Sprint 1 Rollerball

Blueberries!

COLORADO STATE UNIVERSITY

Department of Computer Science

Jared Ham, Alex Enthoven, Daniel Chavez,
Sharon Zu, Robbie Weinel

Agenda

- Rollerball Overview
- Process/Product Decisions made during this sprint
- SCRUM ceremony results
- Project Progress

Rollerball Overview

COLORADO STATE UNIVERSITY

Department of Computer Science

Game Setup

- 7x7 square grid with middle 3x3 grid missing
- 4 Unique Pieces
  - 2x Rook
  - 2x Pawn
  - 1x Bishop
  - 1x King
- White always moves first
- Two Ways to Win
  - Checkmate Enemy King
  - Move king to enemy king starting location through clockwise movement of king

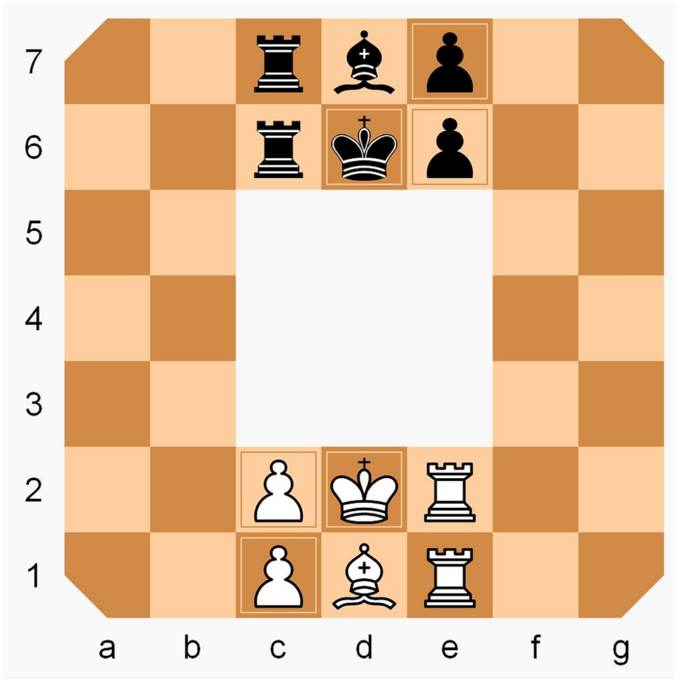


Figure 1: Rollerball Board
https://en.wikipedia.org/wiki/Rollerball_(chess_variant)

Pawn and King Valid Moves

- Pawn can move orthogonally or diagonally forward 1 space. Promoted to Rook or bishop by reaching starting square of opponent pawn
- King can move 1 space in any direction, so long as the move does not result in check

