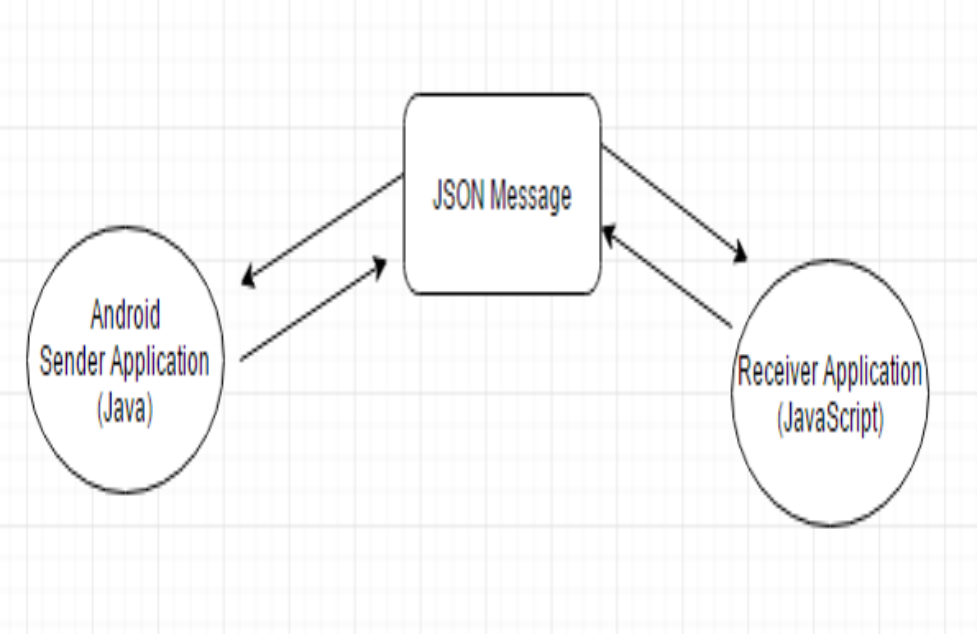


Figure 3 is just to show the Android sender application will encode a JSON message to be sent to the Server receiver application which will decode the messages, and then encode responses back to the Android application.

Figure 4:

