Multiplayer, Multiplatform Web Game – Hyper Bout

A Project Report
Presented to
The Faculty of the Computer Engineering Department

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Bachelor of Science in Software Engineering

By
Erni Ali, Phil Vaca, Randy Zaatri
12/2013

APPROVED FOR THE COLLEGE OF ENGINEERING

Keith Perry, Project Advisor

Professor Keith Perry, Instructor

Dr. Sigurd Meldal, Computer Engineering Department Chair

ABSTRACT

Multiplayer, Multiplatform Browser Based Game – Hyper Bout

By Erni Ali, Phil Vaca, Randy Zaatri

Job postings on Dice indicated a high demand for skills in web oriented technologies which prompted the formation of this project. State-of-the art tools include HTML5, Box2D, and Node.js which allow developers to build real time and complex applications accessible to any HTML5 browser enabled device. Google's GRITS provided the basis of this project which utilized these state-of-the art technologies to handle browser based games. As more applications continue to shift their focus from native applications to a platform agnostic environment, HTML5 satisfies this need as the primary utilized language for web and web based game development. With HTML5's widespread and continuous growth, exploring the possibilities it can provide in an industrial or commercial environment is crucial.

Therefore, with the background and knowledge gathered from the various state-of-the-art research and needs for the project, the objective of this project was to develop a web based multiplayer, multiplatform competitive game using several emerging web development technologies, including HTML5/CSS3, JavaScript, Socket.IO/Node.js and Box2D.

Ultimately, this project provides an applied academic environment to develop the necessary skills required to align with this significant industry shift in web oriented technologies. Individuals in both an academic and commercial environment can reference this report to review the current capabilities of HTML5 applications.

Acknowledgments

The authors are deeply indebted to Professor Keith Perry for his invaluable comments and assistance in the preparation of this study.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Project goals and objectives

As Hyper Bout utilized developing and emerging technologies of the time, its goals and objectives were more performance oriented. Ultimately, the goal of this project was to see how well a real-time application could hold up in an HTML5 enabled browser with Node.js handling the networking aspect of the application. Since games are traditionally the most demanding of applications, Hyper Bout was an ideal choice to view any performance or complexity issues of the technology used. Overall, the goal of Hyper Bout was to develop an HTML5 enabled real-time browser based game to which four players can connect to and interact with minimal latency. The following is a list of objectives that determined the success of Hyper Bout:

- Hyper Bout can provide up to 4 player support, handling real-time interactions by players.
- Hyper Bout has minimal lag noticeable by its users, assuming reasonable latency between the clients.
- Hyper Bout should be developed with HTML5 paired with Box2D, Node.js, and JavaScript.

With all these objectives met, Hyper Bout proved that HTML5 paired with Box2D, Node.js, and JavaScript is a viable gaming platform to develop on.

1.2 Problem and motivation

The problems and needs of Hyper Bout did not arise from the fact that the industry needs more games, but rather from the technologies that Hyper Bout is based upon. As developers currently look for a well performing, platform agnostic development platform, HTML5 paired with a networking library such as Node.js would in theory satisfy their needs. The problem addressed by Hyper Bout was to see whether or not HTML5 was a viable platform to develop on and if so how well it performed. As mentioned earlier, a reason why a game was chosen for this project was so that the project could fully test what capabilities HTML5 has as well as review any performance or complexity pitfalls provided by the development platform. Just like Google's *GRITS* game, the source code of this project was made open to the public through a GitHub repository so that aspiring HTML5 developers, specifically those interested in game development, may use our code as a reference to develop an HTML5/Node.js based game.

As for the motivation of Hyper Bout, the primary driving force behind choosing an HTML5 real time application was from the job postings trends observed on sites such as *Dice.com*. The authors were already familiar with object-oriented languages such as C# and Java. However, when deciding on a project, the team wished to choose technologies that were booming and were expected to be known by any well-versed programmer. With the increased need of experience in the context of web development, the authors chose to develop something more complex unlike a standard website that can be done in basic HTML. Therefore, the team decided to develop Hyper Bout for the senior project.

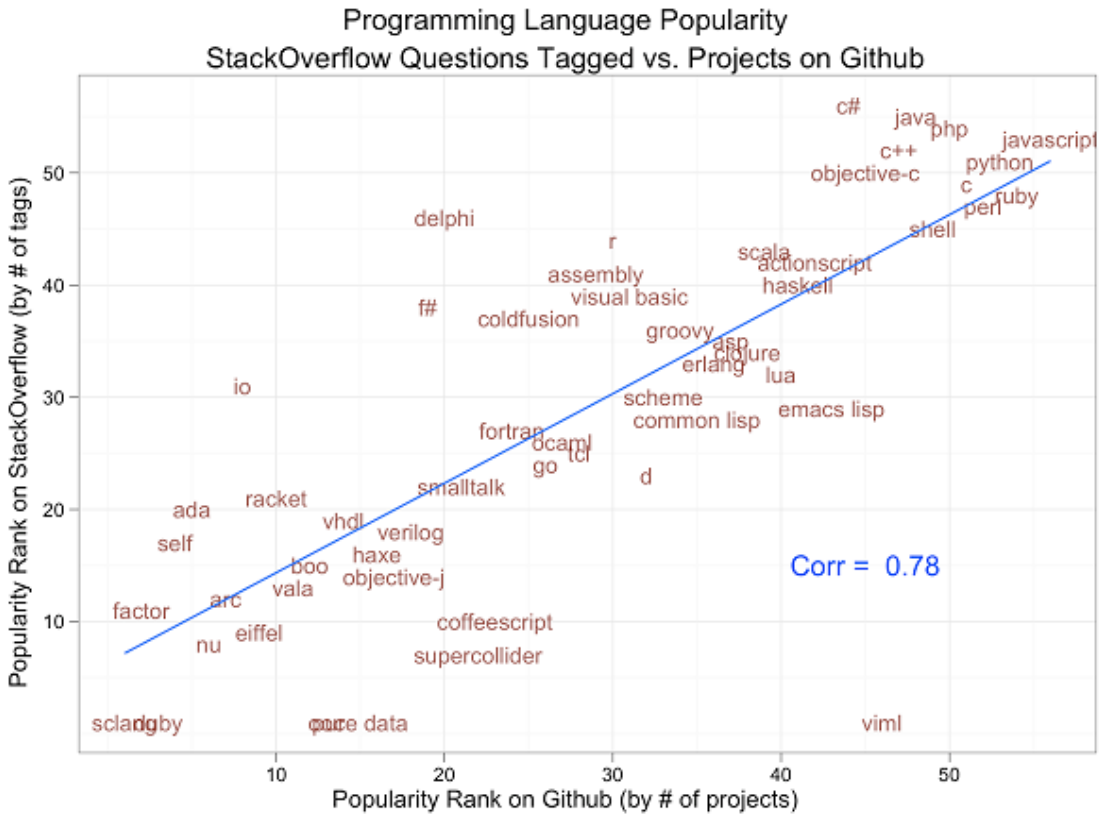


Figure 1: Popularity of Programming Languages (Source: Finley, 2012)

Figure 1 above shows a diagram of the most popular programming languages measured by the number of tags found on *StackOverflow* and the number of projects found on *GitHub*. As observed, JavaScript can be found at the very top right corner of the diagram, indicating its importance in the job market. Typically, JavaScript goes hand-in-hand with HTML development. This diagram provided the team with a reason to develop Hyper Bout on top of an HTML5/Node.js/JavaScript platform.

Another motivating influence for the creation of Hyper Bout was from evidence on *Dice.com* listing of the top 20 languages that companies were looking for developers to fill as seen in Table 1.

Table 1: Program Language Job Popularity (Source: Moore, 2012)

Ranking	Language	Job Postings	% Change ('11-'12)
1	Java	17,599	+8.96%
2	XML	10,780	+11.70%
3	**JavaScript**	10,738	+11.64%
4	**HTML**	9,587	-1.53%
5	C#	9,293	+17.04%
6	C++	6,439	+7.55%
7	AJAX	5,142	+15.81%
8	Perl	5,107	+3.21%
9	PHP	3,717	+23%
10	Python	3,456	+32.87%
11	Ruby	2,141	+39.03%
12	**HTML5**	2,035	+276.85%
13	Flash	1,261	+95.2%
14	Silverlight	865	-11.91%
15	COBOL	656	-10.75%
16	Assembler	209	-1.42%
17	PowerBuilder	126	-18.71%
18	FORTRAN	45	-33.82%

The languages in bold were the ones that were used in the development of Hyper Bout. As mentioned, Hyper Bout utilized HTML5 and JavaScript, which includes Box2D.js and Node.js. From the chart above, right below Java and XML, JavaScript is in the top three languages in demand according to the postings from *Dice*. Since Java and XML were already familiar languages to the team, JavaScript and HTML were the ideal candidates at the time of project selection. However, what was interesting about the chart was the rapid boom of HTML5 jobs. As seen in the period from 2011 to 2012, HTML5 related jobs have experienced a job increase of 276.85%. Since HTML5 was still a new standard at the time, estimates expect even a more rapid rise in its popularity. This trend satisfies the motivation to develop Hyper Bout as an HTML5 based game.

1.3 Project Application and Impact

The following sections provide details of the impacts that Hyper Bout has from an academic, societal, and commercial perspective.

1.3.1 Application of Project Results

From an application standpoint, this project is most relevant to individuals or teams who wish to develop a browser based game using HTML5, Node.js, and Box2D. The code of this project is open source so the public can simply pull this project from the Hyper Bout GitHub repository. The code can either be viewed or used by the public so developers interested in developing future games may pull snippets of code from this project's source code.

The other half of this project emphasizes performance. In other words, since HTML5 was a new technology at the time of Hyper Bout's development, this project was developed to test what could be done using HTML5 and how well strenuous applications could run. Individuals who want to examine the performance HTML5 provides could refer to *Chapter 7: Testing and Experiment*. By examining the contents of this report, readers can determine the complexity of developing an HTML5 application and can additionally look into this project's source code.

1.3.2 Academic Impact

From an academic standpoint, professors and researchers may utilize Hyper Bout as an example to review the capabilities modern HTML5 applications provide. Students

and professors will be able to observe how Hyper Bout handles real time communication between multiple clients and how data mismatch between clients was handled. Classes teaching web development can take this project as a template and work off of it to have students create complex applications. Instead of building typical websites that handle basic synchronous communication between a server and a client, students can be exposed to more valuable and modern programming techniques used in developing real time HTML5 games.

1.3.3 Industry Impact

Hyper Bout can have an impact in the industry in terms of developers wishing to monetize on developing HTML5 games. As of the time of the project's completion, Hyper Bout does not handle accounts or money transactions, however, developers can use Hyper Bout's code and develop an F2P model from it. Some companies have already monetized on browser based games, yet, most times these games were basic and limited by the technologies that were used. Additionally, by moving on to HTML5, existing game developers developing for the iOS, Android, or Windows Mobile platforms do not have to worry about developing native applications anymore as an HTML5 game can be developed and run on any HTML5 enabled browser.

1.3.4 Societal Impact

Inspiration

Hyper Bout was inspired by two popular game series. The first inspiration was drawn from *Bomberman*, a classic game that allowed the player to take the role of

characters called Bombermen, who can use bombs strategically to blow up walls to open paths and defeat enemies. The second inspiration was drawn from *Super Smash Bros Brawl*, a game on the Nintendo Wii console, where four characters are put into an arena / environment and have to knock each other off the arena to score points. Hyper Bout features the ability to throw bombs or projectiles in order to defeat the opponents while having a four player competitive system where the winner was the player who had reached a particular point limit.

Super Smash Bros Brawl

Super Smash Bros Brawl was a very popular game in Japan, selling over 500,000 units of its initial 600,000 unit shipment. This forced Nintendo to ship more units due to the game becoming frequently out of stock (Stephany, 2008). This situation does not occur very often because usually the sales numbers do not meet the numbers that are predicted by the company. Hyper Bout's game play is very similar to *Super Smash Bros Brawl* and based on the sales figure of *Super Smash Bros Brawl* in Japan, Hyper Bout has a high change of being very well received for the Japanese culture. While the Japanese may or may not appreciate the 2D sprite graphics in Hyper Bout due to varying graphic affinities that are popular in Japan, Hyper Bout's networking performance is critical because in *Super Smash Bros Brawl*, the online multiplayer feature was not as well received due to its limited network capabilities.

Additionally, Japanese culture typically does not promote realistic violent games. Even though Hyper Bout is a competitive game, it does not feature a lot of violence and is playful and cartoon-like by nature. Players simply damage each other using

projectiles. There is no blood or gore animation to show that the character was damaged as this was shown through the player's decreasing health points as well as sprite reactions. In an interview with a *Yakuza* game director, Toshihiro Nagoshi, Nagoshi stated that western shooter games were not popular in Japan because it had "too much violence for our [Japanese] taste" (Jenkins, 2011). Since Hyper Bout did not adopt a typical "western style" of violent gameplay that kept Japanese consumers away from the authors' target market, Hyper Bout would blend into and mix well with the Japanese culture without many changes made to the game design or gameplay itself.

Bomberman and Art Style

Bomberman was also a Japanese classic game which was published in 1983 and became very well-known all over the world, especially in Japan and America. Hyper Bout fits into Japan and America's gaming culture since Hyper Bout drew its inspiration from *Bomberman* by having projectile based gameplay. Even if there was no problem with the theme to be ported to Japan, there were several adjustments that could be made in order for Hyper Bout to survive in the Japanese game industry. For example, making Hyper Bout more "anime stylized" instead of "retro stylized" would probably fit more into the Japanese culture. However, this would mean Hyper Bout would have more difficulties fitting into the American gaming culture and Hyper Bout would end up not being well-received in America. A modern retro style was used because focus on the American market was emphasized.

Game Length

A "bout" or a round in Hyper Bout usually lasts about 3-5 minutes. Three to five minutes is well suited for busy working people or for casual gamers. Somebody who does not have too much time would only need three to five minutes in order to play Hyper Bout. Since the game is not time consuming, it gives players a lot of flexibility, which is well suited for many cultures. Hyper Bout is well suited in the Japanese society because traditionally, the Japanese tend to work long hours and barely have free time to play games. The length of the game attracts both hard-core and casual gamers.

People want shorter and shorter games. Take for example, Angry Birds, a game consisting of ten to twenty second stages that allowed players to advance to the next stage by completing the previous stages. According to an article on CNN, experts believed that console gaming is dying because of the rise of social and casual gaming (Snow, 2012). There are less and less dedicated gamers willing to spend time to play long console games. This is why Hyper Bout is adapted to today's culture. It has a social aspect and the game only lasts for 3-5 minutes which appeals to casual gamers.

Game Graphics

Paul Neurath, a creative director at Zynga stated that "People aren't as more motivated by cutting-edge graphics as they once were" (Snow, 2012). Society is changing and people do not only care about graphics anymore. Gameplay and aspects of social networking are also important aspects to consider in a game. In addition, Paul Neurath also stated that graphics cannot advance anymore further than how it is now other than to have holographic displays. While there are certainly gamers who still care about graphics, there are less and less of them now (Snow, 2012). Gamers' perspectives

have changed. Graphics no longer matter as much anymore. Hyper Bout further emphasized this perspective. Hyper Bout uses simple 2D sprites for its graphics. There is nothing fancy about the artwork or the graphics. However, with exciting gameplay and multiplayer networking capabilities, Hyper Bout fits well into the modern gaming community.

China's Market for Games

Hyper Bout is free for everyone to play. Earning money was planned to be done through advertisements. If the authors were to globalize the game to China and put a price on Hyper Bout, it would probably be pirated and income from the game would be difficult to earn. Therefore Hyper Bout would be offered for free and the way to earn money would be to have in game advertisements to generate income. This method would generate less income, but would help navigate around China's societal issue.

1.4 Project results and deliverables

Hyper Bout consists of server side and client side code, both packaged with this document. As the names imply, the server side code is intended to be run on a Node server while the client side code is to be provided by the clients when they connect to the server.

In addition to the code, the Node.js installer is provided for Windows, Mac, and Linux based systems. Additionally, a user guide and setup manual are provided for users who wish to setup the game and have user connect to it. When the user runs the game, up to four players can join on a single server and play the game to completion.

Deliverables

- System V1.0
  - Code
    - Node.js Modules (Socket.IO/WebSockets)
    - Server Side Code (“ServerBout.”)
    - Client Side Code (“Public”)
    - Test Scripts (“Tests”)
  - Resources
    - Images
    - Audio
  - Installers
    - Node.js Client for Windows, Mac, Linux
- Documentation
  - Hyper Bout Project Report
  - User Installation Manual
  - Gameplay Manual

1.5 Market Research

In terms of profits, the video game industry has been rapidly growing and consists of now an approximate $50 billion global industry as seen from the graph below (PricewaterhouseCoopers, 2011). From 2007 to 2011, there has been a growth of sales by 30.4% in a mere period of four years. Ubisoft's CEO Yves Guillemot predicted that sales of games would in fact even take up a larger growth, as much as 50%, within the four year period (Anderson, 2007). While he was not particularly correct, 30.4% is still a considerable amount. From these predictions and trends observed in the previous 10 year period, it should be clear that video games are going to continue to be economically feasible for quite some time considering the room for growth.

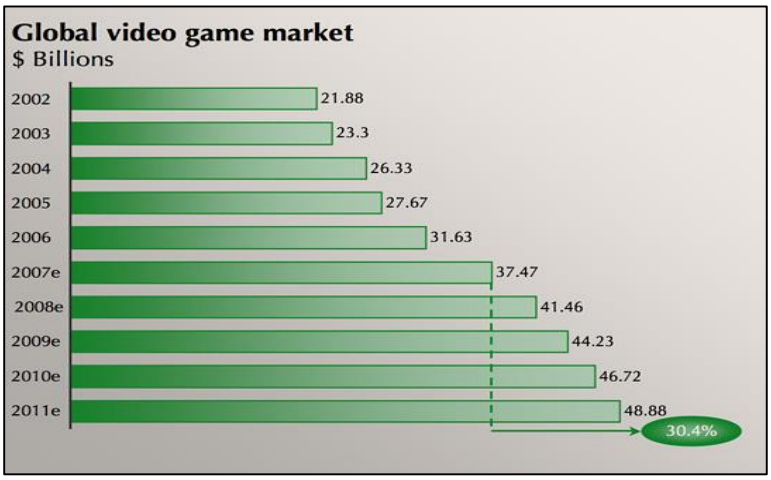


Figure 2: Global Video Game Market Sales Figures

In terms of sales purely in the United States, the US sales consist of about 33% of total video game sales with the majority of other sales consisting of East Asian countries (e.g. Japan, China, Korea) along with major European countries (e.g. France, Britain, Spain, etc.). The diagram below provided by the NPD Group/Retail Tracking Service shows a large boom in video game sales particularly starting in 2007. The main reason

for this boom was due to the emergence of online distributed sales systems such as Steam (Aldridge, 2012). People no longer have to walk into stores and purchase their games for their home consoles or PCs anymore as most games are now distributed through linked user accounts. As seen in the diagram, store sales (represented in gray/black in 2009-2011) went down while online delivery formats went up significantly. Hyper Bout would work well in sales as it follows a distributed delivery format. It is convenient to play Hyper Bout as players are not required to go into a store and purchase a disk based game. Players also have immediate access to the game just from their web browser.

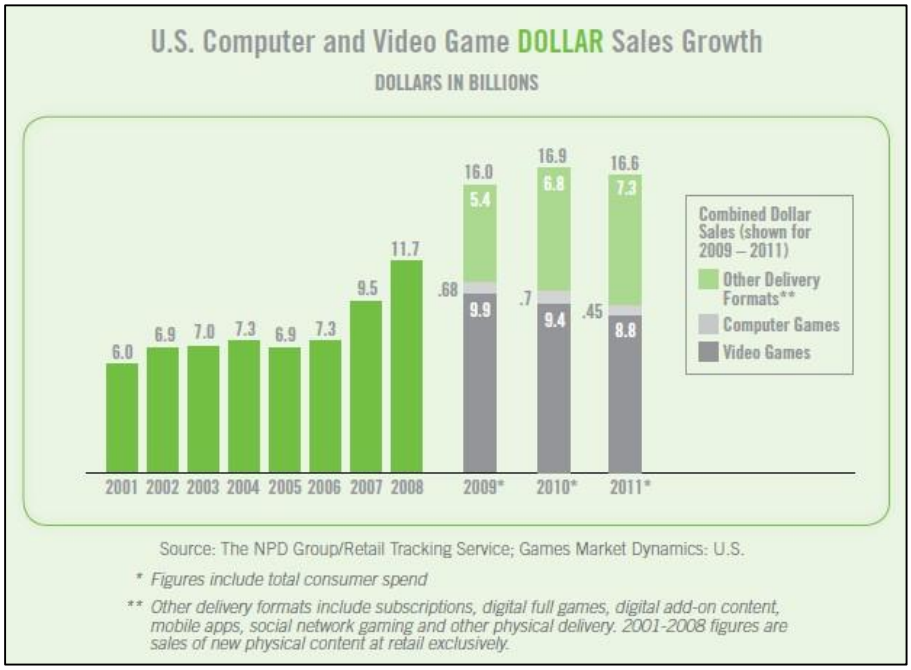


Figure 3: U.S. Computer and Video Game Dollar Sales Growth

1.6 Project report Structure

The following table will describe the details and contents of the following sections in the document.

Table 2: Document Section Descriptions

Section	Title	Description
2.1	Background and Used Technologies	This section describes the background of the project and technologies that were used in Hyper Bout. The reason why Hyper Bout was chosen as a senior project topic is described here and the different modern technologies used are also discussed in detail.
2.2	State-of-the-art	Different existing projects and products in the market that are related to Hyper Bout are presented and discussed here.
2.3	Literature Survey	Existing related research results and research projects are presented here.
3.1	Domain and Business Requirements	Different requirements for the business domain is presented and explained here along with UML Diagrams such as state chart diagram, activity diagram, and class diagram that will help individuals from the business domain visualize the project.
3.2	System functional requirements	System requirements describing what the system did are given here.
3.3	Non-functional requirements	Requirements that specifies restrictions for the system is specified here, such as performance or standard compliance.
3.4	Context and interface requirements	Requirements for the interface and also context on which the project was developed, tested, and deployed on is described.
3.5	Technology and Resource Requirements	The minimum hardware requirements and software requirements are specified.
4.1	Architecture Design	The architecture design used by Hyper Bout is explained here.

4.2	Interface and Component Design	Here lies the interface and component designs of Hyper Bout.
4.3	Structure and Logic Design	Structure and logic design that went into the project.
4.4	Design Constraints, Problems, and Trade-offs	Different challenges, constraints, and the trade-offs that were made in order to overcome challenges.
5.1	Implementation Overview	Overview of the programming languages that were used, scope of implementation, and also the dependencies in the implementation are described here.
5.2	Implementation of Developed Solutions	The details of Hyper Bout implementation, techniques, methods, and algorithms that were used.
5.3	Implementation Problems, Challenges, and Lesson Learned	Lessons that were learned, the problems and challenges that were faced, and how solutions were found.
6.1	Tools Used	The development tools and testing tools that were used in the project was presented here. Reasons why the tools were chosen and where they were applied are also explained.
6.2	Standards	Explanations of the different standards that were followed during the development of the project.
7.1	Testing and Experiment Scope	The scope of testing, focus, objectives, and test criteria are detailed here.
7.2	Testing and Experiment Approach	The approach on implementing a test framework into the system and different test cases are explained.
7.3	Testing and Experiment Results and Analysis	The results of the tests and whether or not the test cases passed or failed.
8	Conclusion and Future Work	Description of the conclusion of the project and plans for the future of the project.

Chapter 2 Background and Related Work

2.1 Background and Used Technologies

The following section discusses the background of this project, the concepts that were applied throughout the project, and contains a list of the different courses in San Jose State University that provided the necessary knowledge, leading to the completion of the project.

2.1.1 Concepts and Knowledge

1. Project Estimation

Project estimation was a necessary skill to have when managing a project that spanned for a year, such as Hyper Bout. Projects in the school environment typically only lasted for one semester or less. Most times projects only lasted for two weeks to a month. Therefore, knowing how to manage, plan, and estimate a long term project was crucial.

There were a lot of factors that had to be taken into account when doing project estimation for Hyper Bout, such as individual schedule, availability, and skill levels. All of these factors were needed to accurately determine how many features could be implemented given the time constraints. In addition to development, there was also a design and documentation phase to take into account. Reducing the duplicated work was also needed to be ensure so that no waste was produced as a result.

Knowing how to estimate a project led to Hyper Bout's success. The project was finished on time by November and deadlines and time constraints

were all met. Therefore, being able to estimate accurately takes away a lot of stress and last minute rush that goes into a project.

2. UML

Knowledge of how to draw UML diagrams came in use during the design phase. UML diagrams were really important in design because it helped convey and visualize the Hyper Bout's design in the form of a standardized diagram. That way, no confusion was present on how the system was to be implemented amongst the team during the implementation phase.

If implementation was not done as designed, there would be problems when it came to integration and having to put the pieces together. Not only that, but UML diagrams clearly drew out the state and the actions that users would take in order to interact with the system.

3. JavaScript Prototype Patterns

The prototype pattern was used throughout Hyper Bout's code. Prototyping in JavaScript is essentially inheritance in Java. Because of the vast knowledge that the authors have learned in the university regarding inheritance in Java, this design pattern was chosen. Since JavaScript did not have any method for inheritance like the Java programming language, prototyping is used to implement inheritance. Prototyping reduces code duplication and makes the code looks cleaner.

By using prototype, properties and methods of an object can be accessed easily. Not only that, relationships between objects are defined, which makes the code easier to understand and also adds an integral component to coding Object Oriented JavaScript.

2.1.2 Technologies

There were several modern technologies adapted by this project. The primary technologies used for this project were:

- *Client Technologies:* HTML5 and JavaScript
- *Middle-Tier Technologies:* JavaScript, Node.JS, Box2D, Socket.IO

Client Technologies:

1. **HTML5**

   HTML5 was chosen as the host language for displaying Hyper Bout in a web browser. It is the most used markup language for web development and is highly extensible and portable. It allowed Hyper Bout to reach as many users as possible since HTML is supported by all web browsers and is extensively documented on the web. HTML5 were used to display graphical assets, as well as creating the interface for users to interact to navigate Hyper Bout's menus.

2. **JavaScript**

   JavaScript is a programming language designed specifically for web development. This programming language was chosen because it allowed the creation of an application that ran on most web browsers. It is also closely

coupled with HTML5, allowing the creation of powerful and efficient client side software applications that do not require users to have powerful systems to run efficiently. Hyper Bout's game engine was written using JavaScript, which allowed handling of game mechanics as well as graphics processing.

JavaScript is also widely used for web development and additional features are being added through the use of libraries. One library that was extensively used in Hyper Bout was Box2D for JavaScript. Box2D was used in Hyper Bout to handle game physics and collision detection between entities. These features of JavaScript creates a development environment that gives all of the features needed to create a web based game, without requiring the aid of many other specialized technologies to complete the functionality of this game.

Middle-Tier Technologies

1. **Socket.IO**

In order for users to connect to the Hyper Bout server, cross-communication between the users' systems and the host server are required. Socket.io allows achieving real time connectivity between users with little delay. It is also capable of asynchronous communication, which allows users to communicate with one another without having to set up a direct connection.

2. **Node.js**

Node.js is a software system that is powerful and flexible enough to allow for real time network communication to be sent to multiple users simultaneously.

Additionally, Node.js is cross platform and supported by many server hosts. Node.js allowed Hyper Bout to have users be able to quickly log in to this project's site and play a game with multiple users quickly and efficiently. No additional software is required for installation and all web browsers that support Socket.IO were compatible with Node.js.

3. Box2D

Box2DWeb is a port of the original Box2D that was written in C++. Box2DWeb is contained in a single file, which made it easy to host on a web server and does not require any additional work from the user. This library is also kept up to date by the developer and currently has the most features out of the two JavaScript ports of Box2D. This part of Box2D also does not require the use of prototyping within JavaScript, which allows Hyper Bout's code to be written with fewer constraints.

2.1.3 Courses Helped

There were several courses that gave the authors the necessary knowledge for completing the project. These courses were the courses that were taken in San Jose State University. Below is a table that list out these courses.

Table 3: List of Courses

Course Number	Course Title	Useful Subjects and Knowledge Learned
SE131 and SE133	Software Engineering I and II	These two classes taught software development processes in detail. Different documentation requirements, functional requirements, non-functional requirements, system requirements, and different UML diagrams, and implementation design was covered in this class. These topics were really useful for the completion of the documentation required for Hyper Bout. For example, on chapter 3, UML diagrams are presented, which is one of the useful skills learned from these two classes.
CS174	Server side Web Programming	HTML5 standards and web programming skills are taught in this class. For an HTML5 web game project like Hyper Bout, this is a really useful skill to have to ensure that Hyper Bout conformed to the W3C HTML5 standard in the industry.
SE165	Software Engineering Process Management	This class gave the knowledge on how to manage a project with a small team. Not only that, it described different values of a leader, how to complete a project and meet deadlines by estimating and determining schedules, and also risk management. With these skills, the authors were able to manage and complete this project on time.
SE187	Software Quality Testing	Different methods and concepts for testing were covered here and this information gave an idea on a way to test this project and produce a high quality output. Testing is covered in chapter 7 of this document. Both white box and black box testing were used to test Hyper Bout to produce a high quality product.
ENGR100W	Engineering Reports	The authors learned skills for writing high quality technical reports from this course. This skill, of course, was applied throughout this document to provide a high quality documentation worthy of publication for future individuals who wished to know more about this project.

2.2 State-of-the-art

As of the publication of this document, HTML5 games are still an emerging product and therefore are limited in its application due to the limitations in the technologies used to build the game. That said, there are a few examples of complicated HTML5 games that are found in the market. There exist many sample games on sites such as http://html5games.com/, however these games are typically single player and simple in terms of game logic. For the context this project, two specific implementations of browser based multiplayer games were studied: (1) GRITS and (2) Command and Conquer.

The first of these two games was developed by a six member team at Google and presented during the Google I/O 2012 Conference. GRITS played as an overhead shooter where robots eliminated each other in an arena. This game was built upon an authoritative server model and provided well documented code on how the Google team handled such challenges including client prediction and using network resources effectively. (McAnlis, 2012)

The second of these two games was a remake of a 1995 game based of the same name. This HTML5 based remake was created by Aditya Ravi Shankar who has also written other books including *Pro HTML5 Games*. As *Command and Conquer* was a game filled with much content and complexities including several hour long campaigns and artificial intelligence, this remake has gained much attention with HTML5 development communities.

2.3 Literature survey

As Hyper Bout is a multiplatform, multiplayer web based game, this section reviews HTML5, Node.js, and other necessary technologies used to run a HTML5 enabled game.

In terms of the literature survey, HTML5 and Node.js are relatively new technologies that are being adopted by startups and large companies globally. These technologies emerged from noticing a trend that websites were no longer simple pages that provided information and interacted synchronously with the user. When HTML4 was drafted in 1997, web developers did not give much thought to dynamic pages. Websites were filled with multimedia and provided real time information flow between clients and servers. These more complex web services were achieved through implementation of HTML5 and Node.js without the need for exterior plugins.

One key aspect that HTML5 addressed was platform and plug-in independence. Through HTML5, applications were no longer bound to a specific operating system nor need specific plug in extensions. For instance, Adobe Flash was often used to implement multimedia applications on the web. Slowly though, Flash has been dying out as large companies such as Apple dropped support of Flash for the iOS. Google also dropped support of Flash as well, stating that they were not supporting it for Android 4.1 and above (Lawler, 2012). Ultimately, this meant that applications written in HTML5 had no problems with any hardware or platform dependencies as long as a HTML5 supported browser was available on the platform of the user's choice. This ensured that once the application was written once it could be run on many devices.

While HTML5 holds many promises, it is important to note that HTML5 is still a developing technology as stated by the W3C (W3C, 2012). As of the time of this writing, HTML5 has not been fully implemented in any major browser. While it was not expected for HTML5 to be fully supported by browsers for some time, a majority of features have been handled in the latest versions of Firefox, Chrome, and Internet Explorer and are slowly making its way to mobile web browsers.

As HTML5 handles client side visuals and content, Node.js along with WebSockets are the primary system running on the server machine that handled data flow and communication. A key aspect behind Node.js is that it is a system that maximized an application's scalability while keeping overhead to a minimum. Node.js's prevalence has rapidly expanded into the industry, now being used by a majority of large companies. Some of the companies listed on GitHub included eBay, Microsoft, and LinkedIn where networking was an essential aspect of their own business (GitHub, 2012).

As for the book Aditya Ravi Shankar wrote, *Pro HTML5 Games* provided a functional architecture that one would need to build a scalable network game. Networking, texture atlasing, and collision detection were described in detail in his book. This book also provided examples of how to build two browser based games including an *Angry Birds* like physics game and an implementation of the *Command and Conquer* game currently running on the book authors' site. More importantly, Shankar provides a clear implementation of how to handle networks through WebSockets. Previously, the only way to do communication was through "polling and long polling with a steady stream of requests" and while they worked, they "had very high network latency as well as high-bandwidth usage" (Shankar, 2012). Obviously this would be very problematic

for a real time game, hence why Shankar dedicates a chapter solely to resolve this issue with the newly introduced HTML5 WebSockets.

In addition to the coverage of browser based games in the industry, much research has been done to see how well multiplayer based gaming can be handled and improved over a network. Many frameworks have been proposed, but one framework that stood out was one proposed using HTML5 with WebSockets with multiple servers. According to the framework built, their tests show that their "Three.js 3D Engine and WebSocket based MOGs [Multiplayer Online Games] can easily support the interaction of a small group of users" (Chen & Xu, 2011). While the focus wasn't necessarily on Three.js for graphic rendering, the system architecture includes two servers: (1) A web server to handle web interactions and (2) A game server to handle solely game processing. This is one implementation method that one can consider in order to halve data transmissions to a single server so that one server does not have to handle all interactions.

Additionally, much research has been done on Node.js. As an overview, Node.js focuses on a high performance, low memory consumption model, and supports JavaScript in modern web browsers such as Google Chrome. Performance of Node.js can also be improved by using multithreading techniques which allows the full utilizations of multicore hardware. This means that networking programs will run smoothly and efficiently without user lagging. Node.js can run in multiple instances in multiple threads in parallel. A multi-node library is available in order to allow for the operation system to be able to take Node.js' sockets and run them in parallel (Tilkov et. al, 2010).

Lastly, there is also the study of how to handle collision detection over a network. Tom Chen's article on collision detection describes a solution to dealing with collision detection across a multiplayer based game. Within his paper, he describes a protocol called "motion-lock" (Chen, 2010). This protocol allows for a smoother experience for users game play wise, since this will allow for better object collision prediction and object trajectory prediction. This protocol also has the benefit of not using a lot of resources from the server, which allows for smoother latency between users. Conclusively, as seen in the aforementioned studies and industry implementations, HTML5 and Node.js will become standard technologies used in web based interactions and will be essential to any developer who wishes to construct a web based multiplatform, multiplayer game.

Chapter 3 Project Requirements

3.1 Domain and Business Requirements

The following sections review the diagrams and specifications constructed to meet Hyper Bout's requirements.

3.1.1 State Machine Diagram

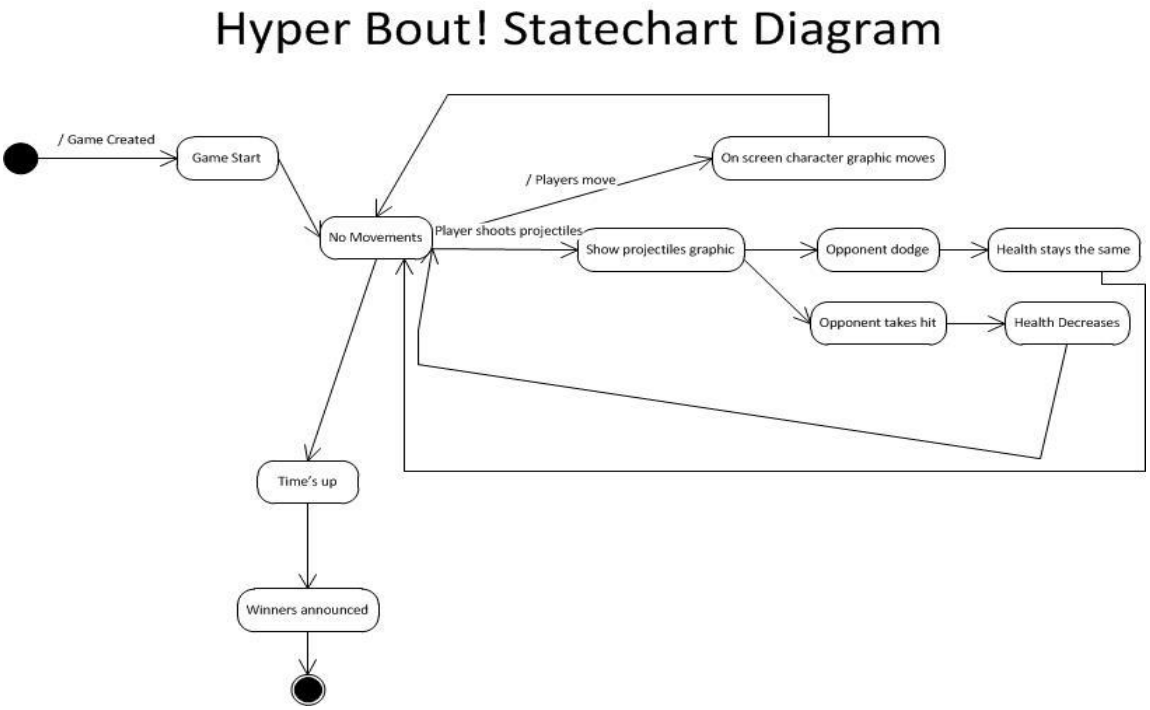


Figure 4: Hyper Bout State Chart Diagram

The state diagram above is representative of a Hyper Bout game. When a game is started, there is a very brief idle/no movement period where no players make any movements or take any actions. Afterwards, players are allowed to take action. Players can either move or shoot projectiles. When projectiles are shot at an opponent, the opponent can either dodge or take a hit. If the opponent dodged, their health remained the same. On the other hand, if the opponent took the hit, his/her health bar decreased. A game lasts about 2-3 minutes on average. [Extra Features Content Removed Here]

3.1.2 Activity Diagram

The activity diagram found on the following page shows how one player can go from one screen to another. As shown in the diagram, at first, a player enters the lobby. Once the lobby has enough players and all the players pushes the start game button, the game will start. Inside the game, players have several choices, such as moving, firing projectiles, or simply jumping. When the projectile hit the player, the player who got hit health reaches zero, the player will gain a point. However, if the projectile misses, then the player will not gain any point.