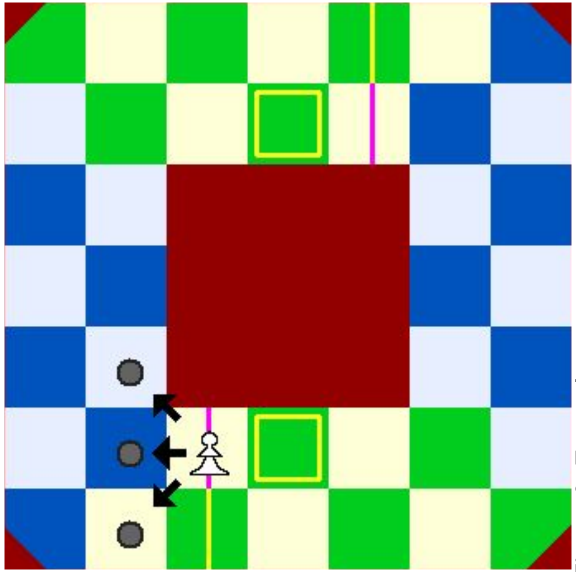


Figure 2: Example pawn move http://history.chess.free.fr/rollerball.htm

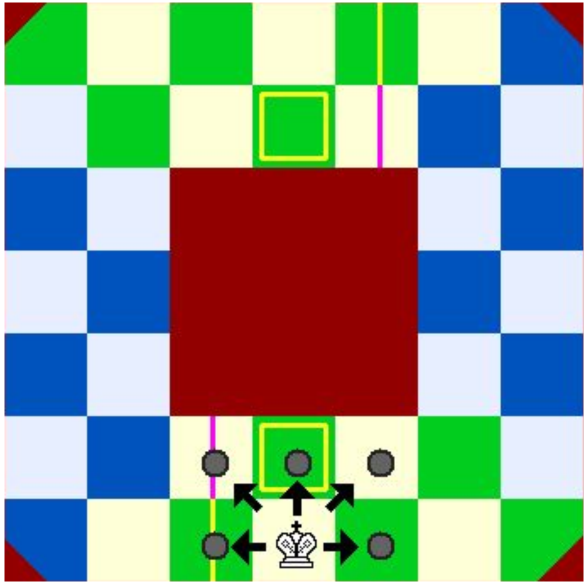


Figure 3: Example king move http://history.chess.free.fr/rollerball.htm

Bishop Valid Moves

- Slides Diagonally forward any number of squares, with one rebound off an internal or external wall allowed. Can also move diagonally backwards a single square

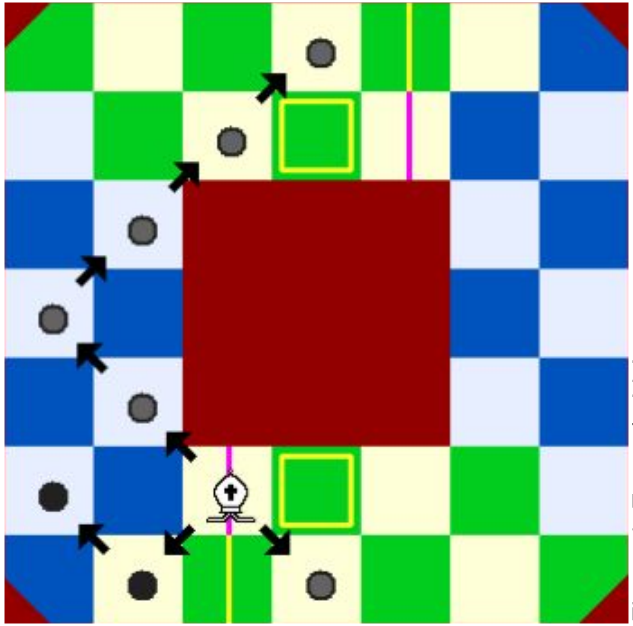


Figure 4: Example bishop move
http://history.chess.free.fr/rollerball.htm

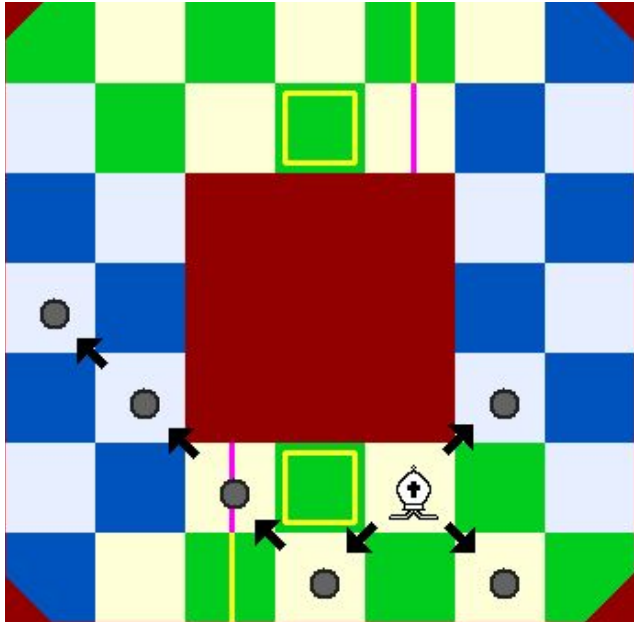


Figure 5: Example bishop move
http://history.chess.free.fr/rollerball.htm

Rook Valid Moves

- Rook can move orthogonally forward any number of squares, backward a single square. It is also allowed a single rebound off of a corner square