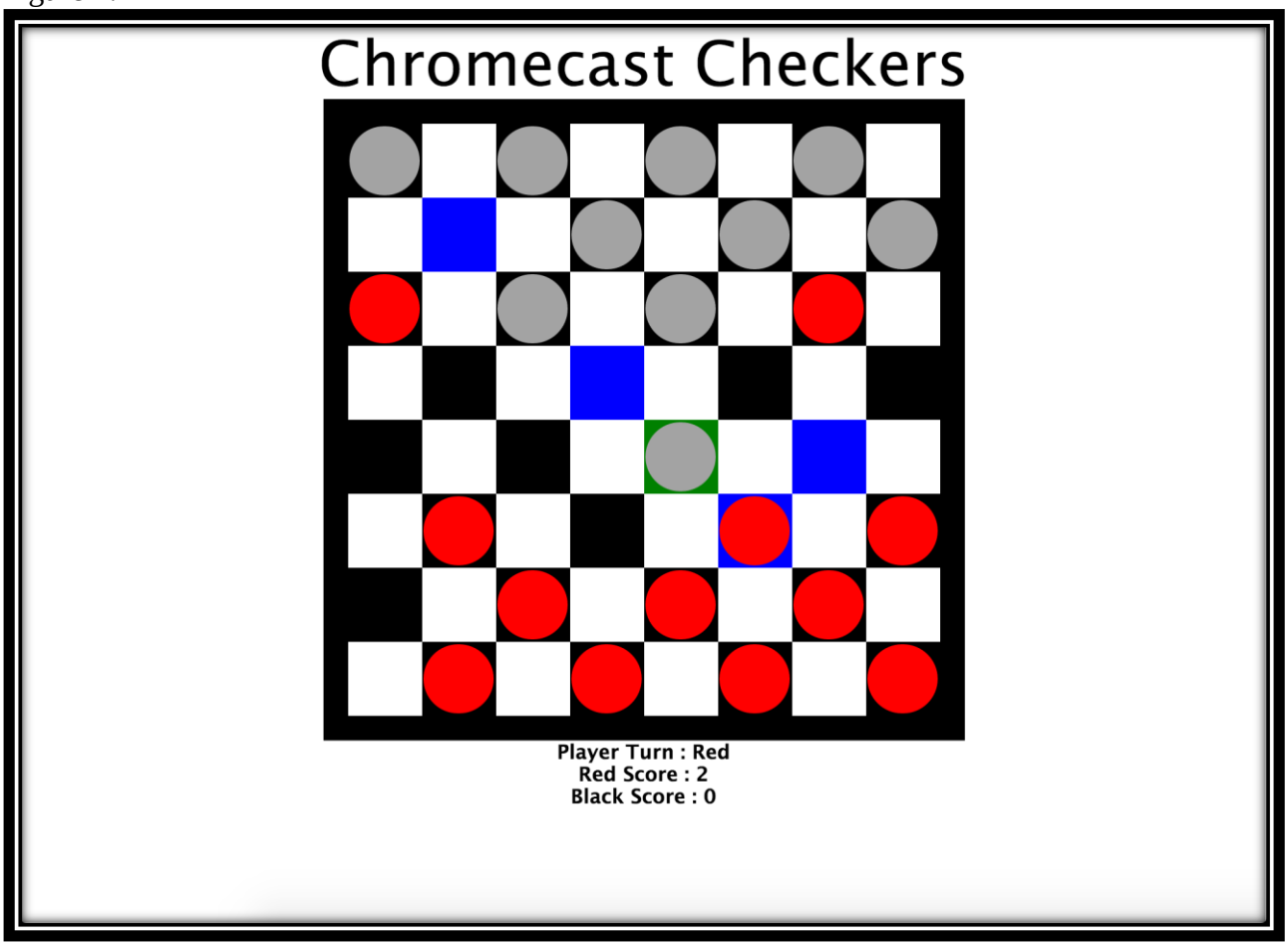
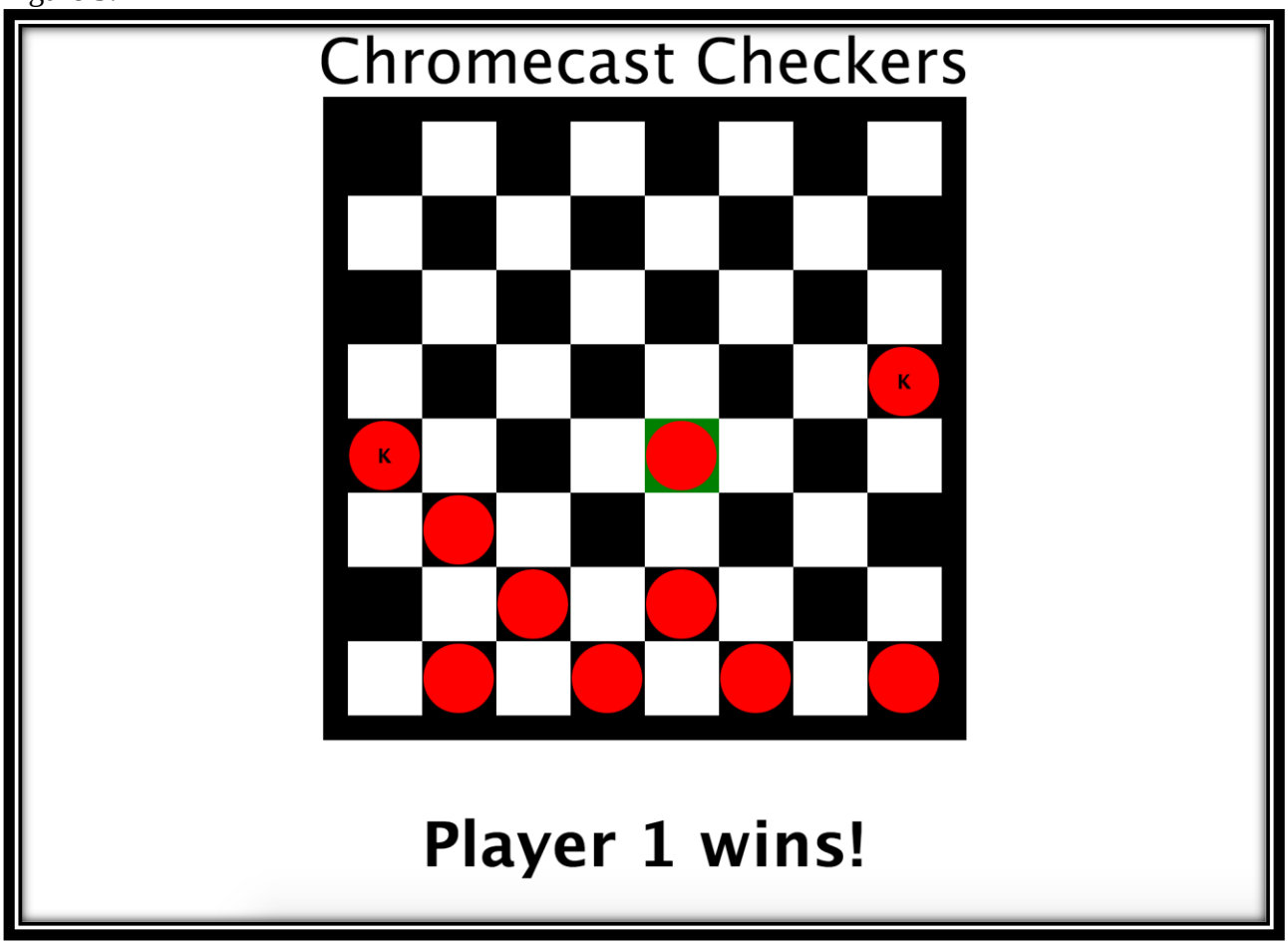


