Chromecast Checkers

Authors: Zach Almon, Matt Dunbar, Omid Omidi

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

Program Requirements:

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

Program Environments:

Chromecast Application / Google API for the Chromecast
Host / Server JavaScript that handles communications and Checkers Logic
Android Controller Application / Google API for Android

Program Use Scenarios:

1. User starts Chromecast
2. User starts Checkers for Chromecast App
3. User(s) starts Android controller companion app
4. User(s) connects Android controller to Chromecast
5. User(s) start a new game
6. User with first move will select and move whichever piece they wish
    a. Second user will be greyed out at this time, unable to do anything
7. Second user will gain the ability to move after the first user
    a. First user will be greyed out at this time, unable to do anything
8. Repeat 6-7 until someone has no pieces' left
9. 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).

Program Diagrams:

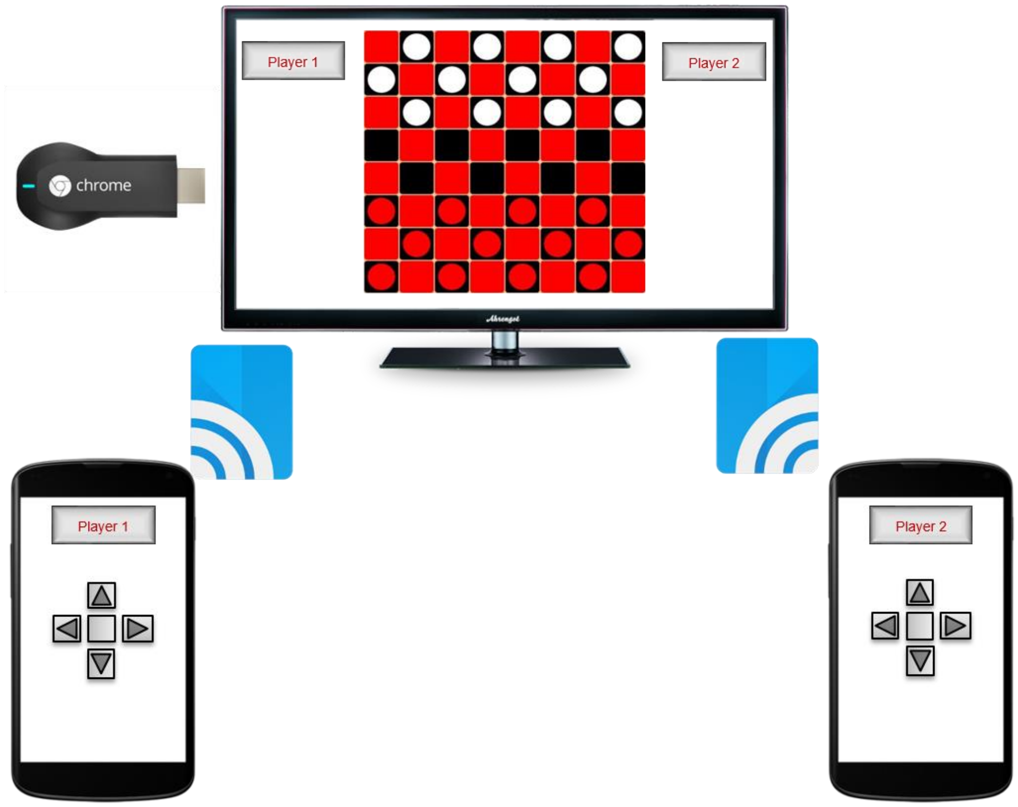


Diagram one is to show the components and interfaces. The server side runtime program is not seen by users. The Chromecast application is mainly a display, but will have to be turned on, and the application will have to be run. The Android controllers have the user input to move pieces, as well as maintaining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

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.

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.