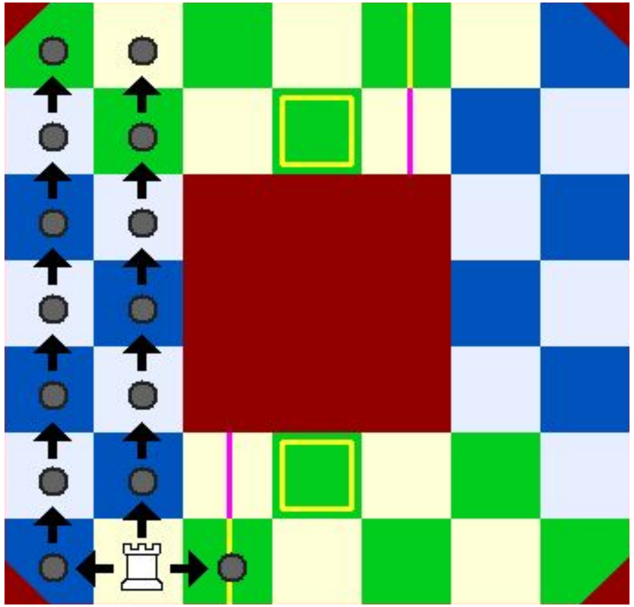


Figure 6: Example rook move
http://history.chess.free.fr/rollerball.htm

