Smashville

Joshua Petherick & Kyle Tuckey

University of Brighton

Contents

[Game Summary 3](#_Toc481161369)

[Objectives 3](#_Toc481161370)

[Rules 3](#_Toc481161371)

[Game Play 3](#_Toc481161372)

[Background Research 4](#_Toc481161373)

[Screen/Level Map 5](#_Toc481161374)

[Screen Map 5](#_Toc481161375)

[User.js 5](#_Toc481161376)

[Menu.js – Local Mode 6](#_Toc481161377)

[Chat.js 6](#_Toc481161378)

[Menu.js – Net Mode 7](#_Toc481161379)

[Play.js 7](#_Toc481161380)

[Network Utilisation 8](#_Toc481161381)

[Network Architecture: 8](#_Toc481161382)

[Component Interactions: 8](#_Toc481161383)

[Client 8](#_Toc481161384)

[Objects 8](#_Toc481161385)

[Player.js 9](#_Toc481161386)

[Enemy.js 9](#_Toc481161387)

[Session.js 9](#_Toc481161388)

[Sound.js 9](#_Toc481161389)

[Code Structure 9](#_Toc481161390)

[Assets 9](#_Toc481161391)

[Connect\_button.png 9](#_Toc481161392)

[Level1\_button.png 9](#_Toc481161393)

[Level2\_button.png 10](#_Toc481161394)

[Character1.png 10](#_Toc481161395)

[Character2.png 10](#_Toc481161396)

[Character3.png 11](#_Toc481161397)

[Start\_button.png 11](#_Toc481161398)

[Back\_button.png 11](#_Toc481161399)

[Tiles1.png 11](#_Toc481161400)

[Tiles2.png 11](#_Toc481161401)

[Server 11](#_Toc481161402)

[Data Structures 11](#_Toc481161403)

[Code Structure 12](#_Toc481161404)

[Implementation Evaluation 12](#_Toc481161405)

[Strengths 12](#_Toc481161406)

[Easy communication between Phaser and Socket.IO 12](#_Toc481161407)

[Tiled JSON 12](#_Toc481161408)

[Weaknesses 12](#_Toc481161409)

[Partially Peer-to-Peer 12](#_Toc481161410)

[Lack of built in textbox support in Phaser 13](#_Toc481161411)

[References 13](#_Toc481161412)

[Appendix 1: Presentation 13](#_Toc481161413)

[Appendix 2: GitHub Repository 13](#_Toc481161414)

# Game Summary

## Objectives

Smashville is a 1 to 4 player, 2D physics based fighting game where the objective is to try and knock players off the stage and out of the arena.

## Rules

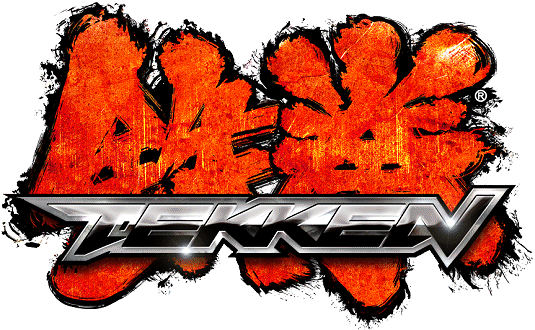
There are a small set of rules in Smashville:

* Players can move left, right and double jump
* Players can be knocked left, right, up or down by other players
* Getting hit will increase the knockback effect
* Getting knocked off the screen will result in the player losing a life, up to 3 times
* The last player remaining will be the victor

## Game Play

Smashville is a medium paced fighting game that makes use of Phasers Arcade physics engine to provide an unusual spin on the traditional fighting games. None of the players have health bars, instead each player has a base percentage that gradually builds up the more a player is hit. The percentage influences the velocity of the knockback caused by other player’s attacks. This gameplay provides a strategic element to the fighting game, as players plan their attack against the other player(s) whilst prioritising their own survival.

# Background Research



Tekken Logo (Upload.wikimedia.org. 2017).

Street Fighter Logo (Upload.wikimedia.org. 2017).