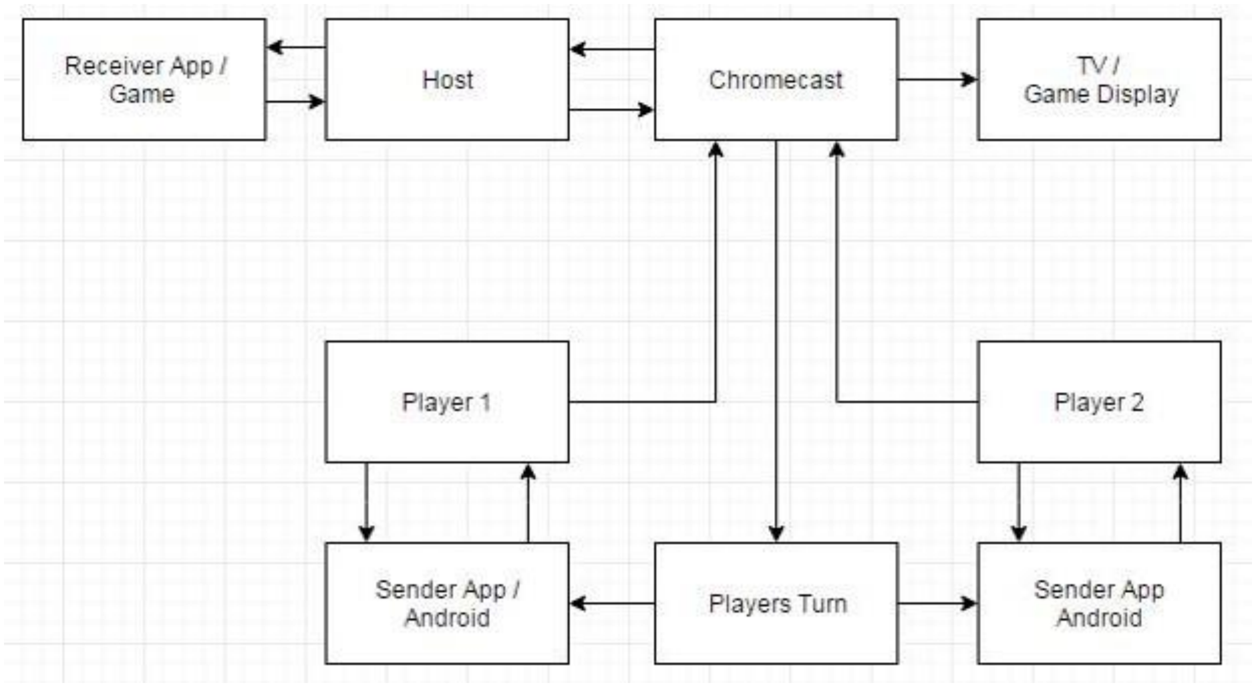
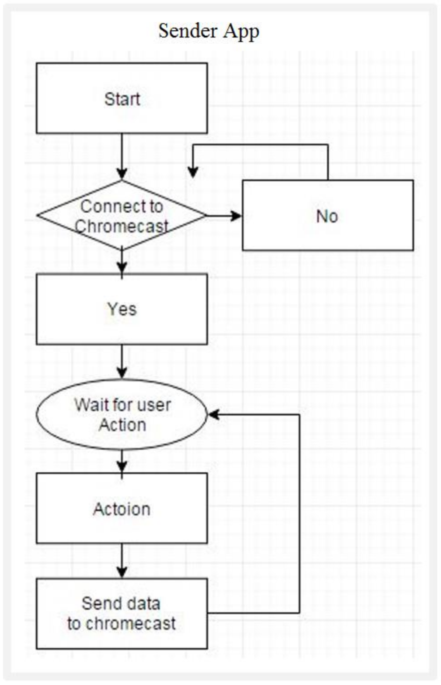


Diagram two is similar to diagram one, but more technical in showing the actual communication flow between the programs and applications. Starting with the Chromecast, it displays to the TV/game display. It also communicates to the hosting server and receiver app. Both of these programs process the messages 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.