Afterwards, after one of the players gain enough points to win the game, the game will end and the winner is determined by the player who has the highest points.

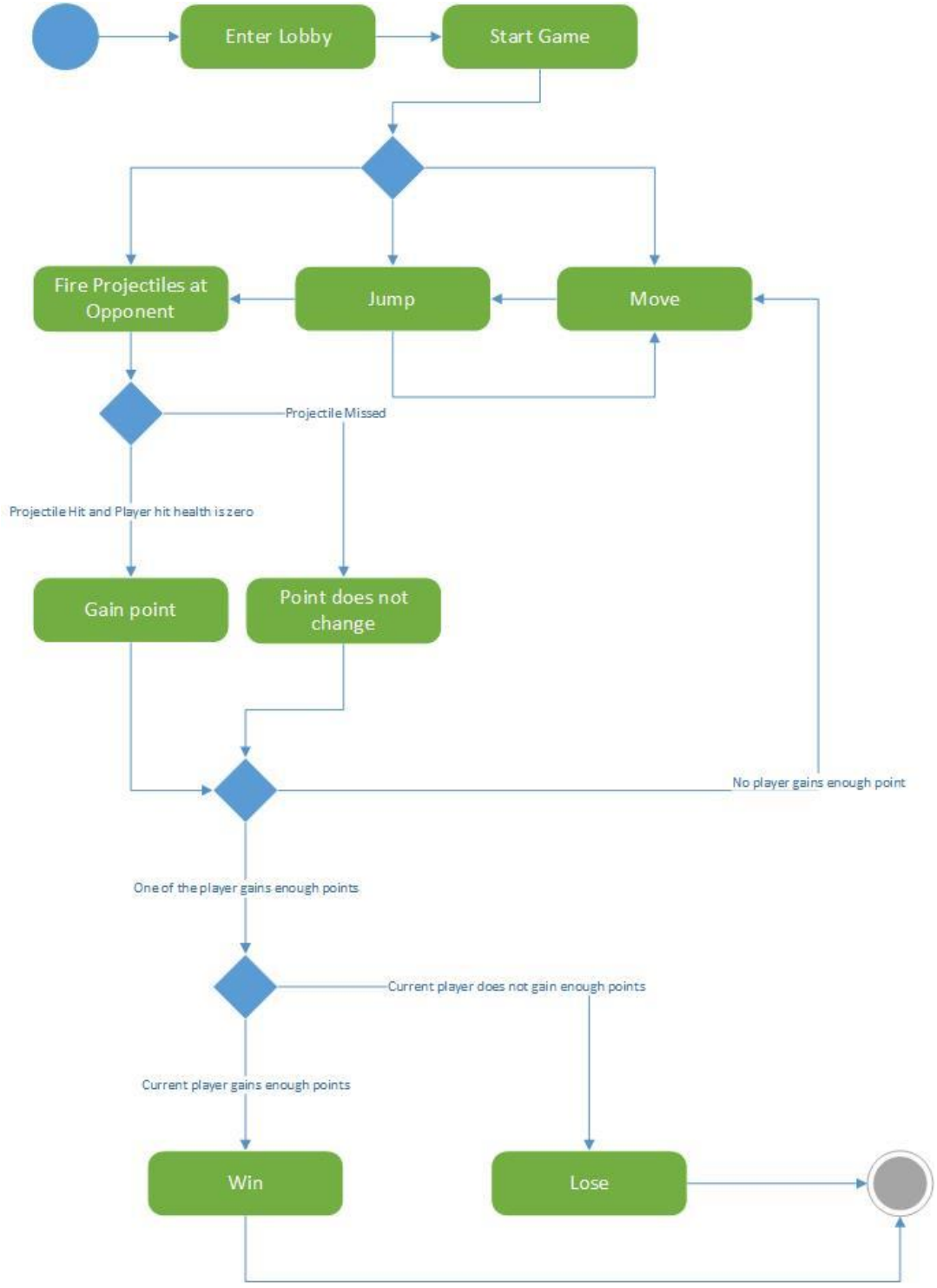


Figure 5: Activity Diagram

3.1.3 Class Diagram

The object oriented design model in Hyper Bout can be seen in the classes inside the Hyper Bout project. There are 5 classes inside Hyper Bout. Three of those classes are client-side code (Hyper Bout.js, HyperPlayer.js, and HyperPowerUp.js.) while the other two are server-side code (ServerBout.js and ServerPlayer.js).

In the Hyper Power up class of Hyper Bout inside HyperPowerUp.js file, generalization was applied. HyperPowerUp is essentially a parent class of all the power ups that are implemented inside Hyper Bout. For example, the health power up is an extension of Hyper Power Up.

All classes in Hyper Bout also communicate to one another in order to be able to work which depicts associations between this classes. For example, in order to be able to draw power ups and assign logic to these power ups, for example, increase player health when power up is picked up, the Hyper Bout class needs to communicate with the Hyper Power Up class.

The Hyper Bout project also implement the singleton design pattern where there is only one class with a single instance, Hyper Bout. The Hyper Bout class contains all the core engine, mechanic, and also logic that happens in the game. Hyper Bout is the one who handles and put everything together in the game. In other words, Hyper Bout depicts the singleton pattern because Hyper Bout is restricted to only have one instance and it is the class that coordinate actions across Hyper Bout.

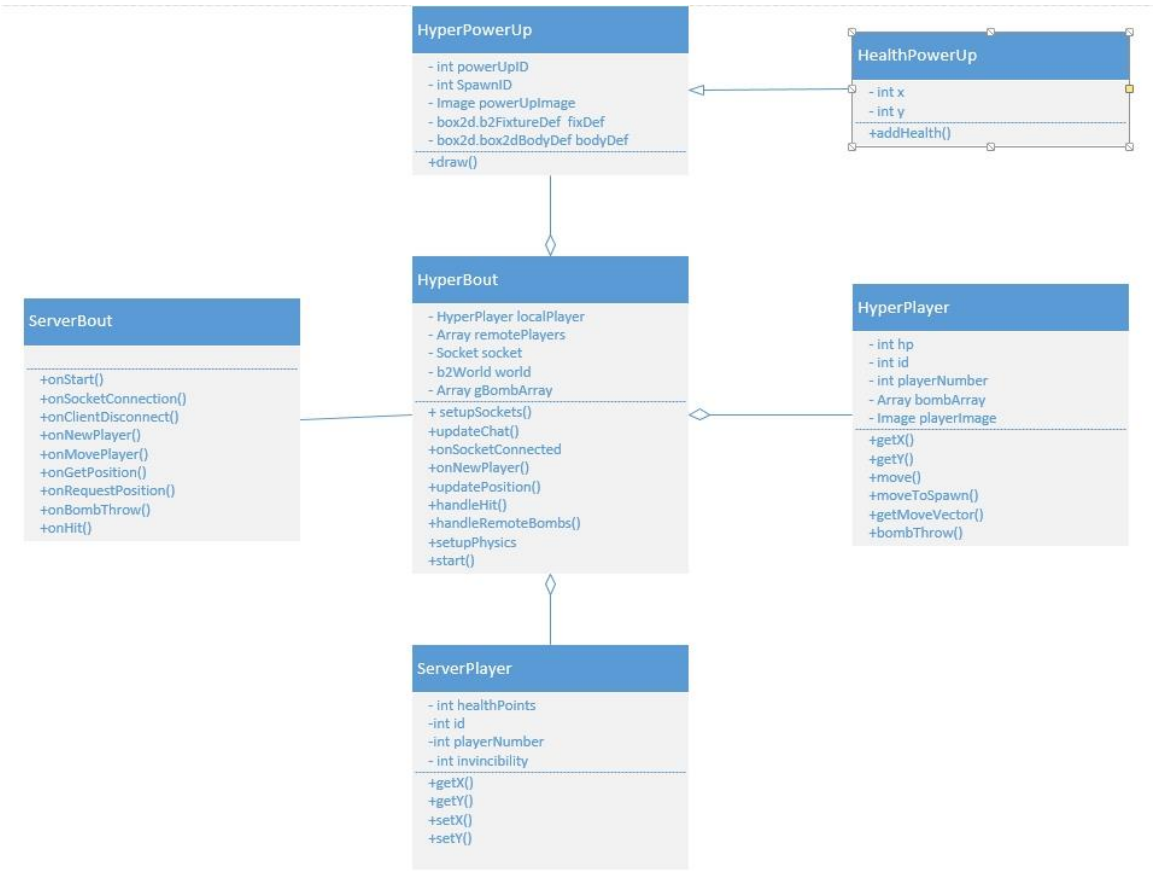


Figure 6: Hyper Bout Class Diagram

3.2 System (or component) functional requirements

Table 4: Functional System Requirements

Requirement ID	Description	Priority
S-F-1	Users shall be able to damage other players by throwing projectiles at their associated on screen character.	High
S-F-2	User's health shall decrease when they take damage.	High
S-F-3	Users shall take damage from other player's projectiles.	High
S-F-4	Users shall be able to interact with platforms and the environment of the level.	High
S-F-5	Users shall be able to close the game at any time, thus disconnecting from the game server.	Medium
S-F-6	When a user disconnects from a game, the server shall close the connection with that user.	High
S-F-7	The system shall use 4 different character sprites inside the H TML page to be selected by players.	Medium
S-F-8	The system shall support a level background.	Medium
S-F-9	The system shall keep track of the position of each player as the game progresses.	High
S-F-10	The system shall relay the positioning of each player to the other corresponding players in game.	High
S-F-11	The system shall utilize Box2D to tell when two or more objects are colliding on screen.	Medium
S-F-12	The system shall utilize Box2D for physics simulation.	Medium
S-F-13	The system shall have a how to play link that displays how to play the game.	Low
S-F-14	Users shall be able to score points by damaging other players.	High
S-F-15	The system shall contain moving and animated backgrounds, such as clouds or lights.	Medium
S-F-16	When a user shoots a projectile, the projectile shall explode upon impact.	Medium
S-F-17	The lobby should include a chat feature where members of the lobby can communicate with each other.	Medium
S-F-18	The system shall have power-up items that players can pick up and use to their advantage.	Medium
S-F-19	The system should keep track of player's points through a heads up display (HUD).	Low

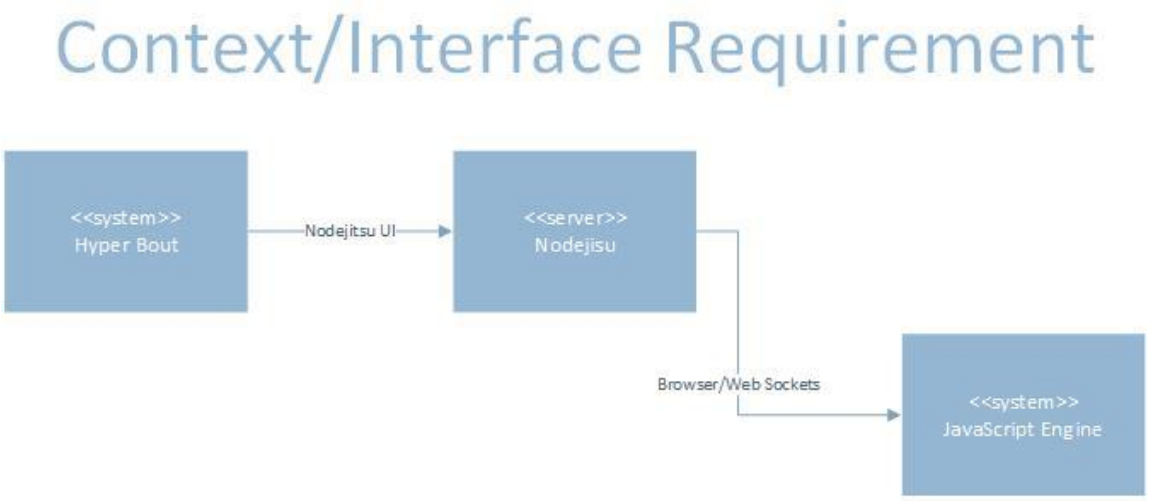
S-F-20	The system shall display user's screen names inside a game lobby.	High
S-F-21	The system shall prevent more than 4 players from joining the same game lobby.	Low
S-F-22	The system shall only allow game to start when at least 2 players are inside the game lobby.	Medium
S-F-23	At the end of a game session, players should be able to find out who won and receive post gameplay statistics.	Low

3.3 Non-functional requirements

Table 5: Non Functional Requirements

Requirement ID	Description	Priority
S-NF-1	The server should be able to host up to 4 players per game with a maximum of 150 ms of latency.	Medium
S-NF-2	The game server shall be able to support up to 4 concurrent users.	Medium
S-NF-3	The system shall not allow users to transfer malicious data to other users.	Low
S-NF-4	The system shall be able to handle at least 20 sprite images on screen at once.	Medium
S-NF-5	The game shall run at a steady frame rate of 30 frames per second.	Medium
S-NF-6	The game shall be able to run in current version of Firefox and Chrome browsers on Windows operating system.	High
S-NF-7	The system shall utilize Node.js to allow connectivity between users and the server.	High
S-NF-8	The game shall be written using JavaScript and HTML5.	High
S-NF-9	The system should use Nodejitsu to deploy and host the game.	Low
S-NF-10	The system shall utilize timers to handle image processing and ensure smooth animation.	High
S-NF-11	The game's control shall be written using jQuery to handle user inputs on multiple platforms.	High
S-NF-12	The system shall allow users to play the game using only keyboard and mouse.	High

3.4 Context and interface requirements



Context:

- HyperBout <system> - System that handles game logic, physics, and network aspects between users.
- Nodejitsu <server> - Server system that handles communication between the game logic and the users. Information is passed between users and the Nodejitsu server.
- JavaScript Engine <system> - The JavaScript engine is dependent on the browser that the user is using. For example, in Chrome the JavaScript engine is Google's V8 engine and in Firefox it is the SpiderMonkey engine. This engine handles functionality from the JavaScript files that the user is accessing with their browser.

Interface:

- Jitsu UI: An interface used to upload the HyperBout system to the NodeJitsu servers.
- Browser/Web Socket Interface: This interface is used for the user to communicate with the game by graphical queues and inputs via keyboard and mouse inputs.

Figure 7: Hyper Bout Context/Interface Requirement

3.5 Technology and resource requirements

Below is a list of hardware and software requirements for both client and server machines for this project.

Table 6: Hyper Bout Minimum Hardware Requirement for Client and Server

Minimum Hardware Requirement	Description
Intel® Core™2 Duo CPU P8600 @ 2.40 GHz 2.40 Ghz	Any computers or laptops on retail during the writing of this document should have processors that were above this requirement.
4GB RAM	By the time of this writing, laptops and desktops mostly have RAMs higher than 4GB.
ATI Mobility Radeon HD 3400 Series graphics card	A graphics card was necessary to display the game.
Keyboard	Any keyboard should work
Mouse or trackpad	Any mouse or trackpad should work.
Internet Connection	DSL or above.

Table 7: Hyper Bout Software Requirement for Client

Software Requirement	Description
IE9 and above, newest version of Google Chrome, newest version of Firefox	These were the web browsers that were supported by this project. Anything below might not have compatibility with HTML5.
Windows XP, Vista, 7, or 8 Operating System	Any operating systems that supported the above mentioned web browsers will be able to run Hyper Bout.

Table 8: Hyper Bout Software Requirements for Server

Software Requirement	Description
Node.js client	Version 0.10.18 with socket.io

Chapter 4 System Design

This section of the document describes the design choices for the system architecture, the interface, and the interface logic that runs behind the scenes. Additionally, this section will also discuss the design choices, challenges and trade-offs our team faced.

4.1 Architecture design

Hyper Bout relies heavily on communication between other players. Player movement, player created projectiles, current health points and collision detection must be communicated to all players in real time as the game progresses. Our team realized that the architecture that would best fit this required communication between multiple users is client-server architecture.

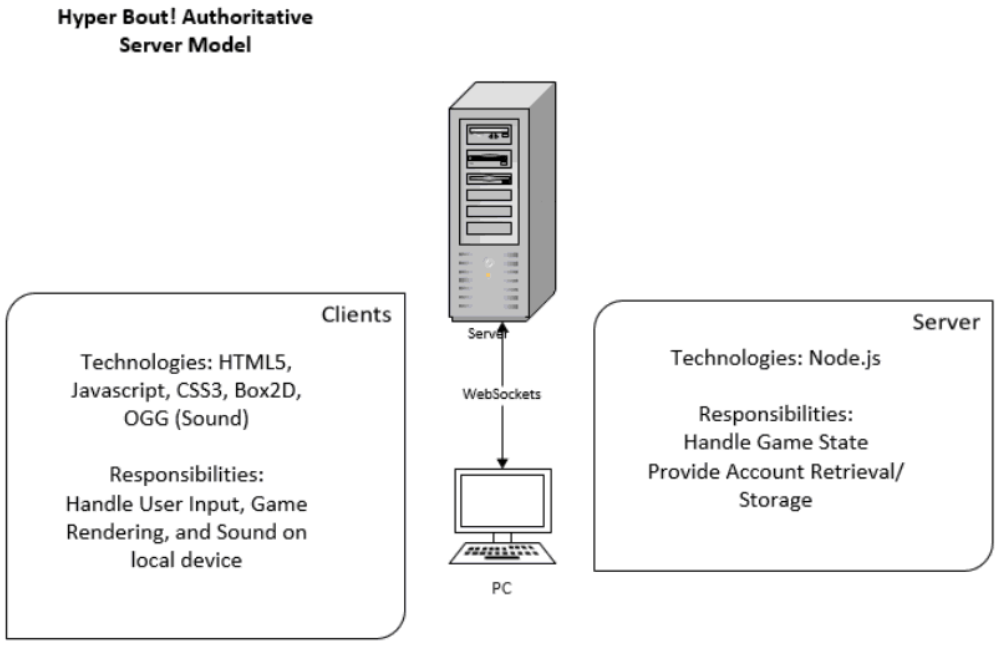


Figure 8: Hyper Bout Server Model

Each user currently logged into the Hyper Bout game will create a connection to the Hyper Bout server. The Hyper Bout server stores the game assets such as images, sounds and game logic. This information is sent to each of the user's computers so that they can play Hyper Bout from their screen. This architecture allows users to also send information to the server, which is then rerouted to the other players in the game.

Additionally, this architecture allowed us to design Hyper Bout in such a way that allows for slight modification to discrepancies that might arise while the game is in progress. Since we chose an authoritative server, when a user is out of sync due to latency issues the server can request a fix to be made to all users currently in the game. By choosing this architecture we have designed the system in a way to fix these types of errors quickly, so that users may not even notice the adjustments are being made.

4.2 Interface and component design

Hyper Bout is broken into 3 main interface components. The UserRegistration component, which consists of users entering in their name into a text box. The design for this component is meant to be simple, with everything centered on the screen so users can easily join a lobby as easily as possible.

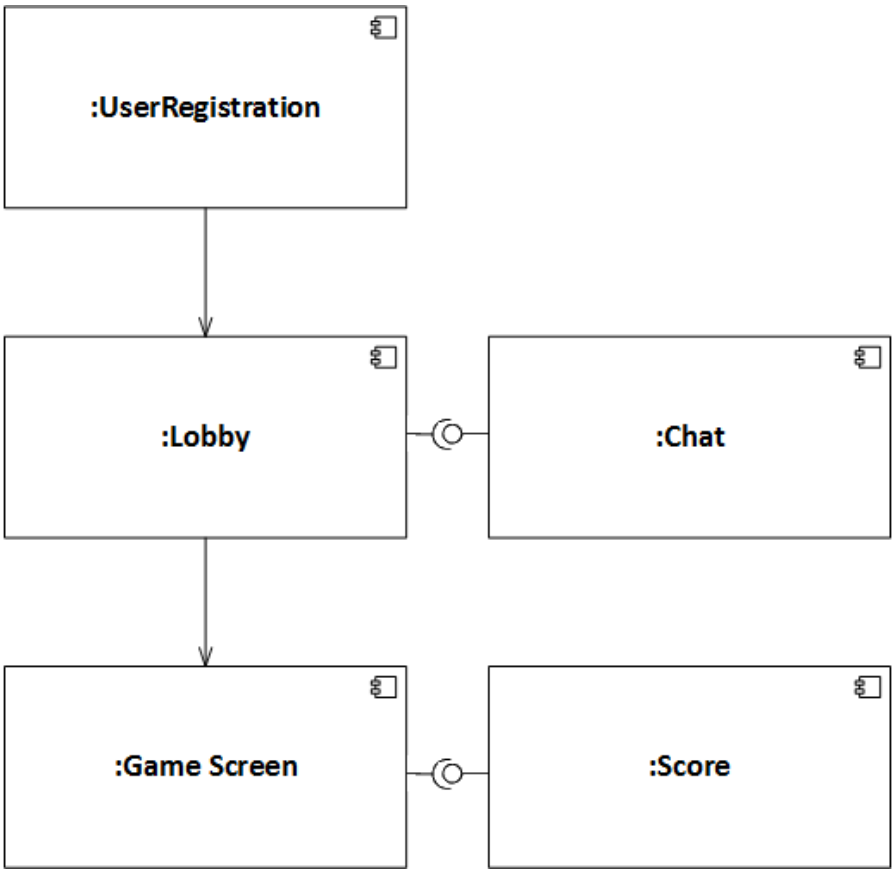


Figure 9: Hyper Bout Interface Component Diagram

The second component is broken into two different components, a lobby which displays the users currently in that lobby and the chat component where users send each other messages. Our team looked at different game lobbies for several popular multiplayer games currently out on computers. We found that chat boxes are usually

located on the bottom of the screen so that the text does not cover up other objects on the screen. To keep uniformity our team decided to stick with this general rule and created a similar lobby system where players are displayed on top and the chat is displayed on the bottom.

The final component of the game is the actual game screen. This is where players interact with the game and try to score points by hitting other players with projectiles. This interface can also be broken down into multiple components, that is, the game screen and the score box.

The game screen is centered and is displayed in a resolution of 1122x548. Our team decided to go with a smaller resolution so that we could display the game properly on even smaller monitors that only run resolutions up to 1280x800. We needed to make the game screen somewhat smaller than the maximum resolution to make up for the toolbar for each type of browser as well.

The score box is located in the top left and displays users' scores and health points. Our team designed this in such a way that it is easy to quickly glance over to see your current scores and health, but it also does not get in the way of other parts of the screen too.

4.3 Structure and logic design

The user interface for Hyper Bout consists of several different screens. The first screen that the user interacts with is the ‘User Registration’ page. This page consists of a text box where users enter in their desired screen name. Once they have entered in a name they click the join lobby button which transitions their screen into a lobby screen. Their screen name is taken from the ‘User Registration’ component and transferred over to the ‘Lobby’ component so that it can be displayed for all the other users to see.

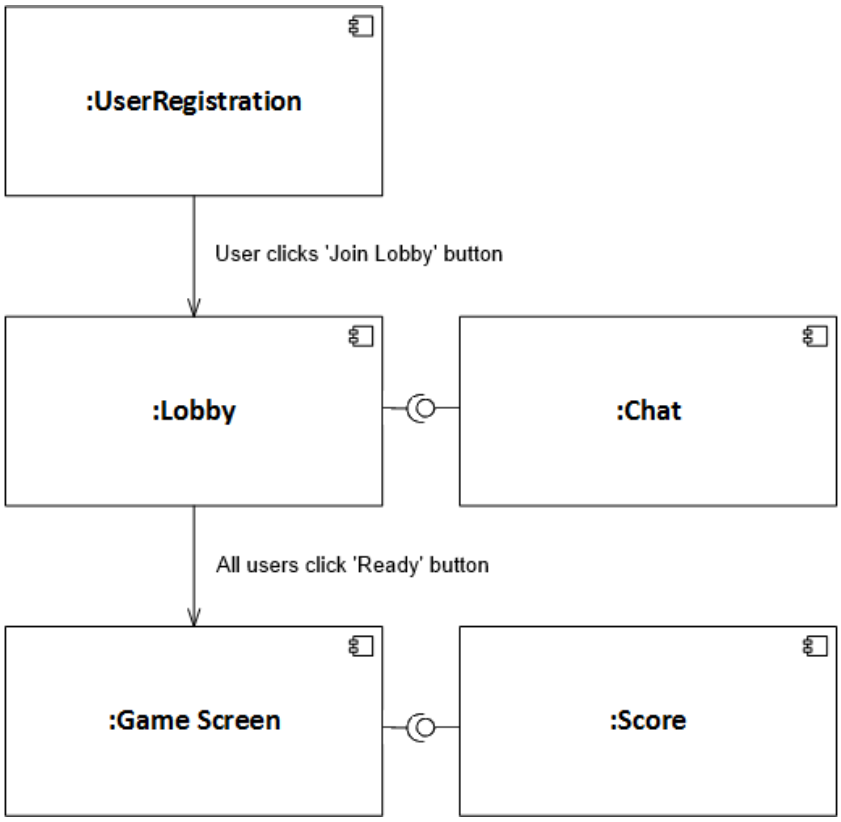


Figure 10: Hyper Bout Logic Interaction

The first component is the lobby itself, which stores the users that are currently in the lobby and displays their screen name next to their player number. This component

also displays the interface buttons such as the ‘Ready’ button. This button when clicked by a user establishes to all other users that they are ready to start the game. Once all players in the lobby have readied up the lobby interface transitions into the game play screen where the players can start playing.

The second component for the lobby interface is the ‘Chat’ component. Messages can be entered and sent to the other users by having a user type with their keyboard. These messages are displayed in a small box on the lower part of the lobby interface for other users to read.

The final interface is the ‘Game Screen’. This screen consists of all of the graphical assets that make up the Hyper Bout game, including players, projectiles and animated 2D sprites. A user interacts with this interface using their keyboard for player movement and their mouse for throwing projectiles.

Listeners are implemented into the game engine loop that waits for users to push a button on their keyboard, or click their left mouse button.

These event handlers are registered to the main game engine and wait for the player object to push a button. Once a button is pushed it is handled dependent on which type of input it was. For example, if it is a keyboard button then the button that was pushed is analyzed and the type of movement is applied to that player.

```
72      Engine.RegisterInputHandler(new Engine.InputHandler('player', function(event) {
73          if (HyperPlayer.IsMovementKey(event.which)) {
74              if (event.type == 'keydown') {
75                  self.onKeyDown(event);
76              }
77              else if (event.type == 'keyup') {
78                  self.onKeyUp(event);
79              }
80          }
81      }));
```

Figure 11: Hyper Bout Input Handlers

However, if the input was a mouse click, then the coordinates of the mouse cursor are captured at that moment and a project is applied towards that area of the game screen. Additional to the game screen is the 'Score' component that shows all of the users scores and current health points at a given time. This allows users to see who is currently winning the game and how many health points they currently have in the game.

Listeners were also created for when a user collides with another player's projectile. This sets off a flag and the number of the player is captured and their health is decremented appropriately. If the user loses all of their health points, then the player who through the projectile must gain a point. In order to do this both player numbers needed to be captured during the collision of the player and projectile. This was done by making sure that player numbers are attached to each projectile. The score interface is then updated to display all the players' current health points and the current points for each player as well.

This is done by sending a signal to the component when a player's health has reached 0, then the player who got the last hit on that player will receive a point.

4.4 Design constraints, problems, and trade-offs and solutions

This section discusses the constraints to the Hyper Bout system, as well as some of the problems and challenges our team faced when developing our project.

4.4.1 Design constraints

Several constraints needed to be placed on Hyper Bout due to resources and software constraints due to limitations with the HTML5 and Node.js platforms. Resource constraints involved graphical assets that needed to be created for each of the players, the background objects, the background images, projectiles and sprite animations. Since our team consists for 3 software engineering majors it was difficult for us to design and draw each of these pieces of the game. So in order to save time and focus on development we decided to minimize the amount of artwork that needed to be created by constraining the game to a single level as well as limiting the number of players to 4. By applying these constraints we reduced the number of resources required to create all of the graphical assets required.

Additionally, since our team only has several computers and laptops available for demoing the finished product at the expo we also needed to constrain the number of players to 4. Our team does not have the server capacity to allow more than 4 players to be in a single game at once since this would put a strain on our server.

Software constraints from HTML5 limitations also needed to be taken into account when designing our software. The total number of sprites on screen at once needed to be limited by placing a timer on each player. This timer corresponds to the number of projectiles they can throw in a given time limit. When we first developed the game players had an unlimited number of projectiles they could throw, so if a player

clicked their mouse 10 times a second, they would create 10 projectiles within that time period. Since there are 4 simultaneous players at once, this could equate to 40 projectiles being created each second on screen.

Our team found that when there were this many projectiles on screen at once the game would dip in frame rate and become unresponsive to keyboard and mouse inputs. So this constraint was put in place to allow for better interaction with our users and reduce the possibility of crashing due to overproduction of animated projectiles.

In addition to the HTML5 limitations Node and networking limitations also came into play when designing our application. Similar to the projectiles being launched in quick succession reducing the frame rate and responsiveness to the user client side, creating this many projectiles and sending them over the network also put a strain n our server.

Every time a projectile is created it must be sent over the network so that all users see it on their screen at once. When so many projectiles are being sent back and forth through the network this would cause the server to drop some packets so some projectiles would not be created on some of the user's screens. So this was resolved in the same way that the frame rate problem was fixed, by limiting the number of projectiles a player could create in a specific time period.

4.4.2 Design problems and challenges

4.4.2.1 Code Design Structure

There were several design issues when developing Hyper Bout. One of the biggest issues was developing the structure in which the Hyper Bout game was written. Developing a game where components are tightly coupled together made the creation of the code structure very difficult. Our team, coming from a Java background where class based structure is normally used made it difficult to transition to writing JavaScript where the main game engine resides in one JavaScript file.

4.4.2.2 Control Design

Designing the controls for the game was another issue that was difficult to overcome. Initially we thought that using the keyboard only was going to be sufficient for controlling the game mechanics. However, we realized that we wanted to allow for more freedom for throwing projectiles.

Using keyboard inputs for moving players around was also another issue, since keyboard keys have different numerical values per browser type, our team had to make the decision of learning JQuery as well, so that the controls would be universal across all browsers. Adding in another technology that our team was not familiar with created an even steeper learning curve.

4.4.2.3 Character Design

Character sprites needed to be animated when the user is pushing a key to make their character run. Prior to designing the animation for the sprites, characters would slide along motionless as they moved from one point to another point within the game world. When designing the running animation for the character, we found taht there were

several design choices that we made that inhibited us from implementing the running animation in a simple way. Since, each character is moved by pushing in a direction their character would start animating in that direction when the key is pushed. However, when the character is jumping or throwing a projectile different inputs are being joined together to create different types of movements. Running, jumping and throwing projectiles could be triggered all at the exact same time if the player hits all of the buttons at once. This would cause issues with the running animation.

4.4.2.4 Game Mechanic Design Challenges

Creating the general game play mechanics for Hyper Bout was also a challenge for our team. Projectile based game play made it difficult for us on deciding what type of objects could collide with other types of objects. For example, should projectiles collide with other projectiles? Should they collide with players? When should they go through platforms? All of these questions needed to be asked in order to make a balanced game that would not only be fun for users to play, but also balanced for each of the players on the screen.

4.4.3 Design solutions and trade-offs

4.4.3.1 Code Design Structure Solution and Trade-off

While designing the code structure to the game, our team decided to take the class based route when creating each of the object types in Hyper Bout. For example, the main game engine, Hyper Bout, resides in the Hyper Bout JavaScript file, while the player objects and power up objects reside in different files. This made our code structure more similar to that used when developing projects in an object oriented type of language such as Java.

This code structure also made it easy to use JavaScript's prototype feature, which is similar to that of Java's class based inheritance. This design choice made it easier for our team to familiarize ourselves with the code structure and made it so that object types could inherit functions based on object specifications. That is, all player based objects would inherit all of the player prototypes functionality, such as move and throw projectile. This made it so our team did not have to rewrite the code for each of the players within the game, but could reuse the same code for each player object.

By doing this however, this made it very difficult to host on Nodejitsu since many of the files had connections to each other through inheritance Nodejitsu would throw errors. Our team had to go through the code and rewrite some of the sections specifically for hosting on Nodejitsu as well as when running the game over a LAN.

4.4.3.2 Control Solution and Trade-off

Implementing the game with JQuery controls made it easier for our game to be played cross platform, but also added complexity to the code of our game. We no longer

had to write control specific inputs for each browser, but instead had to learn how to implement JQuery into the project effectively.

4.4.3.3 Character Design Solution and Trade-off

The character sprite constraint was overcome by choosing a sprite that looked like he was flying as he moved across the screen. This not only limited the number of sprites that needed to be stored on the server, but also made it easier to animate because each direction of movement only consisted of 1 static frame, rather than several animated frames.

4.4.3.4 Game Mechanic Design Solutions and Tradeoffs

Our team decided that when throwing projectiles, projectiles should only collide with the floor and not other players, or other player projectiles. By doing this we made it so that in order to hit other players, a user would have to throw a project directly at another players feet to be able to hit them with their projectiles explosion. This made it so when a user is above a player, or below a player they both have an equal chance to hit each other and no player is given an unfair advantage over another.

The trade-off with this is that users will not be able to bounce other projectiles back towards enemy players. This was a design choice that needed to be made since when originally designing the game play mechanics for Hyper Bout, this was one of the key features that we wanted to add to the game.

Also, since players throw projectiles so quickly, we found that it was very difficult to dodge enemy projectiles when they would collide with player bodies. This made it very easy to score points just by throwing projectiles directly at another player. This ended up being a game of who could throw the projectile at each other faster than

the other. Balancing these game play mechanics was difficult for our team, since there are not many other games that have attempted this type of game play before.

Chapter 5 System Implementation

5.1 Implementation Overview

The following section reviews the implementation aspects of the Hyper Bout system including the system scope, the platform and languages used, and the dependencies found in the project.

5.1.1 Implementation Scope

In terms of the scope of the project, the design only applies to a single dedicated server to four client model. The primary reason for this limitation was cost and time. That said, our system is designed to be scalable so that if Hyper Bout were to be released as a dedicated gaming service, the developers would simply have to develop a server load balancer and implement controls for mobile devices.

The purpose of the server load balancer would be so that multiple instances of Hyper Bout can be run concurrently at once. Since dedicated servers are required for smooth game play, a load balancer would be needed to assign players to a dedicated server. The load balancer would be responsible for seeing which lobbies require more players and which lobbies are empty so that players can be assigned to them. This configuration, as seen in Figure 12, would represent an ideal configuration of how players would connect and access to our game. Due to resource constraints, this project handles only a single instance of a dedicated server.

