Portfolio 2: Checkers

By: Matt Bechtel

HOW TO RUN THE CODE:

As of 12/09/17, the code is currently running on a Node server hosted at “proj-319-P62.cs.iastate.edu:3000”, so all that you have to do is visit that url on Iowa State’s VPN and you can access the Checkers website. If you would like to run it locally, just unzip it and type “npm install” in the current directory, this will install all of the dependencies. Then, type “npm start”, and the application will serve out of port 3000. To visit the site, type localhost:3000 into the web browser.

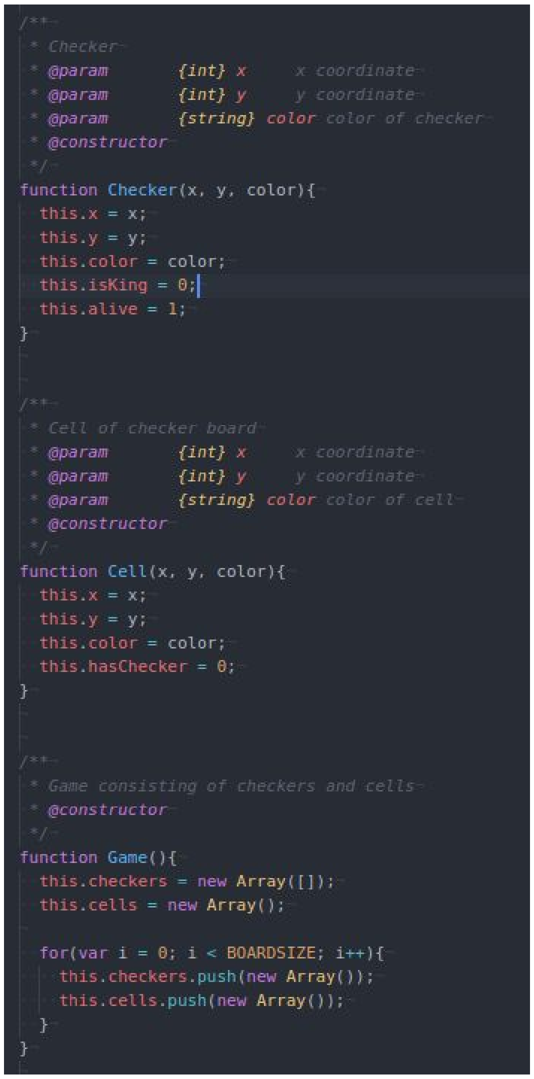
To play, just type in a username and click either create game or join game. If there is no one else playing, there will be no active games, meaning for testing purposes you must open two instances of the site and create a game on one and join it on the other, this will put you in a match with yourself. The game is simply classic checkers with no special rules. When the server detects a winner each player will be notified of the outcome of the game.

FRAMEWORKS/LIBRARIES:

As far a frameworks go, I utilized Node’s Express framework, allowing for quick setup and much customization. The bulk of what was done for the game was done using the Socket.io library, meaning, all information that was passed between the client and the server was done using sockets. Because of the use of Node, all server and client side code is written in JavaScript, and because of the use of the Express framework, all views are written in Jade.

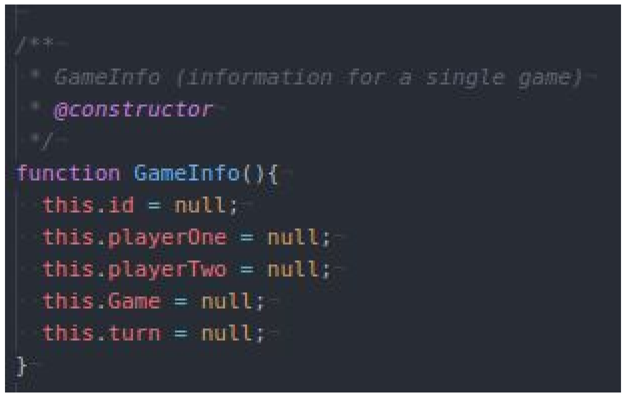
CHECK LIST:

- middleware/game.js: Probably the most important file in the whole project. Here, all of the server side game logic and matchmaking is done. Within this file there are a few important objects.
    - First, the Game object. The Game object is an instance of a game of checkers, containing a checkers array (an array of objects of type Checker), a cells array (an array of Cell objects representing the board).



