GAME450 Project Proposal

By: Jeff Rose and James Zinger

# Introduction

For our project, we propose to create a checkers game website complete with a lobby and 3D-rendered checkers gameplay. Players will visit the website and enter their name, at which point they will enter the lobby and create a game or join an existing game. When play begins, both players will enter into a 3D gameplay screen where their game will play out, mediated by the server (which contains the master copy of the game state).

# Technology

We intend to build this game using modern web-technologies, which demands that we use JavaScript for the client-side code (since it runs in the browser). Since James has some prior experience working with node.js for server-side website backends, we have opted to code the server-side in JavaScript with the node.js application framework. This backend solution offers our webapp a significant scalability advantage over native C/C++ code, and presents us with the opportunity to deploy the service to cloud application stores that can serve the game to a large number of clients simultaneously if necessary.

**The client-side browser application is founded upon the following dependencies:**

|  |  |
| --- | --- |
| Software Library | Description |
| *AngularJS* | An MVC single-page-application framework for websites by Google. This allows the frontend to easily process DOM events and separate the DOM (view) from the data model that is being rendered by communication with the server. |
| *jQuery* | A DOM manipulation and parsing library that provides helpful functionality for searching the DOM and modifying document contents. |
| *Socket.IO* | A WebSocket communication library that provides event-based socket communication between a server and client. |
| *three.js* | A 3D rendering engine for the browser that provides a subset of OpenGL 4.x rendering functionality according to the WebGL specification. |

**The server-side application requires the following dependencies:**

|  |  |
| --- | --- |
| Software Library | Description |
| *node.js* | A JavaScript application framework written in C and built on top of Google Chrome’s V8 JavaScript interpreter. Node.js applications are inherently asynchronous and extremely scalable by nature, making them ideal for the web. |
| *Socket.IO* | A WebSocket communication library that provides event-based socket communication between a server and client. |

# Protocol

In order to achieve the goals we have in mind for this game, we have developed a protocol to guide communication between the server and its clients while they are in the lobby or in-game.

## JSON Format

**// Request is an RPC-like call**

**var request = {**

**cmd: ‘C’,**

**data: null, // Any data**

**name: ‘Tabby’ // Username**

**};**

**// Response approves request**

**var response = {**

**approved: true,**

**data: null // Any data**

**};**

**// Push is an RPC-like call**

**var push = {**

**cmd: ‘GO’,**

**data: ‘Tabby’ // Any data**

**};**

## Lobby

|  |  |  |
| --- | --- | --- |
| Client Actions | Command | Additional data |
| *Create Game Request* | ‘C’ | -- |
| *Leave Game Request* | ‘L’ | -- |
| *Join Game Request* | ‘J’ | Player name of the game host (used as a unique key) |
| *Set Ready Request* | ‘R’ | -- |
| *Set Wait Request* | ‘W’ | -- |
| *Set Name Request* | ‘N’ | Requested player name |
| *Connect Init Request* | ‘I’ | -- |

|  |  |  |
| --- | --- | --- |
| Server Actions | Command | Additional data |
| *Create Game Response* | -- | (TRUE + created game) or (FALSE + error message) |
| *Leave Game Response* | -- | TRUE or (FALSE + error message) |
| *Join Game Response* | -- | (TRUE + joined game) or (FALSE + error message) |
| *Set Ready Response* | -- | TRUE or (FALSE + error message) |
| *Set Wait Response* | -- | TRUE or (FALSE + error message) |
| *Set Name Response* | -- | TRUE or (FALSE + error message) |
| *Lobby State Response* | -- | Lobby (players list and games list) |
| *Game Create Push* | ‘GC’ | Created game |
| *Game Remove Push* | ‘GR’ | Player name of the game host (used as a unique key) |
| *Game Update Push* | ‘GU’ | Updated game |
| *Player Create Push* | ‘PC’ | Created player |
| *Player Remove Push* | ‘PR’ | Player name (used as a unique key) |
| *Player Update Push* | ‘PU’ | Updated player |
| *Gameplay Start Push* | ‘SP’ | Initial gameplay state |

## Gameplay

|  |  |  |
| --- | --- | --- |
| Client Actions | Command | Additional data |
| *Move Piece Request* | ‘M’ | Piece move data |

|  |  |  |
| --- | --- | --- |
| Server Actions | Command | Additional data |
| *Move Piece Response* | -- | (TRUE + turn end data) or (FALSE + error message) |
| *Piece Position Push* | ‘P’ | Piece move data |
| *Piece Dead Push* | ‘D’ | Piece identifier (‘B2’) |
| *Piece Kinged Push* | ‘K’ | Piece identifier (‘B2’) |
| *Turn Begin Push* | ‘B’ | Player name (player to begin turn) |
| *Game Over Push* | ‘GO’ | Player name (winner) |