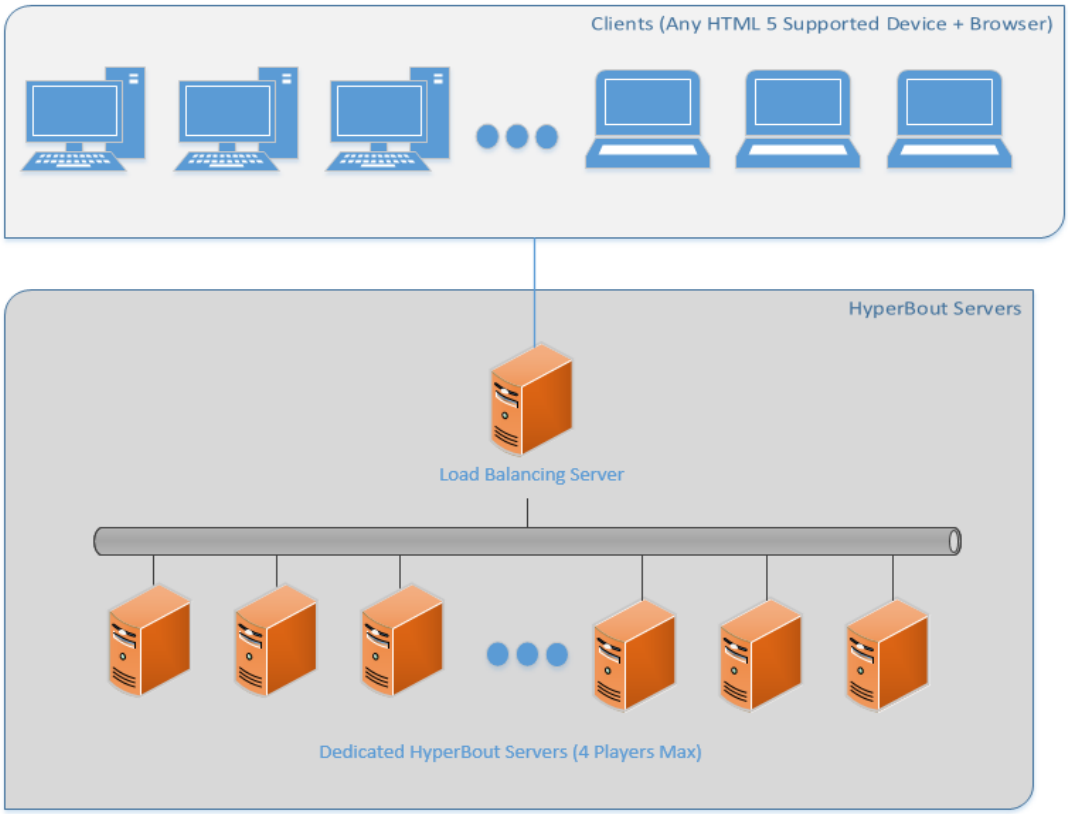


Figure 12: Full Scale Hyper Bout Commercial Configuration

Additionally there is the concern of porting our Hyper Bout project to mobile devices such as tablets and smart phones. In the early stages of project planning, mobile devices were considered for development as HTML5's multiplatform nature was ideal for this situation. That said there were three reasons why we limited the scope to desktops and laptops: (1) Gameplay elements of Hyper Bout required a keyboard and mouse for proper play, (2) HTML5 was not as well supported on major mobile devices, and (3) time constraints.

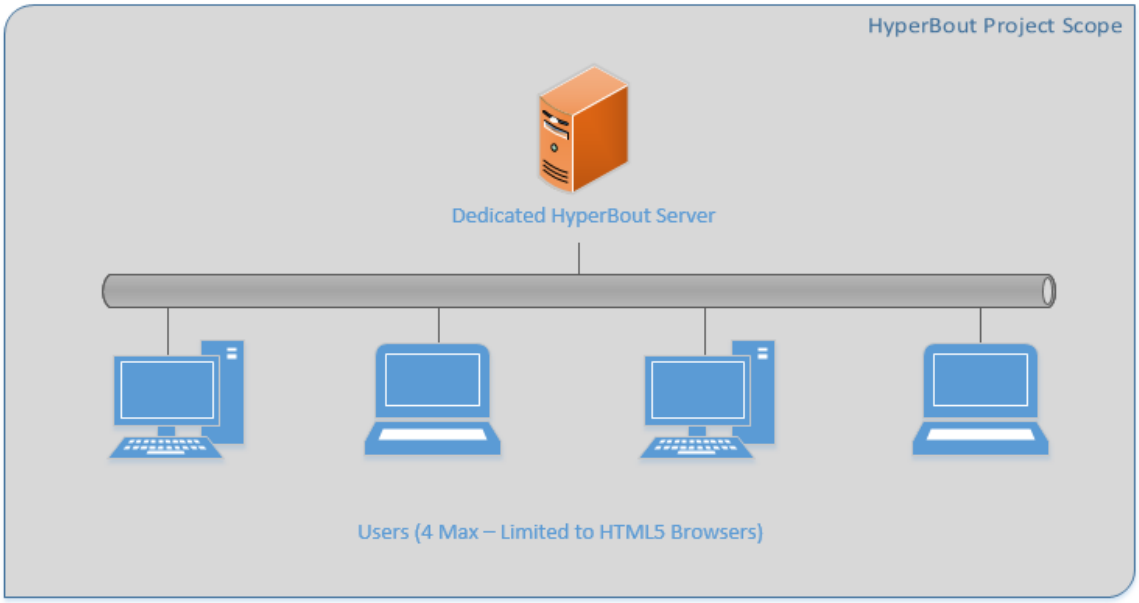


Figure 13: Scope of Developed Hyper Bout Project

Within the scope of the project, the Hyper Bout system requires that separate client side and server side code be developed for the game to properly run. Primarily the server side code was written in JavaScript with Node.js and Socket.io libraries while the client side code was written in HTML5 and JavaScript with the Box2D.js library. These technologies are discussed in further detail in the next section.

5.1.2 Used Platform and Language

Languages

1. **JavaScript**

JavaScript is the primary language that allows us to handle the logic of the system. The game engine, lobby system, user interactions, etc. are all handled by JavaScript code. Due to JavaScript's versatile nature, many tricks with encapsulating data and sending various data types over the network could be accomplished. At the

same time, this proved a challenge to the developers as some times it was not clear or as well defined what was actually being sent over the network.

The snippet of code below in Figure 14 comes from the Hyper Bout.js file. At 1,700+ lines long, this file contains a majority of the logic behind Hyper Bout's system. In particular, the game engine function can be seen below where object arrays and listeners are configured and the logic is handled. In the logic below, the listeners retrieve two bodies that have made contact and check the object data of each body (e.g. player, floor, bomb, etc.). Depending on the two colliding body types, statements would execute as a result of those two particular objects colliding. Once again, due to JavaScript's versatility, Hyper Bout's game logic can be easily managed and updated so that new functionality can be added such as additional collision listeners or animations to be run.

```
668 Engine.prototype.start = function()
669 {
670     var graveYard = new Array();
671     var pGraveYard = new Array();
672     var listener = Box2D.Dynamics.b2ContactListener;
673
674     //Listener to handle collision between to Box2D Objects
675     listener.BeginContact = function(contact)
676     {
677         var contactA = contact.GetFixtureA();
678         var contactB = contact.GetFixtureB();
679
680         //Listen to when Bomb Hits the Floor
681         if(contactA.GetUserData().charAt(0) == 'B' || contactB.GetUserData().charAt(0) == 'B')
682         {
683             //If contact B is the bomb, then push contactB's body into the graveYard.
684             if(contactA.GetUserData() == 'Floor')
685             {
686                 contactB.SetUserData('dead'+contactB.GetUserData().charAt(4)); //for the gBombArray and the blade bomb
687                 contactB.GetBody().SetUserData('dead'+contactB.GetUserData().charAt(4)); //for passing into graveyard
688                 // console.log(contactB.GetUserData());
689                 graveYard.push(contactB.GetBody());
690             }
```

Figure 14: JavaScript Engine Logic

2. HTML5

With HTML5, the newly introduced <canvas> tag has made game development in web browsers a viable option as traditional web browser games were limited to standard HTML elements that were styled through CSS. With canvas, drawing can directly be done on the browser with sprites and animations being drawn on screen as often as the engine specifies. Figure 15 shows the canvas element on screen with JavaScript handling the drawing functions. The procedure to draw items on screen is still handled by JavaScript, but the rendering is all done within the HTML5 canvas tag.

In addition to the canvas feature, new HTML5 tags such as <audio> are used to play the background music during the course of the game. Additionally the index.html file contains the references to all needed files and the basic layout of the page.

3. CSS3

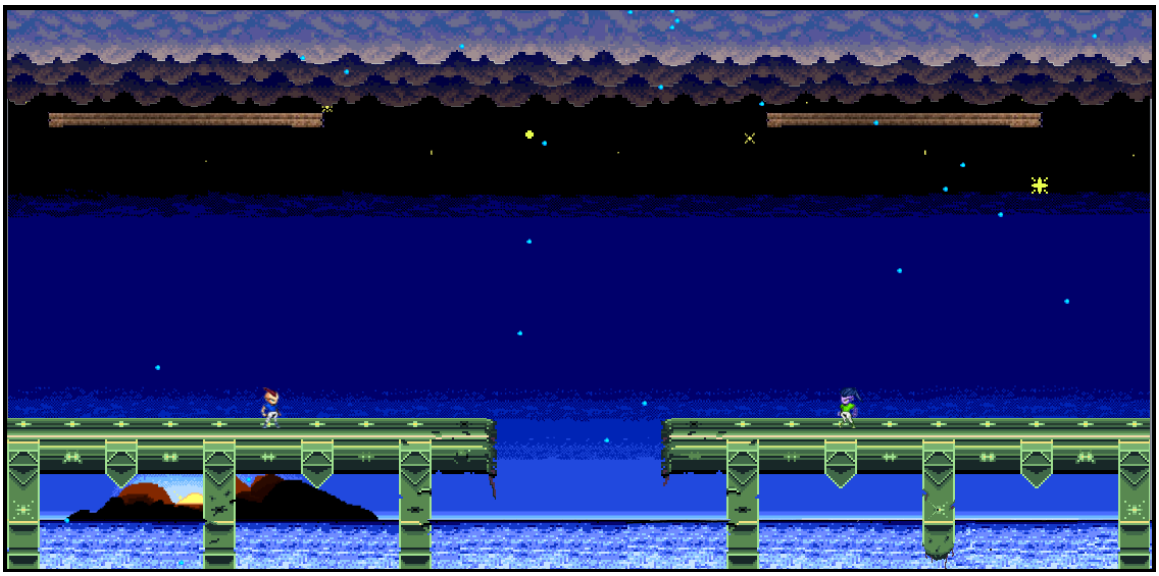


Figure 15: HTML5 <Canvas> Element with Drawing

CSS3 allowed the developers to add visual enhancements to the elements found on the HTML page. By using element selectors, we can add additional features such as fading in and background stylizing to provide a visual appeal to the game. Figure 16 shows an example of how the lobby looks with the CSS3 style applied and without and CSS3.

Platform and Libraries

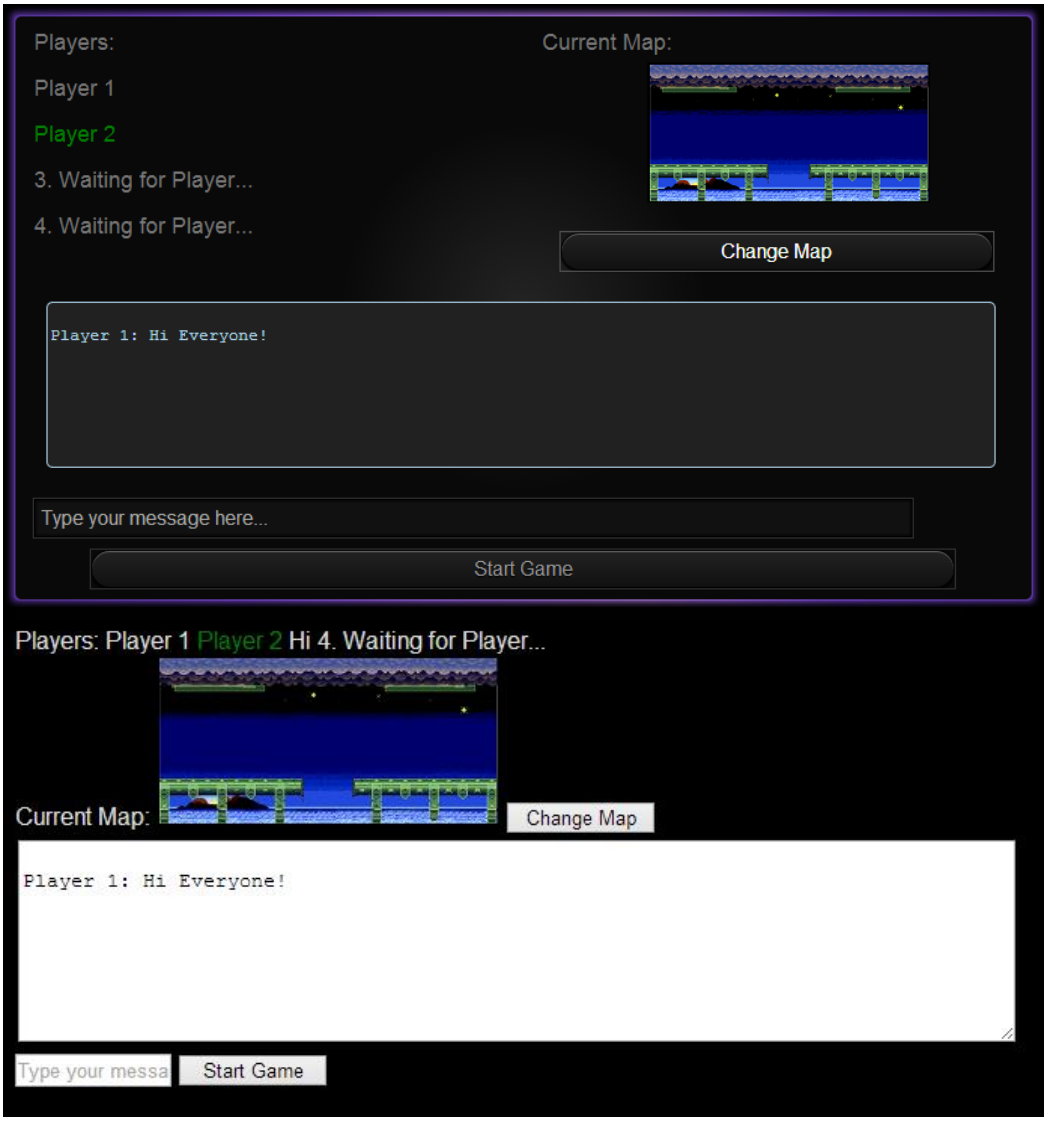


Figure 16: Lobby With CSS3 (Top)/ Without CSS3 (Bottom)

1. **Node.js / Socket.io**

The networking aspects of Hyper Bout were handled by the Node.js and Socket.io networking libraries. These libraries were used to consistently have the server and clients emit messages back and forth to each other. Depending on which messages were sent and received, particular functions would be executed on the server and client machines. Due to the asynchronous nature of these two technologies combined, a real time connection between clients could be accomplished.

2. Box2D.js

Box2D provided the team an existing physics library that could be integrated into the Hyper Bout project. One of the more complicated aspects of developing a game is designing a physics engine and with the physics engine already implemented in the form of Box2D, the team had only needed to learn how Box2D functioned in order to use its features.

One of the features provided by Box2D is automated physics calculations. In other words, the team did not have to concern itself how projectiles would be drawn midflight as they are affected by gravity. Box2D automatically handles these calculations assuming the input provides the vector of motion and the force at which the object was created. In terms of the jumping implementation, an upward vector is applied at a force which eventually becomes canceled out by gravity and brings the player back down to the floor.

Another important feature that Box2D provides is collision detection. Whenever two Box2D bodies collide, a signal is provided to the collision listener that takes in the two bodies that collided as inputs. By taking in the user data assigned to

each of the bodies, the engine can tell which bodies have collided. In the situation that the two bodies are “player” and “floor”, the system does nothing. However in the situation that the body “player” and the body “explosion” make contact, the system executes the logic that reduces the player’s health points by one, draws a shield around the player, and fires a socket.emit() to tell other players that this player has been hit. This example can be visualized in Figure 17 below.

Figure 17: Box2D Collision Detection

Box2D can also be used in situation to draw non-animating images falling off screen as a substitution to drawing animations in the standard JavaScript engine. This is done with the rain images by spawning rain objects at random locations on the top region of the screen.

5.1.3 Dependent Hardware and Software

As platform independence is a key component of HTML5, hardware and software dependencies from a client’s perspective are minimal. As long as a browser is properly equipped with HTML5 capabilities, it should have no issue running Hyper Bout so long

as it meets the very basic of hardware requirements specified in *3.5: Technology and Resource Requirements*.

That said, the visual appearances may vary from browser to browser. For development, Google *Chrome* was primarily used as our base platform to develop on. However, Hyper Bout should also be runnable in Mozilla Firefox and Internet Explorer as an HTML5 application. It turns out that the game both run in these two browsers, however they may appear slightly different. For instance, the text boxes in Mozilla Firefox may render in a slightly different location while the glowing animation for the boxes may not play in Internet Explorer. The core components are still there however that make the game playable by users on these three major browsers.

From the server's perspective, an installed version of Node.js is required to run the game. Compared to other server instances such as Apache's PHP server, Node is considerably easier to setup and configure and easier to develop for as JavaScript is still used to code the Node logic. With Apache, in addition to JavaScript to handle the page logic, PHP is still required to handle the back end logic to develop the elements on the page.

5.1.4 Implementation Dependencies

In terms of Hyper Bout's implementation, the system can be divided into three major components: (1) Lobby, (2) Networking, and (3) Game Logic. Of these components, the Lobby and Networking systems are tightly coupled and the Game Logic and Network subsystems are tightly coupled. In actuality, the Lobby and Game Logic do not do much in terms of interactions and can be treated as almost separate systems. Of

course, the lobby is still required to keep track of player names of each player and ensure that only four player can join a match.

In that sense, the networking subsystem is the glue that holds the project together and is the one critical subsystem that the implementation depends on.

5.2 Implementation of Developed Solutions

The following section reviews aspects of the project in where the developers have developed particular solutions for issues encountered during the design and development of Hyper Bout.

5.2.1 Hybrid Client/Server Approach

One of the challenging aspects faced during the development of Hyper Bout was whether the processing should be handled on the client or on the server. If the system ran on a "thin client, fat server" approach, the benefit would be that the clients could include low performance end devices that only had to take care of rendering images and animations. However, this would put more burden on the server so that in addition to listening for and routing messages to clients, it'd also need to perform the calculations.

On the other hand, a "fat client, thin server" approach was also considered so that processing was handled all locally on each client's machine. Of course this meant less burden on the server, but required each client to put in more effort with processing. This methodology was considered, however part of the definition of a "fat client, thin server" is that the client can run mostly independent of the server's functionality.

Table 9: Client-Server Model Types

Model Type	Local Storage	Local Processing
Thin Client, Fat Server	No	No
Fat Client, Thin Server	Yes	Yes
Hybrid	No	Yes

Since neither the "Thin Client, Fat Server" nor the "Fat Client, Thin Server" seemed appropriate for Hyper Bout, the development team relied on a hybrid approach. This hybrid approach still mostly resembles the "Fat Client, Thin Server" architecture, since Hyper Bout cannot function without the server. The processing is still mostly done on the client machines while the server simply acts as the messenger that simply receives messages and updates players on all other players' status. Traditionally more processing would occur on the server machine, however since the server is receiving at least 120 messages per second (i.e. 30 emits sent by each player a second), the socket listening and message emits to each player should already put enough load on the server as is.

A representation of this model can be seen in Figure 18 below. The figure below goes over the responsibilities that the client and server has as well as the main technologies that the two components run on top of. The responsibilities are not necessarily exclusive to either system, but these are the primary roles they provide service to.

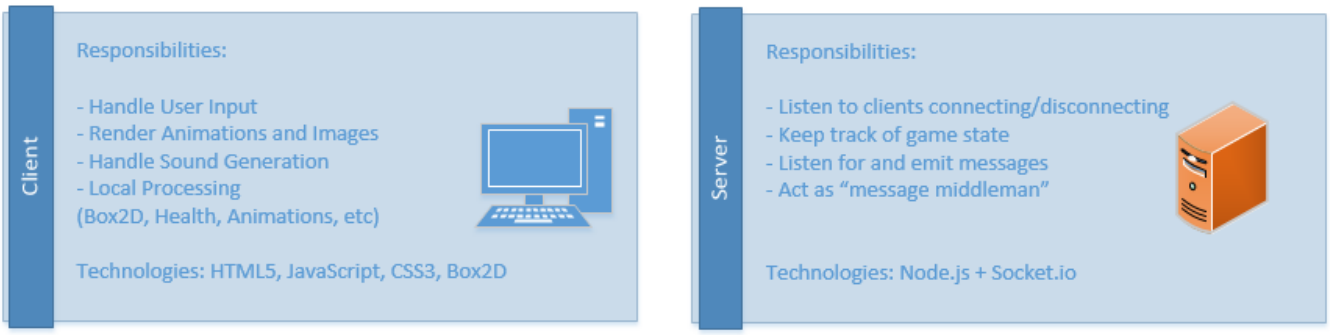


Figure 18: Hybrid Model Configuration

5.2.2 Box2D Integration with Node.js

Another challenging aspect was to see how the system could be developed so that Box2D objects could be appropriately sent over the network. In the initial phases of development, the team's primary goal was to develop a single player version of Hyper Bout. During this phase in development, multiplayer with Node and Socket.io was not taken into considerable account. When it came time to send Box2D objects over the network, the team initially sent over only a portion of the data that was needed to render the objects on the other clients' displays.

In order to have Box2D fully work, we needed to send the Box2D bodies and fixtures along with data such as the starting coordinates of the body and the force applied to that body. A Box2D body contains the data including the mass, velocity, location, and angle. Basically the Box2D body contains information that describes the properties of the object. A Box2D fixture describes some additional properties, but more importantly how the object should appear. It includes properties such a shape, restitution, friction, and density. By sending these two objects, Hyper Bout could finally render Box2D objects on other clients' screens.

In particular, this challenge came up when trying to get projectiles to match for each player. For projectiles, the Hyper Bout system utilizes a global array and every time a player clicks on the screen to throw a projectile, the projectile is added to that global array and the client provides a socket.emit() to tell other players to pull the most recent bomb from the array to display.

5.2.3 Separation of Client and Server Code

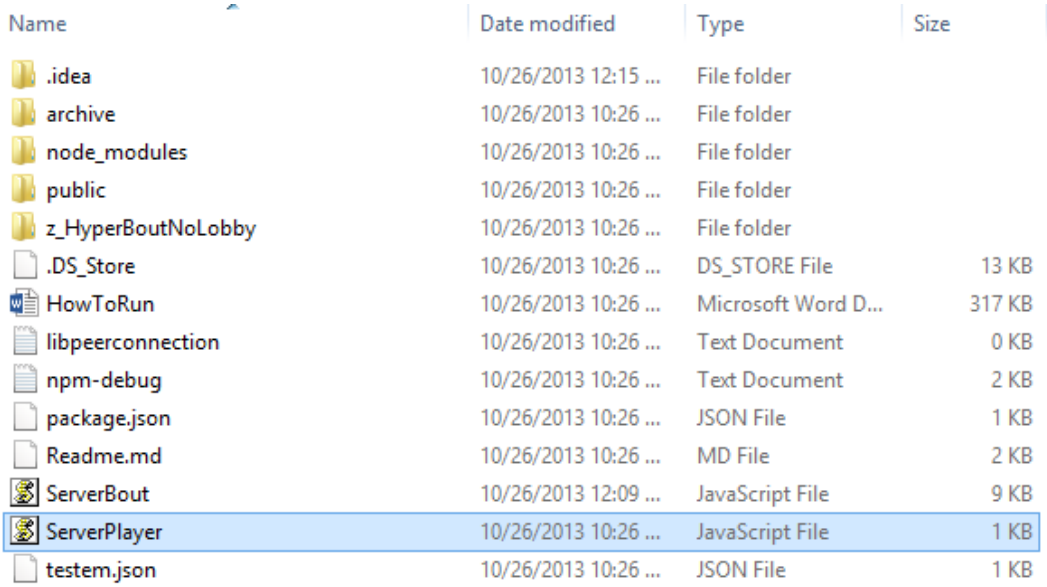


Figure 19: Server Code on Game Root and Client Code in Public Folder

Another concern that was brought up was how the server logic would be separated from the client logic. In Figure 19 above, it was determined that all server oriented files would be located in the game's root directory. In particular, three files/folders are required to run the server side code: (1) ServerBout, (2) ServerPlayer, and (3) the node_modules folder. ServerBout contains the main code to start up the node server while server player simply serves as a class to which data can be stored in for server side information about each of the players. The node_modules folders contain the socket.io library needed to communicate with client devices.

On the other hand, the client code is located within the “public” folder. This folder contains all the code and resources that would need to be provided on the client’s end.

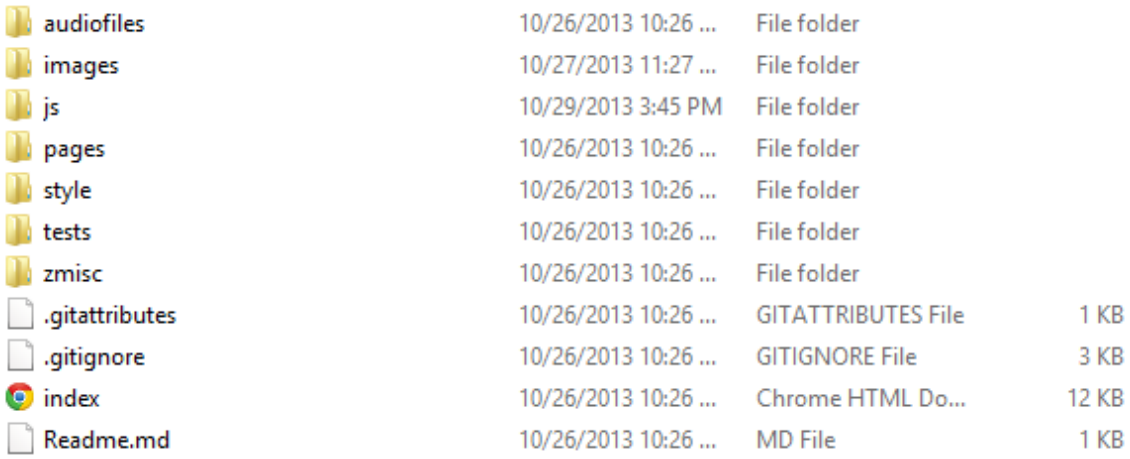


Figure 20: Layout of Client Side Code

The contents of each folder is as follows:

- **AudioFiles**: Contains BGM and sounds played during the duration of the game
- **Images**: Visual resources used that include player sprites, projectiles, and background images and animations
- **JS**: Includes JavaScript logic code that is required to connect to the Hyper Bout server and process game interactions
- **Pages**: Separate web pages (such as the “Help” page)
- **Style**: Includes CSS3 style sheets used for visual enhancement for pages
- **Tests**: Jasmine and Selenium tests run for the system
- **Index**: Primary HTML file to access and run the game.

5.3 Implementation Problems, Challenges, and Lesson Learned

The following sections review the challenges faced during the course of this project as well as the solutions found in addition to the ones mentioned in Section 5.2.

5.3.1 Implementation Problems and Challenges

Hosting

One of the bigger challenges of developing a network orient application is finding a suitable place to deploy the application. Typically most websites aren't heavily dependent on continuous dependable communication as most sites simply deliver resources to the user at one time and close the connection. This however does apply to a game as a continuous connection is needed and additionally messages are continuously being sent back and forth between the server and its clients.

In addition to this issue, the developers needed to find a hosting service that provided hosting of Node.js applications. The original plan was to use Google's app engine to host our site, but with realization that they do not support Node applications, the team had to look elsewhere. What was needed a hosting service that provided both a reliable connection and supported node applications. Additionally, the team looked for a hosting service that was free or at the very least provided a trial period so that the game could be at least tested. Nodejitsu seemed to provide these services so after some research, the Hyper Bout project was then hosted on Nodejitsu after some code refactoring.

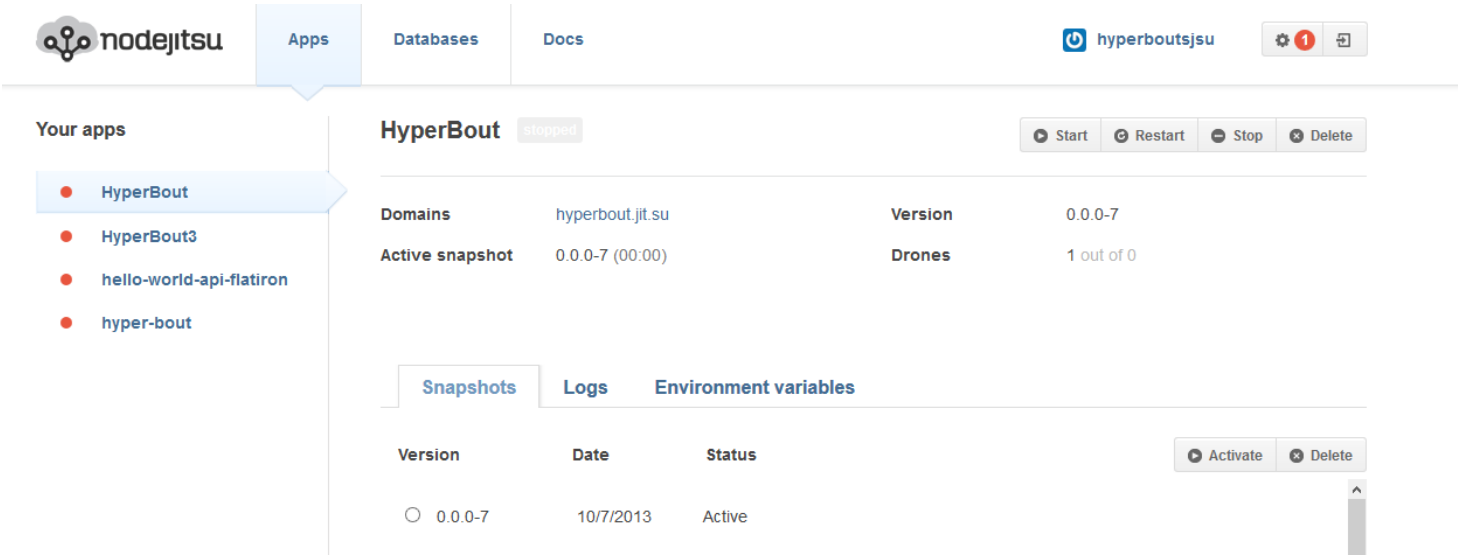


Figure 21: Nodejitsu Dashboard

While in theory Hyper Bout should run on nodejitsu with no hassle, it turned out that the connection to the nodejitsu servers were not reliable enough to handle loading the game as quickly as was planned. In addition, the connection at SJSU is unreliable at times so combining both situations would be unsuitable for a demonstration of the project.

As such, the team had developed a backup LAN configuration utilizing one system as the dedicated Hyper Bout server while the other four systems would act as clients. These systems would all be hooked to a router so that the clients can all connect to the local dedicated server and be able to play a game.

Player Position Mismatch

Another issue that was brought up during game development was that the position between players would sometimes be incorrect on two different screens. Generally the positioning wasn't drastically off, but still noticeable by a few pixels. The reason for this was that initially the client side prediction algorithm was implemented, but there was no check algorithm to update player positions.

Essentially Hyper Bout uses extrapolation CSP where the received packet provides the other character's position and its current force vector. Using these characteristics, the system assumes the player will stay in that direction with the same effects of the force occurring on the screen of player who sent out the position data. Of course, this form of CSP is only used to best predict what will happen on a player's screen and what the next data packet actually says in terms of position.

To fix the position mismatch issue, we run an update loop every 5 milliseconds that prompts the user position. At the same time while CSP algorithm is running, we update the player position every 5 millisecond interval so that the player positions are as accurate as they can possibly be. Of course this will heavily depend on every player's connection, however the CSP algorithm will better handle players with slower connections. The key behind the timing of the updates was to pick a time interval that didn't bombard the server with messages, but at the same time was often enough so that character movement was as accurate and "real-time" appearing as they could possibly be.

Testing

In addition to hosting and positioning issues, testing proved to be another challenge as testing a multiplayer game is much different than testing a traditional website with standard forms and buttons. For one, automated testing cannot be done as easily and the places it can be done are limited with features that closely resemble that of traditional website. For example, the lobby feature shares characteristics by being able to submit fields the chat box, being able to ready up by clicking the "ready up" button, and submitting your name to the lobby. These are all features that can be tested with automation. For lobby oriented features, the Selenium Testing framework was used to run the test suites.

Other parts that can be tested with automated tests are parts of the page that Hyper Bout should load on start up. Using the Jasmine Testing framework, the development team can ensure that elements have loaded to the page properly and that images and resources are all there.

However, when it came to the core gameplay, manual testing was required. Not only that, but typically manual testing was faster in the short term as automated test scripts did not have to be written up. Manual testing was required for features that required two or players live to play and test the movement and interaction between the two. That said, tests could still be run and often times the testing methodologies used helped the team catch lurking bugs. Hyper Bout's testing solution can be found in detailed more in depth in *Chapter 7: Testing and Experiment.*

5.3.2 Lessons Learned

As the developers have had limited experience with real time web development and game development in general, there were many lessons and techniques that were learned that could be useful in future projects.

For one, the project provided members experience with emerging web technologies, specifically HTML5 and Node.js. The team had picked up how Node works and learned the messaging model used with Node.js and Socket.io. At first it took the team a while to code up new server functions, but writing up a few server and client socket calls, developing the communication method between the server and client has become almost automatic.

Additionally the project allowed the team to develop optimizations for a system heavily dependent on performance. As networking was a critical aspect of this project, the team learned to keep network transmissions as minimal as possible and perform as much processing that can be done on the client end. The developers additionally learned how to keep game "lag" to a minimum by using client side prediction and by continuously updating the player positions.

Furthermore, the group gained experience with Box2D and learned how to utilize Box2D with the Node/Socket.io communication calls. This proved as one of the most challenging aspects of this project, but after the team got a solid grasp on the fundamentals of Box2D objects, more advanced features could be worked on such as health pickups.

Chapter 6 Tools and Standards

6.1 Tools Used

Listed below are software tools that were used throughout this project's lifecycle.

6.1.1 Design Tools

Microsoft Visio

Microsoft Visio is a software tool for drawing UML diagrams. Visio is a standard software that it used in the industry when it comes to the design phase. This tool allows the team to be able to draw diagrams and communicate the design of the project clearly to one another. Not only that, Microsoft Visio is also used to draw Pert Chart during the planning phase of the project.

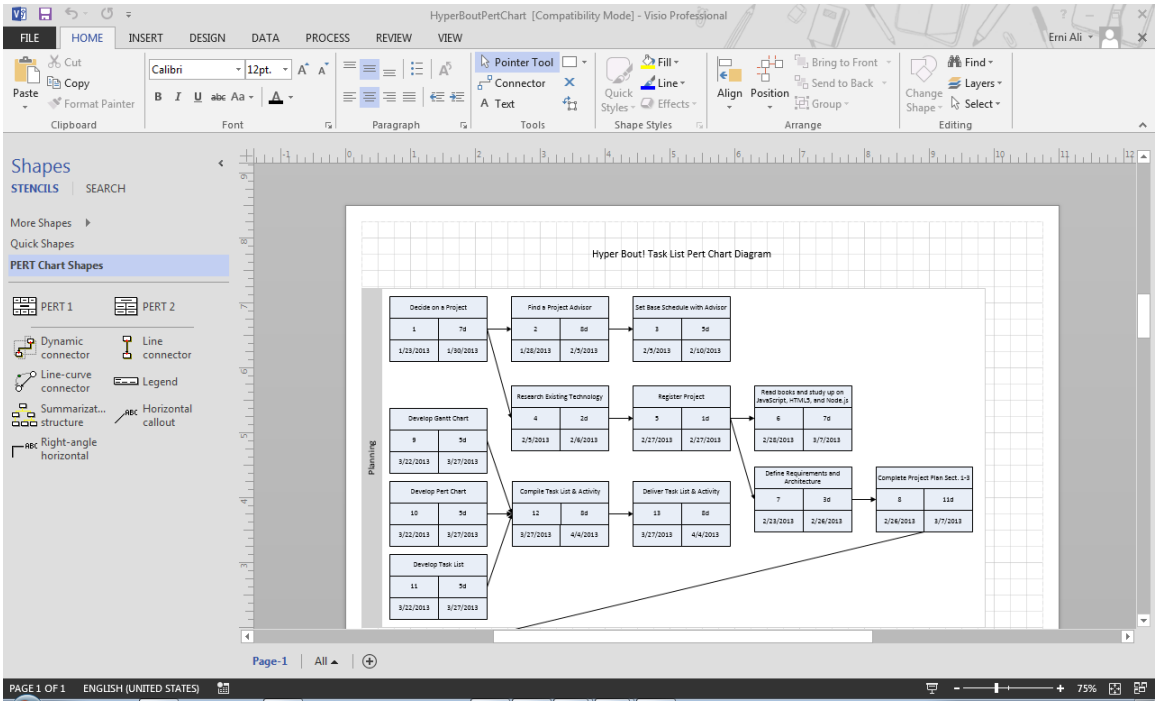


Figure 22: Microsoft Visio Interface

6.1.2 Management Tools

1. Microsoft Project

Microsoft Project was used as a management and planning tool for Hyper Bout. Microsoft Project is able to draw a detailed Gaant chart. This Gaant chart is very useful for calculating how much time it would take to do a certain task and how much time is left until the deadline and until the next milestone. Using the Gaant chart generated by Microsoft Project, the Hyper Bout team is able to meet deadlines in a timely manner.