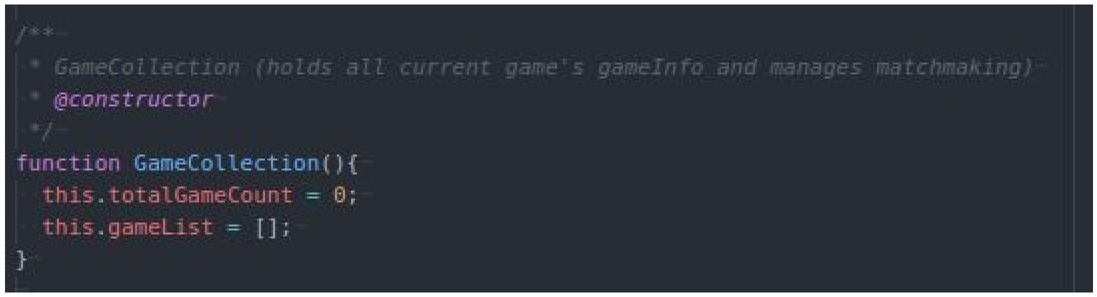
```
/**
 * Checker
 * @param       {int} x      x coordinate
 * @param       {int} y      y coordinate
 * @param       {string} color color of checker
 * @constructor
 */
function Checker(x, y, color){
  this.x = x;
  this.y = y;
  this.color = color;
  this.isKing = 0;
  this.alive = 1;
}


/**
 * Cell of checker board
 * @param       {int} x      x coordinate
 * @param       {int} y      y coordinate
 * @param       {string} color color of cell
 * @constructor
 */
function Cell(x, y, color){
  this.x = x;
  this.y = y;
  this.color = color;
  this.hasChecker = 0;
}


/**
 * Game consisting of checkers and cells
 * @constructor
 */
function Game(){
  this.checkers = new Array([]);
  this.cells = new Array();

  for(var i = 0; i < BOARDSIZE; i++){
    this.checkers.push(new Array());
    this.cells.push(new Array());
  }
}
```

- Second, is the GameInfo object. This object contains information about a game between two players. The object keeps information such as game id, players, an instance of the Game between two players, as well as a integer value representing which player's turn it is in the game. Upon creation of a "game" between two clients, an instance of GameInfo is created with a brand new Game, and that GameInfo instance is placed in an array in the third object that is in the file, GameCollection.

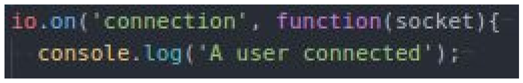


```
/**
 * GameInfo (information for a single game)
 * @constructor
 */
function GameInfo(){
  this.id = null;
  this.playerOne = null;
  this.playerTwo = null;
  this.Game = null;
  this.turn = null;
}
```

- The GameCollection object will, at anytime, have access to information about all active games. From this, the matchmaking is done.

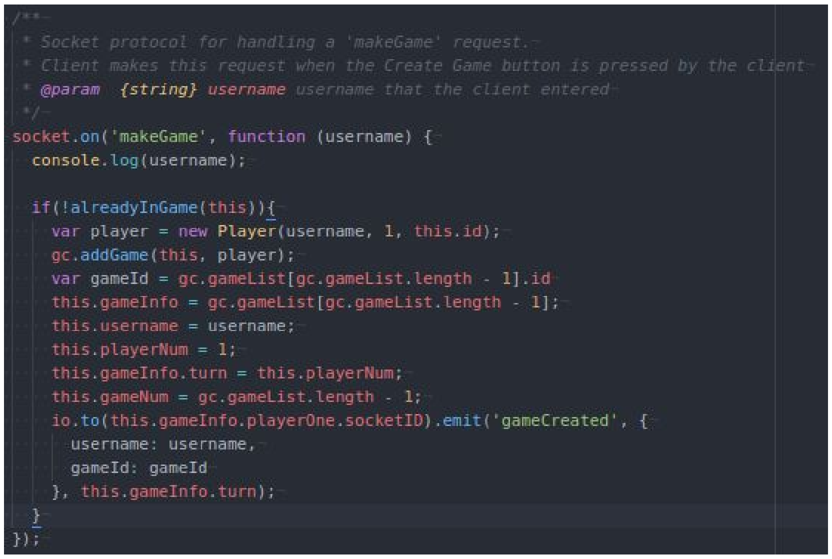


```
/**
 * GameCollection (holds all current game's gameInfo and manages matchmaking)
 * @constructor
 */
function GameCollection(){
  this.totalGameCount = 0;
  this.gameList = [];
}
```

- modules/socketApi.js: This file is where all of the socket protocol is done. List of server side actions, each responds to client accordingly:
  - 'connection' 