Early in the design phase of the assignment there was a lot of research surrounding traditional fighting games. More specifically, games such as Street Fighter and Tekken were inspirations for the game as both franchises have been very prominent in the Arcade fighting scene. However, unlike Smashville, these games make heavy use of combo’s and chaining abilities to overwhelm your opponent. Although the combo system in these games is interesting, it can be rather complicated to develop, and fighting games are traditionally developed with a controller in mind, not a keyboard and mouse. Therefore, a simpler approach to the genre was needed. This lead us too…

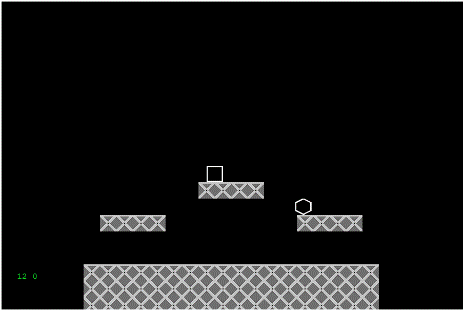
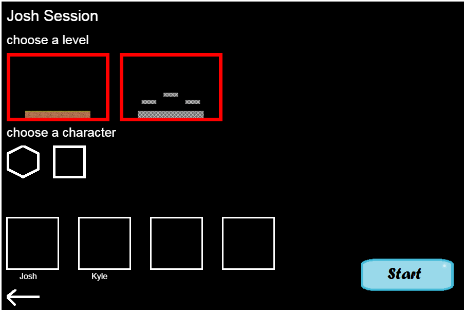
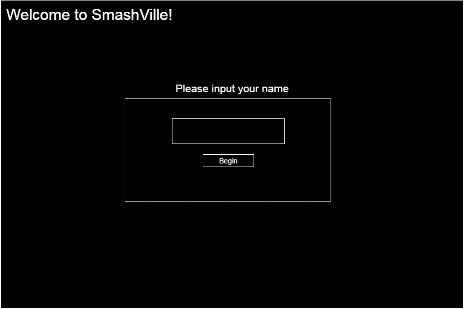
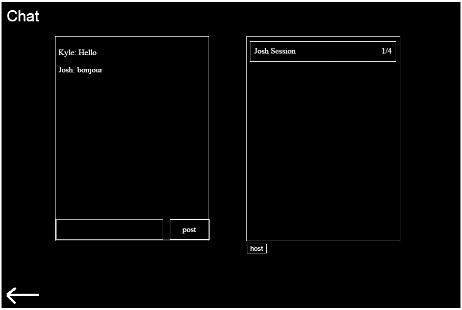
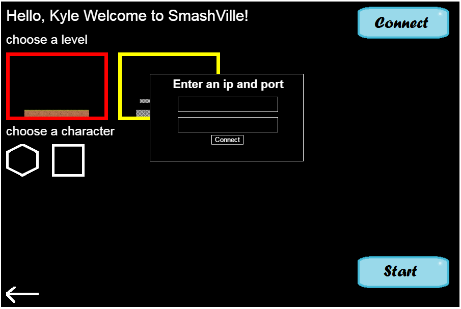


Super Smash Bros logo (Upload.wikimedia.org. 2017).

… the Super Smash Brother’s series, a physics based fighting game on the Nintendo platform. What is great about the Smash Brothers formula is that it is remarkably easy to replicate on a PC. This is because combo systems and complex button mapping do not play a prominent role in the game itself. Smash Brothers is a game that can be picked up by anyone of any age and not require them to know every combo combination, or the best way to chain them.