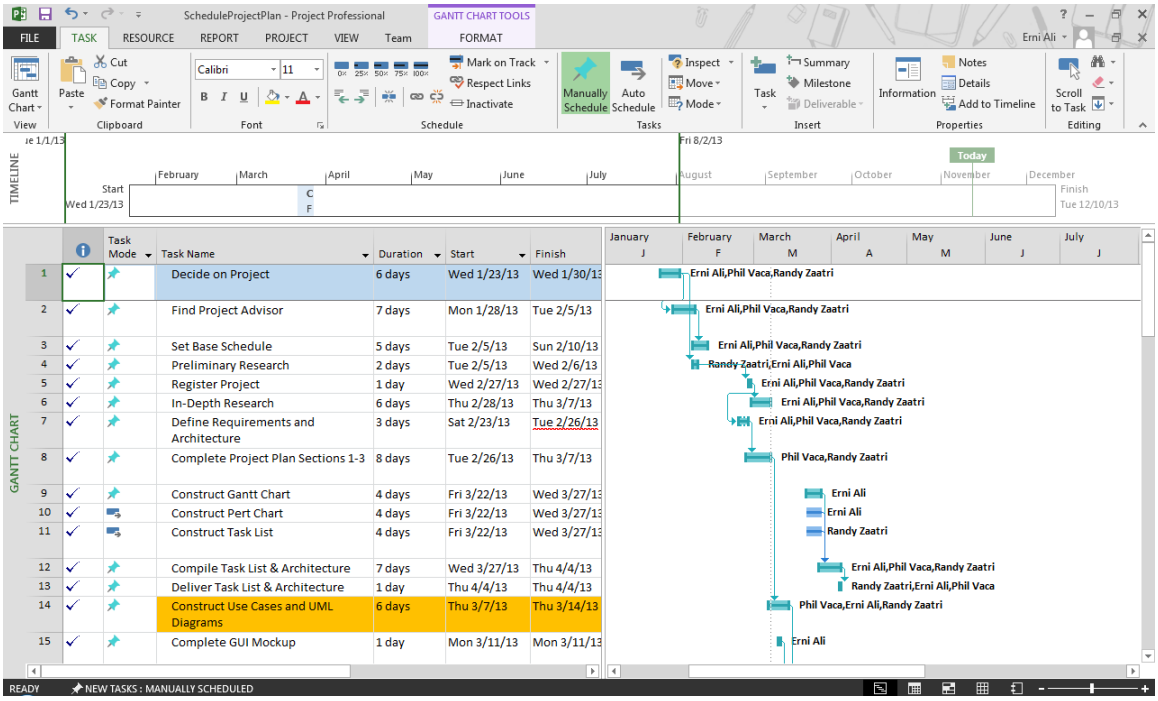


Figure 23: Microsoft Project Interface

2. Google Calendar and Google Hangout

Google Calendar is very useful as a management tool for the Hyper Bout team. Using Google Calendar, the team was able to schedule time for meetings with the adviser and also weekly meetings with the team. Using Google Calendar,

the team is able to find out when the next meeting is going to be because Google Calendar sends out notification a day before the event happens. Google Calendar is also used alongside with Google Hangout. The Hyper Bout team uses Google Hangout in order to hold online meeting and collaborate with one another in writing code and documentation and also look over one another's work.

6.1.3 Development Tools

1. Sublime Text 2

Sublime Text 2 is an open source text editor that is able to color code and detect JavaScript and HTML files. The reason why Sublime Text 2 was chosen as Hyper Bout's development text editor was because its ease of installation and use. The interface of this tool is also very slick and modern as shown in Figure 24. This tool is used in the industry in one of the author's internship.

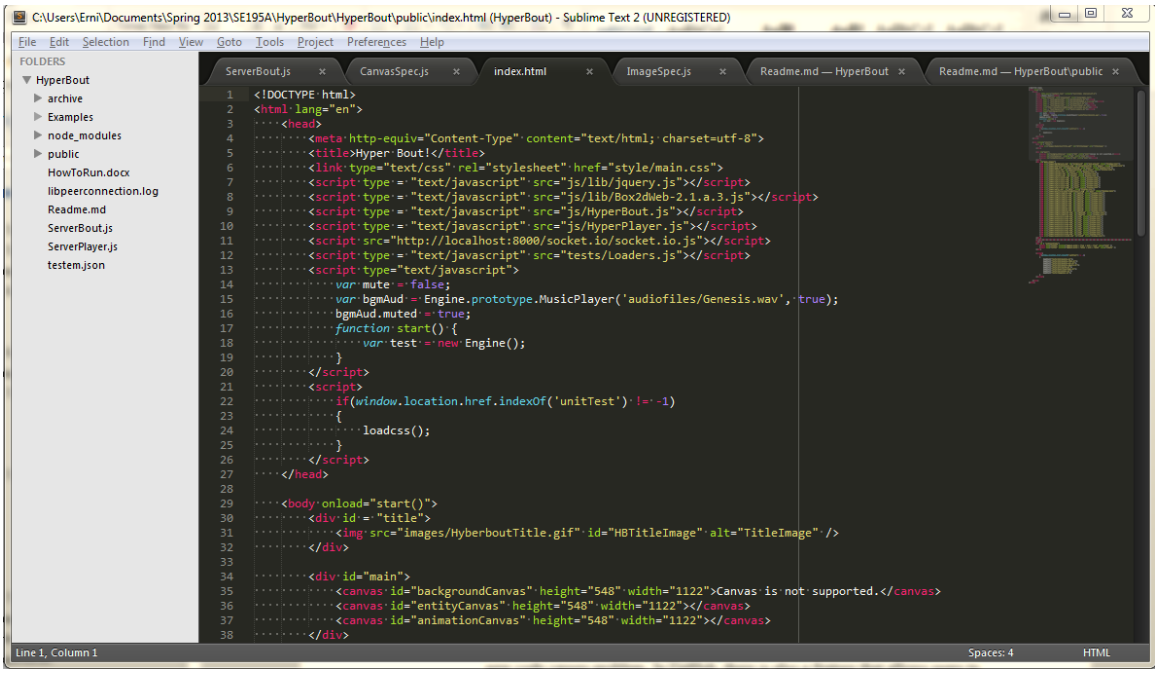


Figure 24: Sublime Text 2 Interface with Folder Browsing

As can be seen, if configured properly, Sublime Text 2 is able to show and display the contents of a whole folder. So, from the text editor, there is no need to open the files one by one. Instead, we can easily browser through the folders and the contents inside the folder from the left hand side of the editor as shown in the figure above.

If browsing through the folders are too confusing, there is also an option to search for files from the folder that Sublime Text 2 is opened in. So when a project gets really big and there are thousands of files involved, this feature saves the developer a lot of time. This feature can be accessed using Ctrl+P and the search menu will show up as shown in Figure 25.

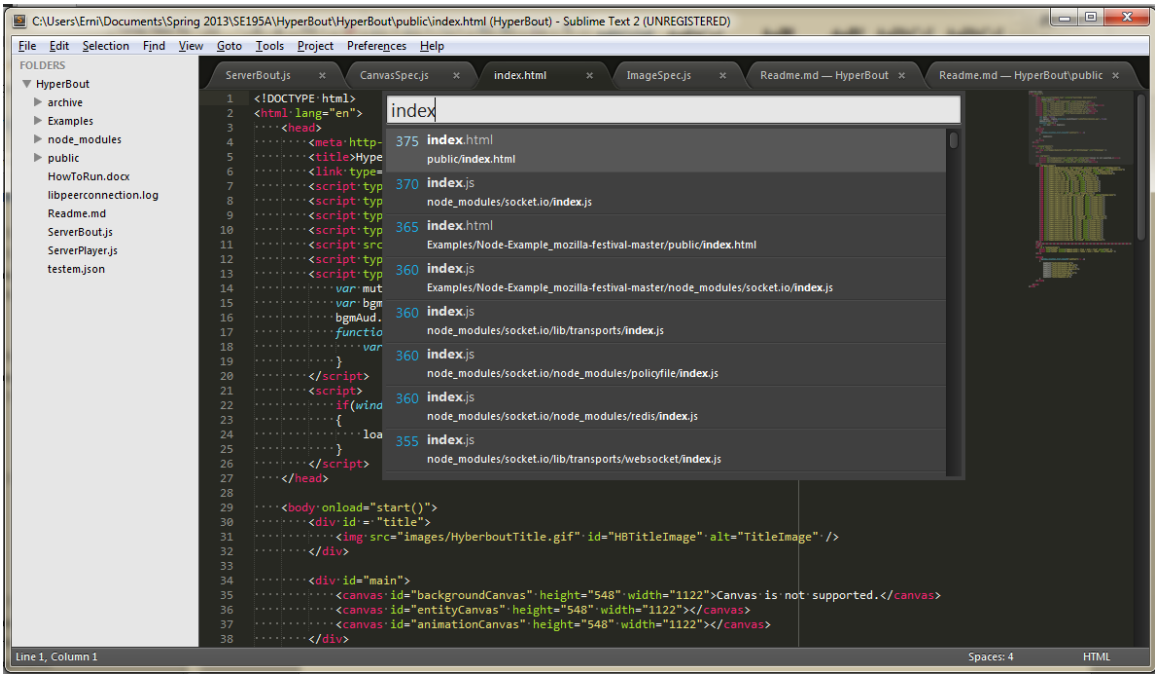


Figure 25: File Search Feature in Sublime Text 2

Sublime Text 2 is used throughout the whole development phase of Hyper Bout. We edited all of our JavaScript and HTML source codes using this tool.

2. Node.js command prompt

Node.js is used to handle the networking aspect of Hyper Bout. Since development was done on windows operating system, the Node.js command line tool was really useful to start the server side script of Hyper Bout. It also requires less time to set up than having to use the windows command prompt or the Cygwin environment on Windows.

In order to start the server side script of Hyper Bout using the Node.js command prompt, all that was needed to be done was to browse to the root directory of Hyper Bout and type in "node ServerBout.js" to run the script as shown in Figure 26.

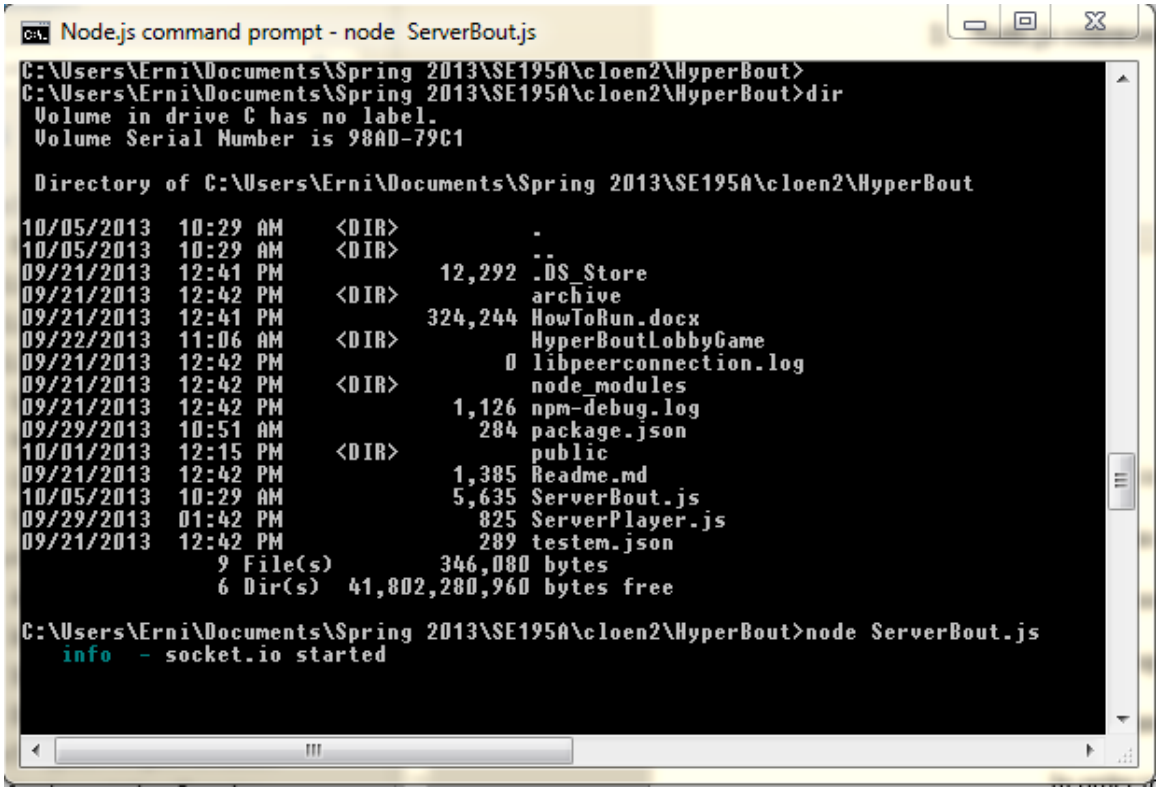


Figure 26: Running Hyper Bout Server Script using Node.js Command Prompt

3. GitHub Repository

GitHub is the source control tool used to share source code. The GitHub public repository is completely free. However, this means that anyone can easily pull our source code and make their own changes. However, without permission, they will not be able to change the authors' Hyper Bout repository. GitHub is easy to set up. An account needs to be created and each user needs to be given permission to push and pull to the repository. GitHub is also the industry standard. Therefore, learning this skill will be useful for the authors' future job in the industry. The figure below shows Hyper Bout's GitHub repository.

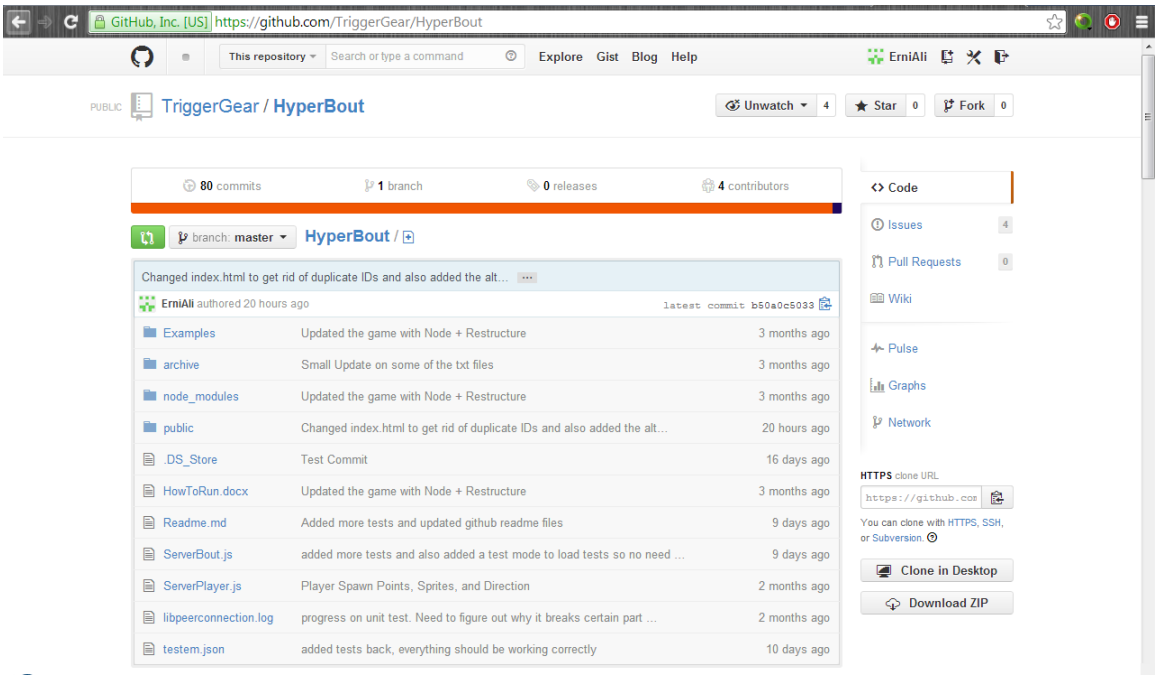


Figure 27: Hyper Bout GitHub Repository

GitHub also offers the ability to roll back to the previous version of the code. So, if for example, a new feature breaks the system, the authors will be able to roll back and erase that changes that were made. That way, not a lot of time

will be spent to fix a defect. There is also a blame feature to find out whose code is responsible for breaking certain aspect for a system. This is useful when the team working on the project is so large that it is difficult to find out who is responsible for which part of the code.

Not only that, the GithHub repository allows for a markdown document that will essentially let the users describe the GitHub repository in an easy to read manner as shown in Figure 28 below.

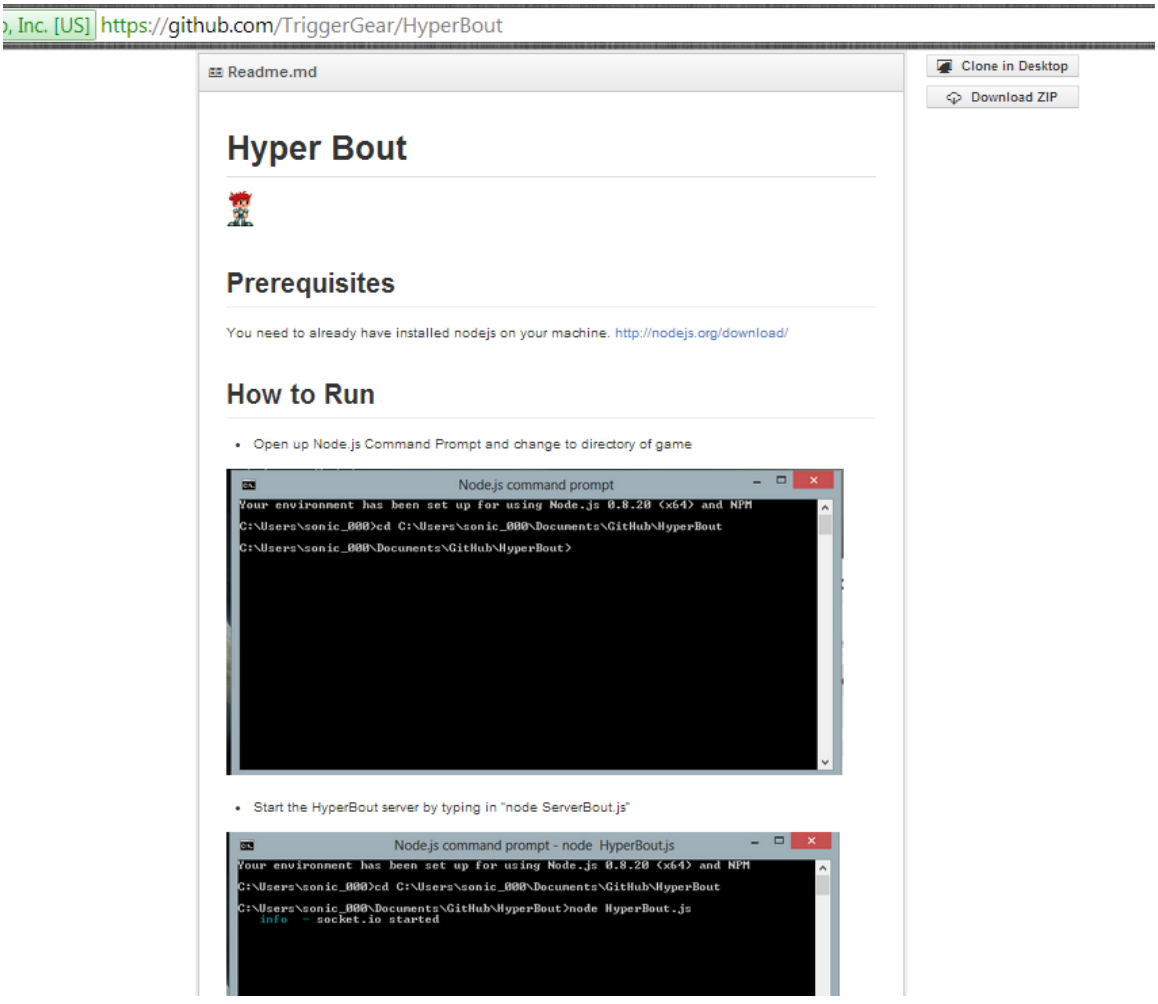


Figure 28: Hyper Bout ReadMe

4. Git Shell and GitHub GUI

Github allows easy pulling and pushing of source code from the Git Shell or from the GitHub graphical user interface. The Git Shell requires some knowledge of Git command line instructions to push and pull code as shown in Figure 29.

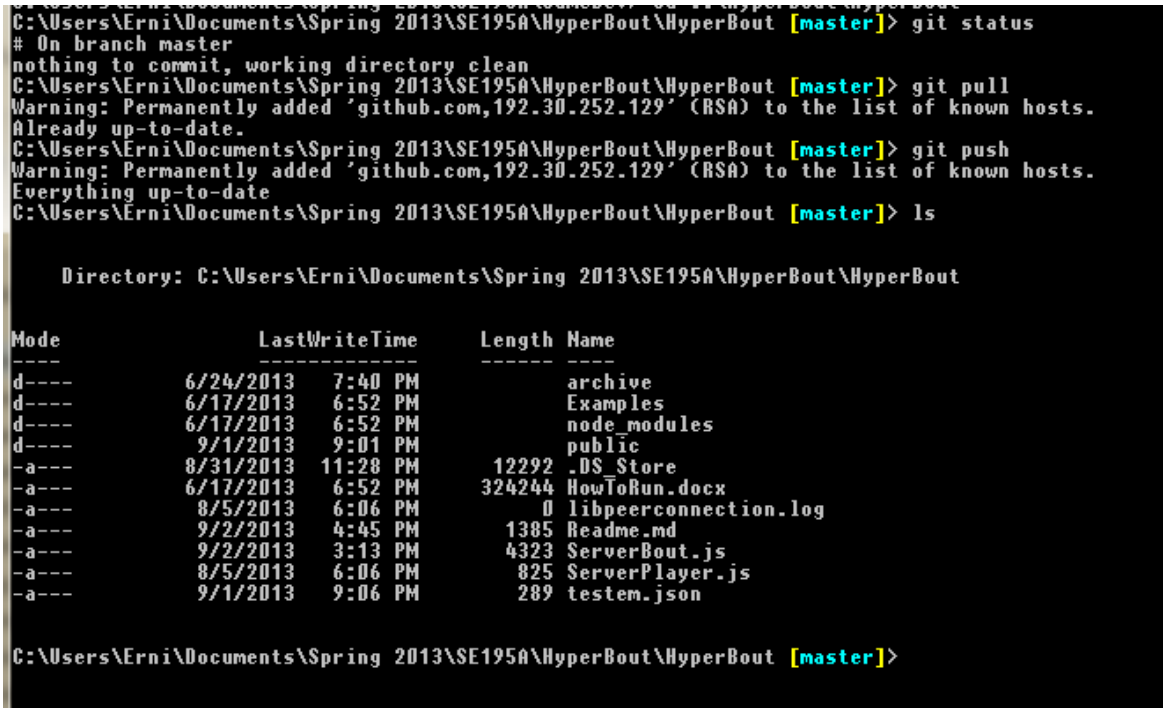


Figure 29: Git Shell Command Line tool

However, the GitHub GUI provides an easy to understand user interface that allows pulling and pushing code without a steep learning curve. The Git Shell and the GUI interface are the two tools that the authors used when dealing with GitHub.

5. Web Browser

Since Hyper Bout is a web based game, web browsers are crucial for the development of this project. The authors use the web browsers in order to test and to cross check compatibilities, such as on Firefox and Chrome. Google Chrome particularly, offers a very sophisticated debugger for JavaScript that allows the setting of breakpoints and finding out the different values of a variable at any given time as shown in Figure 30.

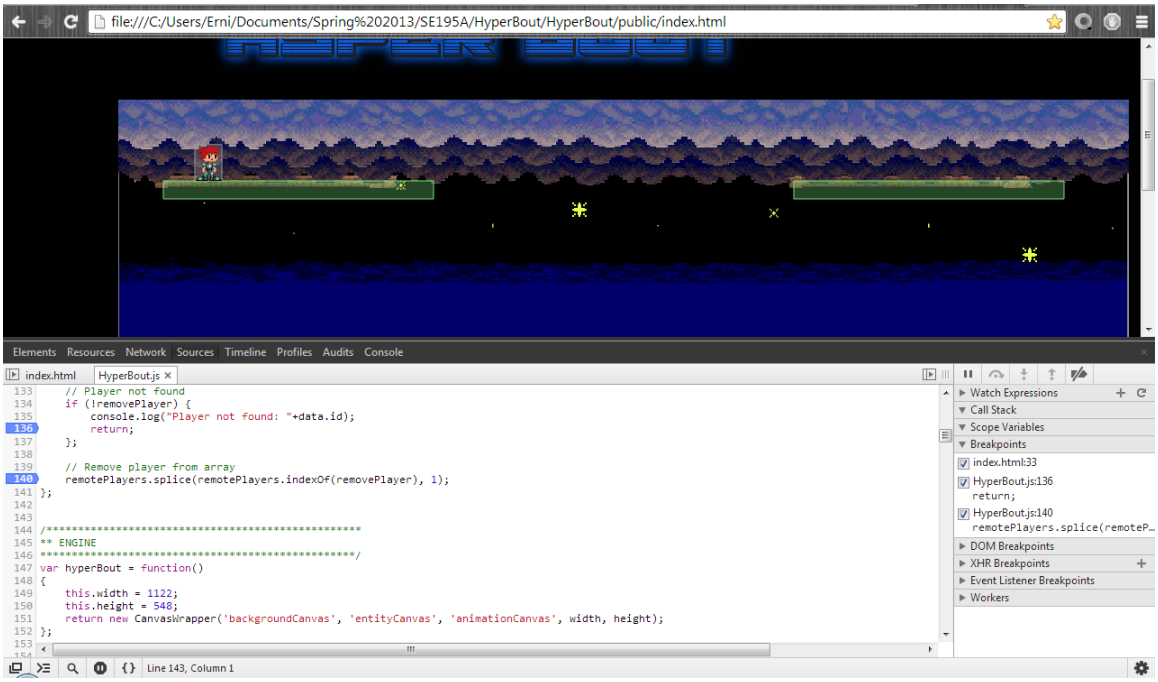


Figure 30: Google Chrome Debugger

Not only that, Google Chrome also allows console logging which lets developers log messages for debugging. Google Chrome developer tool gives the authors an easier time when defects are found and debugging is necessary.

6.2 Standards

The following section describes the standards that are used in the Hyper Bout project.

6.2.1 Behavior Driven Development

Behavior Driven Development or BDD is an addition to Test Driven Development. Instead of following TDD standard where testers are required to write unit test for "something that doesn't exist yet", BDD seeks to solve this problem by "writing tests for specifying behavior" (Helm, 2013). This means that unit tests should be written in a way that specifies how users think that the system should behave.

By writing unit tests in such a way, a project can then focus on the most integral and important feature for the users. This forces developers to think in the shoes of the users which will increase the quality of the product or the project because in the end, the end users are the consumers who will be using the product. The most emphasized standard in BDD is that the syntax is human readable unlike the syntax in TDD unit tests. Using BDD, both the users and also developers or teammates will be able to know what a unit test is about without having to know how to write unit tests themselves. Figure 31 shows an example of Jasmine BDD syntax.

```
describe("Hyper Player", function(){
    var player;
    beforeEach (function(){
        player = new HyperPlayer();
        player.setX(68);
        player.setY(68);
    });

    it("should be able to set position X correctly", function(){
        player.setX(89);
        expect(player.getX()).toEqual(89);
    });

    it("should be able to get position X correctly", function(){
        expect(player.getX()).toEqual(68);
    });

    it("should be able to set position Y correctly", function(){
        player.setY(89);
        expect(player.getY()).toEqual(89);
    });

    it("should be able to get position Y correctly", function(){
        expect(player.getY()).toEqual(68);
    });
});
```

Figure 31: Jasmine BDD human-readable syntax

As shown from Figure 31 above, an individual with no coding experience will be able to know that the test spec is specifying the behavior of a player in Hyper Bout with the various test spec. For example, a player, in the component function, should be able to set and get their X and Y positions correctly. The syntax of BDD is very descriptive and when a test fails, it makes it easy to know which functions fail. BDD also serves as a regression test when run later on after more features are added into the system (Helm, 2013).

6.2.2 W3C Standards for HTML5

Hyper Bout conforms to the World Wide Web Consortium (W3C) HTML5 standards. The W3C is known as the association who sets standards for the World Wide

Web and it also works on making the web accessible to everyone (W3schools, 2013). The World Wide Web Consortium offers a very simple service that enables users to check if their HTML5 documents conform to the standards in the W3C validator. The webpage of the validator is shown in Figure 32.

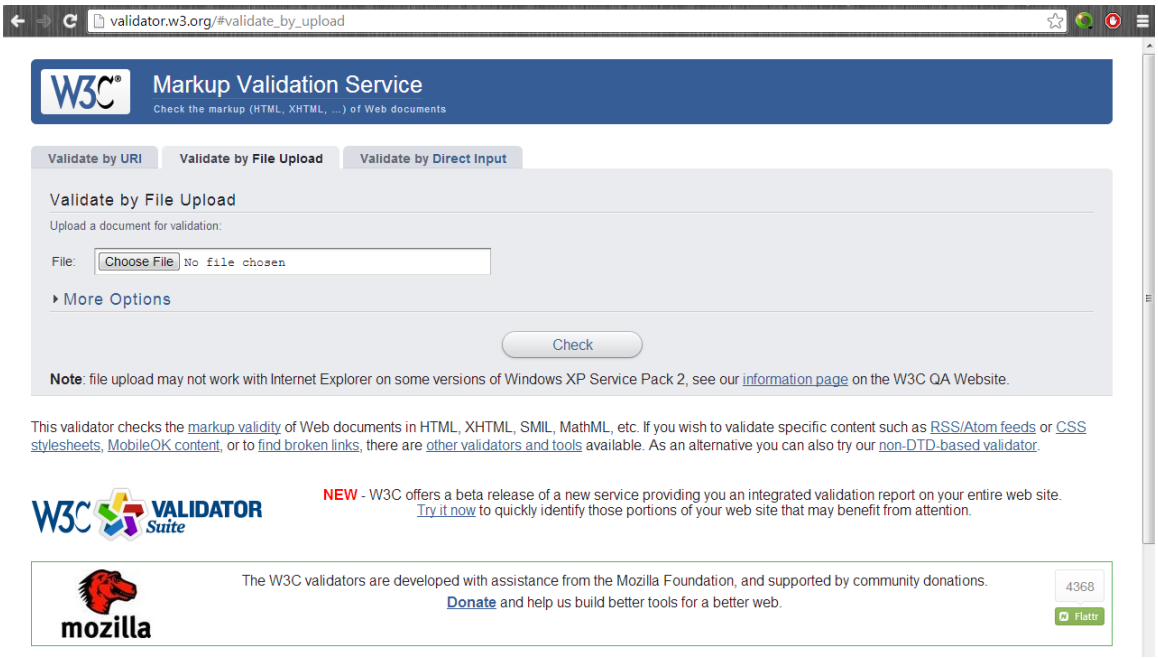


Figure 32: W3C Validator

There are several ways to validate an HTML5 document in the validator, validate by URI where users can enter the URI where their webpage is hosted, validate by file upload where users can upload the HTML5 file that they want to validate, and validate by direct input where users will be able to easily copy and paste their entire HTML5 document and get it validated.

In this case, the authors are going to use validate by file upload for Hyper Bout. Before conforming to the standard, there are a total of 37 errors as shown in Figure 33.

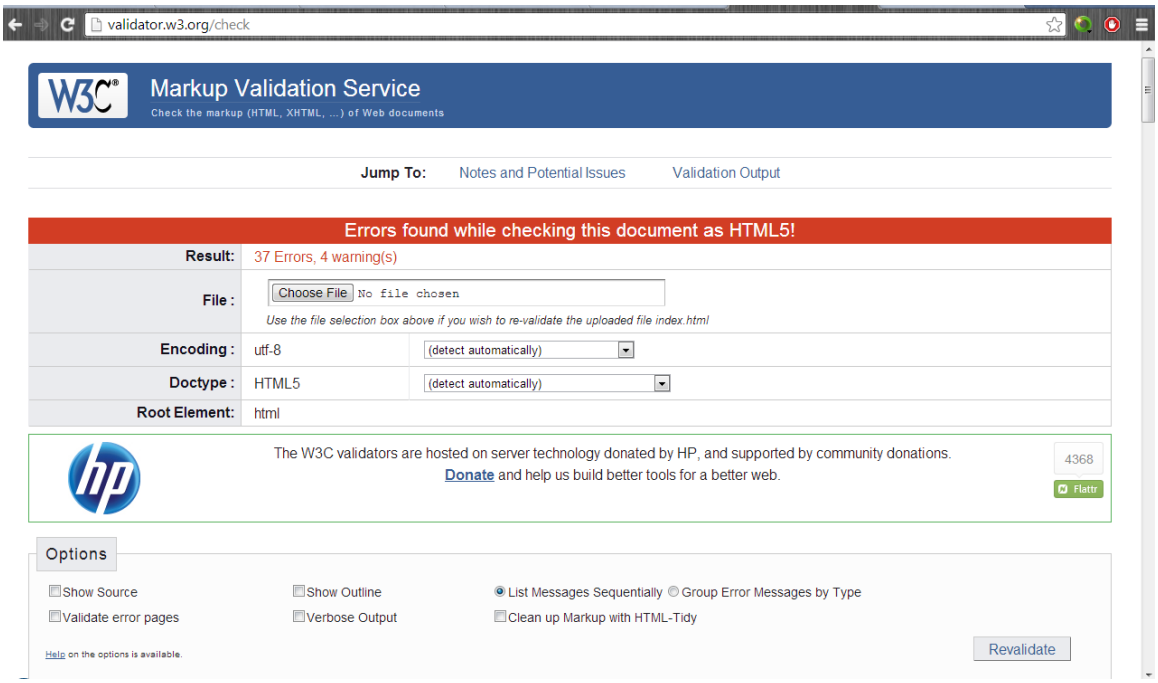


Figure 33: Hyper Bout HTML5 Document Check before Conforming to Standards

There errors are mostly errors on not having a certain attribute that are required by standard. For example, the image tag of HTML5 requires the alt attribute to describe the image so that if the image doesn't show on certain web browsers or if the image leads to a broken link, the description will be there and it will be easier to debug and users will also know what should be showing on the area where the image should be.

Therefore, this standard is also useful for debugging. From the example above, if there are a lot of images in a website, then the alt attribute will help in finding out which image links are broken based on the description of the image. After fixing Hyper Bout's HTML5 document, there are no more errors with one warning that basically says that HTML5 is still in development and the standards may still change as shown in Figure 34.

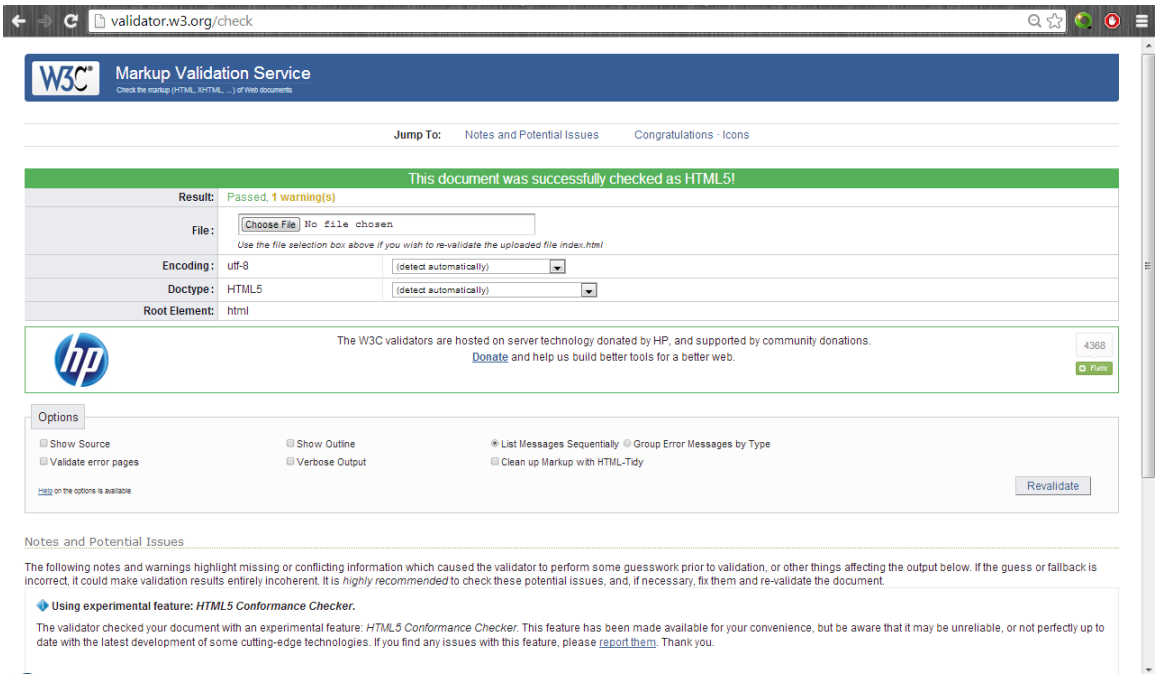


Figure 34: Hyper Bout HTML5 Document Check after Conforming to Standards

Therefore, with no errors, Hyper Bout has conformed to the W3C standards of HTML5.

Chapter 7 Testing and Experiment

7.1 Testing and Experiment Scope

Hyper Bout's test process, focus and objectives, and test criteria are described in this section. Since Hyper Bout is an asynchronous web based multiplayer game, manual testing was heavily relied upon in order to ensure the game's quality. Parts that could use automated testing using Selenium and Jasmine have been tested to save time, yet a good portion of tests still required manual testing.

7.1.1 Test Process

The test process in Hyper Bout involved defining test cases, performing manual and smoke testing, unit testing, and performance testing. If these tests result in a failure and therefore a bug, the bug would be reported and resolved by the developers. This process can be visualized using the chart below.