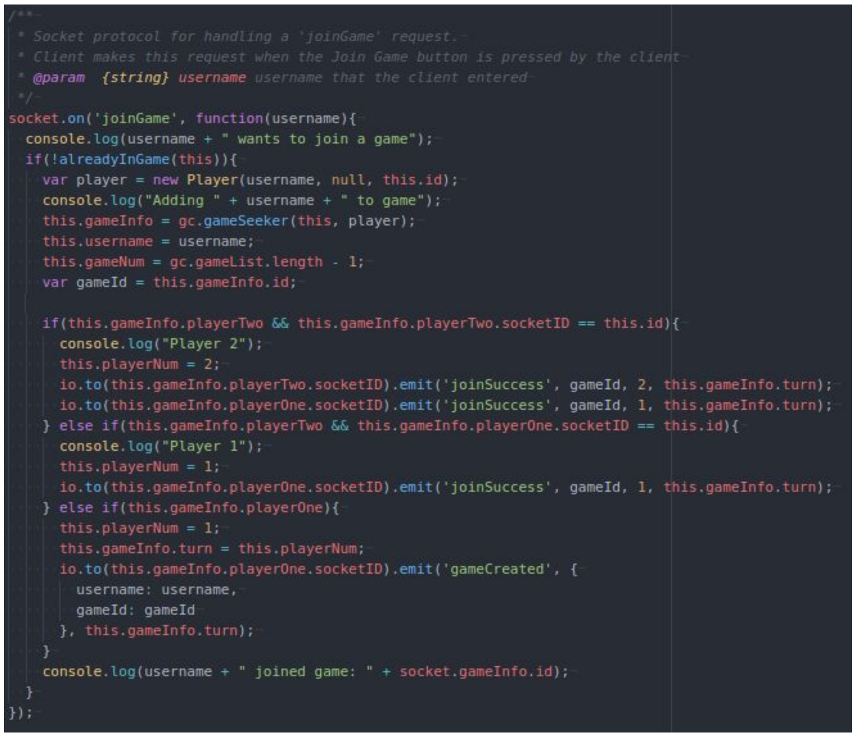
```
io.on('connection', function(socket){
  console.log('A user connected');
```

- 'makeGame'



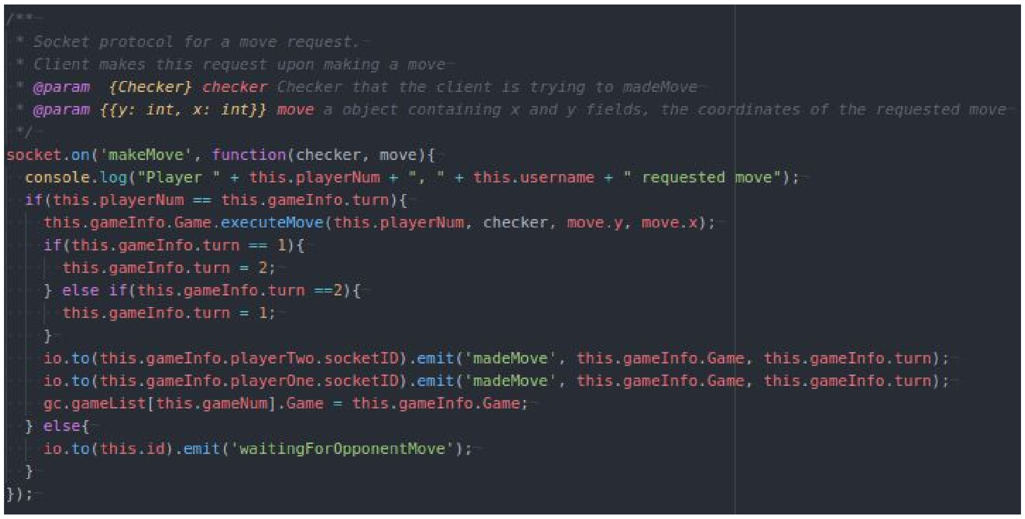
```
/**
 * Socket protocol for handling a 'makeGame' request.
 * Client makes this request when the Create Game button is pressed by the client
 * @param  {string} username username that the client entered
 */
socket.on('makeGame', function (username) {
  console.log(username);

  if(!alreadyInGame(this)){
    var player = new Player(username, 1, this.id);
    gc.addGame(this, player);
    var gameId = gc.gameList[gc.gameList.length - 1].id
    this.gameInfo = gc.gameList[gc.gameList.length - 1];
    this.username = username;
    this.playerNum = 1;
    this.gameInfo.turn = this.playerNum;
    this.gameNum = gc.gameList.length - 1;
    io.to(this.gameInfo.playerOne.socketID).emit('gameCreated', {
      username: username,
      gameId: gameId
    }, this.gameInfo.turn);
  }
});
```