# Screen/Level Map

## Screen Map

********

Net

Local

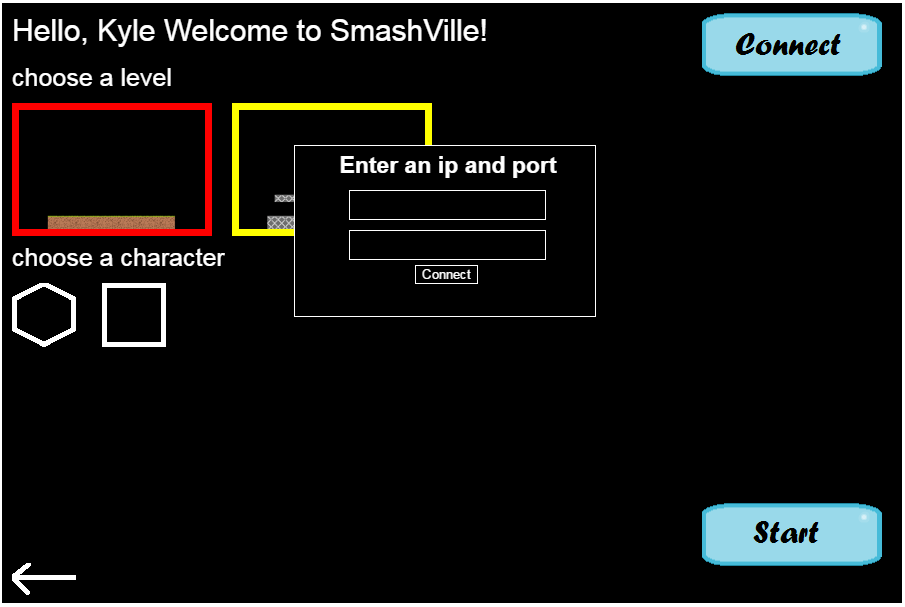
The screen map above shows how each screen state flows into the other.

## User.js



This is the first screen introduced to the user. It is a combination of the Phaser game state and a DOM element, that takes in the name, and stores it into the backend. All the user can do on this page is input their name and click the Begin button.

## Menu.js – Local Mode



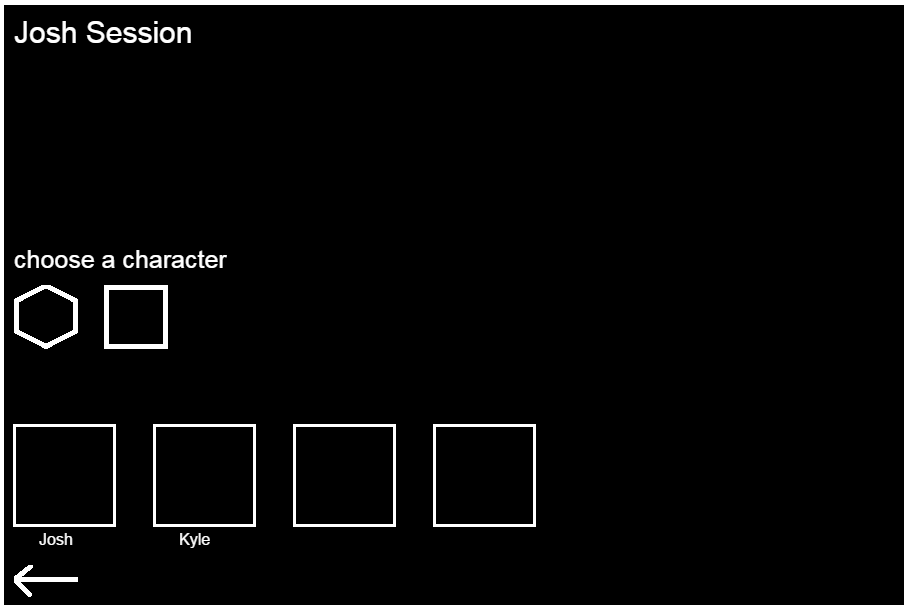
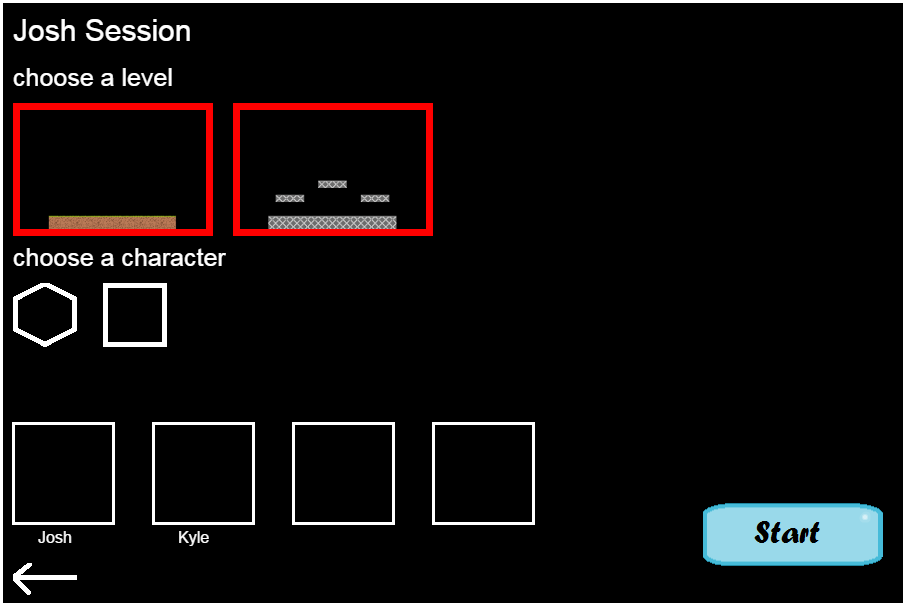
The next screen has more functionality than the previous one, allowing the user to select a level and character then press start - launching a local session of the game. However, the user can instead choose to connect to a server by inputting an IP address and port number to connect to.