Figure 4 is the simple interface of the end game checker board. This is what will ultimately be shown to the users from the Chromecast and the TV. This screen appears from when the user hits the start casting button from the Android companion application. The UI is pretty simple, there is the 8x8 checker board, 12 “black” pieces and 12 red pieces. Due to the color scheme of the board the “black” pieces are shown as grey, we kept the name black as that is what is traditional in checkers. We show the name, “Chromecast Checkers” at the top. Below the board is an indicator of whose turn it is, and what the scores are. Scores are tallied by pieces jumped. If red has a score of 5, then red has “jumped” or taken 5 black pieces. This is how the status of the game is kept. Once a player has a score of 12, the other player will have no remaining pieces left and will lose. See figure 5 for a final game screen UI.

The selector is a green highlight square that is moved around by the Android controller and can “select” pieces and squares. When a piece is selected, that piece is highlighted blue. The spaces open to jump to are also highlighted blue. As seen in figure 4, a double jump can take place, and the player can move elsewhere, take a single jump, or complete the full double jump. Jumps further than a double jump can occur and will show up. Triple jumps have happened and work as intended. Kings are shown with a bold faced K, see figure 5.

Figure 5:



Continuing from the last figure, Figure 5 shows how the game board changes when a player has finally won. Besides when the scores update when a player jumps the other, this is the only time when the screen changes. The player turn line and the player score lines go away and the player who won is notified in big bold print. From here the Android companion app will handle the "teardown", as it is called, for the Chromecast Application to close and resume the normal Chromecast screen.

Figure 6:

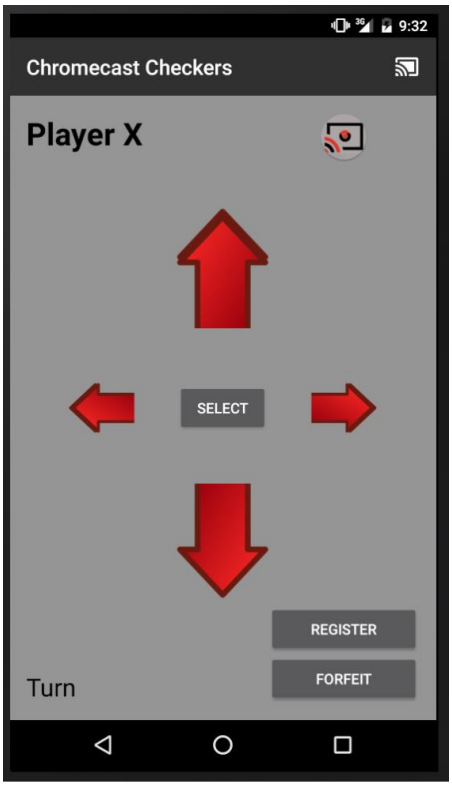


Figure 6 is the GUI of the Android companion application, as the player starts the app. There are 8 buttons. There are 4 movement buttons, up, down, left, and right. There is a select button, register button and a forfeit button. Finally, there is a small cast button in the upper right hand corner across from the Chromecast Checkers title. To start off, when the player opens the application the first thing to do will be to press the cast button and select the Chromecast they want to play on, usually this will only have one Chromecast. Both players will need to do this and connect to the same Chromecast. Next the players will both press the register button, this will prevent other users from trying to interrupt the game and "steal" access by trying to register their own device. The first player to register will be player 1 and will be red, the last to register will be player 2 and be black.

Once the player has connected and has registered, the "Player X" will change to "Player #" to indicate which player they are, "Turn" will change to "Your Turn" or "Waiting for Turn" to indicate if it is their turn or not, and the register button will disappear. At any time, during their turn, a player may forfeit. If the player becomes unable to move pieces they will be forced to forfeit.