- 'joinGame'



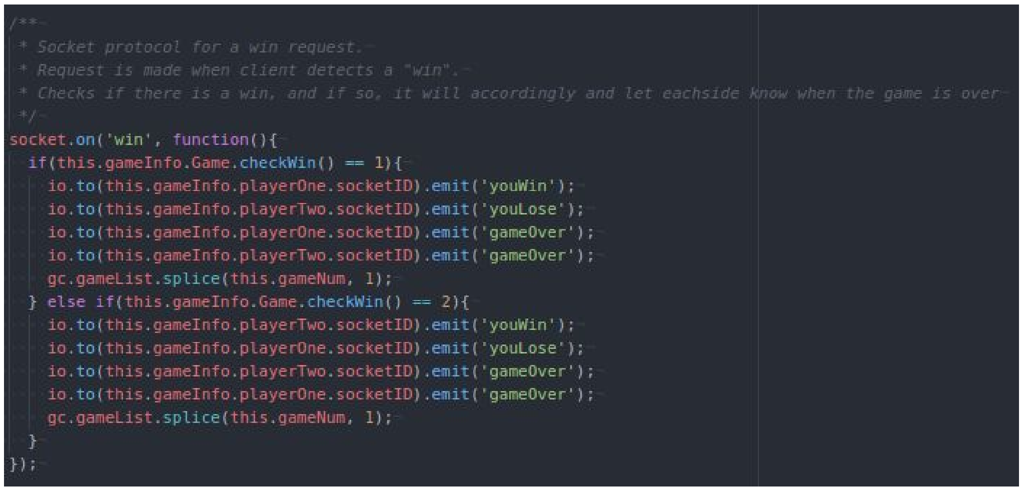
```
/**
 * Socket protocol for handling a 'joinGame' request.
 * Client makes this request when the Join Game button is pressed by the client
 * @param  {string} username username that the client entered
 */
socket.on('joinGame', function(username){
  console.log(username + " wants to join a game");
  if(!alreadyInGame(this)){
    var player = new Player(username, null, this.id);
    console.log("Adding " + username + " to game");
    this.gameInfo = gc.gameSeeker(this, player);
    this.username = username;
    this.gameNum = gc.gameList.length - 1;
    var gameId = this.gameInfo.id;

    if(this.gameInfo.playerTwo && this.gameInfo.playerTwo.socketID == this.id){
      console.log("Player 2");
      this.playerNum = 2;
      io.to(this.gameInfo.playerTwo.socketID).emit('joinSuccess', gameId, 2, this.gameInfo.turn);
      io.to(this.gameInfo.playerOne.socketID).emit('joinSuccess', gameId, 1, this.gameInfo.turn);
    } else if(this.gameInfo.playerTwo && this.gameInfo.playerOne.socketID == this.id){
      console.log("Player 1");
      this.playerNum = 1;
      io.to(this.gameInfo.playerOne.socketID).emit('joinSuccess', gameId, 1, this.gameInfo.turn);
    } else if(this.gameInfo.playerOne){
      this.playerNum = 1;
      this.gameInfo.turn = this.playerNum;
      io.to(this.gameInfo.playerOne.socketID).emit('gameCreated', {
        username: username,
        gameId: gameId
      }, this.gameInfo.turn);
    }
    console.log(username + " joined game: " + socket.gameInfo.id);
  }
});
```

- ‘makeMove’