## Chat.js



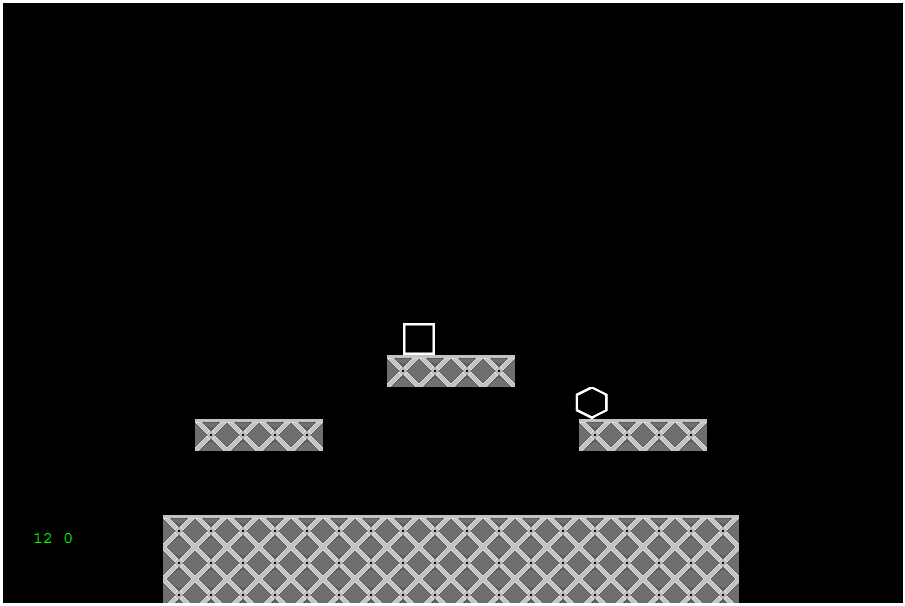
This screen is visible to the user once they have connected to a server. It consists of 2 DOM html elements for chat and sessions. The chat passes posted messages to the server, which then delivers the message to the other connected users. The session’s DOM element works by presenting clickable buttons – which allow users to join the selected session – or click on the host button to start their own session (Producing a button for other users).

## Menu.js – Net Mode



When joining a session the user will arrive back onto the Menu.js screen - except net mode is now activated. The user will see different versions of the screen depending on if the user joined or hosted the session. If the user is the host he can select the sessions level and start the game.

## Play.js



The play screen is accessed by the menu state, regardless of whether the state is local or net mode. The difference is that in local mode there are no adversary’s to play against. Upon defeating your enemy, or being knocked off 3 times, you are returned to the menu screen.