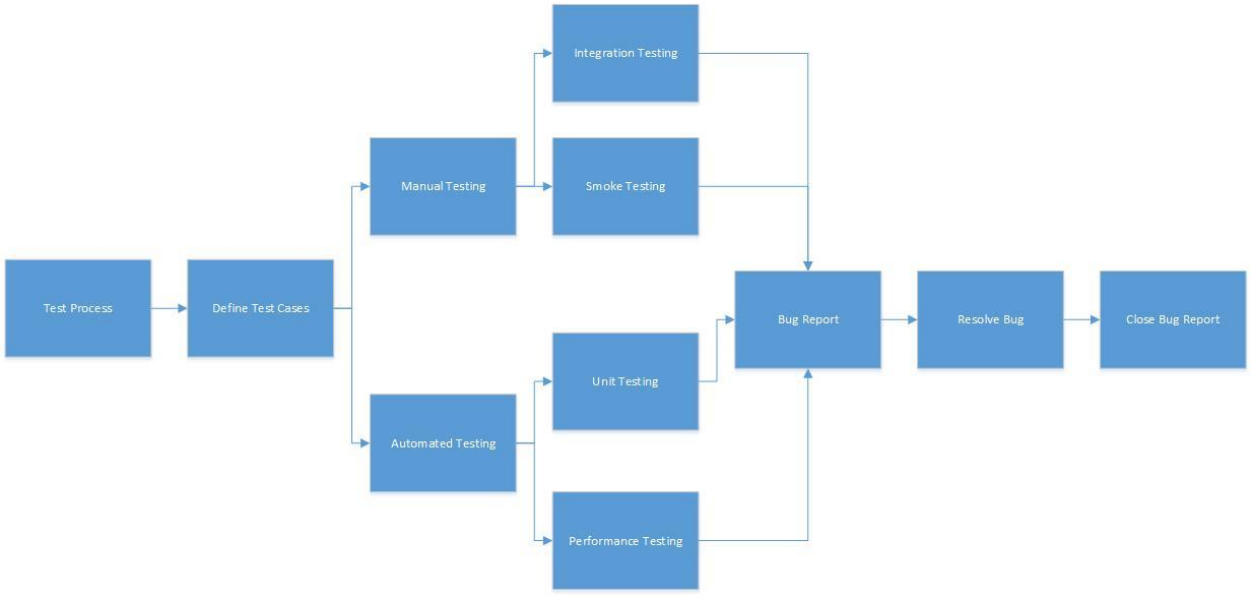


Figure 35: Hyper Bout Test Process

Before test cases could be defined, a test framework needed to be implemented into the project. For one, the project required a bug reporting tool of some kind. It turned out that the Hyper Bout GitHub repository not only functioned as a version control tool, but also as a bug reporting tool. The GitHub repository provides an easy to use tool that allowed members of the team to report defects that were found throughout the testing period of the project. Other than reporting defects, the project members also have the ability to assign the defect to the developers that have access to the repository. More details about the bug reporting tool in GitHub repository will be discussed in a later section.

In addition to bug tracking, test cases were defined using the Hyper Bout test design specification. These tests cases were written to perform manual testing and smoke testing to check the overall usability of the system. To perform tests, testers referenced the test cases, which gave them information on what to test and what to verify.

These manual tests also included integration tests that were performed whenever the Hyper Bout developers integrated parts of the code together. During code integration stage, functions that previously worked would sometimes break due to conflicts with previous code. Therefore, this integration testing was performed to ensure that the code integrated well together and that the previous and new functionalities worked.

To have a less tedious manual testing process, a unit testing framework was implemented into Hyper Bout. These unit tests performed white box tests for the system. Jasmine was chosen as Hyper Bout's unit test framework. Jasmine is a Javascript unit test framework that checks whether specific functions are behaving as they should or if

certain components in Hyper Bout still perform correctly after new features are added. These Jasmine unit tests helped prevent and detect regressions in Hyper Bout.

Lastly, before the project was concluded, performance testing was performed in order to ensure that Hyper Bout would be able to achieve adequate responsiveness and stability. For example, the amount of time it will take for the client and the server to process the player's input. A ping server test was also conducted to test a network connection and if network communication was already established. Ping tests return the time that it takes for the server to respond (Mitchell, 2013). From this kind of test, Hyper Bout's server delay and latency with the clients can be measured.

7.1.2 Test Focuses and Objectives

Hyper Bout tests were written and designed to focus on the overall performance and usability of the system. The ultimate objective was to present an entertaining game to the end user. To achieve this, Hyper Bout needed to run as smoothly as possible with little to no delay, conform to the requirements, and be defect free for its major functions.

The major focus of Hyper Bout tests was for its networking capabilities to be seamless. Networking in Hyper Bout should not be a feature that hinders end users from playing. For example, it should be easy for users to be able to set up connections and play a game with their friends.

In addition to networking, handling game state changes is important, so that games can easily start and terminate. For example, end users should be able to start a game and end the game once they have gathered enough points. Winners of the game should be announced and there should be an exit criteria to the game.

Another important objective was also to ensure that players can interact with one another via the game environment. There would be no point to a competitive game if players merely sit inside the game without the ability to attack or defend. Additionally the game had to ensure that positions and collision detection was handled appropriately across each player's screen. Additionally to enhance the playability and enjoyment of Hyper Bout, we wished for players to be able to run their games at a smooth 30 frames per second.

As for the objectives, due to time constraints and limited resources, the tests had to be run in a way that was both thorough and at the same time not as time consuming. Even though manual testing is a tedious task, it is important to find defects that cannot be normally found through automated testing. Furthermore, manual testing is a faster method to find defects in the short term. For example, if new features were added, having a team member manually test the new feature would allow finding defects to be faster than having to write automated tests that would have taken some time. By finding defects at a faster rate, developers were able to fix the defects before they have forgotten about the feature that they have implemented and have to re-learn the code.

In order to perform manual testing, test cases needed to be written so that the manual testers knew which aspect of the system they have to test to achieve a thorough and complete manual test.

Smoke testing was conducted to have a quick test of the overall system to test usability and to find any glaring defects. Smoke testing was also used to test the simplest features of the system, for example if the game loads or not or if the player or characters show up on the game canvas. If there were defects, then that would mean that more

thorough manual testing needed to be performed to find out if some features were broken in the system. Smoke testing takes less time and was an easy way to see if the system was still working fine after new features were implemented.

Integration testing was performed to ensure that the system integrated together perfectly with new parts of code. This type of testing was done during Hyper Bout's development phase when different components of the system were put together. For example, developer A finished writing the code to shoot projectile and developer B also met the deadline to implement character animation into the game. Together, developer A and B needed to integrate the code together and made sure they worked by performing integration testing. To avoid issues during integration, bottom-up testing was utilized where the lowest level components were tested first so that problems can be found easily from the smallest components.

Automated testing was implemented to reduce the tedious and repetitive task of manual testing. This way, regressions can be found easily. Regressions are cases where features that were previously working are no longer working because of new features that were recently implemented. The faster regressions were found, the sooner it could be fixed before the problems were escalated further. This would have caused the developers to waste a lot of time tracking down the cause of the defect.

Unit tests in Hyper Bout helped find these regressions and reduce the amount of tests manual testers have to perform. However, this situation came at the price that the unit tests had to keep being updated. Unit tests performed basic smoke test, such as if the canvas element is there when user loads the game or if some function calls return the correct value.

Performance testing was the final testing phase for Hyper Bout. This test focused on finding the performance of the Hyper Bout server and its clients. The objective of this test was to ensure that the end users will be able to play a smooth game with no delay and even if there were delays, it should be unnoticeable by the users. This test also ensured that interactions between players would happen in real time or almost real time. Hyper Bout's performance test mostly focused on its networking feature.

7.1.3 Test Tools

1. GitHub Issue Reporting System

The project used GitHub as its source control system. GitHub is the modern industry standard for source control. It allows the authors to share and modify code easily. Not only that, it gives developers the ability to rollback code in cases new code causes problems. In GitHub, there is also a feature that allows users to report issues. This is the feature that the team used for reporting the bugs found in Hyper Bout.

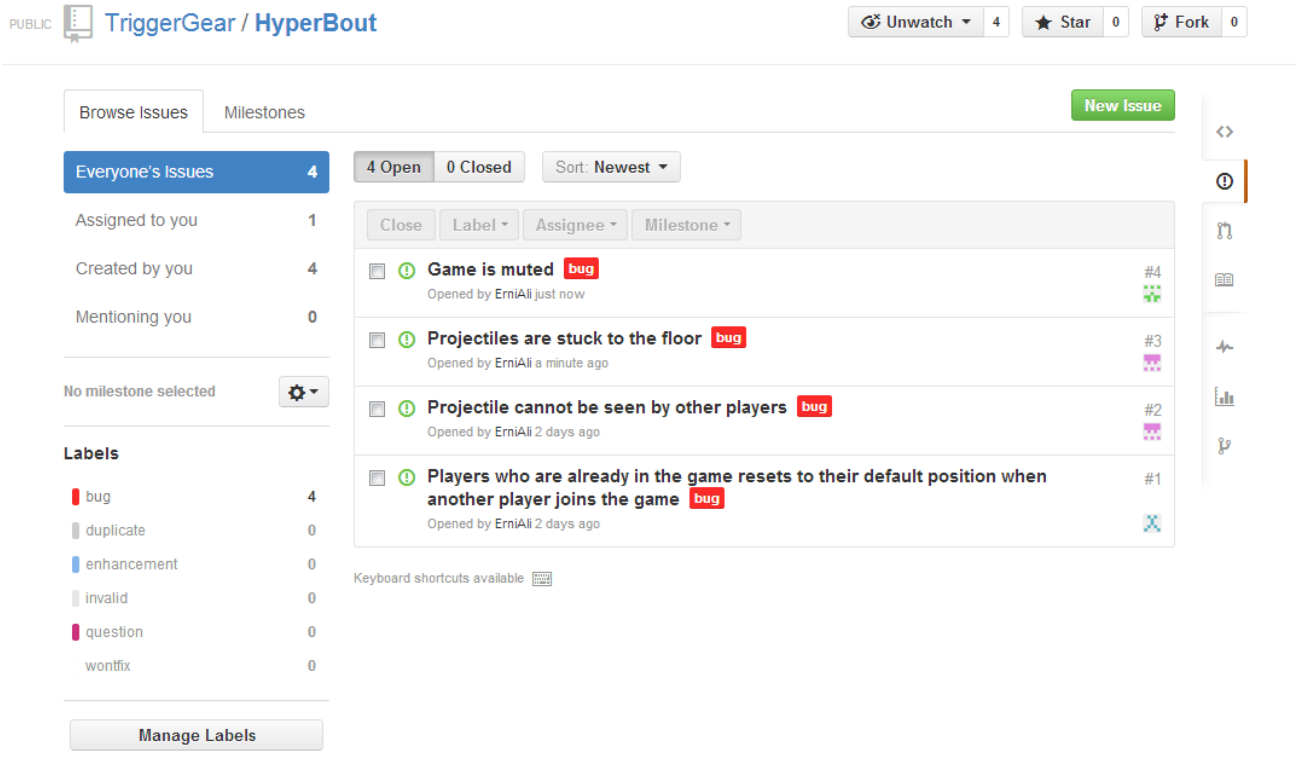


Figure 36: Hyper Bout Bug Reporting System

The GitHub issue reporting system can be found under the GitHub repository webpage. The issue page, as shown on the previous page, displays all the issues that have been reported so far. It also displays the icon of the user that is assigned to fix that issue. The user can then go in and change the status of the defect or the issue when they have already submitted a fix. On the other hand, it is also possible for the developer to mark the issue as invalid as "wontfix".

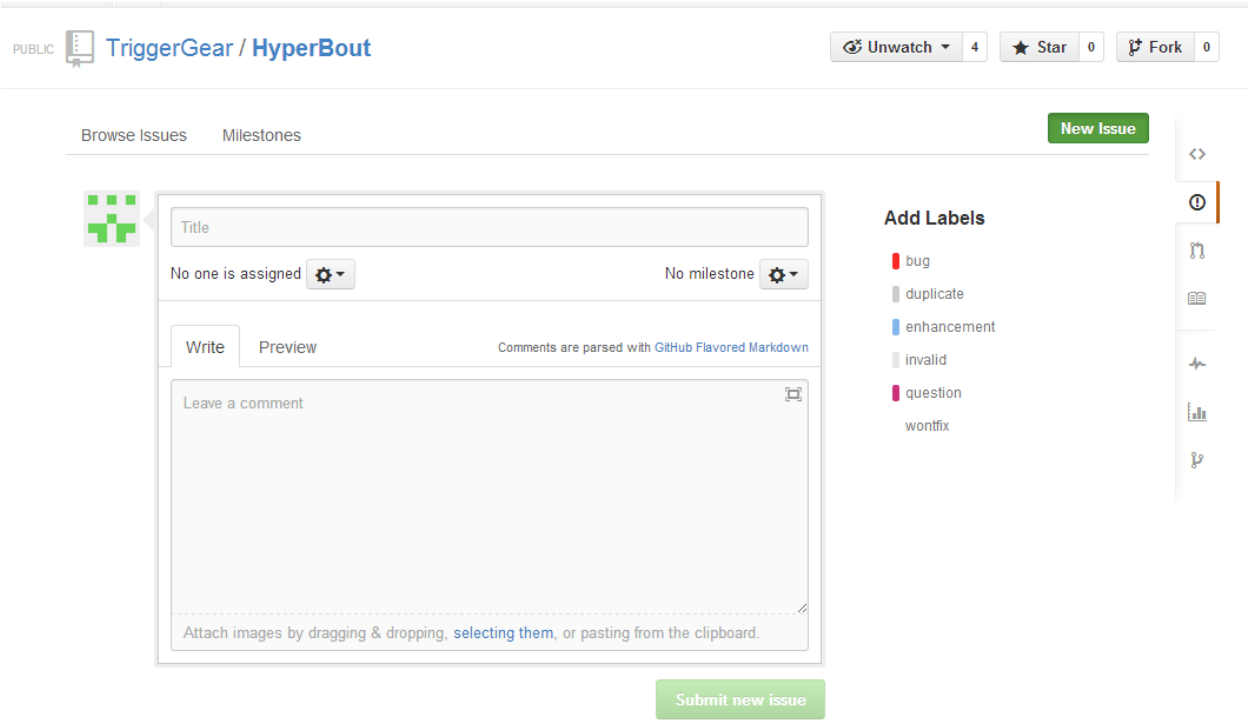


Figure 37: Submitting a new issue/bug

Submitting a new issue can be easily done using this GitHub feature. All the user needs to do is click the new issue button and a form will be displayed with clearly labeled fields as shown in Figure 38.

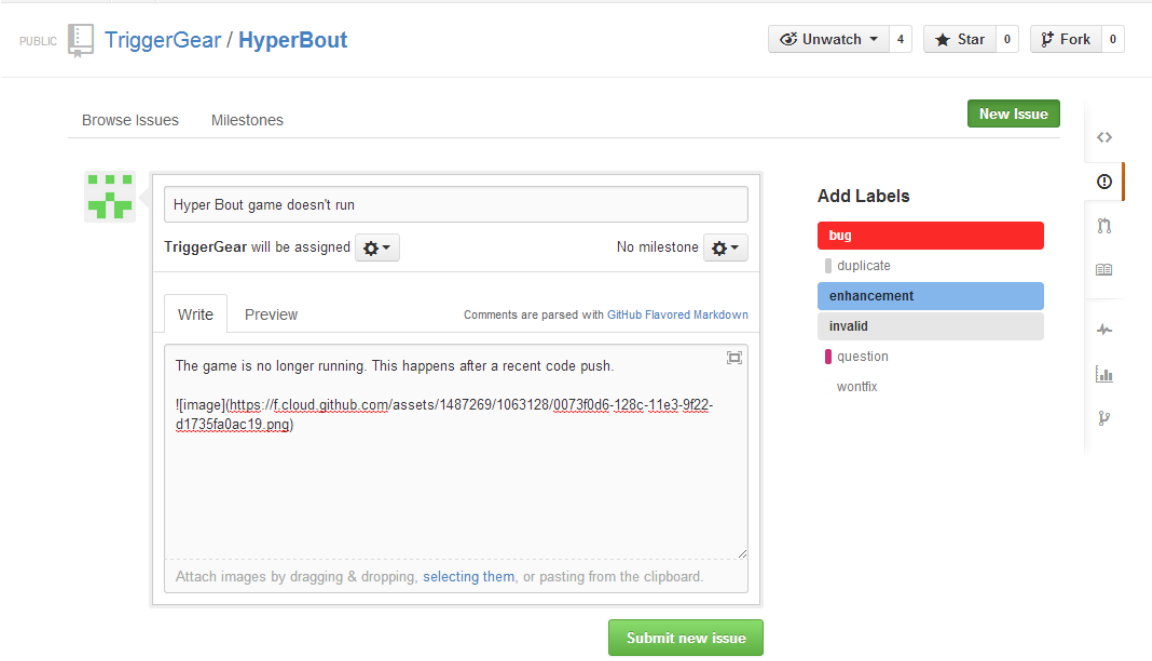


Figure 38: Filled in details of the bug report on GitHub

To submit an issue, the user would fill in the title and the details of the issue, assign it to a developer and click “Submit new issue” so that the issue would be submitted. Labels can also be added, for example, clicking on the “bug” label under “Add Labels” would put the issue under bug. Multiple labels can also be applied, as shown in the screenshot in Figure 39. An image can also be attached along with the bug report. After submitting the new issue, it will appear as the screenshot below with all the selected labels, the assigned developer, and also the image.

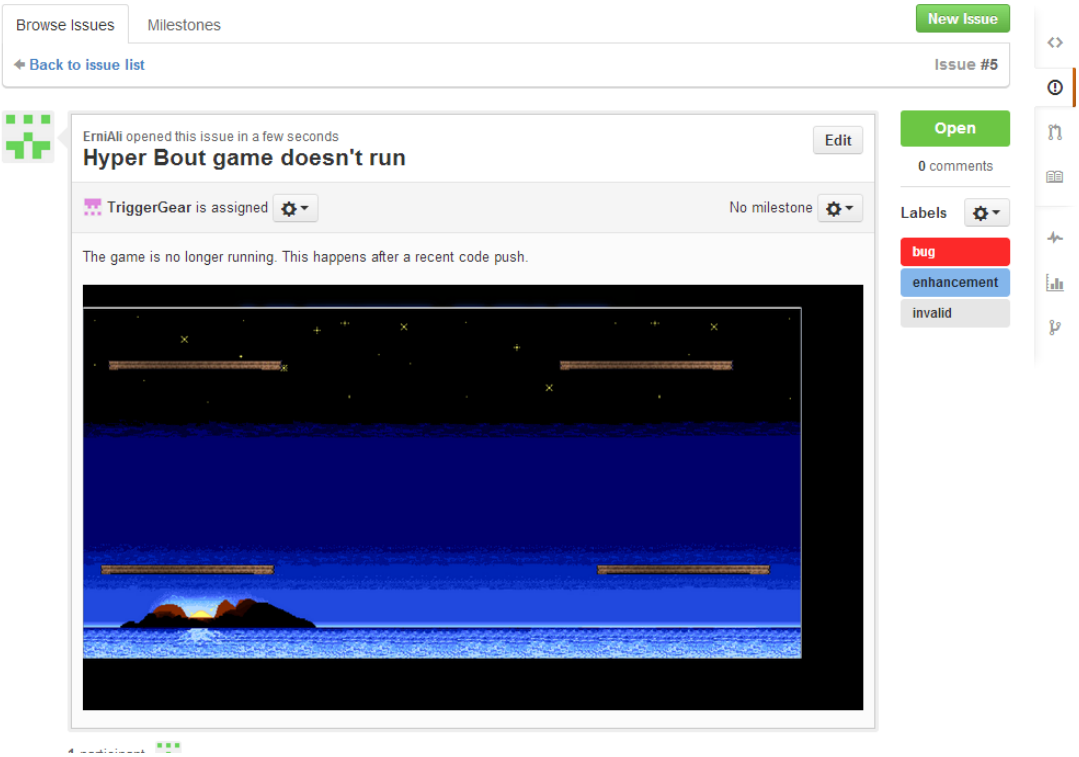


Figure 39: Submitted issue

This tool is used throughout the testing phase of the project. Whenever an issue is found, the individual who found the problem will post a new issue to the GitHub Issue Reporting System. Not only that, after the issue has been posted, if the individual responsible for the issue already fixed the issue, the issue can be closed.

2. Jasmine Testing Framework

Another tool that was used for testing is the Jasmine framework. Taken from the Jasmine documentation, "Jasmine is a behavior-driven development framework for testing JavaScript code" (Pivotal Labs, 2013). The reason why Jasmine is chosen as our testing framework is because Jasmine has a very easy to understand and readable syntax that will allow even non-programmers to be able to understand the tests that are written. This is a unique feature of behavior-driven development where tests are written in a readable way.

A sample test for Hyper Bout written in Jasmine can be seen in the screenshot below.

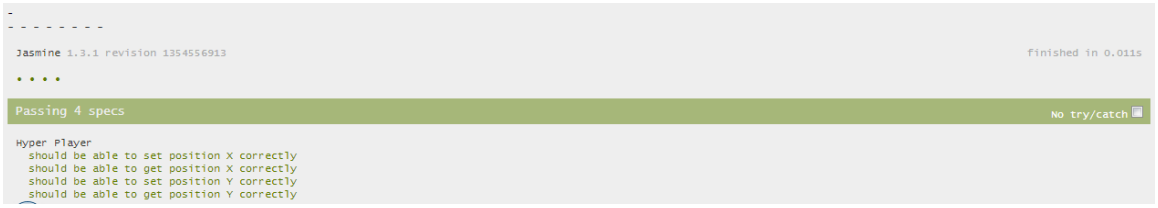


Figure 40: Jasmine test embedded within Hyper Bout Web Page

The Jasmine unit test from Figure 40 is embedded inside the Hyper Bout web page itself. These unit tests only take seconds to run and they are able to automate and reduce the amount of manual tests that need to be performed. For example, the unit test above tests to see if correct x and y position for the players are being returned. This is crucial so that players will be able to see each other's movement in multiplayer.

Another example of the unit test is to check if the title element exists in the page. Therefore, a test can be written to check for the existence of the

element. If, for example, a developer had a merge issue and accidentally deleted the code that loaded the Hyper Bout title, the Jasmine unit test will raise an issue to the problem as shown below.

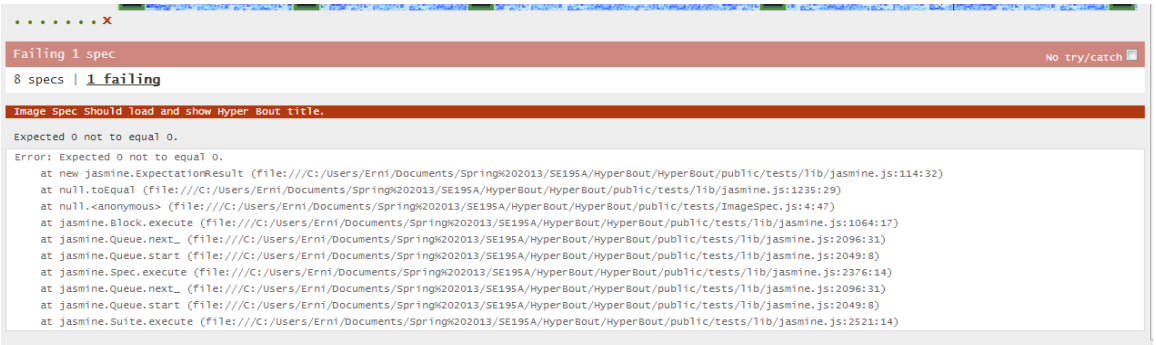


Figure 41: Failing Jasmine Unit Test

In Figure 41, the unit test fails and the issue brought up with Jasmine was that the title image could not be found. This way, developers will be able to find out that the system is not working as expected.

3. Testem

Testem is a node package that allows for Javascript unit tests to be run in continuous integration mode. This means that all the unit tests that are written in Jasmine can be ran automatically without actually having the testers or developers to manually load a website. Testem is open source and can be easily installed using the node package manager with the command "npm install testem -g" (Airportyh, 2013). After testem is installed, it can be ran in the command line.

Testem has a command which will allows it to run in continuous integration mode. Some setup is needed in order to run testem in Hyper Bout. However, after installation and after testem detects all the web browsers that are installed in the computer, testem can run Jasmine unit tests on all available web browsers, as shown in Figure 42.

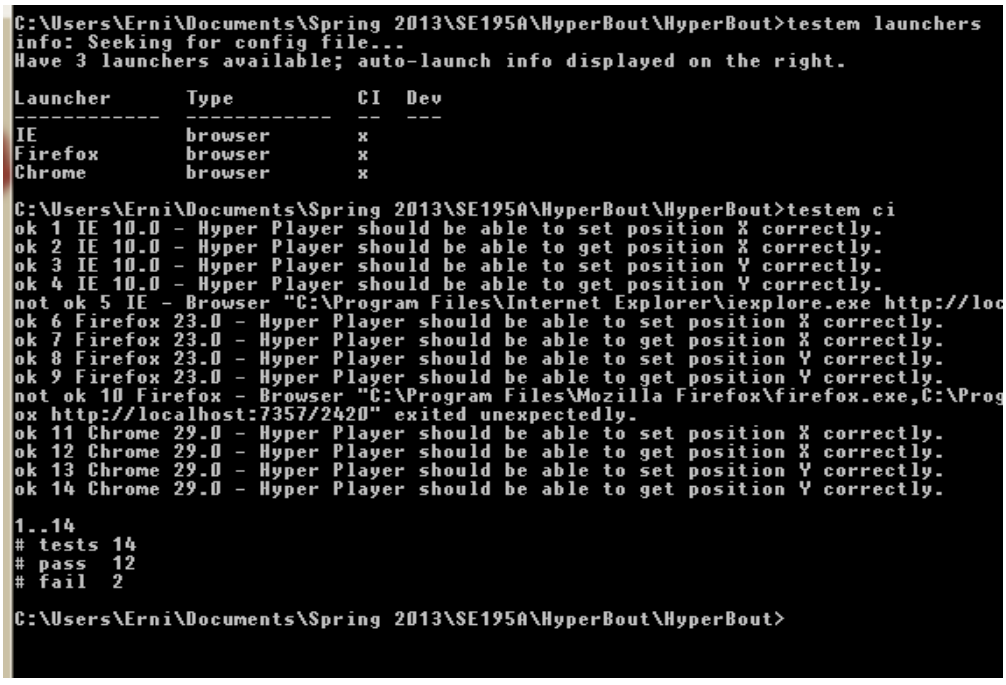


Figure 42: Automated running of unit tests

Using testem, developers and testers will not have to manually test Hyper Bout on every single web browser. Everything can be done automatically using testem. This will save the authors a lot of time and leave more time for the development of Hyper Bout.

4. Selenium

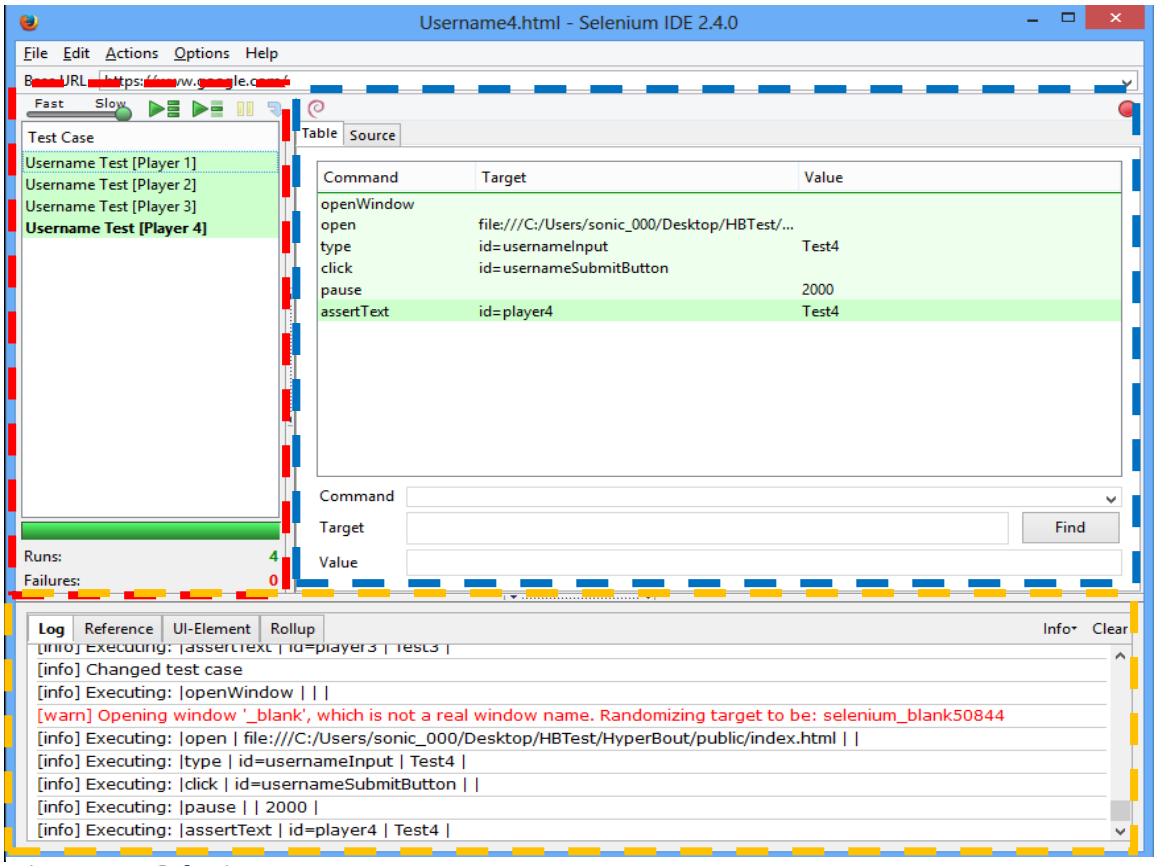


Figure 43: Selenium IDE

Selenium is another automated testing tool that the project primarily utilizes to test its lobby features. In particular, we have created four test suites to test particular features of the lobby including: (1) Username Input Validation, (2) User Readying Validation, (3) Chat Validation, and (4) Misc. Connectivity Error Validation. As the lobby system was the easiest component to perform black box automated test on, Selenium proved to be an ideal candidate for the job.

The panel outlined in red in the figure above shows the tests belonging to a particular test suite. The one shown is the "Username Test Suite" that checks to see that when players enter their user name in the Username Prompt, their username is carried to the lobby screen and is in the appropriate position. For this particular test, four users are simulated as opening different windows for the game where players enter their usernames in a defined order. If Selenium sees that the usernames match with the expected results in each CSS selector value, then the test is marked as a "Pass". If not, it is marked as "Fail". During the test run, the green progress bar will continue to update with a green to red ratio depending on whether or not each test has passed. In the particular example above, the test suite has fully run with 4/4 tests running to completion with a pass.

For the panel outlined in blue, this panel contains the specific commands that each test script in the suite is to be executed. As seen in the example above, a new window is opened, fields are specified with values, and the test either passes or fails depending on the "assert" command found on the last line. Commands that are highlighted in green pass while those in red fail. This helps the team determine at which phase the automated tests failed.

Lastly there is the panel in orange which provides the testers additional information on what exactly is being executed or what has gone wrong with the test execution. Additionally there are tabs for references and script code that the testers could look at for more details about the test.

7.1.4 Test Criteria

Hyper Bout's test criteria were drawn from its requirements from Chapter 4 to determine if its specifications were met. The basic of criteria for Hyper Bout was that it should be playable and it should have an end game state. Specific test criteria details can be found on the table below.

Table 10: Hyper Bout Test Criteria

Test Criteria #	Test Criteria
1	Users should be able to throw projectiles and damage the opponents.
2	User's health should decrease when damage are taken and points should increase for the player who threw the projectile.
3	Environments and platforms should have gravity and character sprites should walk on them.
4	When the game is closed manually by the player, said player should not be in the game anymore.
5	Players should have different character sprites that make them look unique.
6	Position of the player should be correct on all four players screen.
7	There should be a tutorial or how to play link.
8	The winner of the game should be the player who first reached the point limit.
9	The game should end when the maximum number of points are met.
10	The background and clouds in the game should be animated.
11	When a projectile hits the ground, it should show the explosion animation.
12	Chat feature in the lobby allows players in the lobby to chat to one another.
13	Power up items should work to the player's advantage.
14	Numbers on top of the players' heads should indicate the number of points that they have accumulated during the game session.
15	The game lobby should show the user-inputted name.
16	The game should only start when there are 4 players in the lobby.
17	End game statistics should show the users who won and who lost.
18	The game should not lag and should run at 30 frame rate per second.
19	The game should be compatible on Chrome and Firefox for windows.
20	The game should be playable with a keyboard and mouse.

21	The player's character sprite should jump if the space bar is pressed.
22	The player's character sprite should throw projectiles when the left mouse button is clicked.
23	The player's character sprite should move left when the 'a' button is pressed.
24	The player's character sprite should move right when the 'd' button is pressed.

In order to properly categorize our tests, our system can be broken down into three components that work together to create the Hyper Bout system. The following tests found in section 7.2. belong to one of the four categories listed.

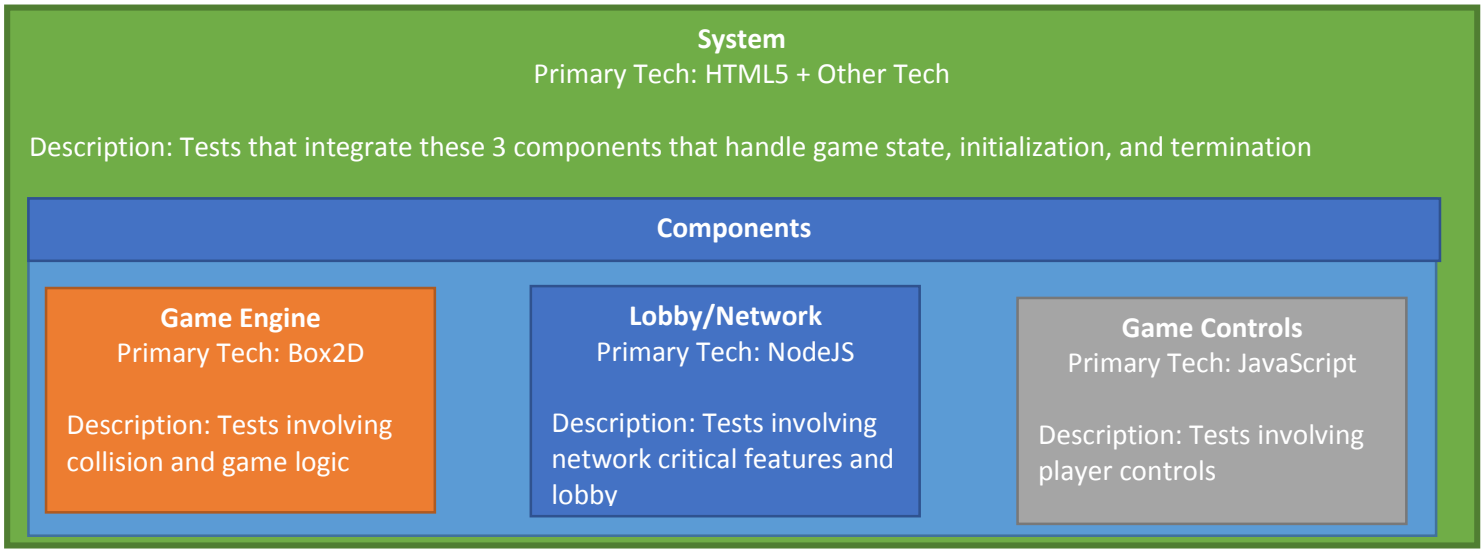


Figure 44: Test Architecture Breakdown

7.2 Testing and Experiment Approach

The following section describes the test strategies, methods, techniques, and coverage criteria in Hyper Bout. Test cases and test summaries are also listed in this section.

7.2.1 Test Strategies

Hyper Bout's testing strategies were to perform manual tests as often as possible and reduce the amount of manual tests needed to be performed as much as possible over time. All of the members of the team are responsible to manually test and play the game whenever they implement new features. They must perform a smoke test to ensure that the game runs at the most basic level so that defects and problems can be detected as soon as possible.

Unit tests were implemented to take some load off the manual tests. These unit tests will check whether or not the canvas draws or that everything displays correctly on the web page. Not only that, the unit tests also tested functions within the Hyper Bout system.

A test mode within the Hyper Bout system was also implemented. This test mode is a separate project that basically loads all the tests whenever the '#unitTest' is appended at the end of the Hyper Bout URL. Originally, the authors had to comment and uncomment some JavaScript loads inside the HTML page in order to run the unit test. However, with the implementation of this test mode, commenting and uncommenting within the index page is no longer needed. JavaScript loaders will immediately load all the necessary files for unit tests whenever it is requested. Basically the URL is checked to see if it contains the string '#unitTest'.

As shown below, without any code modification, the main page of Hyper Bout is loaded normally.

Figure 45: Hyper Bout Main Page Loaded Normally

However, with the appended '#unitTest' at the end of the URL, instead of loading the main page, Hyper Bout test mode will be loaded with a Jasmine framework interface as shown below.