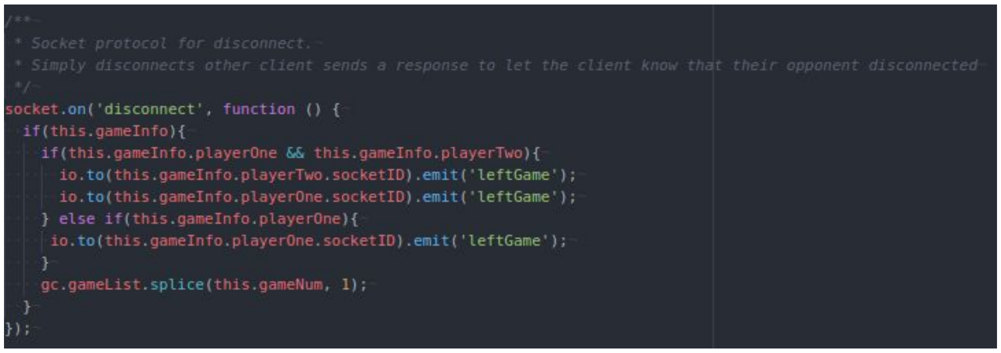
```
/**
 * Socket protocol for a move request.
 * Client makes this request upon making a move
 * @param  {Checker} checker Checker that the client is trying to madeMove
 * @param {{y: int, x: int}} move a object containing x and y fields, the coordinates of the requested move
 */
socket.on('makeMove', function(checker, move){
  console.log("Player " + this.playerNum + ", " + this.username + " requested move");
  if(this.playerNum == this.gameInfo.turn){
    this.gameInfo.Game.executeMove(this.playerNum, checker, move.y, move.x);
    if(this.gameInfo.turn == 1){
      this.gameInfo.turn = 2;
    } else if(this.gameInfo.turn ==2){
      this.gameInfo.turn = 1;
    }
    io.to(this.gameInfo.playerTwo.socketID).emit('madeMove', this.gameInfo.Game, this.gameInfo.turn);
    io.to(this.gameInfo.playerOne.socketID).emit('madeMove', this.gameInfo.Game, this.gameInfo.turn);
    gc.gameList[this.gameNum].Game = this.gameInfo.Game;
  } else{
    io.to(this.id).emit('waitingForOpponentMove');
  }
});
```

- ‘win’



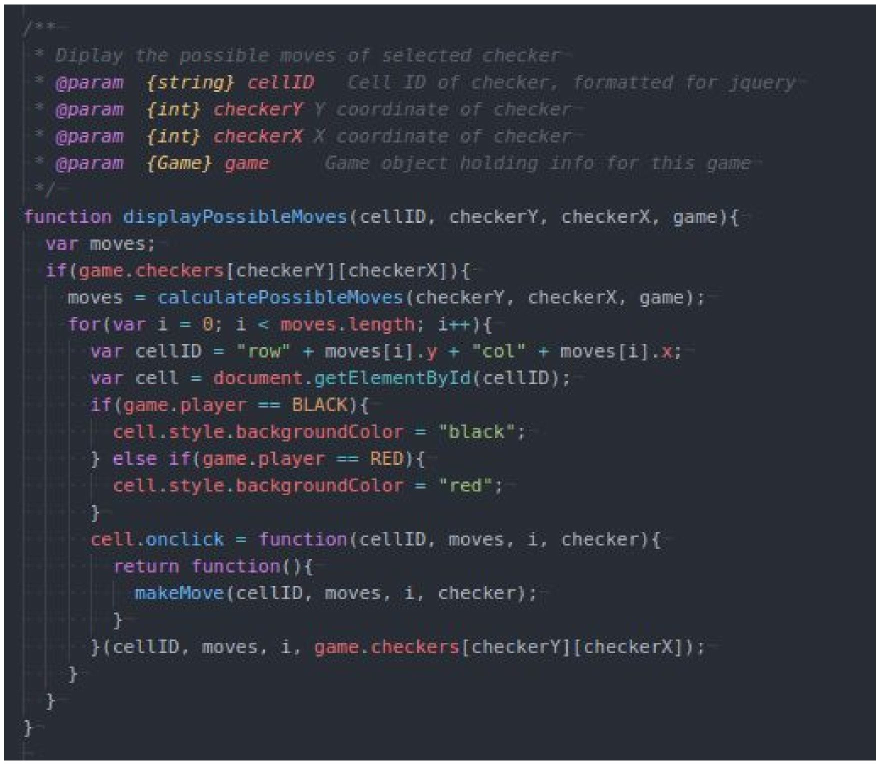
```
/**
 * Socket protocol for a win request.
 * Request is made when client detects a "win".
 * Checks if there is a win, and if so, it will accordingly and let eachside know when the game is over
 */
socket.on('win', function(){
  if(this.gameInfo.Game.checkWin() == 1){
    io.to(this.gameInfo.playerOne.socketID).emit('youWin');
    io.to(this.gameInfo.playerTwo.socketID).emit('youLose');
    io.to(this.gameInfo.playerOne.socketID).emit('gameOver');
    io.to(this.gameInfo.playerTwo.socketID).emit('gameOver');
    gc.gameList.splice(this.gameNum, 1);
  } else if(this.gameInfo.Game.checkWin() == 2){
    io.to(this.gameInfo.playerTwo.socketID).emit('youWin');
    io.to(this.gameInfo.playerOne.socketID).emit('youLose');
    io.to(this.gameInfo.playerTwo.socketID).emit('gameOver');
    io.to(this.gameInfo.playerOne.socketID).emit('gameOver');
    gc.gameList.splice(this.gameNum, 1);
  }
});
```