Figure 7 and 8:

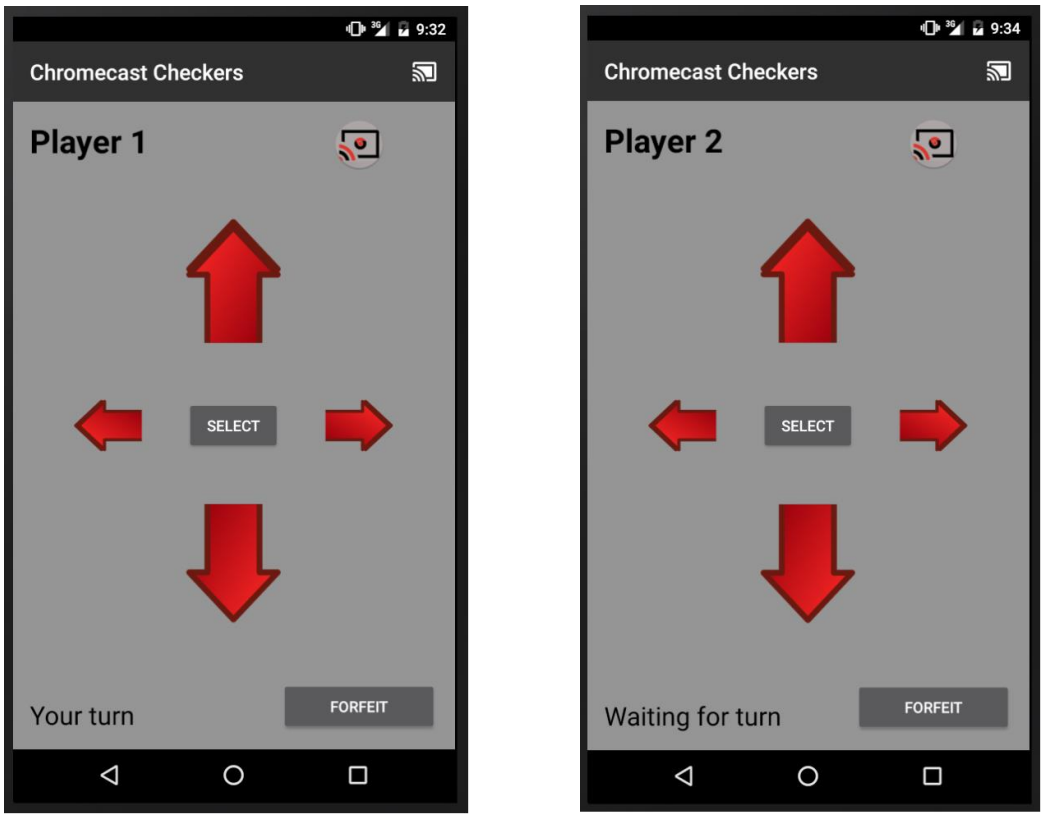


Figure 7 and 8 simply show the different player and turn texts that will be shown to users.

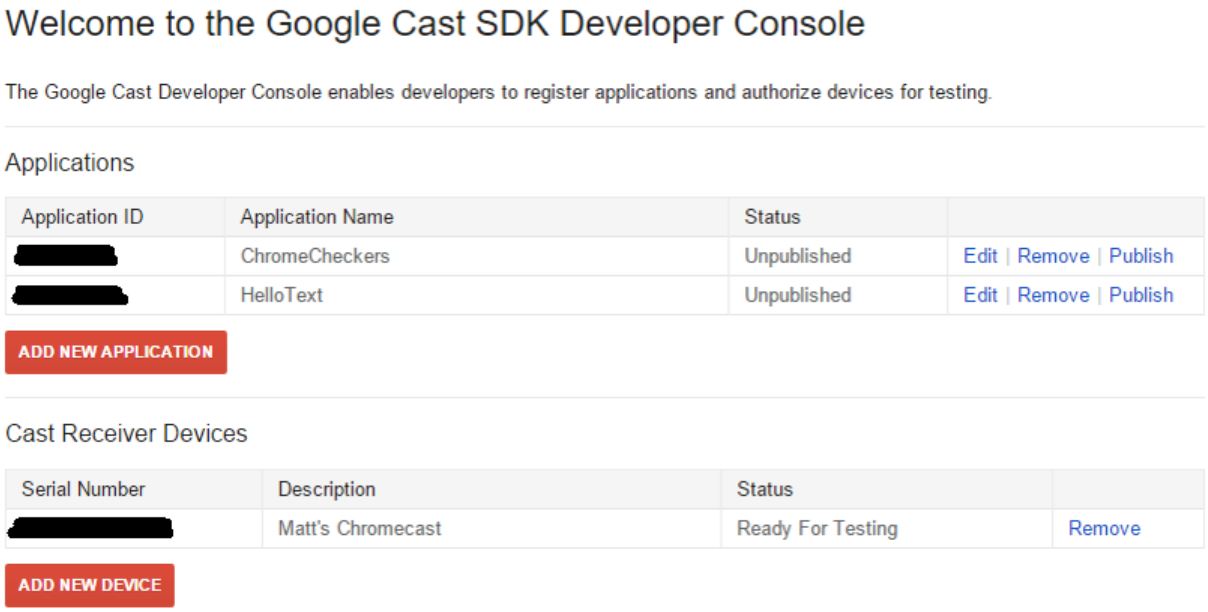
VIII. Google Cast

This section is dedicated to explaining how to setup the cast system to use this application on the Chromecast and Android.