# Network Utilisation

## Network Architecture:

**Server**

Receive Packet

|

Calculate Logic

|

Send Packet to player(s)

**Client**

Send Packet

Receive Packet

|

Apply Packet Logic

The diagram shows a high level approach to how the client and server communicate. The lines indicate the direction of the packets (Who shall be sending and who shall be receiving them).

## Component Interactions:

There are several kinds of messages which are exchanged between the server and the client. This is because we have several client states which are looking for different kinds of packets. This section will cover these different states.

The first state is the menu state, where the client is looking for information regarding chat messages and session statuses. The client will be awaiting for the server to send information when a new session has been created, when a session status has updated or when another user has sent a message. At this stage the user will send packets when they have entered a message, joined or hosted a server.

Next we have the session state where clients have joined a game session with other players, or hosted one that players can join. The client will be looking for packets regarding the other connected players character selections or waiting for the sessions host to begin the game. The client will send a message when they’ve selected a character, left the session or (If they’re the host) begun the session.

Finally we have the in-game state, where the host has begun the session and the players are actually playing the game. Clients will be looking for messages when other players have moved or been hit. The user will also be sending the same kinds of messages that they are expected to receive.

There are also other handling packets for when a client has disconnected from the session or passing players details when a new client has connected.

# Client

## Objects

There is a total of 4 objects on the client side of the application.

### Player.js

The player is responsible for all interaction involving the player, this includes: movement, attacking, checking for hits, checking for ring outs, and dying. The player object also keeps track of its lives (also known as stock), its X and Y position and its sprite dimensions. It also has its own internal update function to update its state

### Enemy.js

The enemy object is very simple, as most of the logic surrounding its interactions with the other players is handled by the server. On the client side of the application all we keep track of is: its X and Y, its sprite dimensions, its characterID which determines its sprite, its name, and its lobbyID, which determines its position in the lobby. We also handle removing the sprite on the client side.

### Session.js

The session object is made up of 4 properties: it’s ID which is used by the server to determine which server it is, its body content which is rendered by the client, its playerCount for the number of players present in the lobby, and its state.

### Sound.js

The sound object handles the music that plays throughout the application. It can play, stop, or loop the music, it can queue up the next song, and it can adjust the volume of the music. It also contains its own update function.

## Code Structure

<https://github.com/jsdoc3/jsdoc>

The code structure has been documented using jsdoc3, you can find the documentation regarding the files and methods used at our GitHub page for the project.

<https://malithium.github.io/Smashville/documentation/Client_Documentation/index.html>

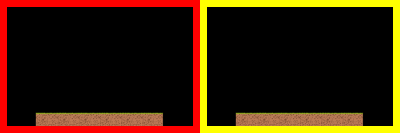
## Assets

### Connect\_button.png

H:\Year3\Smashville-develop\game\assets\connect_button.png

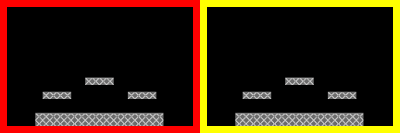
This asset is used on the menu.js screen, it is used to bring up the connect prompt to connect to the server, it was made by Kyle Tuckey.

### Level1\_button.png



This asset is used on the menu.js screen, it is used to select level 1 as the arena, and it was made by Kyle Tuckey.

### Level2\_button.png



This asset is used on the menu.js screen, it is used to select level 2 as the arena, and it was made by Kyle Tuckey.

### Character1.png



One of the test character sprites used in the game. Created by Joshua Petherick.

### Character2.png



One of the test character sprites used in the game. Created by Joshua Petherick.

### Character3.png



One of the hidden character sprites used in the game. Created by Joshua Petherick.

### Start\_button.png

H:\Year3\Smashville-develop\game\assets\images\start_button.png

This asset is used on the menu.js screen, it is used to start the game and move to the play screen. Created by Kyle Tuckey.

### Back\_button.png

H:\Year3\Smashville-develop\game\assets\images\back_button.png

This asset is used on nearly every screen, it is used to return to the previous menu screen. Created by Kyle Tuckey.