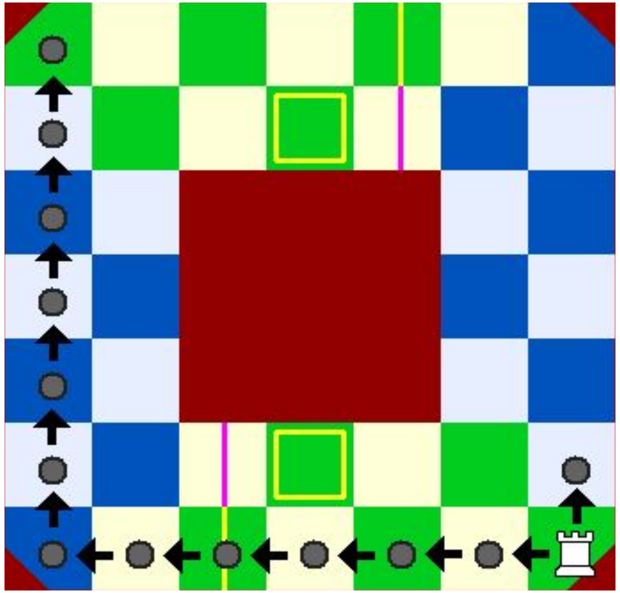


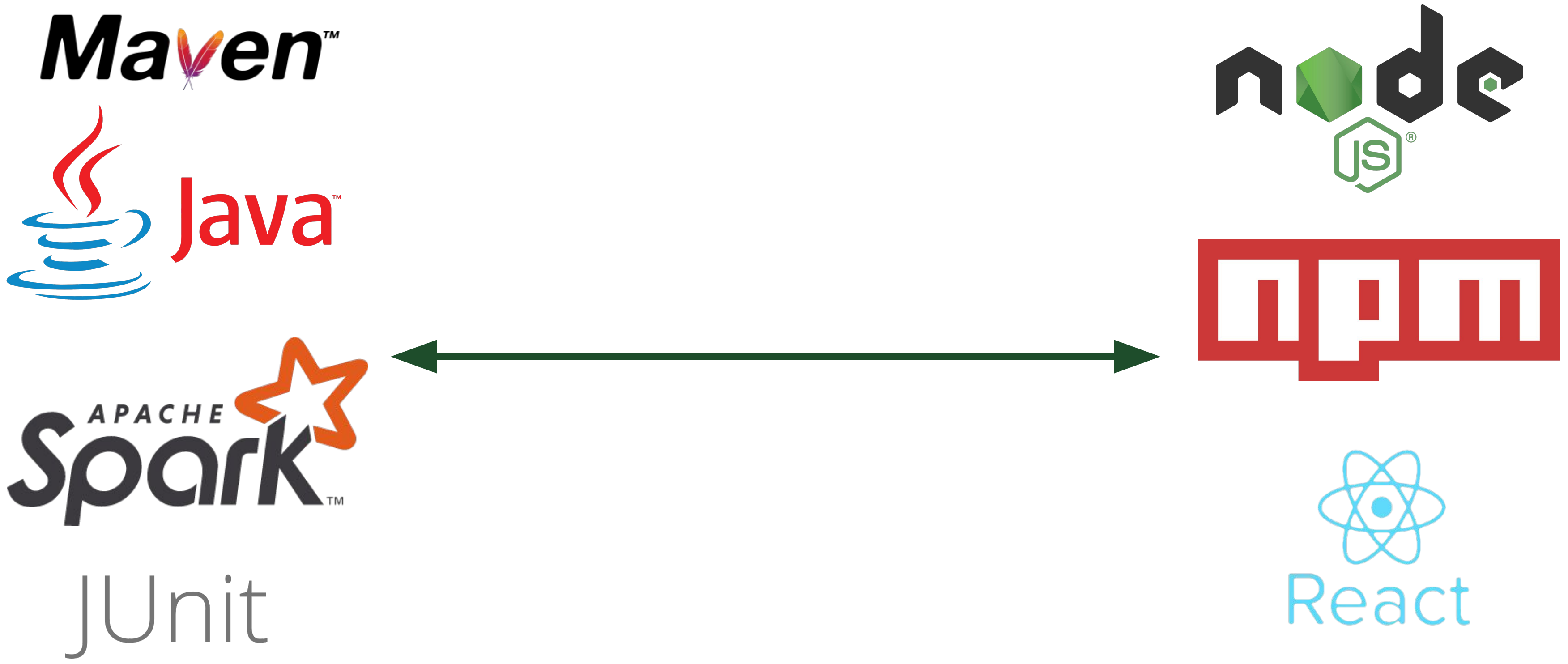
Figure 7: Example rook move
http://history.chess.free.fr/rollerball.htm