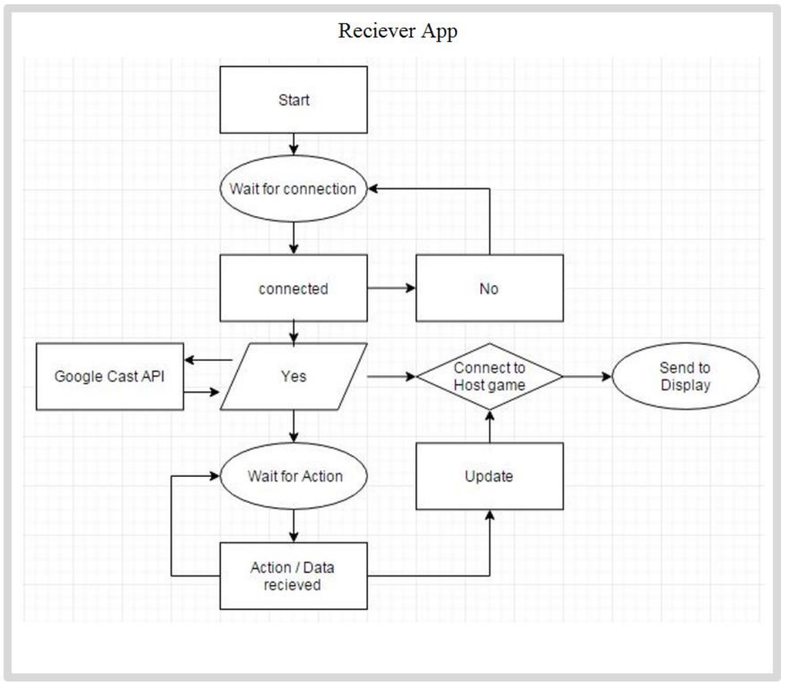


Diagram three has two parts. Starting with the sender application, the Android controller, the app starts up and attempts to connect to the Chromecast. From here it will take user input and send any input to the Chromecast. It is also getting data from the Chromecast, in which could disable the controller 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.

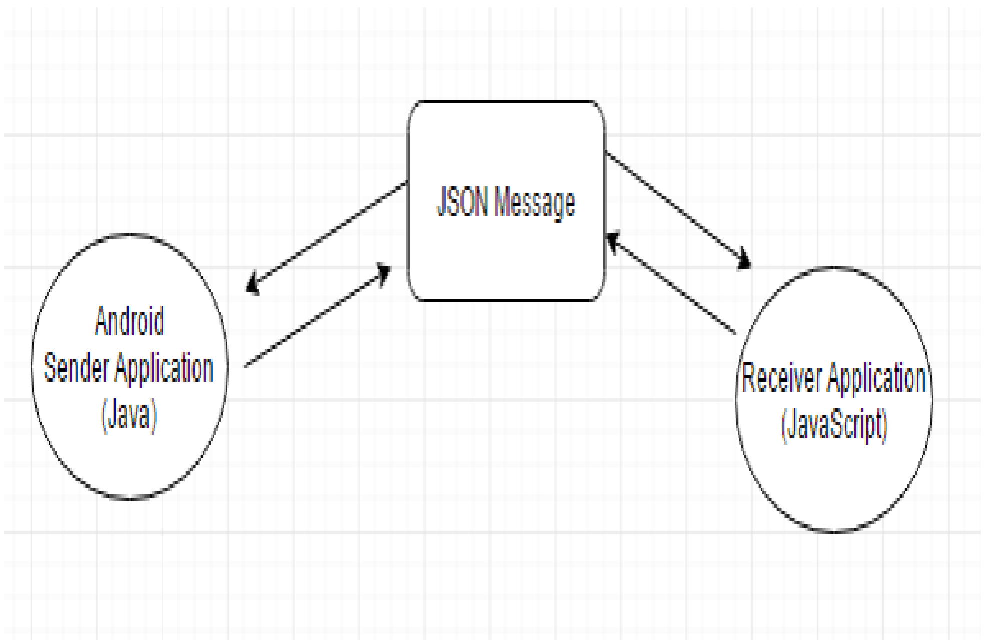
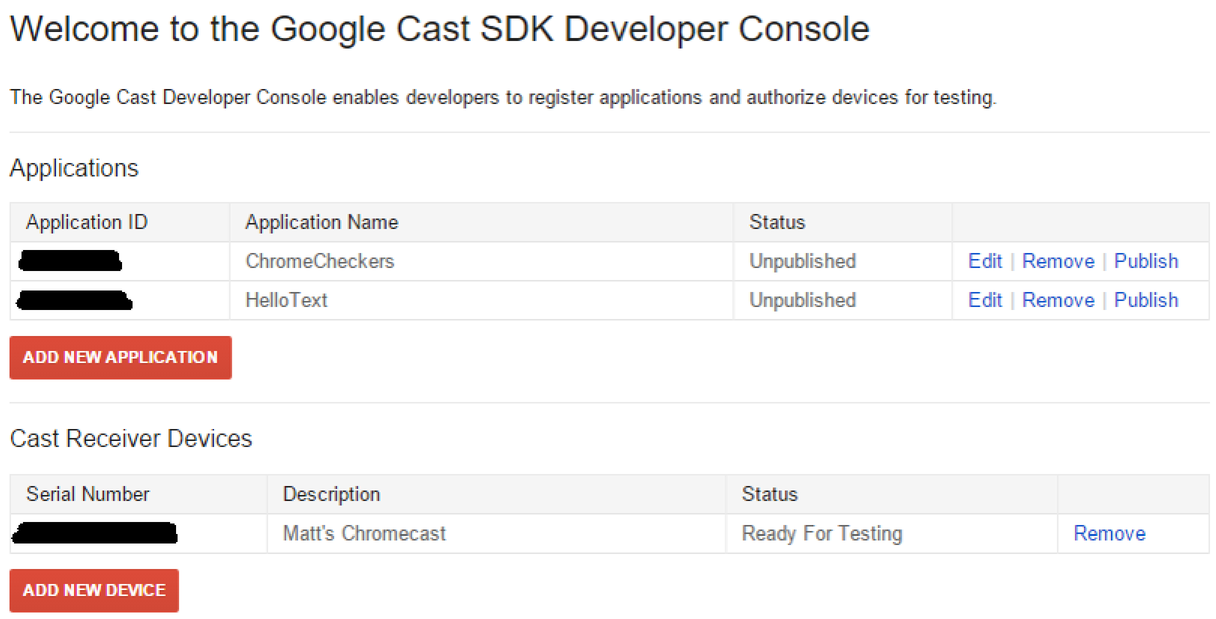