Starting with the Android, the app simply will need to be published to the Google Play Store. From here the application can be downloaded by users and be used as normal to connect to a Chromecast and begin casting Checkers. Alternatively, download the supplied .APK file and install that on your Android device. This file is what Android uses to install applications. This file is the installer / distribution.

The Chromecast application is a bit more complex. There is nothing to install on individual Chromecast devices. What one (the Customer, Chad, for example) would need to do is to publish the Chromecast app on the Google Cast SDK Developer Console, see figure 9. When you publish this application you initialize a place that has the server file hosted. Such as for our team, Matt Dunbar has his own website. We hosted the JavaScript Server file on his website. In his Google Cast SDK Developer Console, we have an application, "Chromecast Checkers", and this SDK points to the JavaScript server file on his website. Every time a user hits the cast button on the Android App, the SDK knows what application to use (this is hard coded within the companion application), finds where to look for this server file and fetches a copy of it. From here the application is cast to the Chromecast and runs.

Figure 9:



In figure 9, we see that there is an application called ChromeCheckers which has an application ID. This ID is hard coded into the Android app so that Google knows what to fetch when someone begins casting. Back to the SDK console, when an application is created there has to be a place initialized where the server file is located, wherever that may be. The status can be set to unpublished or published. When the status is unpublished only a developers registered cast receiver devices will be allowed to test, as shown that Matt's Chromecast is ready for testing. Once an application is published, anyone can begin casting from the Android companion app to their own Chromecast. Be sure that the location of the server file can handle any traffic, because when someone begins casting, Google will fetch the file from the hosting location. Chromecast applications are not stored on the Chromecast or on Google servers.

IX. Implementation

A. Issues that Occurred:
   i. One minor issue is that, with concerning our customer, we didn't prioritize informing him every few weeks of our progress. We have gone since the beginning of march without contacting him for much. This is not to say we haven't worked on the program/project or done anything wrong necessarily, He never said that He strictly wanted contact every few weeks. We have completed the code, based on his requirements, but now that I look back I wish we would have kept him in the loop more. This will be a learning lesson on better communication in the workplace.
   ii. Another problem, albeit not very big, was communication within our team. One day we had scheduled to meet at 10. We never specified 10am or 10pm. Like I said it wasn't a big deal, part of the group met then, then met later as well. We worked together nicely other than that.
   iii. The only other "problems" that occurred were the parts of the project we didn't have time to implement. They were superfluous but I wish we could have added them to the game. Some additions were: adding sound the Chromecast server file, adding a restart option after a game is finished, adding iOS support, and adding single player support.

B. Changes to Design Made:
   i. We didn't end up using Google's GameManager API
   ii. When it is not a players turn the Android screen does not grey out, but the button become un-clickable and the screen displays 'Waiting for Turn'
   iii. We did not end up using a back button, the Select button can be used to reselect any other piece once one has already been selected

C. Restrictions and Limitations:
   i. Due to the time limit of the semester and limitations of how much we can work on this project in conjunction to our other classes, we weren't able to implement as

much as we had hoped for. Such as a restart function, iOS support, and single player support.

X. Testing

A. Unit/Function Testing:

Test Case	**Input**	**Expected Output**	**Actual Output**	**Team Member**	**Comments**
New Game	Someone initiates the Checkers Game	Game starts and waits for another player to connect	Game starts and waits for another player to connect	Zach	Works as Intended
New Game	Board is created	8 x 8 board is created with top and bottom 3 rows filled with 12 pieces each, correctly spaced and colored	8 x 8 board is created with top and bottom 2 rows filled with 12 pieces each, correctly spaced and colored	Zach, Omid	Started off with only 2 rows of pieces, It has been changed to have a full 3 rows.
Player 1 Turn	Game waits for player 1 input to be sent from Android	Upon correct choice the piece moves to players desired spot	Upon correct choice the piece moves to players desired spot	Matt	Works as Intended
Player 1 Incorrect Input	Game waits for player 1 input to be sent from Android	Upon an incorrect choice from player 1, a message is sent back to android and game waits for another choice	User was allowed to move pieces to illegal places.	Omid	Fixed the function that checked for legal places and told the user which place they could move to
Player 2 Turn	Game waits for player 2 input to be sent from Android	Upon correct choice the piece moves to plays desired spot	Upon correct choice the piece moves to plays desired spot	Matt	Works as Intended
Player 2 Incorrect Input	Game waits for player 2 input to be sent from Android	Upon an incorrect choice from player 1, a message is sent back to android and	User was allowed to move pieces to illegal places.	Omid	Fixed the function that checked for legal places and told the user