Process/Product Decisions

COLORADO STATE UNIVERSITY

Department of Computer Science

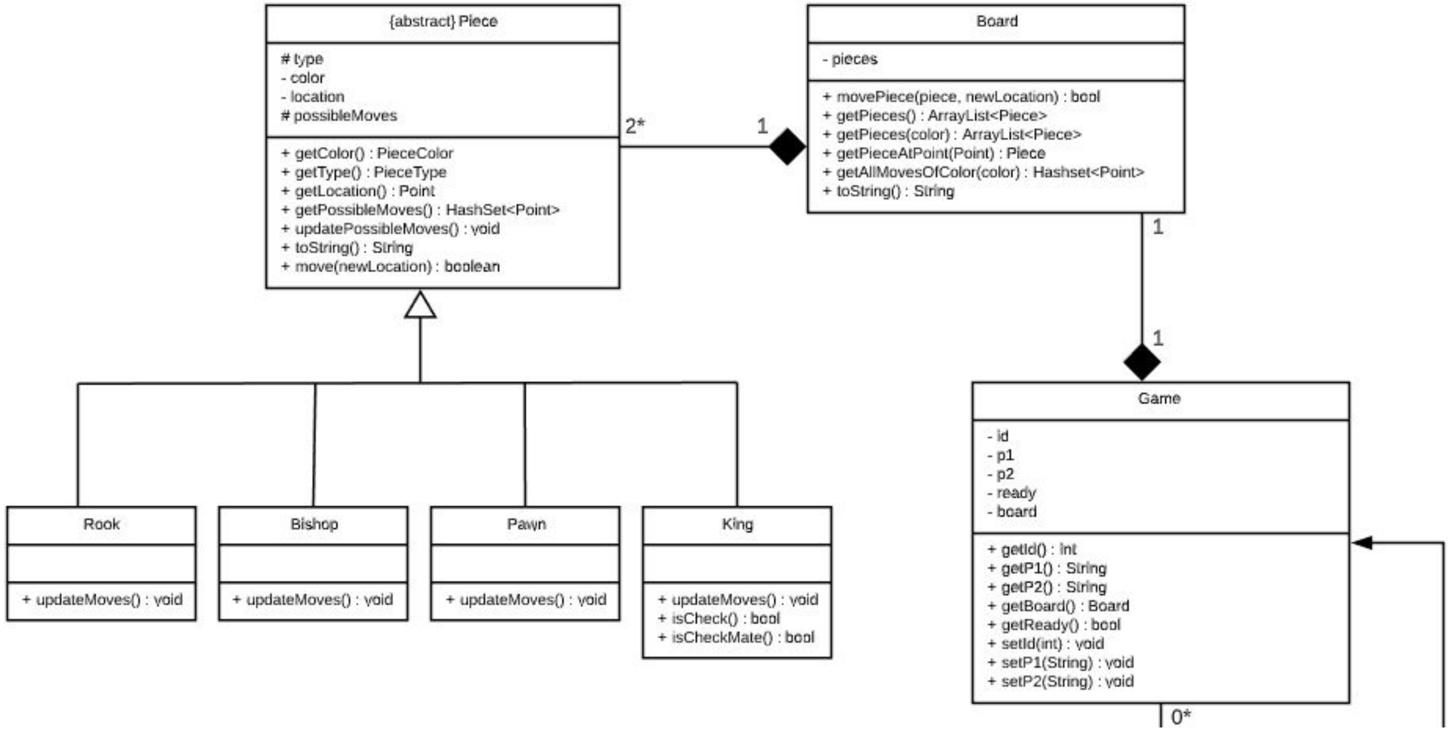
Project Structure



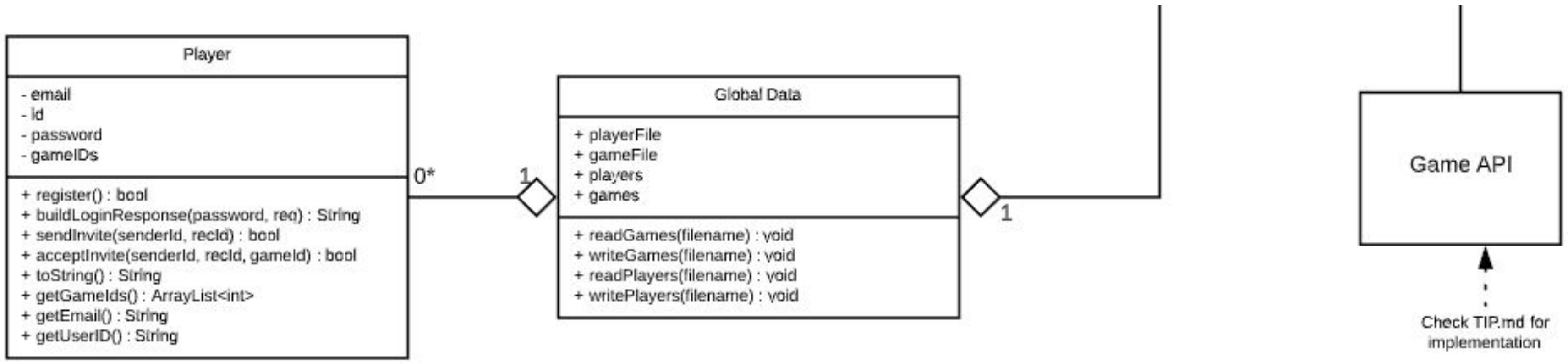
Server Responsibilities

- Loads all players and games from their respective files on startup
    - Players.json serves as a database of all registered players
    - Games.json serves as a database of all past and active games
- Accepts and responds to the following API requests from the client
    - Registration and Login
    - Send and accept game invite
    - Get and Update Game
- In the future it will also validate piece moves

Class Diagram



Class Diagram (continued)