JSON Message
Android Sender Application (Java)
Receiver Application (JavaScript)

Diagram four 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.



Welcome to the Google Cast SDK Developer Console

The Google Cast Developer Console enables developers to register applications and authorize devices for testing.

Applications

Application ID	Application Name	Status	
[redacted]	ChromeCheckers	Unpublished	Edit | Remove | Publish
[redacted]	HelloText	Unpublished	Edit | Remove | Publish

Cast Receiver Devices

Serial Number	Description	Status	
[redacted]	Matt's Chromecast	Ready For Testing	Remove

This image, while not a diagram, is important to note that in order to create Chromecast applications registering the application and authorizing a device to test on is necessary. Application ID's are assigned randomly, and after the application is finished the app can be published to the Google Store.

Index.html

Author: Omid Omidi

Date: 3/5/2016

Description: This is the index page that displays the game by pulling information from other files. This HTML file calls Checkers.js functions to implement the checkers game logic and update the display on the screen. It also handles the communication between the Chromecast App, Android App and itself.

Preconditions: Chromecast is connected to TV and the Server/receiver app.

Postconditions: Game has ended, program has closed.

Size: Around 200 lines of code for all function calls and screen update code

Pseudocode:

```
Use Checkers.js for function calls
Get styling info from Checkers.css
Make a table for checkers board by calling setClass
Insert pieces into board by calling setStyling
Receive messages from AndroidApp.jar
Call newGame
Display Player
Display Red Score
Display Black Score
Receive messages from AndroidApp.jar
Call function Select every time button clicked from
checkers.js
Call king function to see if the piece needs to change to king
Button Up
  On click highlight available upper piece
Button down
  On click highlight available lower piece
Button left
  On click highlight available left piece
Button Up
  On click highlight available right piece
Button Up
  On click select piece
```

Checkers.js
Author: Omid Omidi
Date: 3/5/2016
Description: This is the back end pseudo code that will handle all the game movements and AI
Preconditions: Chromecast is connected to TV and to this Server/receiver app.
Postconditions: Game has ended, program has closed.
Size: Around 900 lines of code