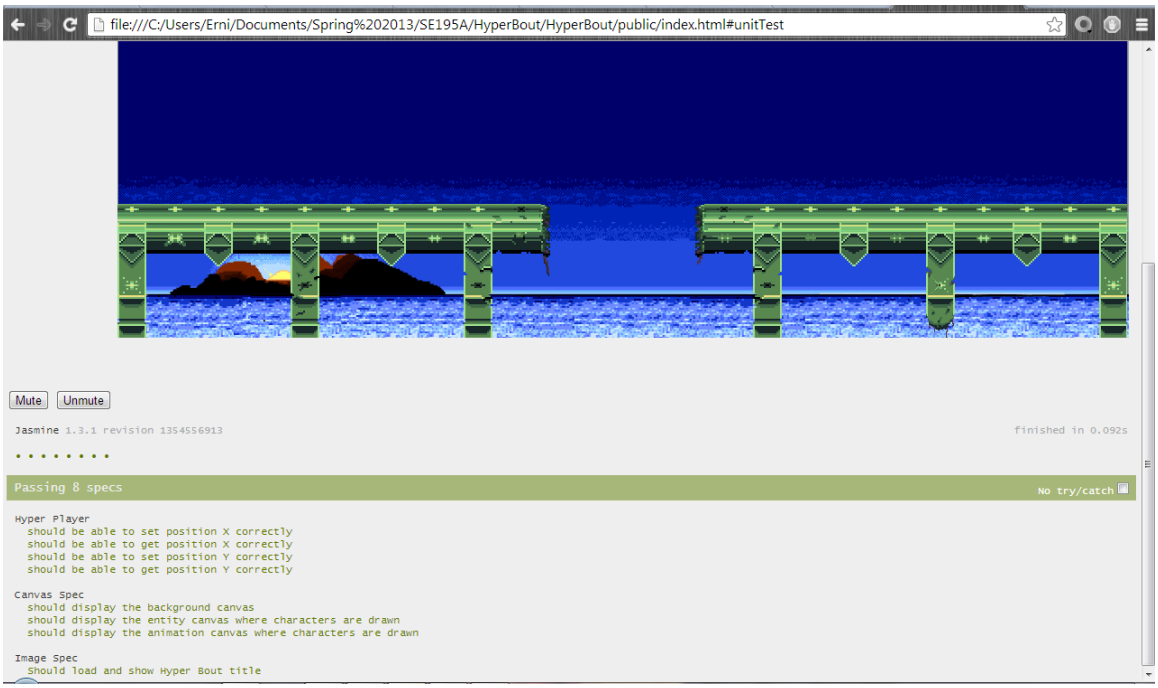


Figure 46: Hyper Bout Test Mode

This makes it easier for the authors to access and run the unit tests by simply changing their URL path.

Performance testing will be achieved through checking the FPS (frame rate per second) of the game. If the number of FPS drops, this means that there is a lag in the game and the players are experiencing a slowdown in their gameplay. Hyper Bout should run at a constant 30 frames per second to ensure smooth gameplay.

7.2.2 Test Methods and Techniques

Testing in Hyper Bout includes both black box and white box testing. Black box testing is achieved through manual testing while white box testing is achieved through unit testing using the Jasmine BDD framework.

7.2.3 Testing Coverage Criteria

The test coverage for Hyper Bout includes having to conform to its requirements and ensuring that each and every one of the requirements that are set in the functional and non-functional requirements are met. The test cases should cover all possible cases, options, and action combination that are allowed within the project. This will ensure the quality of Hyper Bout.

7.2.4 Test Cases

Below the test cases that have been conducted for the Hyper Bout system. Each test case is color coded to represent for which component they belong under. These are representative of the categories described in 7.1.4 Testing Criteria.

* Test Results can be found in *7.3.1: Test Execution and Test Result Summary*

7.2.4.1 Overview of Test Cases

Component 1: Game Engine		
Test ID:	**Description**	**Manual/Automated**
7	Projectile is properly thrown at clicked spot	Manual
8	Projectile explodes on contact with platforms	Manual
9	Players do not hurt themselves with self-projectile	Manual
10	Player's health goes down when hit	Manual
11	Player's spawn shield and invincible for duration	Manual
12	Player positions match on players' screens	Manual
31	Rain animation present and running at 30 FPS	Manual
32	Water animation present and running at 30 FPS	Manual
33	Cloud animation present and running at 30 FPS	Manual
34	Stars animation present and running at 30 FPS	Manual
35	Health power ups spawning at consistent rate	Manual
36	Health pickups disappear on player collision	Manual
37	Unique Player Sprites and Spawn Points	Manual
38	Player respawn when off screen	Manual
Component 2: Lobby and Network		
Test ID:	**Description**	**Manual/Automated**
13	Tests player 1 username input in lobby	Automated
14	Tests player 2 username input in lobby	Automated
15	Tests player 3 username input in lobby	Automated
16	Tests player 4 username input in lobby	Automated
17	Tests player 1 ready state in lobby	Automated
18	Tests player 2 ready state in lobby	Automated
19	Tests player 3 ready state in lobby	Automated
20	Tests player 4 ready state in lobby	Automated
21	Tests player 1 chat capabilities	Automated
22	Tests player 2 chat capabilities	Automated
23	Tests player 3 chat capabilities	Automated
24	Tests player 4 chat capabilities	Automated
25	Tests Player username duplication issue (p1)	Automated
26	Tests Player username duplication issue (p2)	Automated
27	Tests Player username duplication issue (p3)	Automated

28	Tests Player username duplication issue (p4)	Automated
Component 3: Game Controls		
Test ID:	**Description**	**Manual/Automated**
1	Player movement left	Manual
2	Player standing left	Manual
3	Player movement right	Manual
4	Player standing right	Manual
5	Player jumping left	Manual
6	Player jumping right	Manual
29	Player Projectile Second Spam Test	Manual
30	Player Jump Spam Test	Manual
System: Hyper Bout		
Test ID:	**Description**	**Manual/Automated**
39	Game Initialization Test	Manual
40	Game Termination Test	Manual
41	Music mutes when “Mute” is clicked	Manual
42	Music unmutes when “Unmute” is clicked	Manual
43	Explosion sounds play	Manual
44	Players have access to “how to play” link	Manual
45	Player Termination by browser close	Manual
46	Player Initialization by browser Open	Manual
47	GUI update of score and health	Manual
48	Points are added on player termination	Manual
49	System FPS Test	Manual
50	System Ping Test	Manual

7.2.4.2 Component 1: Game Engine Test Cases

Test Case No. 7	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing projectile throwing motion
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	Projectile is thrown
Items to be tested:	
1	Projectile throwing
Specifications:	
Input:	**Expected Output:**
Right click inside the HTML canvas	Projectile originating from the player should be throw at where the mouse click is.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Right click on the canvas in the HTML

Test Case No. 8	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing projectile explosion
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	Projectile is thrown and hits an object
Items to be tested:	
1	Projectile throwing
Specifications:	
Input:	**Expected Output:**
Right click inside the HTML canvas	Projectile originating from the player should be thrown at where the mouse click is and when the projectile hits an object, it should explode with blue fire
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Right click on the canvas in the HTML and watch it hit an object

Test Case No. 9	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character health
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	Projectile is thrown at self and health does not decrease
Items to be tested:	
1	Character Health
Specifications:	
Input:	**Expected Output:**
Right click on your own character sprite	Bomb should be throw at self and the character sprite will be pushed back, but the player's health should not decrease.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Right click on your own character sprite.

Test Case No. 10	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character health
Pre-Conditions:	User already bypass the lobby and there are more than one player in the game.
Post-Conditions:	Projectile is thrown at another player and the player who got hit health decreases.
Items to be tested:	
1	Character Health
Specifications:	
Input:	**Expected Output:**
Throw the projectile so that it hits another player.	The player who got hit will jump from explosion and his health will decrease as indicated from the table on the left.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Right click on the canvas so that the projectile hits another player other than yourself.

Test Case No. 11	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character shield
Pre-Conditions:	User already bypass the lobby and there are more than one player in the game.
Post-Conditions:	Projectile is thrown at another player and the player who got hit are immune for 3 seconds.
Items to be tested:	
1	Character Health
Specifications:	
Input:	**Expected Output:**
Throw the projectile so that it hits another player.	The player who got hit will jump from explosion and a shield will be drawn on said player, indicating the period of immunity.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Right click on the canvas so that the projectile hits another player other than yourself. If the player with a shield on gets hit again, the character's health should not decrease.

Test Case No. 12	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character shield
Pre-Conditions:	Play the game in two different windows or computers.
Post-Conditions:	Players are positioned correctly whenever the character sprites moves or jumps.
Items to be tested:	
1	Positioning
Specifications:	
Input:	**Expected Output:**
Move the character while watching the character on a separate window player.	The player should be shown moving to the correct position and direction in another player's window.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby as player 1.
3	Join the game from another browser tab as player 2.
4	Move player 1.
5	Watch if player 1 moves on player 2's window/tab.

Test Case No. 31	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing rain animation on page load
Pre-Conditions:	Bypass the game lobby
Post-Conditions:	[None – Visual Test]
Items to be tested:	
1	Animation
Specifications:	
Input:	**Expected Output:**
[None – Load Page]	Rain droplets should be dropping from the top of the canvas and should display on top of all other images. Page should remain running at 30 FPS.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	Open up the console in either Chrome or Firefox and retrieve the FPS
4	Verify rain droplets are running smoothly and game is still running at 30 FPS

Test Case No. 32	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing water flow on page load
Pre-Conditions:	Bypass the game lobby
Post-Conditions:	[None – Visual Test]
Items to be tested:	
1	Animation
Specifications:	
Input:	**Expected Output:**
[None – Load Page]	Water flow animation should be rendering at the bottom of the page. Page should remain running at 30 FPS.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	Open up the console in either Chrome or Firefox and retrieve the FPS
4	Verify water flow is running smoothly and game is still running at 30 FPS

Test Case No. 33	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing cloud flow animation on page load
Pre-Conditions:	Bypass the game lobby
Post-Conditions:	[None – Visual Test]
Items to be tested:	
1	Animation
Specifications:	
Input:	**Expected Output:**
[None – Load Page]	Three layers of clouds should be flowing across the top of the screen. Page should remain running at 30 FPS.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	Open up the console in either Chrome or Firefox and retrieve the FPS
4	Verify the cloud flow is running smoothly and game is still running at 30 FPS

Test Case No. 34	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing star glimmer animation on page load
Pre-Conditions:	Bypass the game lobby
Post-Conditions:	[None – Visual Test]
Items to be tested:	
1	Animation
Specifications:	
Input:	**Expected Output:**
[None – Load Page]	Star glimmer animation should be playing right underneath the three layers of clouds. Page should remain running at 30 FPS.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	Open up the console in either Chrome or Firefox and retrieve the FPS
4	Verify star glimmers are running smoothly and game is still running at 30 FPS

Test Case No. 35	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing Health Power Up Spawn
Pre-Conditions:	At least 2 players are in the game
Post-Conditions:	[None – Visual Test]
Items to be tested:	
1	Power Ups
Specifications:	
Input:	**Expected Output:**
[None]	The # of health power ups dropped per 30 seconds should be equivalent to the # of players on screen.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby with two players so that both players are now in the game.
3	Wait for 30 seconds.
4	Verify that the health power ups drop from the sky and the total number of power ups that spawn is equivalent to the number of players.
5	Repeat steps 3 and 4 for more thorough evaluation.

Test Case No. 36	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing Health Power Up Collision with Player
Pre-Conditions:	At least 2 players are in the game
Post-Conditions:	Health of player increased by 1
Items to be tested:	
1	Power Ups
Specifications:	
Input:	**Expected Output:**
Move player to dropped health powerup	The power up should disappear and the player's health who picked up the power up should increase by 1
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby with two players so that both players are now in the game.
3	Wait for 30 seconds for the two health packs to drop.
4	Move player 1 to one of the health packs and player 2 to another of the health packs.
5	Verify both health packs have disappeared and that each player's health has gone up by 1.

Test Case No. 37	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Unique Player Sprites and Spawn Points
Pre-Conditions:	[None]
Post-Conditions:	4 Players appear with unique color pallets and are located at four different spawn points.
Items to be tested:	
1	Player Sprite and Spawn
Specifications:	
Input:	**Expected Output:**
[None]	4 Players should appear in different colors and at different locations
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby with four players so that four players are visible in the game.
3	Verify that players 1 through 4 all have unique images and all have their own unique spawn points when the game start.

Test Case No. 38	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Player Respawn when Off Screen
Pre-Conditions:	[None]
Post-Conditions:	Player is located back at spawn position.
Items to be tested:	
1	Player Sprite and Spawn
Specifications:	
Input:	**Expected Output:**
‘a’ or ‘d’ key to fall off screen	Player should respawn back to starting position.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby with four players so that four players are visible in the game.
3	Take control of player 1 and fall into the middle pit
4	Take control of player 1 and fall off the left edge
5	Take control of player 1 and fall off the right edge
6	Do the same with players 2-4 and verify that they always spawn back at their starting positions.

7.2.4.3 Component 2: Lobby and Network

Test Case No. 13	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that Player 1 in the lobby has his/her name accurately transferred from the username page to the lobby page.
Pre-Conditions:	No other players are in the lobby
Post-Conditions:	Player 1's Username is shown in the player 1 username position
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test1"	"Test1" will appear in the player 1 username slot
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test1" at the Username prompt
4	Click on the "Join Lobby" button
5	Verify the player 1 username output

Test Case No. 14	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that Player 2 in the lobby has his/her name accurately transferred from the username page to the lobby page.
Pre-Conditions:	Player 1 is the only player in the lobby
Post-Conditions:	Player 2's Username is shown in the player 2 username position
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test2"	"Test2" will appear in the player 2 username slot
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test2" at the Username prompt
4	Click on the "Join Lobby" button
5	Verify the player 2 username output

Test Case No. 15	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that Player 3 in the lobby has his/her name accurately transferred from the username page to the lobby page.
Pre-Conditions:	Players 1 and 2 are the only players in the lobby
Post-Conditions:	Player 3's Username is shown in the player 3 username position
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test3"	"Test3" will appear in the player 3 username slot
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test3" at the Username prompt
4	Click on the "Join Lobby" button
5	Verify the player 3 username output

Test Case No. 16	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that Player 4 in the lobby has his/her name accurately transferred from the username page to the lobby page.
Pre-Conditions:	Players 1, 2, and 3 are already in the lobby
Post-Conditions:	Player 4's Username is shown in the player 4 username position
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test4"	"Test4" will appear in the player 4 username slot
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test4" at the Username prompt
4	Click on the "Join Lobby" button
5	Verify the player 4 username output

Test Case No. 17	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that players can reach a ready state from the lobby and that the game can start with more than 2 players.
Pre-Conditions:	No one else is in the lobby
Post-Conditions:	Player 1's username text turns green and is in the ready state.
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test1"	"Test1" will appear in the player 1 username slot and will turn green when the user clicks "Start Game". The game should not start.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test1" at the Username prompt
4	Click on the "Join Lobby" button
5	Click on "Start Game"
6	Verify that the username is lit in green and that the game has not started.

Test Case No. 18	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that players can reach a ready state from the lobby and that the game can start with more than 2 players.
Pre-Conditions:	Player 1 is in the lobby
Post-Conditions:	Player 2's username text turns green and is in the ready state.
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test2"	"Test2" will appear in the player 2 username slot and will turn green when the user clicks "Start Game". The game should start assuming player 1 and player 2 are the only players in the lobby
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test2" at the Username prompt
4	Click on the "Join Lobby" button
5	Click on "Start Game"
6	Verify that the username is lit in green and whether or not the game has started. If only players 1 and 2 are in the lobby and both are ready, the game should have started. Otherwise, both players are still in the lobby waiting for 3 or 4

Test Case No. 19	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that players can reach a ready state from the lobby and that the game can start with more than 2 players.
Pre-Conditions:	Players 1 and 2 are in the lobby
Post-Conditions:	Player 3's username text turns green and is in the ready state.
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test3"	"Test3" will appear in the player 3 username slot and will turn green when the user clicks "Start Game". The game should start assuming only three players are in the lobby.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test3" at the Username prompt
4	Click on the "Join Lobby" button
5	Click on "Start Game"
6	Verify that the username is lit in green and whether or not the game has started. If only players 1, 2, and 3 are in the lobby and are ready, the game should have started. Otherwise, players are still waiting on player 4 to ready.

Test Case No. 20	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests that players can reach a ready state from the lobby and that the game can start with more than 2 players.
Pre-Conditions:	Players 1, 2, and 3 are in the lobby
Post-Conditions:	Player 4's username text turns green and is in the ready state.
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test4"	"Test4" will appear in the player 4 username slot and will turn green when the user clicks "Start Game". The game should start assuming the other three players are readied.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test4" at the Username prompt
4	Click on the "Join Lobby" button
5	Click on "Start Game"
6	Verify that the username is lit in green and whether or not the game has started. The game should have started assuming players 1, 2, and 3 are all ready.

Test Case No. 21	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players can chat with each other and that newly joining lobby players can pull the chat history data.
Pre-Conditions:	No other players are in the lobby
Post-Conditions:	Player 1's message is stored in the chat history on the server.
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test1" Message: "Hi, I'm Test 1"	The username with the message should appear in the chat history box.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test1" at the Username prompt
4	Click on the "Join Lobby" button
5	Type in the message in the message prompt and hit "enter"
6	Verify that the message now appears on the screen.

Test Case No. 22	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players can chat with each other and that newly joining lobby players can pull the chat history data.
Pre-Conditions:	Player 1's message is stored in the chat history.
Post-Conditions:	Player 2's message is stored in the chat history on the server. Player 2 can also see previous chat history
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test2" Message: "Hi, I'm Test 2"	The username with the message should appear in the chat history box along with the previous messages.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test2" at the Username prompt
4	Click on the "Join Lobby" button
5	Verify that the previous messages can be seen.
6	Type in the message in the message prompt and hit "enter"
7	Verify that the message now appears on the screen.

Test Case No. 23	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players can chat with each other and that newly joining lobby players can pull the chat history data.
Pre-Conditions:	Player 1 and 2's messages are stored in the chat history.
Post-Conditions:	Player 3's message is stored in the chat history on the server. Player 3 can also see previous chat history
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test3" Message: "Hi, I'm Test 3"	The username with the message should appear in the chat history box along with the previous messages.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test3" at the Username prompt
4	Click on the "Join Lobby" button
5	Verify that the previous messages can be seen.
6	Type in the message in the message prompt and hit "enter"
7	Verify that the message now appears on the screen.

Test Case No. 24	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players can chat with each other and that newly joining lobby players can pull the chat history data.
Pre-Conditions:	Player 1, 2, and 3's messages are stored in the chat history.
Post-Conditions:	Player 4's message is stored in the chat history on the server. Player 4 can also see previous chat history
Items to be tested:	
1	Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "Test4" Message: "Hi, I'm Test 4"	The username with the message should appear in the chat history box along with the previous messages.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "Test4" at the Username prompt
4	Click on the "Join Lobby" button
5	Verify that the previous messages can be seen.
6	Type in the message in the message prompt and hit "enter"
7	Verify that the message now appears on the screen.

Test Case No. 25	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players with the same username do not break the game's functionality
Pre-Conditions:	No other players are in the lobby
Post-Conditions:	Player 1 has username "test" stored on server
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "test"	"test" should appear in the player 1 username slot.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "test" the Username prompt
4	Click on the "Join Lobby" button
5	Verify that the player 1 slot has the username "test"

Test Case No. 26	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players with the same username do not break the game's functionality
Pre-Conditions:	Player 1 is in the lobby with the username "test"
Post-Conditions:	Player 2 has username "test (1)" stored on server
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "test"	"test (1)" should appear in the player 2 username slot.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "test" the Username prompt
4	Click on the "Join Lobby" button
5	Verify that the player 2 slot has the username "test (1)"

Test Case No. 27	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players with the same username do not break the game's functionality
Pre-Conditions:	Player 1 is in the lobby with the username "test" and Player 2 has "test (1)"
Post-Conditions:	Player 3 has username "test (2)" stored on server
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "test"	"test (2)" should appear in the player 3 username slot.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "test" the Username prompt
4	Click on the "Join Lobby" button
5	Verify that the player 3 slot has the username "test (2)"

Test Case No. 28	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Tests to see that players with the same username do not break the game's functionality
Pre-Conditions:	Player 1 is in the lobby with the username "test", Player 2 has "test (1)", and Player 3 has "test (2)"
Post-Conditions:	Player 4 has username "test (3)" stored on server
Items to be tested:	
1	Username Form / Lobby Form
Specifications:	
Input:	**Expected Output:**
Username: "test"	"test (3)" should appear in the player 4 username slot.
Procedural Steps:	
1	Start up the node.js server (if not already)
2	Open up the index.html file within the public folder
3	Type in "test" the Username prompt
4	Click on the "Join Lobby" button
5	Verify that the player 4 slot has the username "test (3)"

7.2.4.4 Component 3: Game Controls

Test Case No. 1	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character movement animation
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The character sprite moves right and the character sprite shows running to the right animation
Items to be tested:	
1	Game Character Movement
Specifications:	
Input:	**Expected Output:**
Press the ‘d’ button while the HTML Canvas is on focus	The player’s character sprite move to the right with run right animation.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Click on the canvas in the HTML
4	Press the ‘d’ key

Test Case No. 2	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character movement animation
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The character sprite stands still facing right after moving right.
Items to be tested:	
1	Game Character Movement
Specifications:	
Input:	**Expected Output:**
Press the ‘d’ button while the HTML Canvas is on focus and then release the button	The player’s character sprite move to the right with run right animation and then stand still facing to the right after button is released.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Click on the canvas in the HTML
4	Press the ‘d’ key
5	Release the ‘d’ key

Test Case No. 3	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character movement animation
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The character sprite moves left and the character sprite shows running to the left animation
Items to be tested:	
1	Game Character Movement
Specifications:	
Input:	**Expected Output:**
Press the ‘a’ button while the HTML Canvas is on focus	The player’s character sprite move to the left with run right animation.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Click on the canvas in the HTML
4	Press the ‘a’ key

Test Case No. 4	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character movement animation
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The character sprite stands still facing right after moving left.
Items to be tested:	
1	Game Character Movement
Specifications:	
Input:	**Expected Output:**
Press the ‘a’ button while the HTML Canvas is on focus and then release the button	The player’s character sprite move to the left with run left animation and then stand still facing to the left after button is released.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Click on the canvas in the HTML
4	Press the ‘a’ key
5	Release the ‘a’ key

Test Case No. 5	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character jumping animation
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The character sprite jumps with jump animation facing right and lands with character facing right.
Items to be tested:	
1	Game Character Jumping
Specifications:	
Input:	**Expected Output:**
Press the ‘spacebar’ button while the HTML Canvas is on focus and while character is facing right	The player’s character sprite jumps facing right and lands facing right.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Click on the canvas in the HTML
4	Press the ‘d’ key to face right
5	Press the ‘spacebar’

Test Case No. 6	
Date:	11/15/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing character jumping animation
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The character sprite jumps with jump animation facing left and lands with character facing left.
Items to be tested:	
1	Game Character Jumping
Specifications:	
Input:	**Expected Output:**
Press the 'spacebar' button while the HTML Canvas is on focus and while character is facing left	The player's character sprite jumps facing left and lands facing left.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Click on the canvas in the HTML
4	Press the 'a' key to face left
5	Press the 'spacebar'

Test Case No. 29	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Projectile Spam Test. Test observes that players cannot exceed the 1 projectile/second fire rate.
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	[None – Visual Test]
Items to be tested:	
1	Projectile Fire Rate
Specifications:	
Input:	**Expected Output:**
Continuously press the LMB while the HTML5 canvas is in focus	Tester should observer that projectiles fire at constant rate.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Continuously click the LMB on the HTML5 canvas and observer the projectile fire rate.
4	Verify projectile fire rate results.

Test Case No. 30	
Date:	11/18/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Test ensure that players are able to jump when they are touching platforms
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	[None – Visual Test]
Items to be tested:	
1	Game Character Movement
Specifications:	
Input:	**Expected Output:**
Continuously press the ‘space’ button while the HTML5 Canvas is on focus	The player’s character sprite should jump only when the player in in contact with a platform. The space bar should still work and players should not be able to fly.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Click on the canvas in the HTML5
4	Continuously press the ‘space’ key while moving left and right with the ‘a’ and ‘d’ keys.
5	Verify jumping works as described.

7.2.4.4 System Integration Tests: Hyper Bout

Test Case No. 39	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Game Initialization Test
Pre-Conditions:	[None]
Post-Conditions:	Four players have joined the game with health set to 5 and score set to 0 for each player.
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
Username 1: “Test 1” Username 2: “Test 2” Username 3: “Test 3” Username 4: “Test 4”	Four players are in game with health and score set to default
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Access the game from four different systems/windows
3	Type in “Test X” for each user and ready up
4	Verify all players are in unique spawn positions
5	Verify that score is set to 0 for all players and all players start out with 5 health

Test Case No. 40	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Game Termination Test
Pre-Conditions:	[None]
Post-Conditions:	Game ends with the winning player announced and all players return back to username screen.
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
Username 1: “Test 1” Username 2: “Test 2” Username 3: “Test 3” Username 4: “Test 4”	Game terminates at score limit, a winner is announced, and players return to username screen.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Access the game from four different systems/windows
3	Type in “Test X” for each user and ready up
4	With player 1, continuously attack any of the other players until the point limit (2 points default) is reached.
5	Verify that all 4 players receive a winner notification and are pushed to username screen.

Test Case No. 41	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Music Mute Control
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The BGM is muted
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
LMB on “Mute” button	The BGM is muted
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	Click on the “Mute” button on the left hand side of the page.
4	Verify that the BGM has muted.

Test Case No. 42	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Music Unmute Control
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The BGM is unmuted
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
LMB on “Unmute” button	The BGM is unmuted
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	Click on the “Unmute” button on the left hand side of the page.
4	Verify that the BGM has unmuted.

Test Case No. 43	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Explosion Sound Test
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	The explosion audio file plays when a projectile collides with a platform
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
Press the LMB while the HTML Canvas is on focus and is pointed at a platform	Explosion animation and audio plays.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	LMB click in the direction of a platform
4	Verify that the projectile explodes on the platform and that the audio file is played

Test Case No. 44	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Test Help Link
Pre-Conditions:	User already bypass the lobby
Post-Conditions:	Pop-Up window with a brief text tutorial appears on how to play
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
[None]	Pop-Up window with a brief text tutorial appears on how to play
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Proceed to go through the lobby to head to the main game page.
3	Click on the “How to Play” link
4	Verify that the new window has popped up with instructions on how to play

Test Case No. 45	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Player Browser Close Termination
Pre-Conditions:	[None]
Post-Conditions:	The player image is dropped from the game and the socket connection is dropped.
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
Username 1: “Test 1” Username 2: “Test 2” Username 3: “Test 3” Username 4: “Test 4”	The dropped player has his image removed and his connection data dropped.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Access the game from four different systems/windows
3	Type in “Test X” for each user and ready up
4	As player 1, close the browser
5	Verify that player 1’s sprite image is dropped from player 2, 3, and 4’s screen.
6	Verify in the Node.js logs that the connection has been dropped.

Test Case No. 46	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Player Browser Open Initialization
Pre-Conditions:	[None]
Post-Conditions:	The player image is added to the game and the socket connection is recorded.
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
Username 1: “Test 1” Username 2: “Test 2” Username 3: “Test 3” Username 4: “Test 4”	Connection data with the new player is stored on the Node server
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Access the game from four different systems/windows
3	Type in “Test X” for each user and ready up
4	Verify that all four players are visible on screen
4	Verify in the Node.js logs that there are 4 separate connections each with their own connection ID.

Test Case No. 47	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	GUI Update of Health
Pre-Conditions:	[None]
Post-Conditions:	Hit player loses HP
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
Username 1: “Test 1” Username 2: “Test 2” Username 3: “Test 3” Username 4: “Test 4”	The dropped player has his image removed and his connection data dropped.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Access the game from four different systems/windows
3	Type in “Test X” for each user and ready up
4	Have player 1 hit player 2 so that player 2 loses a health point
5	Player 1’s score should remain at 0 while player 2’s health will decrement by 1.

Test Case No. 48	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Points added on Player Termination
Pre-Conditions:	[None]
Post-Conditions:	A point is added to the player conducting the final blow while the defeated player respawns with 5 health points.
Items to be tested:	
1	Hyper Bout System
Specifications:	
Input:	**Expected Output:**
Username 1: "Test 1" Username 2: "Test 2" Username 3: "Test 3" Username 4: "Test 4"	The dropped player has his image removed and his connection data dropped.
Procedural Steps:	
1	Start the node.js sever (if not already running)
2	Access the game from four different systems/windows
3	Type in "Test X" for each user and ready up
4	Have player 1 hit player 2 five times so that player 2's health goes down to 0
5	Verify that player 1 gets a point while player 2 spawns back to the spawn point with a full 5 health points.

Test Case No. 49	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing Frame Rate
Pre-Conditions:	Game has exited lobby and is currently in gameplay mode.
Post-Conditions:	The game keeps a steady framerate
Items to be tested:	Hyper Bout Engine
Specifications:	
Input:	**Expected Output:**
Create box2d objects over a set period of time.	Game handles additional box2d objects and keeps a steady framerate.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Game has started
4	Create box2d objects

Test Case No. 50	
Date:	11/19/2013
Test Item:	Hyper Bout Web Based Game
Pass/Fail:	Pass
Release and Version #:	Hyper Bout v 1.0
Test Case Description:	Testing Network Capcity
Pre-Conditions:	1. 4 players are currently in the game. 2. Lobby has exited and the game is currently in session.
Post-Conditions:	Total packet length is calculated
Items to be tested:	Hyper Bout Network
Specifications:	
Input:	**Expected Output:**
Move all players at once while sending projectiles as quickly as possible.	Average the packets sent over a specified period of time.
Procedural Steps:	
1	Run Hyper Bout
2	Start the game on the lobby
3	Game has started
4	Make all players jump
5	Make all players fire projectiles
6	Record packet size for set period of time

7.2.5 Testing Approach Summary

Automated testing is a bit of a challenge when it comes to a game project. Therefore, focus on the testing phase for Hyper Bout is mostly manual testing done by the developers and testers in the group. Test cases are used as a reference for aspects of the game that should be tested. Minor automated tests are done using Selenium and Jasmine. Jasmine is used mostly for unit testing the basic JavaScript functions in Hyper Bout while Selenium is used to test user inputs on the game. Overall, while running the manual and unit tests, all test cases passed and defects that were found were filed in the GitHub bug tracking tool.

These bugs were assigned to the appropriate developer and said developer would perform a fix and release a patch on the bug. Afterwards, more manual testing was performed to ensure that the defect was actually fixed.

7.3 Testing and Experiment Results and Analysis

Due to time constraints, there were some unfixed bugs that were found during the software lifecycle. However, these bugs were considered lower priority and will not hinder gameplay. Below are charts and figures that convey the testing and experiment results and analysis in Hyper Bout.

7.3.1 Test Execution and Test Result Summary

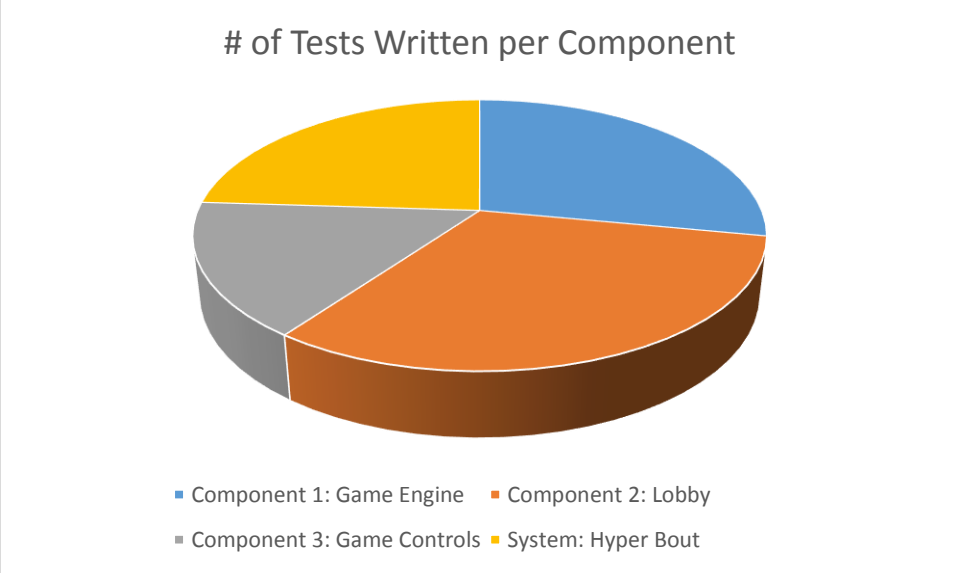


Figure 47: % Tests Written per Component

Component 1: Game Engine	14
Component 2: Lobby	16
Component 3: Game Controls	8
System: Hyper Bout	12

As observed, a majority of the tests focused on the core game engine as well as the network and lobby aspects of Hyper Bout. In addition to these formal tests, individuals of the team would conduct simple smoke test to ensure that the few bugs uncaught by these tests could be resolved. As Hyper Bout passes these 50 test cases, our team can assure the quality of this project.

7.3.2 Performance Test Results Analysis

7.3.2.1 HTML5 Frames Per Second Test

Hyper Bout's game engine loop cycles every 1/30th of a second. Each of these cycles handles several key aspects of the Hyper Bout game, such as animation updates, player location updates, handling collisions and playing music. In order to keep smooth game play for each player our team decided to cap the game at 30 frames per second. In order to test this we decided to stress test the Hyper Bout engine to see how many box2d objects could be displayed while keeping a constant 30 frames per second.

Since there are up to 4 simultaneous players in a single Hyper Bout game these tests would need to be able to handle 4 players creating projectiles at the same time while maintaining a constant frame rate. Additionally, each raindrop in the game is a box2d body that is affected by gravity. At any given time our team found that there are an average of about 200 - 400 rain drops at once.

In order to make sure that the game engine is able to handle this, our team created a test that would randomly spawn box2d objects on the screen that would collide with other box2d bodies placed around the screen. Our team found that the game engine that we had created for Hyper Bout was able to handle well over 500 box2d objects interacting at once with network multiplayer on at once. No frame drops were found even on slower computer hardware, such as a core-duo clocked at ~2.3 GHz.

7.3.2.2 Network Ping Test

In addition to testing the frames per second that we can achieve with our Hyper Bout engine the networking aspect of the game must also be tested. In order to test the

network our team decided to check to see how much data is sent per second over the network to each host. This will give our team a better idea of how much bandwidth Hyper Bout requires.

To achieve this, our team decided to add up each of the packet lengths sent over the network for a small duration of time. The average of this packet sum would be calculated and this would give our team a better idea on how much data is being sent per second.

Each packet that is sent is small in size, but many packets are sent back in forth in short bursts. For example, whenever a character moves, or throws a projectile, or takes damage a packet must be sent to the server, which is then rerouted to the other players in the game. Since the game is looping 30 times a second and there are 4 players that means if each player moves at once, there is a total of 120 packets being sent to the server, then these are rerouted to 3 different users (the user who sent the packet does not receive the same packet). Meaning there are at least ~360 packets being sent just for player movement.

In addition, when a player takes damage or throws a projectile another packet is sent to the router and then rerouted to the players. These packets consist of vectors that must be calculated client-side to dictate where the projectiles are going to be projected to. The amount of times a player can throw a projectile as well has been capped for game balancing reasons as well as to save on bandwidth.

Our team found that each packet is only a few hundred bits each, with a maximum of roughly ~360 packets sent each second. After calculating an average our team found that this means that Hyper Bout is sending a few kilobits each second of game play.

7.3.3 Bug Distribution Report

7.3.3.1 Bug Table Summary

Table 10 shows the list of bugs all the bugs that were submitted and their status.