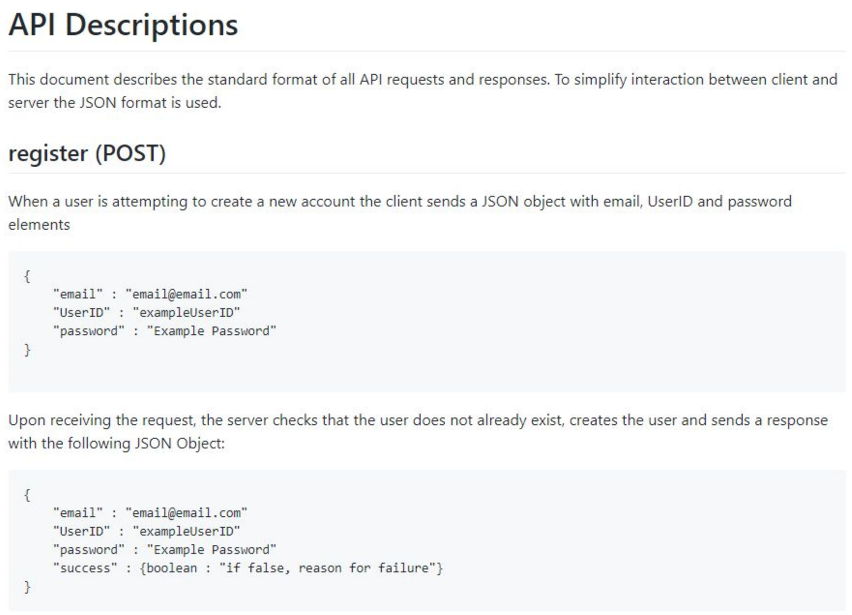

Player
- email
- id
- password
- gameIDs
+ register() : bool
+ buildLoginResponse(password, req) : String
+ sendInvite(senderId, recId) : bool
+ acceptInvite(senderId, recId, gameId) : bool
+ toString() : String
+ getGameIds() : ArrayList<int>
+ getEmail() : String
+ getUserID() : String
0*
1
Global Data
+ playerFile
+ gameFile
+ players
+ games
+ readGames(filename) : void
+ writeGames(filename) : void
+ readPlayers(filename) : void
+ writePlayers(filename) : void
1
Game API
Check TIP.md for implementation

Client Responsibilities

- Have a UI/Frontend
  - Login Page, Register Page, Home page, Profile page, and Game page
- Update pages
  - Update Home for inviting and pending game invites
  - Update Game/board if game has proceeded
- Display information
  - Display information about match history
  - Display information about current games
- Create interactive board
  - Have a interactive board for moving pieces

Interoperability Document

- Each API endpoint that is created is added to this document
- Allows easy understanding of what is going on in API requests
- By documenting the behavior of each API there is one single source of truth



API Descriptions

This document describes the standard format of all API requests and responses. To simplify interaction between client and server the JSON format is used.

register (POST)

When a user is attempting to create a new account the client sends a JSON object with email, UserID and password elements

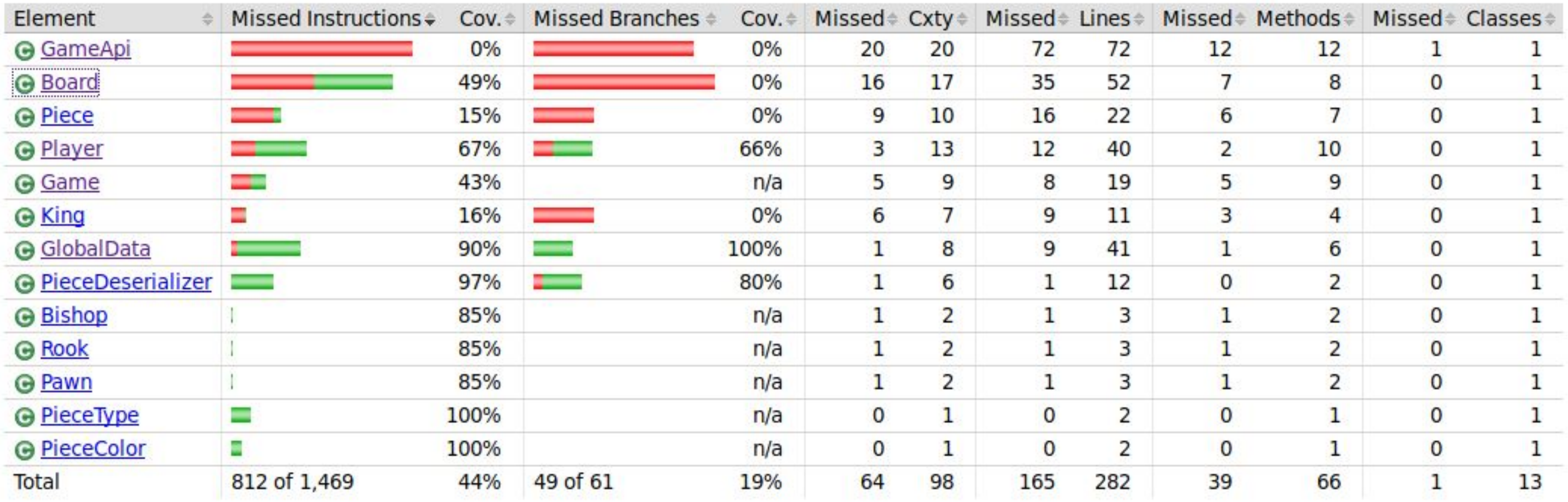
```
{
    "email" : "email@email.com"
    "UserID" : "exampleUserID"
    "password" : "Example Password"
}
```