Function: Piece(x,y)
Preconditions: X,Y integer coordinates. Must be 0 <= X,Y <= 8
Postconditions: Creates a Piece with coordinates X,Y
Description: Pieces of checkers game
Size: Less than 20 lines of code, function is only to initialize each piece
Pseudocode:

```
      chose player
      define x
      define y
      is not King
      Choice is false
```

Function: newGame()
Preconditions: Chromecast is turned on, Android apps are connected
Postconditions: New game has been initialized and the current game has started.
Description: start new game
Size: Around 100 lines of code for creating a new game
Pseudocode:

```
  player = RED
  red Score = 0
  black Score = 0

  board Array
  from i = 0 to BOARD_WIDTH
       board[i] = []
       from j = 0 to BOARD_WIDTH
          if (i = 0 and j % 2 = 0   or i = 1 and j % 2 = 1)
               board[i][j] = new Piece(i,j) BLACK
          else if (i = BOARD_WIDTH - 2 and j % 2 = 0 or i =
  BOARD_WIDTH - 1 and j % 2 = 1)
              board[i][j] = new Piece(i,j) RED
        else
```

```
              board[i][j] = new Piece(i,j) null
```
Function: setStyling()

<u>Preconditions</u>: Styling info has been imported from Checkers.css, Table has been made by setClass

<u>Postconditions</u>: The board has been colored and displayed on the Chromecast

<u>Description</u>: color the board

<u>Size:</u> Less than 30 lines, only to create the style for the checkers board.

<u>Pseudocode</u>:

```
      if square player = RED
          return backgroundColor red
      else if square player = BLACK
          return backgroundColor black
      return backgroundColor none
```

Function: setClass()

<u>Preconditions</u>: Styling info had been imported from checkers.css

<u>Postconditions</u>: The table for the board has been made and displayed on the Chromecast

<u>Description</u>: selection style

<u>Size:</u> Less than 50 lines of code to set the class for each square of the board

<u>Pseudocode</u>:

```
  if square y % 2 = 0
       if square x % 2 = 0
            return (backgroundColor if square choice then
       "green" else "black")
       else
            return backgroundColor": "white"
  else
       if square x % 2 = 1
            return (backgroundColor" if square choice then
       "green" else "black")
       else
            return backgroundColor": "white"
```

Function: select()
Preconditions: Button is clicked, message from AndroidApp
Postconditions: The selected piece will be shown on the screen via highlight
Description: after choosing a piece, this will select it
Size: Less than 100 lines of code for selecting pieces
Pseudocode:

```
    if selected square not = null and not player square
            call function movePiece
            call function reset_choices
    else if square player = player
            selected square = square
            call function reset_choices
            call function set_choices
    else
            selected square = null
```

Function: reset_choices()
Preconditions: Set choices was set for the previous turn. This function has been called on the current turn.
Postconditions: Choices has been reset to empty, so that set choices can be set for the current turn.
Description: reset user choice
Size: Less than 30 lines of code to reset the available choices
Pseudocode:

```
  from i = 0 to BOARD_WIDTH
    from j = 0 to BOARD_WIDTH
         board[i][j] choice = false
         board[i][j] matados = []
```

Function: movePiece()
Preconditions: If piece has been finalized, set, and selected this function is called to move the piece on the board.
Postconditions: Piece was moved and displayed on the screen. This function also handles moving old pieces
Description: move the piece that user selected
Size: Around 50 lines of code to actually move the piece and move jumped pieces off the board
Pseudocode:

```
  if square chosen
  // Jump dude
    From i = 0 to square matados length
```

```
          Jump square matados[i]
          Move Jumped piece to graveyard
              Erase the current position of piece
              Display the Piece in the predefined Graveyard Area
```

Function: is_King()
Preconditions: Piece has been selected, and the coordinates are known.
Postconditions: Piece becomes king if it is within a certain spot of the board.
Description: Checks to see if the piece should become king
Size: Less than 40 lines of code to check if a piece should be a king
Pseudocode:

```
      if player = RED
          if square y = 0
               return true
      else
          if square y = BOARD_WIDTH - 1
               return true

      return false
```