### Tiles1.png

H:\Year3\Smashville-develop\game\assets\images\tiles1.png

These are the level tiles used on level 1. Created by Kyle Tuckey.

### Tiles2.png

H:\Year3\Smashville-develop\game\assets\images\tiles2.png

These are the level tiles used on level 2. Created by Kyle Tuckey.

# Server

## Data Structures

The server holds several objects for managing the different players and sessions. This section will go over each of the objects and explain their purpose.

First we have the Client object which, as the name suggests, holds details regarding each individual client. These details include their id, name, x and y positions and hit percentage. The server will use these details to distinguish which client their communicating with and manage their overall gameplay experience.

Next we have the Session object which holds information about individual sessions, such as their name, state, selected level and an array of all the connected players. The server needs these details to manage the overall session state and cut down the time required to find players who are part of that session.

The Message object is, as the name suggests, designed to hold messages which have been exchanged in the “chat” screen. It simply holds the name of the player who sent the messaged and the body of the text.

Next the Logic object holds several functions which calculate in-game events, such as checking collision, registering hit damage and calculating knockback effect. This object was designed to give the developers the ability to tweak how effective certain actions are or add additional logic later on into the game.

Finally we have the Debug object which runs several small tests on the Logic object (Although more could be added in the future) that get run whenever a client starts the server. This was designed as a mini Unit Testing class so developers could easily expand or alter how testing the server works.

## Code Structure

<https://github.com/jsdoc3/jsdoc>

The code structure has been documented using jsdoc3, you can find the documentation regarding the files and methods used at our GitHub page for the project.

<https://malithium.github.io/Smashville/documentation/Server_Documentation/index.html>

# Implementation Evaluation

## Strengths

### Easy communication between Phaser and Socket.IO

Due to the both technology’s being JavaScript based, having them communicate and work with each other is very straightforward. Simply Initializing a socket object in Phaser with an IP and a port allows the application to connect, and then that socket can be referenced anywhere to send packets. It’s a very straightforward and comprehensible system.

### Tiled JSON

The levels were created using the TILED map editor, Phaser has native support for the JSON exported from TILED which makes creating levels exceptionally easy. This also means that creating multiple maps for the game is a simple process. TILED also allows us to specify where objects can spawn saving us having to hardcode the values internally.

## Weaknesses

### Partially Peer-to-Peer

One major weakness of Smashville is that it’s partially Peer-to-Peer. This is because the program couldn’t run Phaser HEADLESS on the server side due to Node.js executing socket.io from the terminal (Not from a browser). There are better ways that this could be handled, such as using an engine rather than the Phaser library (Unity for example). The server could also have additional checks to prevent players “hacking” or passing across “broken” values. Due to the nature of these issues though there would need to be a lot of checks required to counter this.

### Lack of built in textbox support in Phaser

Phaser does not have any built-in support for textbox’s, this meant that DOM JS had to be implemented to make up for this. This has resulted in an odd combination of Phaser and DOM JS working in tangent between states which has complicated the code base and made it harder to manage, also getting the HTML elements to scale and function within the Phaser canvas and between states is difficult, scaling the screen distorts some of the UI due to this.

# References

Upload.wikimedia.org. (2017). Available at: <https://upload.wikimedia.org/wikipedia/en/e/e1/Tekken_series_logo_as_of_2012.gif> [Accessed 28 Apr. 2017]

Upload.wikimedia.org. (2017). Available at: <https://upload.wikimedia.org/wikipedia/en/e/e9/Street_Fighter_Logo.png> [Accessed 28 Apr. 2017]

Upload.wikimedia.org. (2017). Available at: <https://upload.wikimedia.org/wikipedia/en/a/af/Super_Smash_Bros_4_merged_logo%2C_no_subtitle.png> [Accessed 28 Apr. 2017]

Jsdoc3 (2017). Available at: <https://github.com/jsdoc3/jsdoc> [Accessed 28 Apr. 2017]

# Appendix 1: Presentation



Double click the above link to view the presentation.

# Appendix 2: GitHub Repository

<https://github.com/Malithium/Smashville>

Click the above link to view the GitHub repository.