Table 11: Bug Table Summary

Bug#	Title	Type	Status
1	Players who are already in the game resets to their default position when another player joins the game	Bug	Closed
2	Projectile cannot be seen by other players	Bug	Closed
3	Projectiles are stuck to the floor	Bug	Closed
4	Game is Muted	Bug	Closed
5	Hyper Bout game doesn't run	Invalid	Closed
6	Player Position does not align correctly	Bug	Closed
7	Player and Projectile Collision	Enhancement	Closed
8	Health of the Player	Enhancement	Closed
9	Items and Powerups	Enhancement	Closed
10	The game doesn't end (game termination)	Enhancement	Closed
11	Point system	Enhancement	Closed
12	Player animation	Enhancement	Closed
13	Player's body is still there when the player leaves	Bug	Closed
14	Explosion/Bomb is affected by Gravity	Bug	Closed
15	Health Bar	Enhancement	Closed
16	Jump Issues	Bug	Closed
17	Animation does not show properly on remote player screen	Bug	Closed
18	Health power up doesn't add health points	Bug	Closed
19	Need platforms so that players can jump up to the top platform	Bug	Closed
20	Players health should decrease when player falls off the platform	Bug	Open
21	Box2d block needs to be removed for production	Bug	Closed
22	LAN connection with switches should connect the players	Enhancement	Closed
23	Show the FPS of the game on the top right corner of the screen	Question, Enhancement	Open
24	All players jump when one player gets hit by a bomb explosion	Bug	Closed
25	Canvas Boundary needed	Bug	Closed
26	Lobby: Ready Up Does Not Work for Same Names	Bug	Closed

7.3.3.2 Bug Descriptions

The following images depict the bugs stored on the GitHub repository:

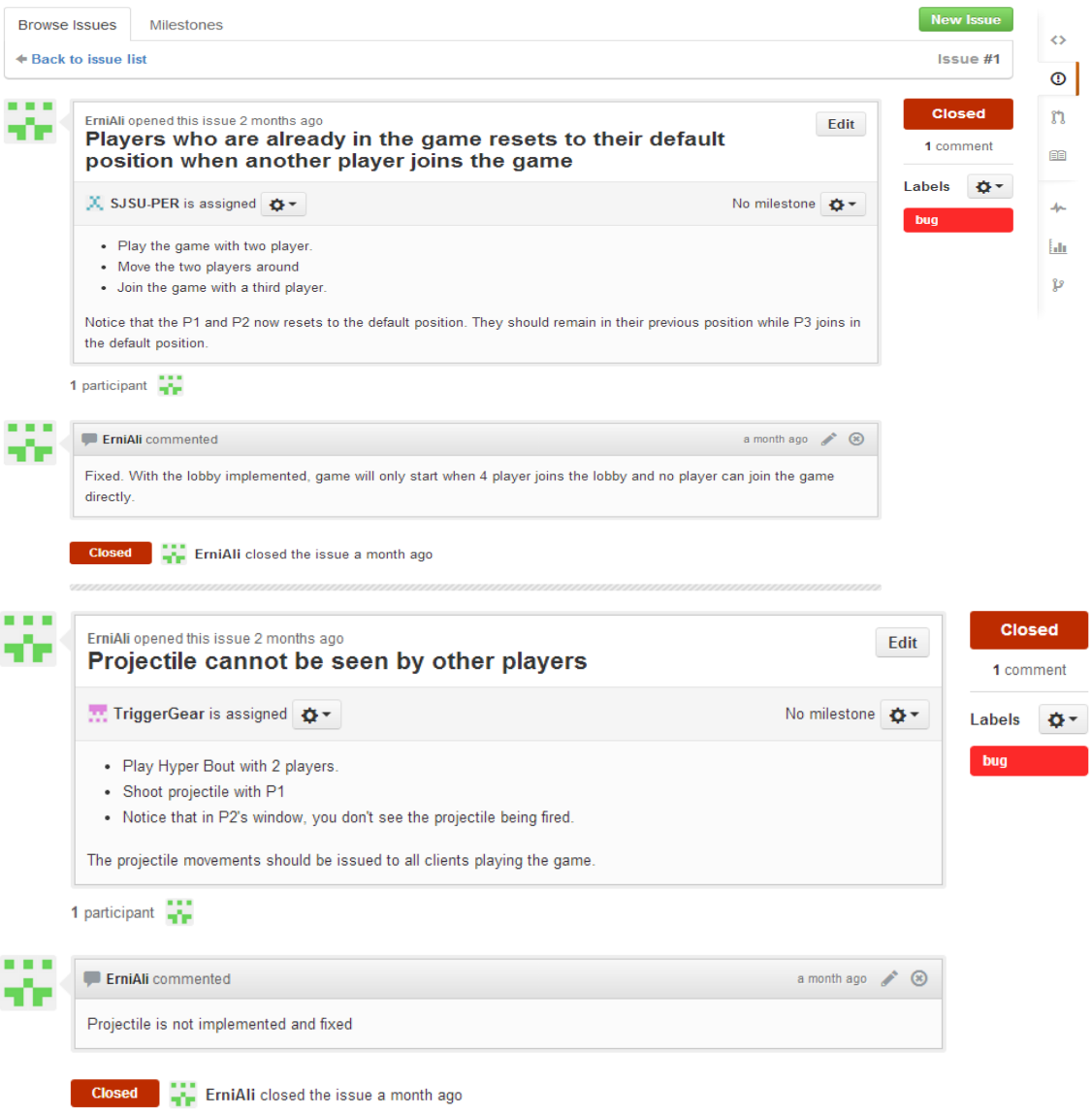
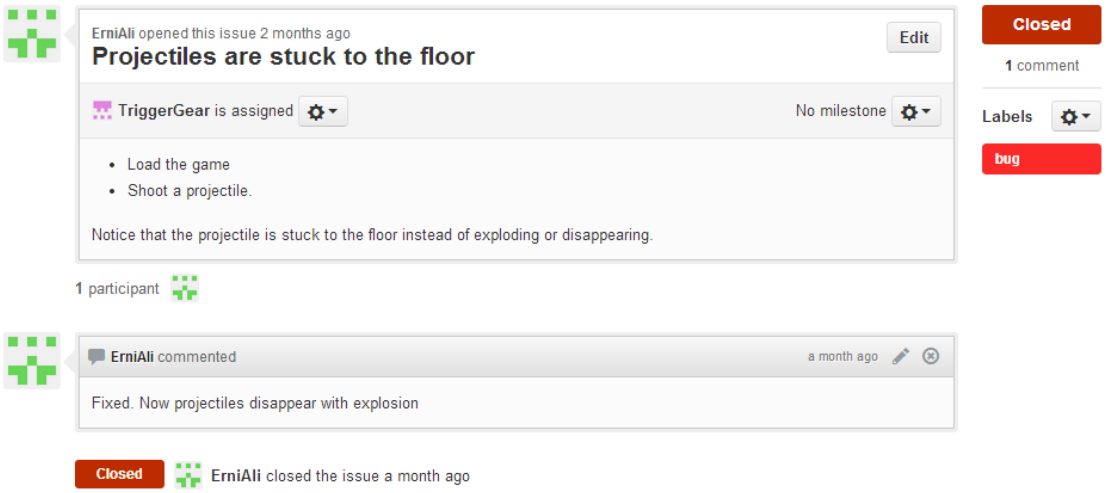
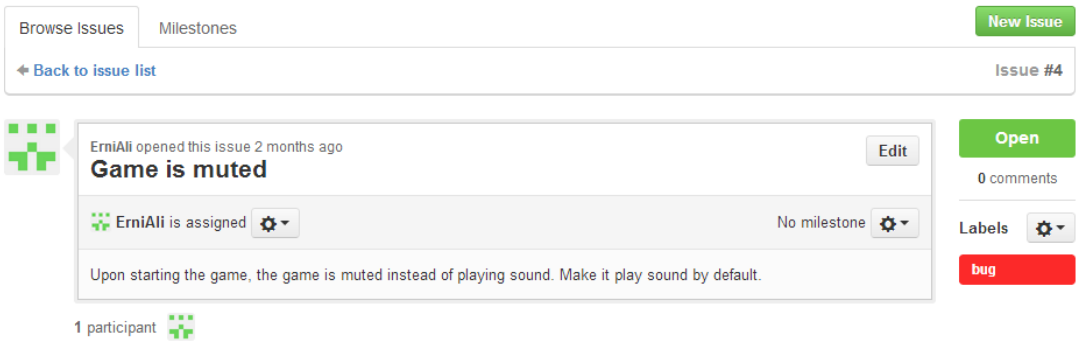


Figure 48: Bug Description List


ErniAli opened this issue 2 months ago
Projectiles are stuck to the floor
Edit
Closed
1 comment
TriggerGear is assigned
No milestone
Labels
bug
Load the game
Shoot a projectile.
Notice that the projectile is stuck to the floor instead of exploding or disappearing.
1 participant
ErniAli commented
a month ago
Fixed. Now projectiles disappear with explosion
Closed
ErniAli closed the issue a month ago


Browse Issues
Milestones
New Issue
Back to issue list
Issue #4
ErniAli opened this issue 2 months ago
Game is muted
Edit
Open
0 comments
ErniAli is assigned
No milestone
Labels
bug
Upon starting the game, the game is muted instead of playing sound. Make it play sound by default.
1 participant

ErniAli opened this issue 2 months ago

Hyper Bout game doesn't run

Edit

Closed

1 comment

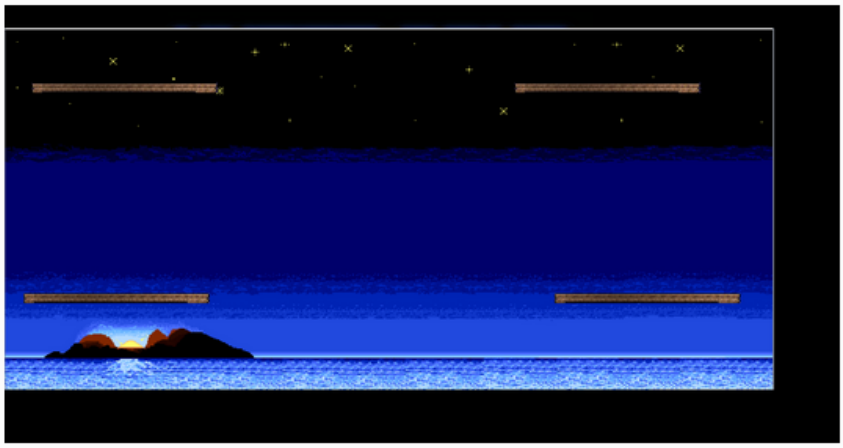
TriggerGear is assigned

No milestone

Labels

invalid

The game is no longer running. This happens after a recent code push.



1 participant

Closed ErniAli closed the issue 2 months ago

ErniAli commented 2 months ago

Need to run node with the server to have the game run.

ErniAli opened this issue a month ago

Player Position does not align correctly

Edit

Closed

1 comment

SJSU-PER is assigned

No milestone

Labels

bug

Run Hyper Bout.

Play the game side by side using the multiplayer feature. Keep pressing jump and move the character around, notice that on the other screen, the player that moves does not align correctly.

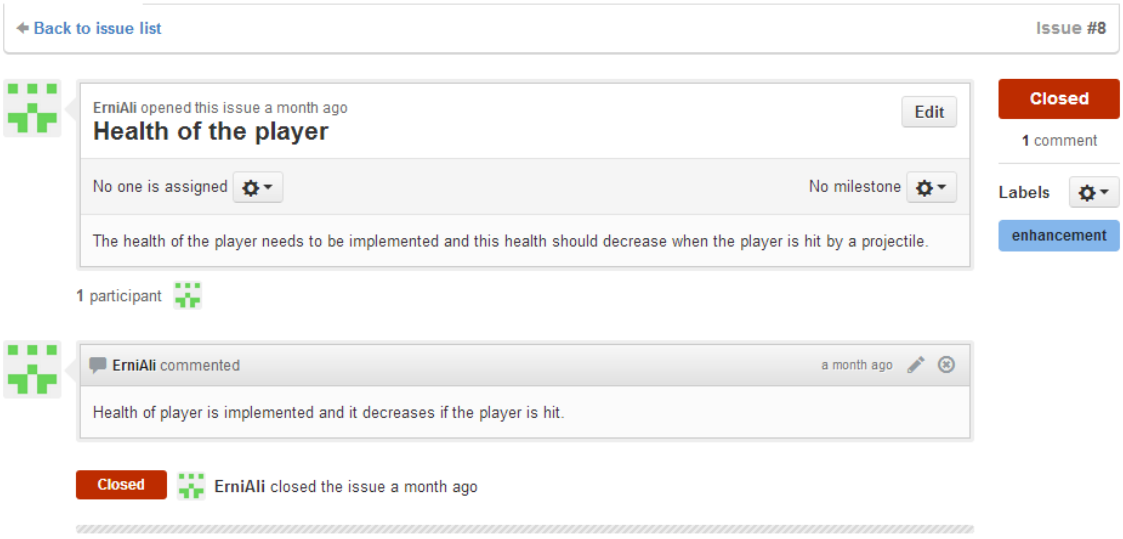
1 participant

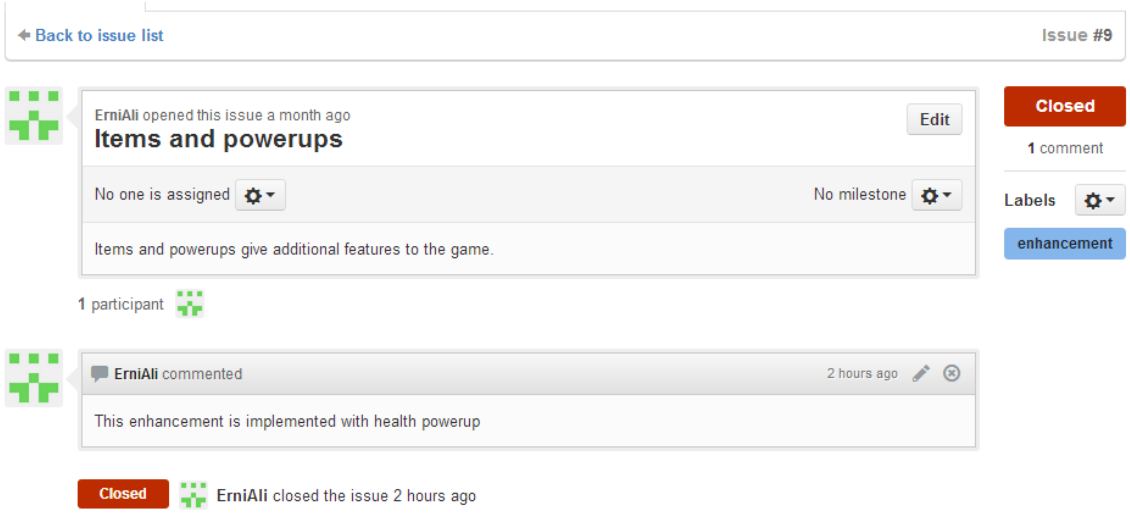
ErniAli commented a month ago

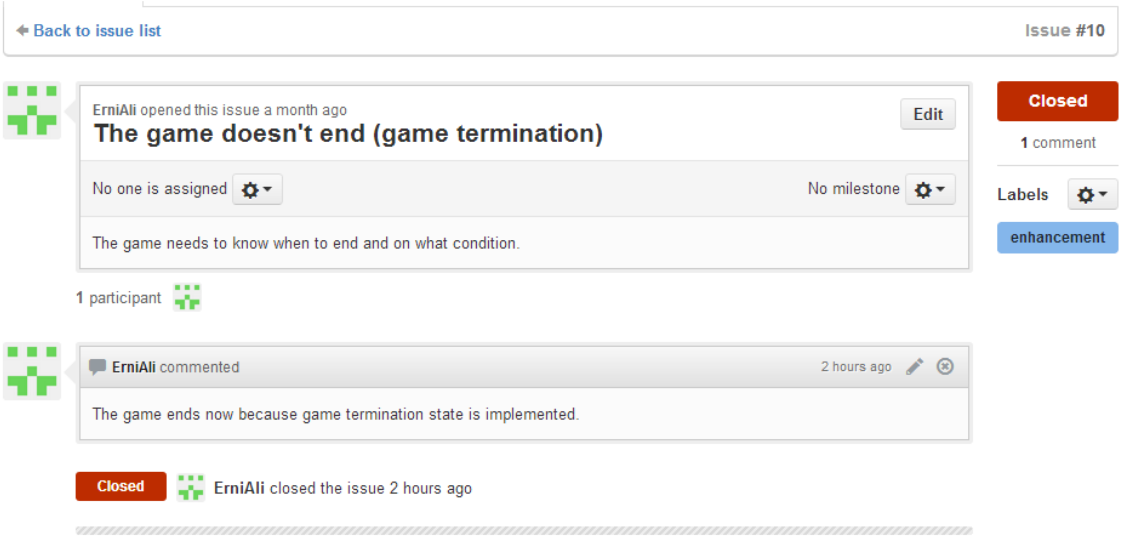
Fixed!

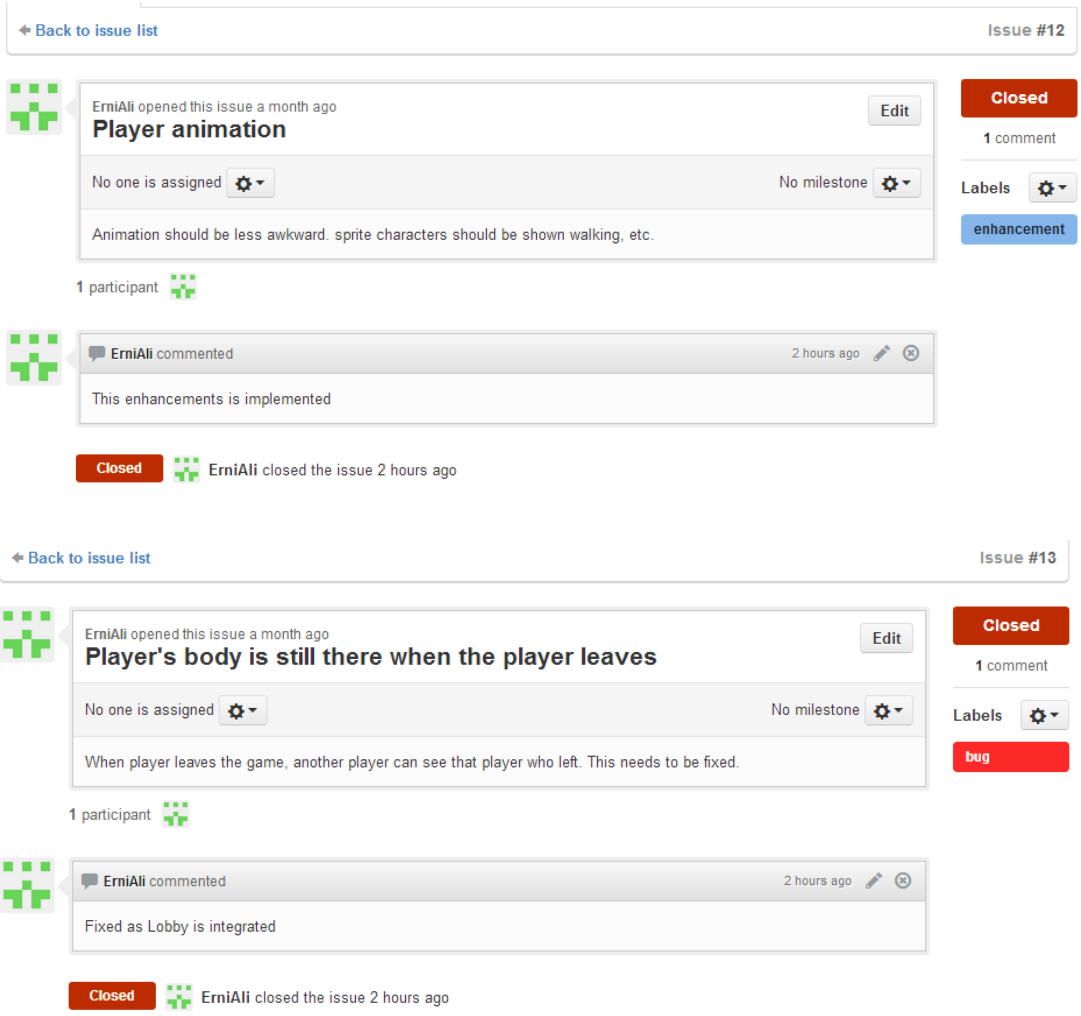
Closed ErniAli closed the issue a month ago

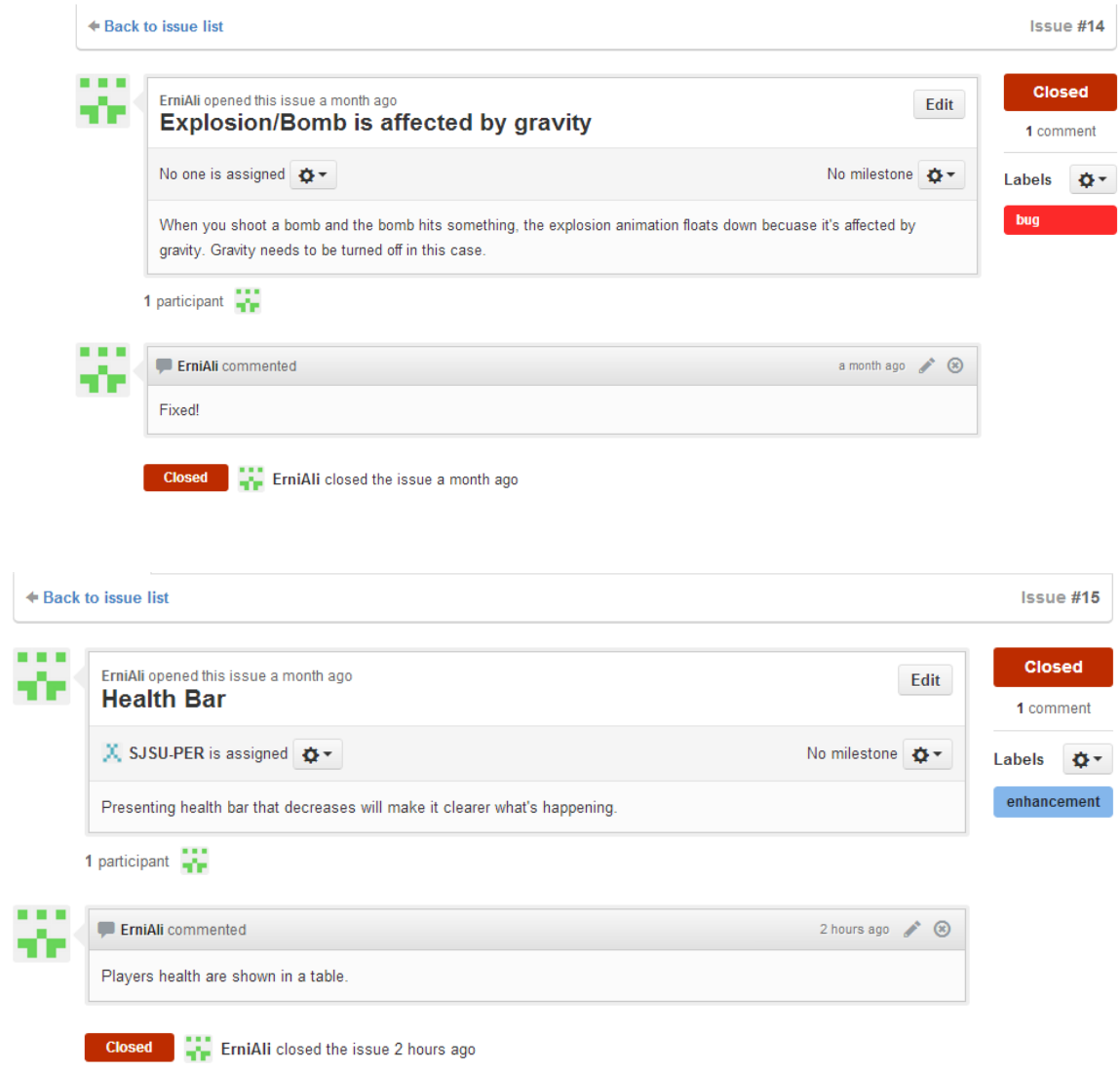

ErniAli opened this issue a month ago
Player and projectile collision
Edit
No one is assigned
No milestone
The game needs to be able to detect collision when the projectile hits a player.
1 participant
ErniAli commented
a month ago
All good. Player jumps when it got hit by projectiles.
Closed
ErniAli closed the issue a month ago
Closed
1 comment
Labels
enhancement

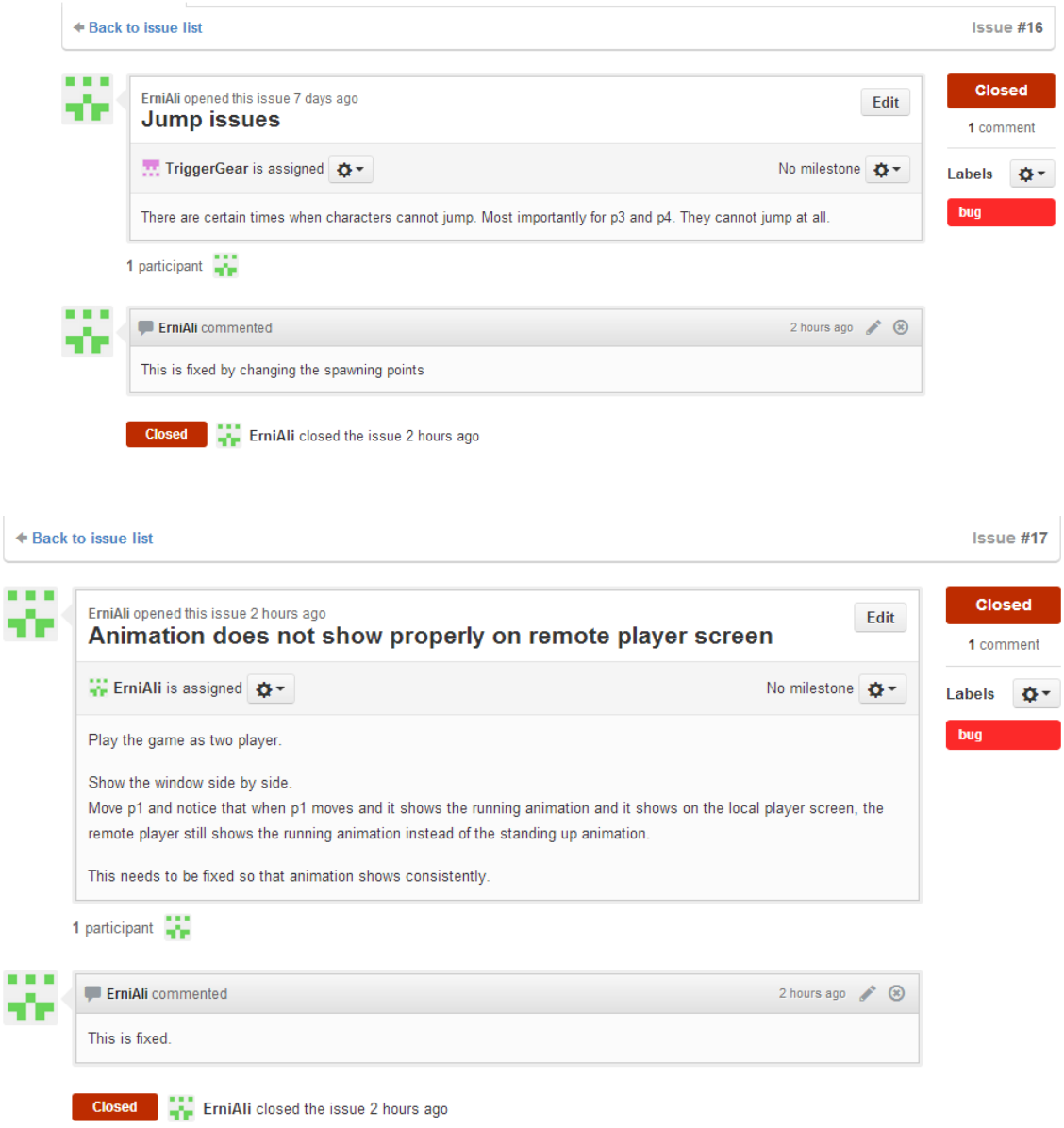

Back to issue list
Issue #8
ErniAli opened this issue a month ago
Health of the player
Edit
No one is assigned
No milestone
The health of the player needs to be implemented and this health should decrease when the player is hit by a projectile.
1 participant
ErniAli commented
a month ago
Health of player is implemented and it decreases if the player is hit.
Closed
ErniAli closed the issue a month ago
Closed
1 comment
Labels
enhancement

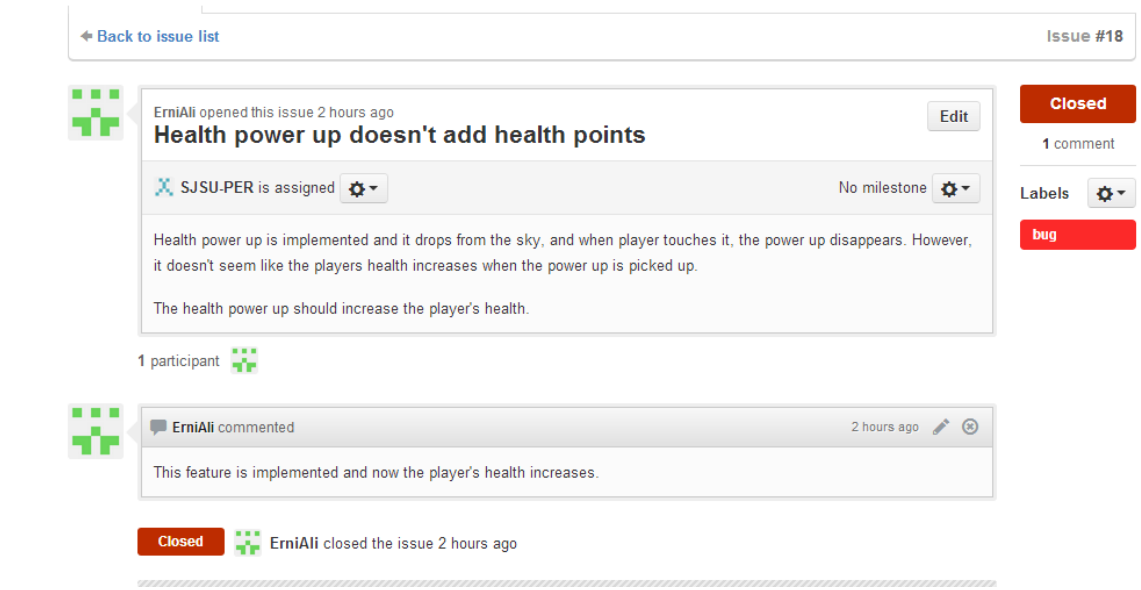

Back to issue list
Issue #9
ErniAli opened this issue a month ago
Items and powerups
Edit
Closed
1 comment
No one is assigned
No milestone
Labels
Items and powerups give additional features to the game.
enhancement
1 participant
ErniAli commented
2 hours ago
This enhancement is implemented with health powerup
Closed
ErniAli closed the issue 2 hours ago

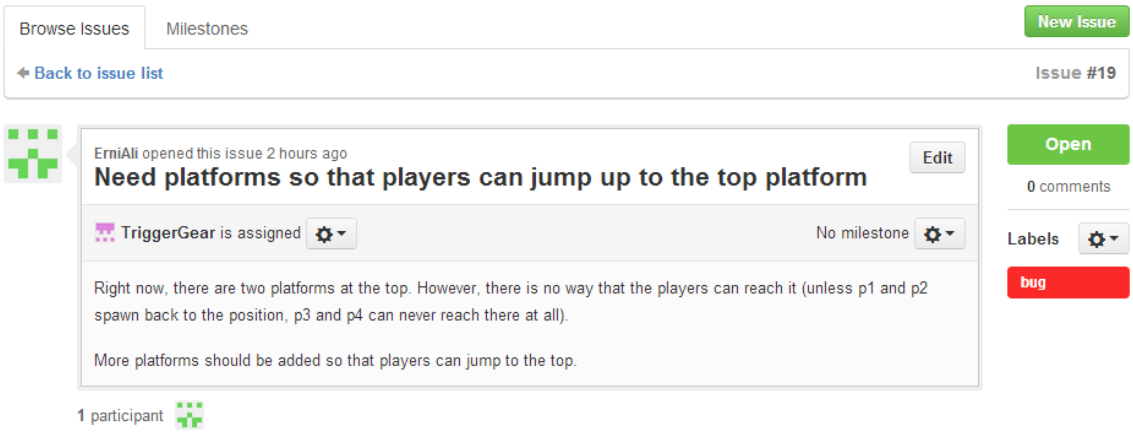

Back to issue list
Issue #10
ErniAli opened this issue a month ago
The game doesn't end (game termination)
Edit
Closed
1 comment
No one is assigned
No milestone
Labels
The game needs to know when to end and on what condition.
enhancement
1 participant
ErniAli commented
2 hours ago
The game ends now because game termination state is implemented.
Closed
ErniAli closed the issue 2 hours ago


Back to issue list
Issue #12
ErniAli opened this issue a month ago
Player animation
Edit
Closed
1 comment
No one is assigned
No milestone
Labels
Animation should be less awkward. sprite characters should be shown walking, etc.
enhancement
1 participant
ErniAli commented
2 hours ago
This enhancements is implemented
Closed
ErniAli closed the issue 2 hours ago
Back to issue list
Issue #13
ErniAli opened this issue a month ago
Player's body is still there when the player leaves
Edit
Closed
1 comment
No one is assigned
No milestone
Labels
When player leaves the game, another player can see that player who left. This needs to be fixed.
bug
1 participant
ErniAli commented
2 hours ago
Fixed as Lobby is integrated
Closed
ErniAli closed the issue 2 hours ago

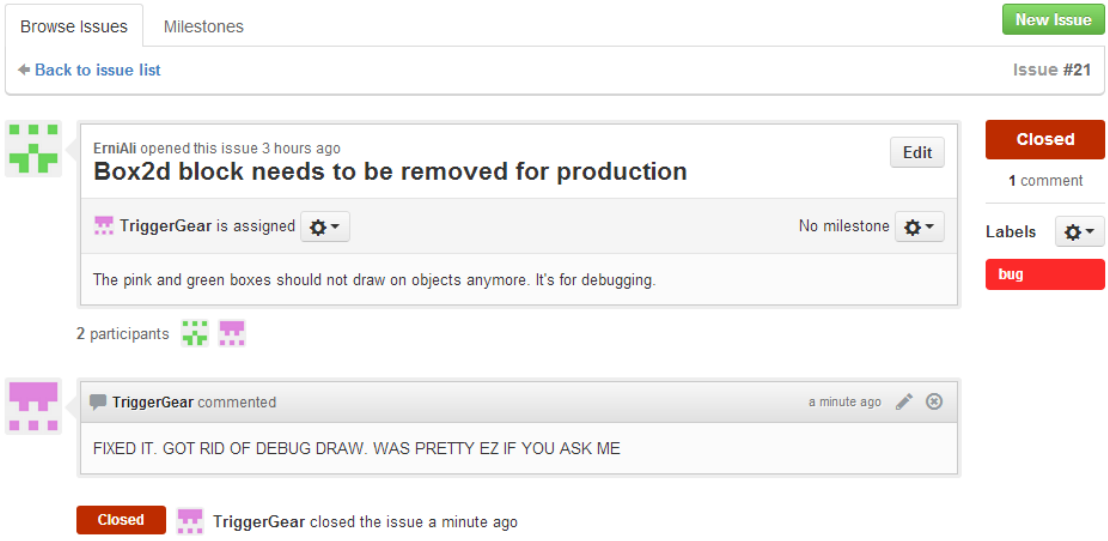

Back to issue list
Issue #14
ErniAli opened this issue a month ago
Explosion/Bomb is affected by gravity
Edit
Closed
1 comment
No one is assigned
No milestone
Labels
When you shoot a bomb and the bomb hits something, the explosion animation floats down becuase it's affected by gravity. Gravity needs to be turned off in this case.
bug
1 participant
ErniAli commented
a month ago
Fixed!
Closed
ErniAli closed the issue a month ago
Back to issue list
Issue #15
ErniAli opened this issue a month ago
Health Bar
Edit
Closed
1 comment
SJSU-PER is assigned
No milestone
Labels
Presenting health bar that decreases will make it clearer what's happening.
enhancement
1 participant
ErniAli commented
2 hours ago
Players health are shown in a table.
Closed
ErniAli closed the issue 2 hours ago

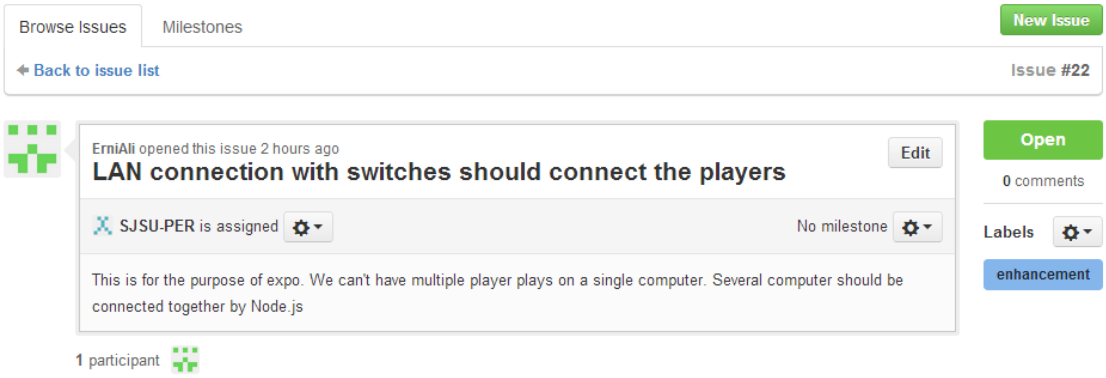

Back to issue list
Issue #16
ErniAli opened this issue 7 days ago
Jump issues
Edit
Closed
1 comment
TriggerGear is assigned
No milestone
Labels
There are certain times when characters cannot jump. Most importantly for p3 and p4. They cannot jump at all.
bug
1 participant
ErniAli commented
2 hours ago
This is fixed by changing the spawning points
Closed
ErniAli closed the issue 2 hours ago
Back to issue list
Issue #17
ErniAli opened this issue 2 hours ago
Animation does not show properly on remote player screen
Edit
Closed
1 comment
ErniAli is assigned
No milestone
Labels
Play the game as two player.
Show the window side by side.
Move p1 and notice that when p1 moves and it shows the running animation and it shows on the local player screen, the remote player still shows the running animation instead of the standing up animation.
This needs to be fixed so that animation shows consistently.
bug
1 participant
ErniAli commented
2 hours ago
This is fixed.
Closed
ErniAli closed the issue 2 hours ago