Function: jump_Jumped()
Preconditions: If a piece can make a double jump
Postconditions: The piece makes a double jump
Description: make the double jumps
Size: Around 40 lines of code to double jump
Pseudocode:

```
      jumped player = null
      jumped is king = false
      if player = RED
               redScore plus one
               if redScore = 8
                    timeout function (gameOver RED)
      else
               blackScore plus one
               if blackScore = 8
                    timeout function (gameOver BLACK)
```

Function: set_choices()
Preconditions: Button is clicked, message from AndroidApp, Select has been called which calls this function to see what choices can be made.
Postconditions: Users available moves are set for current turn.
Description: Sets the choice of user
Size: Around 400 lines of code, this is the longest function to check what choices are available to the user
Pseudocode:

```
      if  depth > 10  return

      // Upper Choices
      if      player = RED or selected square  is king
        // Upper Left
        if  x > 0 and y > 0
            UP_LEFT =     board[y-1][x-1]
          if  UP_LEFT  player   is true
            if  UP_LEFT  player not =     player
              if   x > 1 and y > 1  and not  x - 2 = old X and y
- 2 = old Y
                  UP_LEFT_2 =     board[y-2][x-2]
                if  not UP_LEFT_2  player
                  UP_LEFT_2  choice = true
                    jumpers = matados  slice 0
                  if  jumpers  indexOf UP_LEFT  = -1
                    jumpers  push UP_LEFT
                  UP_LEFT_2  matados = jumpers
                  set choices x-2 y-2 depth+1 jumpers x y


            else if  depth = 1
            UP_LEFT  choice = true


        // Upper Right
        if  x < BOARD_WIDTH - 1 and y > 0
            UP_RIGHT =     board[y-1][x+1]
          if  UP_RIGHT  player
            if  UP_RIGHT  player not =     player
              if   x < BOARD_WIDTH - 2 and y > 1  and not  x + 2
= oldX and y - 2 = oldY
                  UP_RIGHT_2 =     board[y-2][x+2]
                if  not UP_RIGHT_2  player
                  UP_RIGHT_2  choice = true
                    jumpers = matados  slice 0
                  if  jumpers  indexOf UP_RIGHT  = -1
                    jumpers  push UP_RIGHT
                  UP_RIGHT_2  matados = jumpers
                  set choices x+2 y-2 depth+1 jumpers x y

            else if  depth = 1
            UP_RIGHT  choice = true


      // Lower Choices
      if      player = BLACK or selected square  is king
        // Lower Left
        if  x > 0 and y < BOARD_WIDTH - 1
            LOWER_LEFT =     board[y+1][x-1]
          if  LOWER_LEFT  player
            if  LOWER_LEFT  player not =     player
              if   x > 1 and y < BOARD_WIDTH - 2  and not  x - 2
= oldX and y + 2 = oldY
                  LOWER_LEFT_2 =     board[y+2][x-2]
                if  not LOWER_LEFT_2  player
                  LOWER_LEFT_2  choice = true
                    jumpers = matados  slice 0
                  if  jumpers  indexOf LOWER_LEFT  = -1
                    jumpers  push LOWER_LEFT
                  LOWER_LEFT_2  matados = jumpers
                  set choices x-2 y+2 depth+1 jumpers x y


            else if  depth = 1
            LOWER_LEFT  choice = true


        // Lower Right
        if  x < BOARD_WIDTH - 1 and y < BOARD_WIDTH - 1
            LOWER_RIGHT =     board[y+1][x+1]
          if  LOWER_RIGHT  player
            if  LOWER_RIGHT  player not =     player
              if   x < BOARD_WIDTH - 2 and y < BOARD_WIDTH - 2
and not  x + 2 = oldX and y + 2 = oldY
                  LOWER_RIGHT_2 =     board[y+2][x+2]
                if  not LOWER_RIGHT_2  player
                  LOWER_RIGHT_2  choice = true
                    jumpers = matados  slice 0
                  if  jumpers  indexOf LOWER_RIGHT  = -1
                    jumpers  push LOWER_RIGHT
                  LOWER_RIGHT_2  matados = jumpers
                  set choices x+2 y+2 depth+1 jumpers x y
```

```
else if  depth = 1
LOWER_RIGHT  choice = true
```