- ‘disconnect’



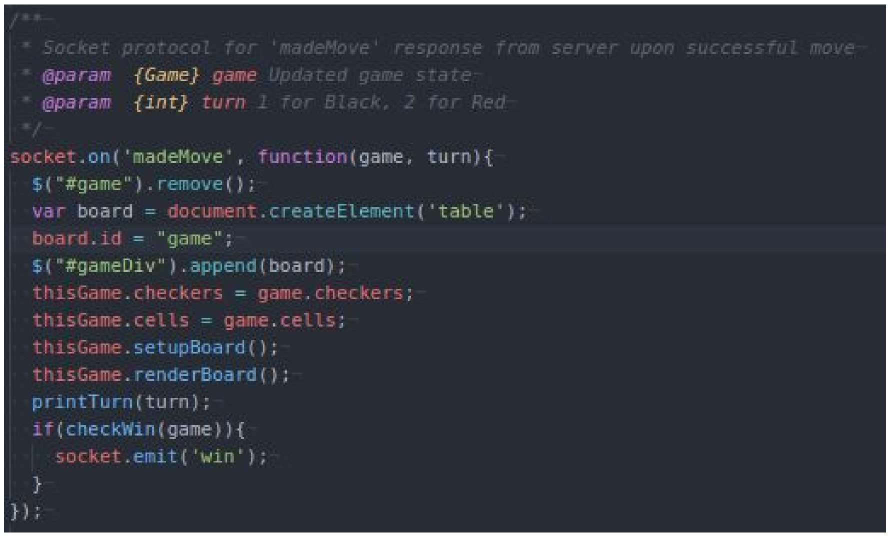
```
/**
 * Socket protocol for disconnect.
 * Simply disconnects other client sends a response to let the client know that their opponent disconnected
 */
socket.on('disconnect', function () {
  if(this.gameInfo){
    if(this.gameInfo.playerOne && this.gameInfo.playerTwo){
      io.to(this.gameInfo.playerTwo.socketID).emit('leftGame');
      io.to(this.gameInfo.playerOne.socketID).emit('leftGame');
    } else if(this.gameInfo.playerOne){
     io.to(this.gameInfo.playerOne.socketID).emit('leftGame');
    }
    gc.gameList.splice(this.gameNum, 1);
  }
});
```

- public/javascripts/checkers.js: This file contains the client side logic for the checkers game. When a client clicks one of their checkers, the displayPossibleMoves method highlights possible moves to the user.



```
/**
 * Diplay the possible moves of selected checker
 * @param  {string} cellID   Cell ID of checker, formatted for jquery
 * @param  {int} checkerY Y coordinate of checker
 * @param  {int} checkerX X coordinate of checker
 * @param  {Game} game     Game object holding info for this game
 */
function displayPossibleMoves(cellID, checkerY, checkerX, game){
  var moves;
  if(game.checkers[checkerY][checkerX]){
    moves = calculatePossibleMoves(checkerY, checkerX, game);
    for(var i = 0; i < moves.length; i++){
      var cellID = "row" + moves[i].y + "col" + moves[i].x;
      var cell = document.getElementById(cellID);
      if(game.player == BLACK){
        cell.style.backgroundColor = "black";
      } else if(game.player == RED){
        cell.style.backgroundColor = "red";
      }
      cell.onclick = function(cellID, moves, i, checker){
        return function(){
          makeMove(cellID, moves, i, checker);
        }
      }(cellID, moves, i, game.checkers[checkerY][checkerX]);
    }
  }
}
```

Whichever move is clicked is sent to the server, where it verifies that the move that was made was valid, if so, the server responds with an updated Game object, which the client side code uses to display the board through setupBoard() and renderBoard()   The rest of the client side protocol has to do with the matchmaking requests.



```
/**
 * Socket protocol for 'madeMove' response from server upon successful move
 * @param  {Game} game Updated game state
 * @param  {int} turn 1 for Black, 2 for Red
 */
socket.on('madeMove', function(game, turn){
  $("#game").remove();
  var board = document.createElement('table');
  board.id = "game";
  $("#gameDiv").append(board);
  thisGame.checkers = game.checkers;
  thisGame.cells = game.cells;
  thisGame.setupBoard();
  thisGame.renderBoard();
  printTurn(turn);
  if(checkWin(game)){
    socket.emit('win');
  }
});
```

DESCRIPTION:

Checkers--A website with matchmaking that allows users to play Checkers against other users. Upon arrival to the site the user is prompted to create or join a game. If there are others who have created a match but no one has joined yet and the user clicks ‘Join Game’ they will be thrown in the first available open game, if they click ‘Create Game’ they will be prompted to wait until another user joins, then they will be thrown into a game with the joined user.