		game waits for another choice			which place they could move to
Player Jumps a piece	Player's piece jumps another piece	The jumping piece moves to its final destination, and the jumped piece(s) are removed from the board	The jumping piece moves to its final destination, and the jumped piece(s) are removed from the board	Omid	Works as Intended
Player Double Jump	If the player can Double Jump and chooses to Double jump [Includes multiple jumps]	The piece moves to players desired spot, and jumped pieces are removed from the board	The piece moves to players desired spot, and jumped pieces are removed from the board	Omid	Works as Intended
Piece reaches other side	ONLY When a player's piece makes it to the other side it becomes king [Function to replace piece with King]	The piece moves to players desired spot, and The piece becomes a "King"	The piece moves to players desired spot, and The piece becomes a "King"	Omid	Works as Intended
King Piece is chosen to move	Kings are allowed more directions to move than regular pieces. [Function to check if piece is King]	Kings will be allowed to move in more directions than regular pieces	Kings will be allowed to move in more directions than regular pieces	Omid	Works as Intended
End of Game	End of game is detected	The game is ended appropriately	The game is ended appropriately	Omid	Works as Intended

The Unit/Function Testing is the JavaScript phase of testing.

To complete to Unit/Function testing we implemented a method to play the game without input from the android applications. We created a small set of clickable arrows as buttons that will act as the "controllers" for the purposes of testing the JavaScript checkers code and functions. This testing feature will be taken out and unusable in the final product. We tested the program as a single JavaScript program to make sure that the game works as intended. We have made sure that the clickable arrow buttons are easily translated to the Android JSON messages that call the same functions to make movements. There is also the test function to make sure the input is valid, otherwise a message is sent that the input was wrong and to try again.

We each played multiple games under this testing version with various people to eliminate any bugs. We have tested the functions that check jumps, this includes double or multi-jumps, that checks when a piece reaches the other side and is changed to a King, and that checks if a piece chosen to move is a King than it is allowed to make a wider range of moves.

The last portion we tested was how the game ends. There is a function that after each player's turn checks the state of the game. If one of the two players has no pieces remaining the game ends, with the one who has pieces winning. Also, if a player cannot make any moves they lose. There is a function that runs before every turn, if a player cannot move any piece the end of game is detected and they lose.

After we tested this JavaScript portion, we took the clickable arrows out and made them hidden from user view. We added back in the functions to communicate with the android applications. The messages received from the applications will be parsed out and the function to check if the input was correct or legal was checked. If the input was not legal we send back an error code JSON message and then wait for another input. Once we get legal input, the appropriate tested functions will be called.

B. System/Integration Testing:

Test Case	**Input**	**Expected Output**	**Actual Output**	**Team Member**	**Comments**
Player 1 Disconnects	Android Phones disconnect	Game waits until player reconnects or the player forfeits after a set time	Game waits until player reconnects or the player forfeits after a set time	Matt	Works as intended.
Player 2 Disconnects	Android Phones disconnect	Game waits until player reconnects or the player forfeits after a set time	Game waits until player reconnects or the player forfeits after a set time	Matt	Works as intended.
Player 1 Reconnects	Android Phone Reconnects	Game resumes where it was left off at	Game resumes where it was left off at	Matt	Works as intended.
Player 2 Reconnects	Android Phone Reconnects	Game resumes where it was left off at	Game resumes where it was left off at	Matt	Works as intended.
Player 1's Turn	Player 2 Cannot Input	Player 2's android screen is greyed out and will not accept input	Android screens could not be greyed out. The buttons were made unclickable instead.	Matt	Had to change from greying out to making the buttons unclickable when it is not your turn
Player 2's Turn	Player 1 Cannot Input	Player 1's android screen is greyed out and will not accept input		Matt	
Player Clicks on Any Button	A Button is pressed	A Correctly Formed JSON Message for that button is formed	JSON Message was not sending correctly	Zach	We had to rework the command, but it is now working
Player Clicks on the Left button	Left button is pressed	1. JSON Message is sent to Chromecast 2. The JSON Message is parsed by the Chromecast 3. If the player cannot move there a response message is sent back with Error	We all three tested these buttons to make sure the commands were being sent correctly, and in turn receiving correct messages and parsing the	Matt, Zach. Omid	We all three worked on this testing and debugging, so we all three worked on fixing the
Player Clicks on the Right button	Right button is pressed			Matt, Zach. Omid	
Player Clicks on the Select button	Select button is pressed			Matt, Zach. Omid	