AndroidApp.jar
Author: Matt Dunbar
Date: 3/5/2016
Description: Android Application to setup cast and to send messages.
Preconditions: Chromecast is Setup and waiting on the Android App
Postconditions: Android Application is connected to the Chromecast; Chromecast is sent messages about user input
Size: 200-300 lines of code, with some room to accommodate things we may need to do for android specific tasks

Function setupCast():
Preconditions: Chromecast is Setup and waiting on the Android App
Postconditions: Android Application is connected to the Chromecast.
Description: The Chromecast button needs to appear in the top right corner of the Android phone. This is handled by the MediaRouteSelector which basically filters through the devices that are displayed to the user when the Cast button is pressed and allows you to pick the Chromecast and start the session. Next, the receiver application must be launched. Once the connection is confirmed, the receiver application will launch itself when the Application ID is specified (assuming it is registered with Google, which it is). After the receiver application is up and running the hard part is basically done. Now Checkers game is being run in JavaScript, and all we have to do through the Android sender application is send commands on what the current player wants to do with their checker piece in JSON.
Size: Less than 50 lines of code
Pseudocode:

```
MediaRouteSelector = new MediaRouteSelector.start()
MediaRouteSelector.applicationID = ("APPLICATION_ID")
if MediaRouteSelector = active
      showButton()
startCast()
if mMediaRouteSelector is active
      call startReceiver()

mApiClient = new GoogleApiClient
mApiClient.connect()
Cast.CastApi.launch()
sessionId = getSessionId()
set applicationStatus = active
```

Function AndroidContoller():

Preconditions: The Android Application has been connected to the Chromecast.

Postconditions: The Android Application has been terminated, program is done.

Description: The main Android function. Setup the controller, wait for user input, send user input to receiver app, check Chromecast connection, get messages from the Chromecast, and wait for the users turn.

Size: Around 100 lines of code to make the controller, handle user input, and call functions

Pseudocode:

```
AndroidController
{
      Setup the controller look
      Show the controller on the phone
      While Game is still being played:
            Check connection of Chromecast
                  If not connected:
                        Try to Reconnect, wait until reconnected
                  Else:
                        Check chromecast message
                        Set if Players turn yet
            Check if players turn
                  If players turn:
                        Display controller
                        Wait for user input for move
                        Check to make sure valid move OR
                        Only let user choose valid moves
                  Else:
                        Wait until player’s turn
            Once user input is received call SendMesg with input
}
```

Function SendMesg():

Preconditions: User has selected some command to send.

Postconditions: The action the user selected is sent to the Chromecast receiver application as a JSON message.

Description: The string “MoveUp” is converted to a JSON Object and sent to the receiver application, on the JavaScript / Receiver end we will then take the JSON, parse it, and know that MoveUp means to move the selector on the screen up. This will be the main source of communication between the sender and receiver and there will be many different commands that we can send back and forth.

Size: Around 40 lines to create and send messages to the chromecast

Pseudocode:

```
sendMesg
{
    mesg = "MoveUp";
    updateSentText( "Sent Msg: " + mesg )

    try {
        JSON message = new JSONObject
        message.put( "msg", mesg );
        sendMessage( message );
    }
    catch( Exception e ) {
        Output error message to sender
    }
}
```

As far as the JavaScript/HTML file is concerned, it will be listening for any messages from the sender application and when it finally does get a message in JSON, it will use its handy built in functions that deal with JSON to parse and carry out the message that was sent by the sender app. JSON.parse() and JSON.stringify().

JSON.parse() parses the JSON data into a string again, and from there it can be used as an actual instruction.

JSON.stringify() can turn a string into a JSON object to be sent back to the sender application.

So for example if Player 1 gets disconnected, the receiver app can send both Player 1 and Player 2 a signal to halt the game until Player 1 can successfully reconnect, or until Player 1 times out. What this will mostly be used for, though, is to tell which player whose turn it is and when it's not their turn to send a message to the sender app to turn off the functionality of the buttons so that there is no extraneous input from the user.