TABLE OF CONTENTS:

OVERVIEW:

In making this website, there was certainly a good amount of hurdles to jump, as I had little to no experience using Socket.io. It helped that I had used Node before, but the use of Socket.io with Express and Node was unknown to me. With this, just gaining the ability to push information through an instance of a socket, to the client, took me a good amount of time to achieve. But, of course once it was done, the power of Socket.io was fully realized.
Surely, this project was by far the most involved programming task that I have embarked on throughout the course of the semester, and likely the most impressive of my career, in my opinion.

At first, just trying to think about how to pair to clients together was a bit daunting. The only real experience that I had had prior to this project with socket protocol, was during assignment 1, in the making of the chat room. Certainly, Java sockets are far different from Socket.io's web sockets, but I did not really understand that coming in, so I expected the matchmaking portion of the project to be very difficult. Luckily, with the use of new library Socket.io, the matchmaking ended up being a doable aspect of the application, taking what is really quite complex under the covers to being quite understandable and usable, at a high level.

In all honesty, all Api's and libraries that I used in making this site were very user friendly, and as a side note, I would like to say that using Node as a beginner is something that I would highly recommend. I would even go so far as to suggest that Node be taught right away in this course alongside the learning of client-side Javascript.

W**HAT IS NEW AND COMPLEX?:**

At the forefront of what is new and complex is, as previously mentioned, is my use of the Socket.io Library in order to create a website that matches users together, in a game of checkers. The most complex of my uses for the library, was making the application put users who were looking to play, into a game together. With that, I'll explain a little bit about how I achieved this aspect of the site.

It all starts with the fact that Socket.io allows the programmer to create custom protocol for each socket and use them as instances. Socket.io gives you an io object that controls all of what happens with the sockets. This means, that you can add custom routines to the io object, in turn giving you a custom socket, which is, of course, extremely useful. Since each socket is created with a unique ID, it is easy enough to use this to keep track of said socket, as you can emit to a specific socket based on ID. From this, I was able to create an object, GameInfo, which stores the information for a single game (between two users). Within this object there are two Player objects, and within each of these player objects is the information about that player, including their socket ID. Because of this, I was able to keep a list of GameInfo instances inside of my GameCollection object, allowing me to track which games were open along with the Game state for each game. This is where Socket.io's power truly shined, giving me the ability to keep track of each user, allowing me to develop methods to search for open games and place new users into them.

Along with the matchmaking aspect, the Socket.io library allowed me to push information to client's instantly, without the client having to request the information. This aspect was essential for overall server load, as the server doesn't have to deal with constant, unnecessary requests from clients, and allows the users to actually play in real time. In regard to the specifics of this particular site, Socket.io granted me the ability to push the new game state to each client in the game after a move was made, making for very little wait time to render the new version of the board.

Beyond Socket.io's application in this site, the use of NodeJS and Express was not new for me, but that does not mean it still wasn't a bit complex the second time around. Granted, I did use Node and Express on my last portfolio, but it really was a bit different this time. For the Checkers game I barely even had to deal with intense routing code in the same sense as last time, again due to the use of Socket.io, but instead I had to do more exporting and importing of classes and the navigating of other aspects of Express and Node. As a result of making a module file for the custom socket protocol I am allowed to simply export that into the ./bin/www file, which is the default file that runs the instance of the server, here, I simply attached the instance of the server to my custom socketApi and there, my custom sockets are up and running. This was certainly a skill that I learned from working on this project that will extremely useful in future Node development.

So, was every part of this project new for me? No. Was it challenging nonetheless? Most certainly. In the making of this project, I really got to put all the skills that I learned this semester to the test, pushing me to complete something that I am undoubtedly proud of. With the experience of the creation of the site, there has certainly come more confidence for future web development, and my efficiency has improved , as I have pushed through yet another barrier as far as development goes.

MY LEARNING JOURNEY, BLOOM'S TAXONOMY:

The six stages of Bloom's Taxonomy are as follows: remember, understand, apply, analyze, evaluate, and create.