Upon receiving the request, the server checks that the user does not already exist, creates the user and sends a response with the following JSON Object:

```
{
    "email" : "email@email.com"
    "UserID" : "exampleUserID"
    "password" : "Example Password"
    "success" : {boolean : "if false, reason for failure"}
}
```

Tracking Test Coverage with Jacoco

- Server side unit test code coverage is tracked with Jacoco



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
GameApi		0%		0%	20	20	72	72	12	12	1	1
Board		49%		0%	16	17	35	52	7	8	0	1
Piece		15%		0%	9	10	16	22	6	7	0	1
Player		67%		66%	3	13	12	40	2	10	0	1
Game		43%		n/a	5	9	8	19	5	9	0	1
King		16%		0%	6	7	9	11	3	4	0	1
GlobalData		90%		100%	1	8	9	41	1	6	0	1
PieceDeserializer		97%		80%	1	6	1	12	0	2	0	1
Bishop		85%		n/a	1	2	1	3	1	2	0	1
Rook		85%		n/a	1	2	1	3	1	2	0	1
Pawn		85%		n/a	1	2	1	3	1	2	0	1
PieceType		100%		n/a	0	1	0	2	0	1	0	1
PieceColor		100%		n/a	0	1	0	2	0	1	0	1
Total	812 of 1,469	44%	49 of 61	19%	64	98	165	282	39	66	1	13

SCRUM Ceremonies

September 9 Team SCRUM

- Implementation Decisions
  - The application will be split into a client and a server
  - The server will keep track of games and players and receive API calls from client
  - React will be used for the client
  - Apache spark will be used to handle API calls on the server
- Major Action Items
  - Use maven and npm to create project skeleton
  - Set up server to perform automated testing using JUnit
  - Create user stories
- Schedule Decisions
  - None

September 16 Team SCRUM

- Implementation Decisions
  - Players and Games will be stored on the server in two JSON formatted files
- Major Action Items
  - Complete the list of tasks in ZenHub
  - Add specifications for Players and Games files to TIP.md
  - Add login and registration API calls to TIP.md
  - Make sure unit tests are added for any code written on the server side
- Schedule Decisions
  - Focus on game logic for remainder of sprint
  - UI will remain as simple as possible for this sprint

September 23 Team SCRUM

- Implementation Decisions
  - None
- Major Action Items
  - Robbie and Dan will continue to implement game logic for the pieces
  - Finish up the login and registration on client side. Make sure that both are functioning properly.
  - Start making slides for the in class presentation
  - Begin gathering deliverables for sprint 1
- Schedule Decisions
  - Simple game logic will be the last added functionality to the application for this sprint

Project Progress

Client Side

- Login page UI is complete, along with functioning login request
- Register page UI is complete, along with functioning register request
- Basic home page is up, along with logout link, displaying user ID, and user’s associated games
- Login and register page gives alerts for invalid requests
- Login, register, and home pages are linked together to redirect based on valid requests

Server Side

- Added API class for supporting login, register, game, game invites, and moves requests
- Added Player class for supporting register, login, and invites
- Added Pieces class for supporting king, rook, bishop, and pawn
- Added Board class for supporting a board and moves
- Added GlobalData class to handle available player and game storage
- Added King, Rook, Bishop, and Pawn classes for support what moves each piece can do
- Added Game class for supporting games