Player Clicks on the Up button	Up button is pressed	4. Player will then have to choose again and the process repeats 5. If the choice is good the Chromecast sends back a message that it is no longer that players turn	return message correctly. There were minor bugs such as the return message not being correctly formed and the message not being parsed correctly	Matt, Zach. Omid	bugs that arose.
Player Clicks on the Down button	Down button is pressed			Matt, Zach. Omid	
Player Incorrect Input	JavaScript sends back an Error JSON Message	When an Error JSON Message is received back the application repeats for new user input while also displaying an Error message that the previous input was not valid	When an Error JSON Message is received back the application repeats for new user input while also displaying an Error message that the previous input was not valid	Matt	Works as intended

The System/Integration Testing is the Android phase of testing.

Unfortunately for the Android Application there was not a simple thing we could do to test the different individual functions. Fortunately, the Android Application is pretty simple. The simple UI only has a few buttons for input, which send JSON messages to the server, and text. Along with the buttons functions there are also functions for when it is not that player's turn and a function to handle return JSON messages.

The first thing we tested when we connected the Android Applications to the JavaScript server was to test what happens upon disconnect. The applications attempt to reconnect. If there is no reconnection after a set time limit that player forfeits. If both players disconnect at the same time and do not reconnect the JavaScript program will display that both players have forfeited and exit the game. The JavaScript program is always listening for the applications to reconnect, while the applications are programmed to always try to connect.

After a player gives a valid input the JavaScript will send back a success message. When the application receives this message it will grey out the UI so that no input will be accepted. When this happens the JavaScript will also send a message to the opposite player unlocking their screen so that they will be able to input for their turn. This then repeats as players input valid moves.

When buttons are pressed a function switches based on which buttons were pressed. This function then creates a JSON message based on what button was pressed and what the player wants to do and sends it back to the JavaScript program. The application will be listening for a reply message, if this reply is an error message than the application will repeat itself waiting for new user input. An error message/box will be displayed letting the player know that the previous input was incorrect.

C. User Testing:

Test Case	**Input**	**Expected Output**	**Actual Output**	**Team Member**	**Comments**
Game Plays Smoothly	Input from android to move pieces	Game smoothly and quickly processes moves and displays the moves on the board	Game play went smoothly and pieces responded quickly to input	Zach	Worked well
Everything is Readable from a distance	Any words on the Chromecast screen is readable in its size and font	Readable Text	Chromecast had readable text	Zach	Looks good
Messages are Readable on the Android Devices	Any messages that need to be displayed on the screen are readable and stay long enough to be read	Readable Messages	Some text boxes were not sized correctly, and made text unreadable	Zach	Text Boxes were resized and looks good now
Text is Readable on the Android Devices	Text and Button Text are readable on the Android screen	Readable text on the Android screen, users have no problems differentiating buttons	Some text boxes were not sized correctly, and made text unreadable	Zach	Text Boxes were resized and looks good now
Android App works on Android 4.0.4 and above	App runs on devices 4.0.4 and above	App runs and UI is the same	Some commands used made the app not work on Android versions lower than 4.0.4	Matt	At the moment only Android version 4.0.4 and above are stable and the app can be installed and played on

User Testing is the last phase of testing. This testing is to ensure that the UI on both the JavaScript/Chromecast display and the Android Application are readable and everything is played smoothly. An example would be, a new game starts without error, there are no display glitches, the Android Applications connect and stay connected, the input communications happen timely, the board is updated quickly and smoothly, the application UI is locked and unlocked appropriately, and the JavaScript detects the end of the game correctly and displays the winner before finally closing out.

XI. Future Enhancements/Maintenance

A. Future Enhancements:
    a. iOS Support
    b. Single Player Support
    c. Restart functionality after finishing a game
    d. Background sound to be played through the Chromecast
        i. One option is to add some soothing background music while you play, at the moment nothing plays through the TV speakers and seems like a wasted opportunity.
        ii. Another option to help the visually impaired was suggested at the final presentation and is as follows:
            1. Adding sounds to indicate where the "cursor" is at, which piece is selected, and where you can move to.
            2. Adding onto this other option, adding sound to the Android companion application for what buttons the user is pressing on and if it is your turn or not can also help.
    e. Animations
        i. This will be the hardest or most complex.
        ii. Animations would have to be done through JavaScript, for better jumping and moving motions, as well as a better king symbol and animation.

B. Issues raised during Testing:
    a. We have the code for the restart functionality in our final code.
        i. It is just commented out and is not called in this version.
        ii. The code for some reason is not working as intended and it was taking too long to debug when we were closing in on having to be done.
        iii. We decided as a group to take the functions out of use but keep them in the files for future use and testing.