Remember. Something that I like to think that I'm getting better at. As far as this project goes, this, the first stage of Bloom's Taxonomy, was certainly related to my novice knowledge of the libraries that I was using. So, at first, when I was beginning the Checkers site, I tried out a few different things with Socket.io, just to explore, and see what I could do. The first thing I did was go through Socket.io's very convenient tutorial, which shows you how to make a chatroom with about 20 lines of code. From that initial experience with Socket.io, I began to realize how powerful it really is. The simplicity of this 'Hello World' style program astonished me, as the only experience I had had building a chatting application was during homework 1, where the application was developed in Java. Comparing chatrooms, the Java one was significantly more difficult to make than the one that I made using Socket.io and Javascript. With this little experience with Socket.io under my belt, I took what I remembered and pushed on to apply it to my idea of a project, a online, multiplayer, Checkers site.

With the little understanding that I had about how I would go about making what I wanted to make, I kept on working, and step-by-step began to understand more about what I was doing. Here, the more time I spent developing, the more efficient I got at interacting with the server-client relationship. Once I had really got into the deep end of the matchmaking development, I finally understood whatI had to do with these sockets to produce the application that I wanted. One key aspect of this, was my ability to store the ID of each socket with it's client, within the gameInfo instance, allowing me to push certain things to player one in the game and other things to player two. Also, all of this was able to be done with constant time access, no stepping through all the games in order to send messages to specific clients because I could store an instance of the game within each socket.

In my application of my newfound understanding of sockets I went forth and began to churn out socket protocol almost effortlessly, all thanks to the learning that I had just done on the topic. Because of my understanding of how to interact with the sockets, the speed at which I was building the application definitely went up. This was undoubtedly due to the fact that I didn't have to second guess and look up how to do what I was trying to do. At this point, I knew how to send and receive data on both the client and server side, allowing me to focus my time on how the game logic was going to work. Going to prove that, in the end, starting out slow with something new will help you. In other words, if you want to learn how to do something, don't skip any steps along the way, as those skipped steps will only leave you less informed than those who took all of the necessary steps. At this point in the project, there was certainly lots of analysis that had to be done in regard to how I wanted to continue.

Just about the time when the matchmaking was getting finished up, I took a bit of time away from the computer to just think, to think about how I wanted to proceed. We'll call this little time out, a period of reflection and analysis. During this period, since I had a good amount of experience with the technologies that I was using, I made decisions about how I would plug in this Checkers front end that I had made, days earlier, with the matchmaking backend that I had just completed using Sockets. This deciding point, was crucial to the overall success of the application, as here, is where, I decided to plug in an instance of the client's game with their socket. In doing that, it allowed me constant time access to every game through it's socket, meaning there was no need to step through the gameList to find which players are in which game. With these decisions made on how I wanted to proceed, came a direction in which to work, something that is very important, as everyone needs a goal. From this point on, I worked on realizing my game, and the usual things that go along with that. First, I wanted to create a GUI that resembled a checkerboard, easy enough, I simply used HTML elements within the client-side code to do this. Second, I needed the two players that were in the same game to be able to see the updated game state in real time, so that when one makes a move it updates both boards, also fairly easy as I was using sockets, I just pushed a new game object to each user that was in the game, and from this object the client side code rendered the DOM.

Once I could update game states on the player's screens the application was reaching a finishing point. Meaning, it was time for another assessment, this time an evaluation. Here, I looked at how I had gotten to this point, with a nearly completed Checkers game, and how else I could've gotten to this point, as well as things for me to improve on next time. I believe this stage of the development is a crucial thing to do, because it allows one to really think about the project that was just completed with the perspective of an individual who has experience in the exact thing that was accomplished. So, if I ever have to create something similar to this again (I will), I will be able to remember everything that I learned and hopefully apply it, in order to be significantly more efficient than those without this experience.

So, for the Create stage of my journey of learning sockets, I can safely say that I now have the ability to use the sockets library to the extent where I can make multiplayer games, and other real time applications where clients interact with one another. From this, there is no doubt that I will now have the ability to create other similar applications utilizing the Socket.io library. In fact, I plan to do just that in the near future, with plans to make additions to the Checkers site. In these additions I am going to add a chat feature that allows users to chat with their opponents, as well as a login system that keeps user’s information on file to produce a matchmaking environment that will match players based on their skill. All of this is of course to be done using sockets.

Overall, in the making of this project, the Checkers website, I learned much about client-server relations and about matchmaking, along with general development using the Socket.io library. I believe, the experience that I had in creating this application, has opened doors for me as far as programming goes. What I mean by that, is that when I first thought of the idea for this project, I thought it was a little much to be done in a week, but I did it, and there is no doubt that my confidence and knowledge in programming has increased a good amount from the creation of this site alone. So, with that, I thank you Professor Mitra, for pushing us all as students and for always making us think assignments are due sooner than they really are.