C. Maintenance Documentation
    a. We have added more comments and spacing to enhance the readability and

understandability to our code. This is in conjunction to our professor's feedback on the matters.

XII. Conclusions

The only real change to the customer requirements was that a back button wasn't needed. With how we coded the program the select button automatically reselected a new piece when you highlight over and select a new different piece. Everything else was straight forward, and the customer wasn't strict in requirements, features, or UI looks. The change made didn't affect the design or schedule at all. It was only one less button and function we didn't have to implement into the Android companion application and JavaScript server file, respectively. One major difficulty we faced during this semester was time. In conjunction with other classes and work, as seniors we have difficulties with procrastination. We managed fine between the 3 of us although, and we pulled together to bring these applications to fruition. Some of the lessons learned during this semester and project are communication. Whether between team members, superiors, or customers having constant reliable communication channels is always a good thing so everyone is up to date and in the clear on what is happening. Another lesson we learned is about company / production level API / source code. We learned a lot about what production level code looks like and how to work with it with dealing with Google's API in both the Android library and Chromecast library. Both of these libraries took a while to learn, in how to use and how they worked. That was a lesson on its own apart from just learning how production level code looks like. One more lesson we learned was how to work with AngularJS and JavaScript. Individually both are powerful, but together a lot can get done in relatively small amounts of code. As seen with this program, the entire checkers game calculations were done in under 500 lines of code. Some things we would do differently if given the opportunity would be time management. If we had better time management and were more effective and efficient with our time we might could have been able to implement a few more of the things we wished to implement.

XIII. References

A. Google's Chromecast API documentation -
    a. https://developers.google.com/cast/docs/developers#components
B. Google's Chromecast Hello world example for Android -
    a. https://github.com/googlecast/CastHelloText-android
C. Google's Chromecast Hello world example for iOS -
    a. https://github.com/googlecast/CastHelloText-ios
D. Google's AngularJS Developer dependencies -
    a. https://docs.angularjs.org/misc/downloading

IXV. Game Guide

A. Android Controller:
   a. How to Play:
      i. There are 8 buttons. There are 4 movement buttons, up, down, left, and right. There is a select button, register button and a forfeit button. Finally, there is a small cast button in the upper right hand corner across from the Chromecast Checkers title.
      ii. To start off, when the player opens the application the first thing to do will be to press the cast button and select the Chromecast they want to play on, usually this will only have one Chromecast. Both players will need to do this and connect to the same Chromecast.
      iii. Next the players will both press the register button, this will prevent other users from trying to interrupt the game and "steal" access by trying to register their own device. The first player to register will be player 1 and will be red, the last to register will be player 2 and be black.
      iv. Once the player has connected and has registered, the "Player X" will change to "Player #" to indicate which player they are, "Turn" will change to "Your Turn" or "Waiting for Turn" to indicate if it is their turn or not, and the register button will disappear.
      v. At any time, during their turn, a player may forfeit.
      vi. If the player becomes unable to move pieces they will be forced to forfeit.

   b. How to Install:
      i. The app can be published to the Google Play Store. From here the application can be downloaded by users and be used as normal to connect to a Chromecast and begin casting Checkers.
      ii. Alternatively, download the supplied APK file and install that on your Android device. This file is what Android uses to install applications. This file is the installer / distribution.

B. Chromecast JavaScript File:
   a. How to Play:
      i. Below the board is an indicator of whose turn it is, and what the scores are. Scores are tallied by pieces jumped. If red has a score of 5, then red has "jumped" or taken 5 black pieces. This is how the status of the game is kept. Once a player has a score of 12, the other player will have no remaining pieces left and will lose.
      ii. The selector is a green highlight square that is moved around by the Android controller and can "select" pieces and squares.
      iii. When a piece is selected, that piece is highlighted blue. The spaces open to jump to are also highlighted blue.
      iv. The player can move to whichever place they want like a single space over, take a single jump, or complete the full double jump. Jumps further than a double jump can occur and will show up. Triple jumps have

happened and work as intended.

v. Kings are shown with a bold faced K, and can move in any direction, not only up.

b. How to Install:
    i. The Chromecast application more complex. There is nothing to install on individual Chromecast devices. What one (the Customer, Chad, for example) would need to do is to publish the Chromecast app on the Google Cast SDK Developer Console, see figure 9. When you publish this application you initialize a place that has the server file hosted. Such as for our team, Matt Dunbar has his own website. We hosted the JavaScript Server file on his website. In his Google Cast SDK Developer Console, we have an application, “Chromecast Checkers”, and this SDK points to the JavaScript server file on his website. Every time a user hits the cast button on the Android App, the SDK knows what application to use (this is hard coded within the companion application), finds where to look for this server file and fetches a copy of it. From here the application is cast to the Chromecast and runs.