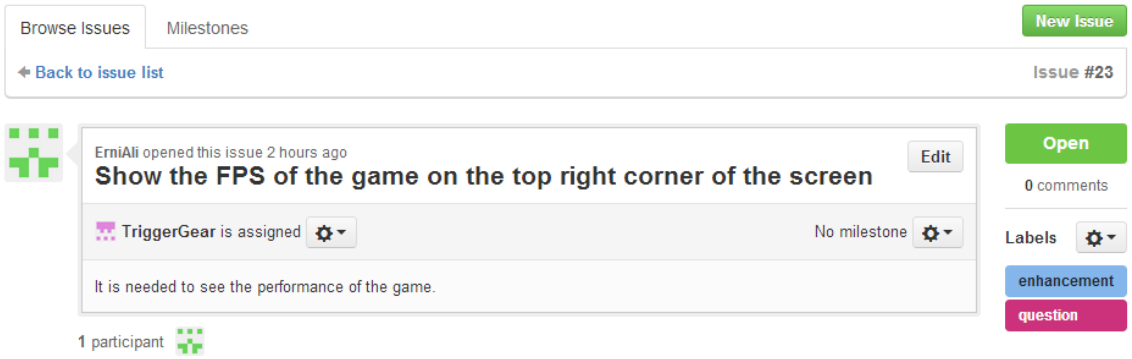

Back to issue list
Issue #18
ErniAli opened this issue 2 hours ago
Health power up doesn't add health points
Edit
SJSU-PER is assigned
No milestone
Health power up is implemented and it drops from the sky, and when player touches it, the power up disappears. However, it doesn't seem like the players health increases when the power up is picked up.
The health power up should increase the player's health.
1 participant
ErniAli commented
2 hours ago
This feature is implemented and now the player's health increases.
Closed
ErniAli closed the issue 2 hours ago
Closed
1 comment
Labels
bug

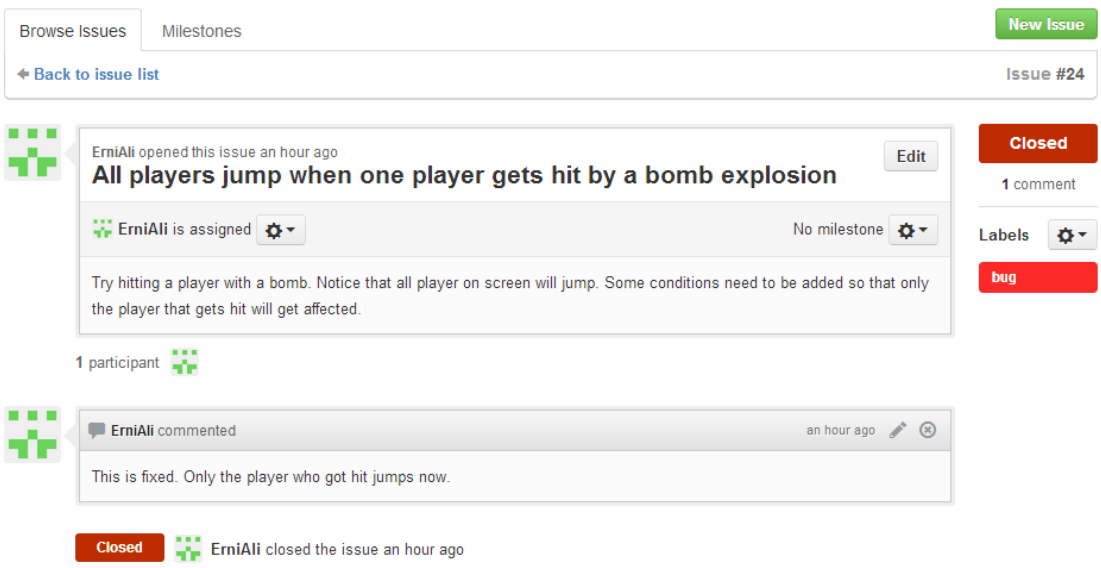

Browse Issues
Milestones
New Issue
Back to issue list
Issue #19
ErniAli opened this issue 2 hours ago
Need platforms so that players can jump up to the top platform
Edit
TriggerGear is assigned
No milestone
Right now, there are two platforms at the top. However, there is no way that the players can reach it (unless p1 and p2 spawn back to the position, p3 and p4 can never reach there at all).
More platforms should be added so that players can jump to the top.
1 participant
Open
0 comments
Labels
bug

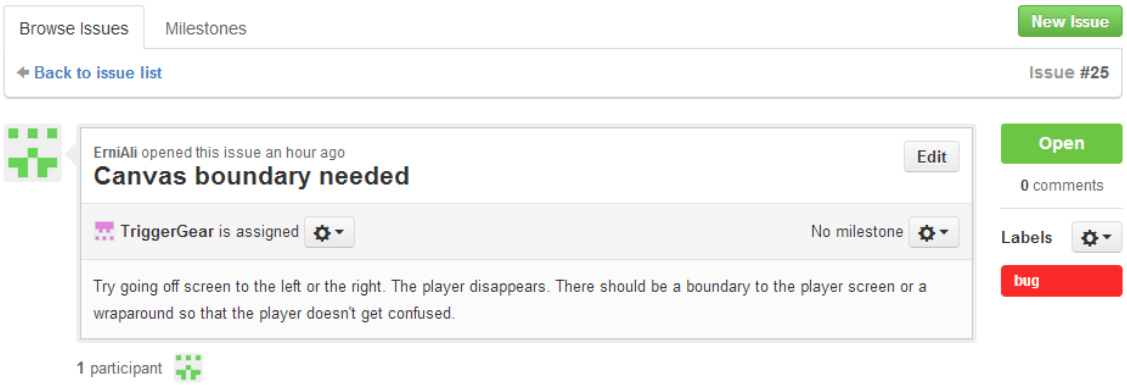

Browse Issues
Milestones
New Issue
Back to issue list
Issue #20
ErniAli opened this issue 2 hours ago
Edit
Players health should decrease when the player falls off the platform
Open
0 comments
TriggerGear is assigned
No milestone
Labels
enhancement
Notice that when a player falls off the platform, their health does not decrease.
It should decrease by one.
1 participant

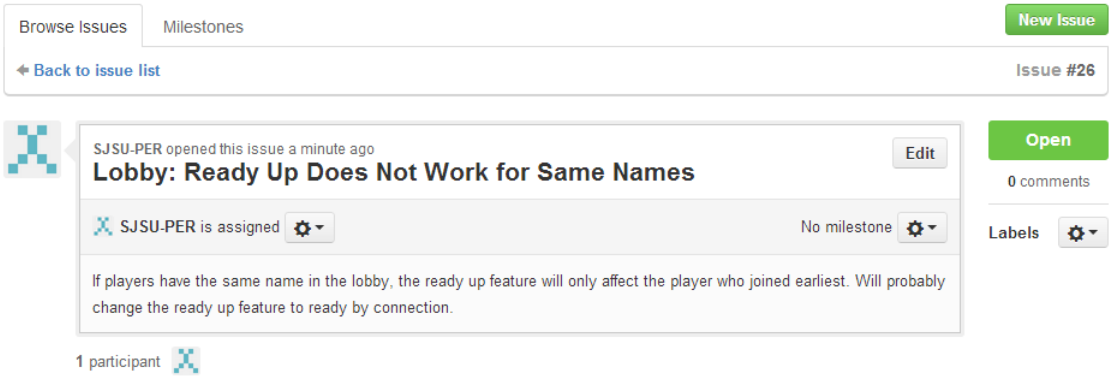

Browse Issues
Milestones
New Issue
Back to issue list
Issue #21
ErniAli opened this issue 3 hours ago
Edit
Box2d block needs to be removed for production
Closed
1 comment
TriggerGear is assigned
No milestone
Labels
bug
The pink and green boxes should not draw on objects anymore. It's for debugging.
2 participants
TriggerGear commented
a minute ago
FIXED IT. GOT RID OF DEBUG DRAW. WAS PRETTY EZ IF YOU ASK ME
Closed
TriggerGear closed the issue a minute ago


Browse Issues
Milestones
New Issue
Back to issue list
Issue #22
ErniAli opened this issue 2 hours ago
LAN connection with switches should connect the players
Edit
SJSU-PER is assigned
No milestone
This is for the purpose of expo. We can't have multiple player plays on a single computer. Several computer should be connected together by Node.js
Open
0 comments
Labels
enhancement
1 participant


Browse Issues
Milestones
New Issue
Back to issue list
Issue #23
ErniAli opened this issue 2 hours ago
Show the FPS of the game on the top right corner of the screen
Edit
TriggerGear is assigned
No milestone
It is needed to see the performance of the game.
Open
0 comments
Labels
enhancement
question
1 participant


Browse Issues
Milestones
New Issue
Back to issue list
Issue #24
ErniAli opened this issue an hour ago
All players jump when one player gets hit by a bomb explosion
Edit
ErniAli is assigned
No milestone
Try hitting a player with a bomb. Notice that all player on screen will jump. Some conditions need to be added so that only the player that gets hit will get affected.
Closed
1 comment
Labels
bug
1 participant
ErniAli commented
an hour ago
This is fixed. Only the player who got hit jumps now.
Closed
ErniAli closed the issue an hour ago


Browse Issues
Milestones
New Issue
Back to issue list
Issue #25
ErniAli opened this issue an hour ago
Canvas boundary needed
Edit
Open
0 comments
TriggerGear is assigned
No milestone
Labels
Try going off screen to the left or the right. The player disappears. There should be a boundary to the player screen or a wraparound so that the player doesn't get confused.
bug
1 participant


Browse Issues
Milestones
New Issue
Back to issue list
Issue #26
SJSU-PER opened this issue a minute ago
Lobby: Ready Up Does Not Work for Same Names
Edit
Open
0 comments
SJSU-PER is assigned
No milestone
Labels
If players have the same name in the lobby, the ready up feature will only affect the player who joined earliest. Will probably change the ready up feature to ready by connection.
1 participant

Chapter 8 Conclusion and Future Work

In the end, the Hyper Bout team has accomplished the goal that was first defined before the project was started. One of the members of the team received a job offer and is already working full time from showcasing this project.

Hyper Bout was chosen to be implemented using modern web technologies including Node.js, HTML5, JavaScript, and Box2D with the purpose of educating the Hyper Bout team about web technologies and learn new skills to get a job in the industry. With the completion of Hyper Bout, the Hyper Bout team has learned essential web development skills. Not only that, the team also learned about handling and completing an entire software development lifecycle. Hyper Bout met all of its original objectives which are to provide four player support and also handle real time interactions of the players, has minimal user lag, and also is developed with HTML5 paired with Box2D, Node.js, and JavaScript.

Having met all of its original objective, the Hyper Bout team wishes to keep working on the project by adding more features, such as add-ons and more power-ups. Add-ons will allow Hyper Bout to be marketable so users will be given the option to buy add-ons to be able to win the game more easily by having additional items in the game. While having more power-ups will improve the variety of the game.

References

Airportyh (2013). *Github airportyh testem.* Retrieved from https://github.com/airportyh/testem

Alderidge, Greg. "How Much Of The Video Games Market Is Digital?" The Sixth Axis. 6 July 2012. 20 Apr. 2013. http://www.thesixthaxis.com/2012/06/07/how-much-of-the-video-games-market-is-digital/

Anderson, Nate. 2007. "Gaming to Surge 50 Percent In Four Years." Ars Technica. 30 Aug. 2007. 20 Apr. 2013. http://arstechnica.com/gaming/2007/08/gaming-to-surge-50-percent-in-four-years-possibly/

Bijin Chen; Zhiqi Xu; , "A framework for browser-based Multiplayer Online Games Programs," *Internet Computing, IEEE* , vol.14, no.6, pp.80-83, Nov.-Dec. 2010. doi: 10.1109/MIC.2010.145

Chen, T.C.L.; Verbrugge, C.; , "A protocol for distributed collision detection," *Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on* , vol., no., pp.1-6, 16-17 Nov. 2010. doi: 10.1109/NETGAMES.2010.5680187

Finley, Klint. 2012. 5 Ways to Tell Which Programming Languages are Most Popular. Retrieved from http://readwrite.com/2012/06/05/5-ways-to-tell-which-programming-lanugages-are-most-popular#awesm=~omJ3Wa5NfWn7YE

GitHub. 2012. Introducing Organizations. Retrieved from https://github.com/blog/674-introducing-organizations

Helm, Clemens (2013). *Testing Tuesday #2: From Test-Driven Development to Behavior-Driven Development*. Retrieved from http://blog.codeship.io/2013/04/22/from-tdd-to-bdd.html

Jenkins, David (2011). Why Japan hates first person shooters: Yakuza's Toshihiro Nagoshi talks Binary Domain. Retrieved from http://metro.co.uk/2011/09/19/why-japan-hates-first-person-shooters-yakuzas-toshihiro-nagoshi-talks-binary-domain-156025/

Lawler, R. (2012, June 30). How Apple Won the War Against Flash. *TechCrunch.com.* Retrieved February 24, 2012, from http://techcrunch.com/2012/06/30/steve-jobs-war-against-flash/

McAnlis, C. (2012, June 27). GRITS: PvP Gaming with HTML5. *Google I/O 2012*. Retrieved February 24, 2012, from https://developers.google.com/events/io/2012/sessions/gooio2012/210/

Mitchell, Bradley (2013). *What is a Computer Ping Test.* Retrieved from http://compnetworking.about.com/od/homenetworktroubleshooting/f/pingtest.htm

Moore, Dennis. 2012. IT Jobs Recovery Continues, Picks Up Steam. Retrieved from http://www.enterpriseirregulars.com/35176/it-jobs-recovery-continues-picks-up-steam/

Pivotal Labs (2013). *Jasmine*. Retrieved from http://pivotal.github.io/jasmine/

PricewaterhouseCoopers. 2011. Global mobile gaming revenues (US$ mn) and share of total video game segment (%) 2008-2017. Retrieved from http://www.pwc.com/gx/en/global-entertainment-media-outlook/segment-insights/video-games.jhtml

Shankar, A. (2012). *Pro HTML5 games*. New York, NY: Apress.

Snow, Blake (2012). Why Console Gaming is Dying. Retrieved from http://www.cnn.com/2012/11/09/tech/gaming-gadgets/console-gaming-dead

Stephany (2008). Leaked 'Smash Bros Brawl' Sales Figures from Japan. Retrieved from http://www.gamefront.com/leaked-smash-bros-brawl-sales-figures-from-japan/

Tilkov, S.; Vinoski, S.; , "Node.js: Using JavaScript to Build High-Performance Network using WebGL and WebSocket," *Multimedia Technology (ICMT), 2011 International Conference on* , vol., no., pp.471-474, 26-28 July 2011 doi: 10.1109/ICMT.2011.6001673

W3C. 2012. HTML5 Definition Complete, W3C Moves to Interoperability Testing and Performance. Retrieved from http://www.w3.org/2012/12/html5-cr

W3schools (2013). *W3C Tutorial*. Retrieved from http://www.w3schools.com/w3c/default.asp

Appendices

Appendix A – Communication Variety

There were several modern collaborating tools that the team used in order to work on a task together. The following is a list of tools that were used.

Collabedit

Collabedit is an online text editor that allows multiple people to collaborate online on a single text editor. Collabedit is not only a simple text editor as it additionally supports various programming languages and has the ability to color code these languages so that the team is able to communicate online and code on a single document with ease. The team frequently uses Collabedit to collaborate together on code to implement features and give opinions about one another's code, especially on how certain features are implemented.

GitHub

GitHub was used in order to host Hyper Bout's code and also acted as a software configuration management tool. By using GitHub, the team was able to share code easily by simply pushing and pulling to the central repository without having to email each other the zip file of the code.

Not only that, GitHub provided a very nifty feature to be able to roll back code to the previous working state. This is very useful when one of the team members has written code that broke the application which needed to result in a rollback.

GitHub Bug Tracking Tool

Since Hyper Bout's project is on GitHub, the team decided that there was no point in using another tool such as Bugzilla in order to do bug tracking since GitHub already provided its own feature to do this. The GitHub bug tracking tool allowed the team to submit defects related to the project by opening new issues. Additionally, defects can be marked with different labels, such as a "bug", "enhancement", etc. This issue can also be assigned to the team members, so the team knew who was responsible for certain defects. When a team member submit a new issue, the other members are notified through email. That way, everyone knows that there are new issues that need to be resolved.

Once the issue is resolved, the person responsible for the issue can close the issue and another email is sent by Github to let the collaborators know that the issue is closed. The rest of the team can then do a final check to ensure that the same defect doesn't occur or replicate.

Dropbox

While our GIT repository handles the code oriented aspects of Hyper, another kind of repository was used to handle all of our document oriented deliverables. For document oriented deliverables, Dropbox was utilized to maintain multiple version of the documents for submission. Therefore, instead of consistently emailing documents back and forth, the group always had an active copy of the latest document and can jump back to previous development iterations of the same document on Dropbox.

Conferencing through Google Hangout

The team used Google Hangout because of its easiness to set up and because of the different features it provides including the ability to screen share during a team meeting. In addition, since all the team members already have Google accounts, this was a tool that was easily adopted by the team without having to create a new account for a separate web tool.

Google Hangout was used by the team for meetings during the weekends in order to meet the planned objectives and finished tasks that were assigned for the week. This allowed the team to collaborate together in writing code easily. The way the team conducted the meetings was by having one of the team members share his or her screen and that team member would be the one writing the code while the other team members provided feedback, support, and thoughts on how to write the code.

In a way, the team used a pair programming approach by using Google Hangout which resulted in a high quality code and an easier time in solver bugs and problems found. Not only that, Google Hangout also saved the team members' time because the meeting need not be conducted in person and there was no time wasted in commuting to a designated meeting place.

Scheduling and Communication Tool

Gmail and Gmail Chat

Hyper Bout team members used Gmail as a primary email client. As such, Gmail provides the desirable means for the team to communicate with each other. Whenever there were urgent matters regarding the project or simply thoughts on submissions or

deliverables, emails were used as the primary communication tool. It was essential that the group was very prompt on replying to these emails so that problems could be resolved quickly and effectively. While emails are effective, they are limited by their very definition of being inherently asynchronous. To address this limitation, Gmail has a built in online chat system, appropriately named “Gmail Chat”, which the team utilized to achieve synchronous communication. Using this feature, the team can instantly chat and communicate with each another in a synchronous manner whenever we were online. Through Gmail Chat, ideas and thoughts can be sent in a more expedient manner than in email. Additionally, Gmail Chat provides the means for a more natural communication method. For this reason, Gmail Chat was primarily used to handle a group meeting like environment online while email was primarily used to handle quick file sharing, status updates, not so urgent questions, or to contact members when they were not online or reachable by phone. Google also provides a “hangout” feature similar to Skype to construct a more natural face to face online discussion environment. The team makes use of this feature as a conferencing tool.

Additionally, email and canvas assignment submissions are also used in order to inform our professor of our status through writing individual weekly reports. These weekly reports act as status updates that consist of 3 tasks that the individual plans to achieve by the end of the week and the status of the tasks from the previous week. Not only is the advisor kept informed about the progress of our senior project, but these reports gave us an idea of where the team currently is in development.

Telephone

Phone numbers were exchanged for easier communication. Phone numbers were used in more urgent situations, such as trying to find a project adviser or informing other members that one of our team members cannot make it to the adviser's office hours or cannot finish some tasks on time. In the situation that a team member cannot attend the weekly meeting, the missing member is informed by phone of the content he/she had missed for that meeting.

Google Calendar

To handle all of our schedule meetings, Google Calendar was used to set up times and locations to meet up at. As Google Calendar is linked up to the team's Android devices, 30 minute reminders are given to our phones in situations that members forget that there is a meeting. In addition to using Google Calendar to handle meeting scheduling, deadlines and milestones have been added to the group's Google Calendar. Seeing the milestones depicted on the calendar gave us an idea of where we were in the project and whether or not the group was ahead or behind schedule.

Weekly Face to Face Meeting

Hyper Bout team members know each other very well and took similar classes during the semester. Hence, it was easy to be able to talk to each other face to face in a hallway conversation like manner. The benefits to these types of conversations are that they are cheap and effective in resolving simple issues and delegating simple tasks. However, the downside to these hallway conversations is that there is sometimes a loss of

information or misunderstandings in verbal exchange. To address the limitations of informal verbal exchanges, weekly hour long meetings are arranged in order to discuss the project and plan the next course of action. These meetings are generally conducted in the SCE room, the Student Union, or a reserved library room in order to limit exterior distractions during the meeting. We also have a weekly face-to-face meetings with our project adviser in order to inform him of our status update and what we have done in the duration of the previous week.

Appendix B – Techniques, Skills, and Modern Engineering Tools

Several techniques were used to identify solutions to the various problems the team encountered. The following text reviews the techniques and skills used and the reasoning behind them.

Hosting Problem

Hyper Bout required an online host so that several users could interact with the game simultaneously. The team had decided to use Nodejitsu in order to host the project, however several obstacles were encountered during the process of hosting the project. The process took some time, but our team was finally able to host Hyper Bout and have the project accessible to the public.

The team found out that there were numerous changes that needed to be made to the code in order to enable core features in Hyper Bout to be able to work properly. The team decided to look into different hosting platforms in order to solve this problem first. The team also found out that the internet connection at school would be unreliable during the day of the expo.

Therefore, in order to solve this hosting problem, the team decided to simply go with a LAN configuration during the expo day. A router will be used in order to connect all the computers together to allow the multiplayer feature of the project. Four laptops would be used as clients while a fifth would be used as the server.

Character Invincibility Period Problem

At first, the team wanted to implement sophisticated features where the player character would blink whenever a projectile hit the player. However, this blinking would

increase the load of the client system. More animation was required to achieve this and it would have taken additional time to implement this feature. Considering all these disadvantages, the team came up with a new solution to draw a shield on top of the player instead whenever the player was on an invincibility period. The player with the shield would not lose health whenever he or she was hit. This alternative solution was simple enough to implement so that it didn't take time away from the team.

Engineering Practices

Throughout the course of the project, Team Hyper Bout used many modern engineering tools to aid in the software engineering lifecycle. The engineering tools that were used are listed and explained below:

Google Hangout

As a state-of-the-art tool, the team used Google Hangout because of its easiness to set up and because of the different features it provides, such as the ability to screen share during a team meeting. In addition, since all the team members already have Google accounts, this is a tool that was easily adopted by the team without having to create a new account for a separate web tool.

Google Hangout was used by the team for meetings during the weekends in order to meet the planned objectives and finished tasks that were assigned for the week. This allowed the team to collaborate together in writing code easily. The way the team conducted the meetings was by having one of the team members share his or her screen and that team member would be the one writing the code while the other team members provided feedback, support, and thoughts on how to write the code.

In a way, the team used a pair programming approach by using Google Hangout which resulted in a high quality code and an easier time in solver bugs and problems found. Not only that, Google Hangout also saved the team members' time because the meeting need not be conducted in person and there was no time wasted in commuting to a designated meeting place.

GitHub

GitHub was used in order to host Hyper Bout's code and also acted as a software configuration management tool. By using GitHub, the team was able to share code easily by simply pushing and pulling to the central repository without having to email each other the zip file of the code.

Not only that, GitHub provided a very nifty feature to be able to roll back code to the previous working state. This is very useful when one of the team members has written code that broke the application which needed to result in a rollback.

Chrome Developer Tools

The Chrome Developer Tools were extensively used throughout the project. This tool is very useful in order to debug JavaScript code, a core programming language used by Hyper Bout. By using the Chrome Developer Tools, the team were able to find out what different variables contained during the execution of the program. Not only that, the team was also able to find out if certain code was executed during runtime.

Node.js Command Line Tool

The Node.js command line tool that comes with the installation of Node.js was very crucial during the development of Hyper Bout. It saved the team's time in having to set up the windows command line tool to work as expected. By using the Node.js command line tool, the team was able to run the project easily and also install the required Node.js packages using the npm (Node Package Manager) command.

Sublime Text 2

While Sublime Text 2 is not an IDE (Integrated Development Environment), it is a very useful text editor that is able to color code Javascript and HTML code correctly. Not only that, Sublime Text 2 allowed the team in order to navigate through files and folders easily. This sped up development time and also provided a uniform text editor for the team. Also, there was virtually no learning curve for this text editor at all because it is very lightweight and easy to use.

GitHub Bug Tracking Tool

Since Hyper Bout's project is on GitHub, the team decided that there was no point in using another tool such as Bugzilla in order to do bug tracking since GitHub already provided its own feature to do this. The GitHub bug tracking tool allowed the team to submit defects related to the project by opening new issues. Additionally, defects can be marked with different labels, such as a "bug", "enhancement", etc. This issue can also be assigned to the team members, so the team knew who was responsible for certain defects.

Once the issue is resolved, the person responsible for the issue can close the issue and a final check by the team is done to ensure that the same defect doesn't occur or replicate.

Nodejitsu

Nodejitsu was used as Hyper Bout's hosting platform. Nodejitsu allows the deployment of Node.js application. Nodejitsu was chosen by the team because it was pretty well known and there was a lot of support. It also gave a 30 days free trial. It required a bit of a learning curve in order to host Hyper Bout and even when it was hosted in the end, there were features that didn't work correctly. However, knowing how to deploy a Node.js application is a very good learning experience for the team, especially in the professional environment.

Collabedit

Collabedit is an online text editor that allows multiple people to collaborate online on a single text editor. Collabedit is not only a simple text editor, it supports various programming languages and has the ability to color code these languages so that the team is able to communicate online to write together on a single document with ease. The team frequently uses Collabedit to implement features and give opinions about one another's code, especially on how certain features are implemented.

Appendix C – Design, 4+1 Model, Architecture, and OOD Models

4+1 Model

1. Logical

Class Diagram

Refer to Section 3.1.3.

Sequence Diagram

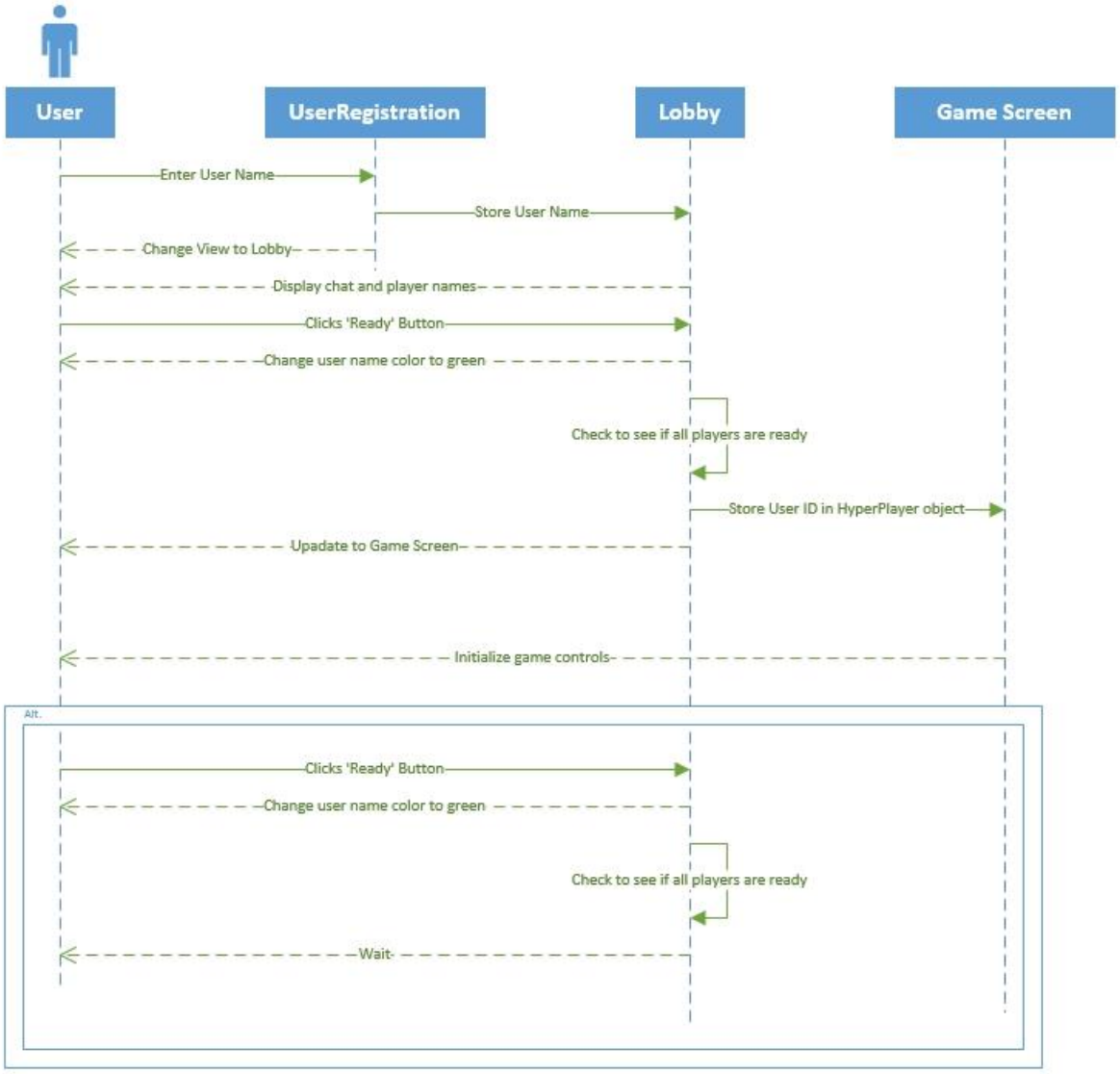


Figure 49: Hyper Bout Sequence Diagram

Development: Component Diagram

Refer to Section 4.2.

Process: Activity Diagram

Refer to Section 3.1.2.

Physical: Deployment

Below depicts Hyper Bout's deployment diagram:

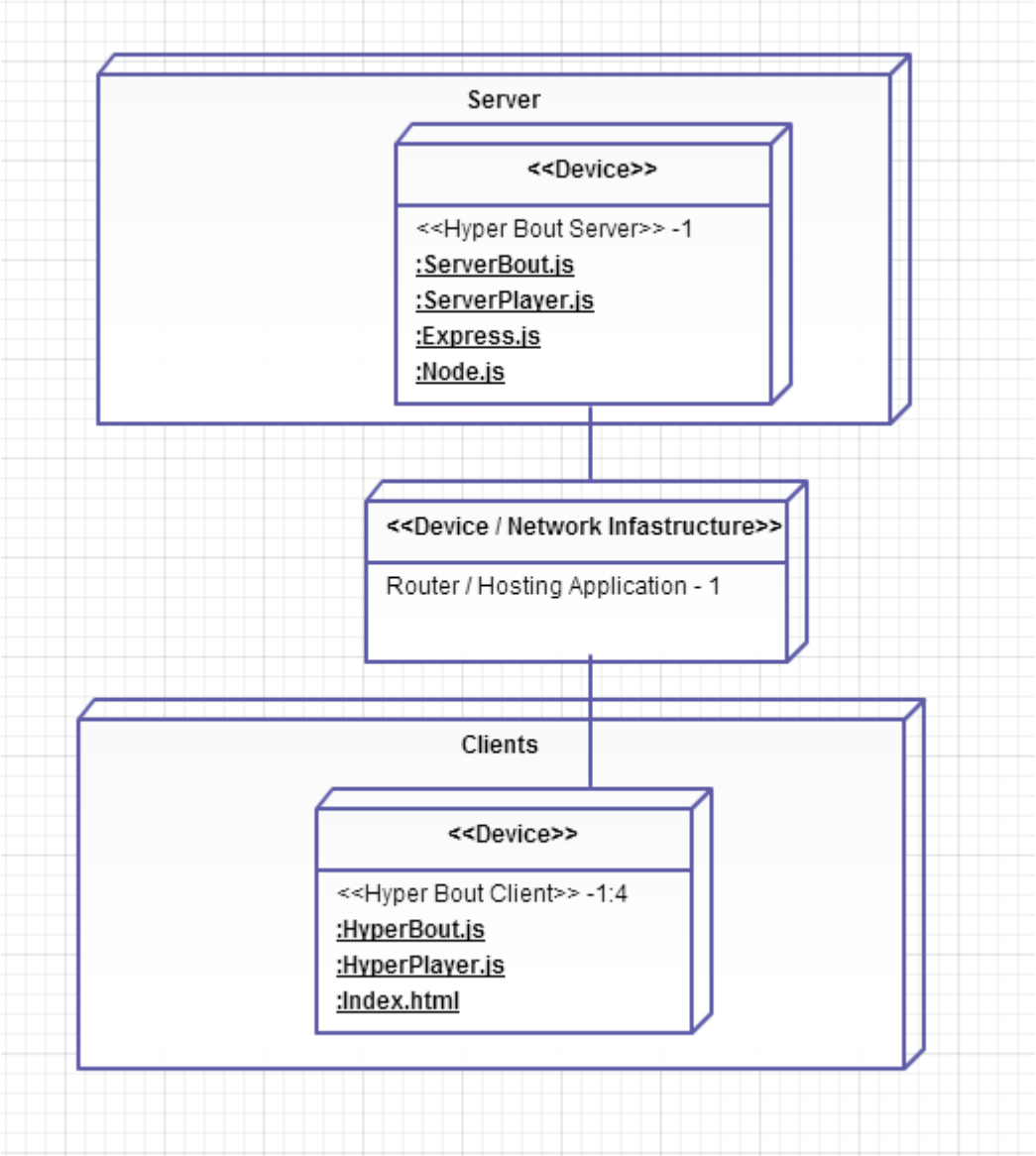


Figure 50: Hyper Bout Deployment Diagram

Scenarios: Use Cases

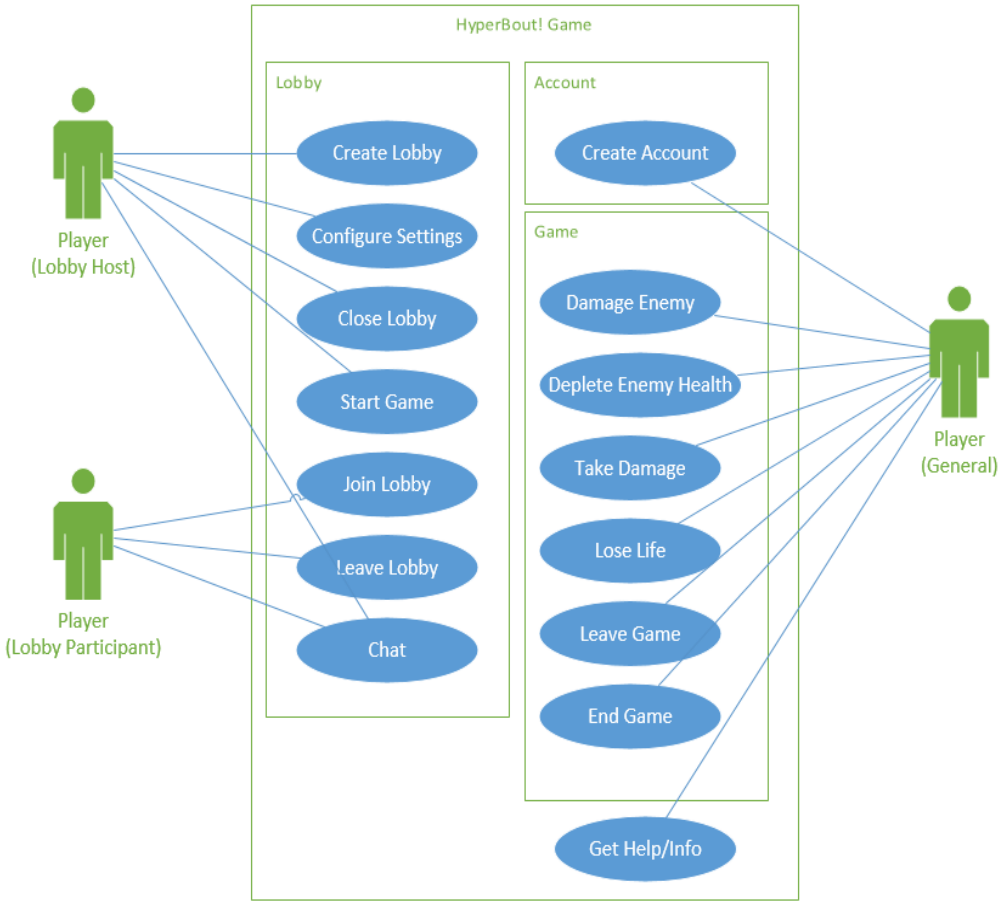


Figure 51: Hyper Bout Use Cases

The use case model portrays two kinds of players within the lobby system (Lobby Host/Participant) which extend Player during account creation and game play.

Use Case Descriptions

Use Case Name	UC #1: Create Lobby
Participating Actors	Initiated by *Player [Lobby Host]*
Flow of Events	1. *Player [Lobby Host]* logs into the Hyper Bout! System and clicks on the "Create Lobby" button. 2. *Player [Lobby Host]* supplies a "Lobby Name" that will be displayed on the server lobby list. 3. The lobby GUI is brought up to the *Player [Lobby Host]* and has his/her name designated as the lobby host. 4. The "Lobby Name" with the number of users in the lobby is displayed on the lobby list server. 5. *Player [Lobby Host]* waits until at least one other *Player [Lobby Participant]* joins the lobby before the game can start.
Entry Condition	• *Player [Lobby Host]* is logged in.
Exit Condition	• *Player [Lobby Host]'s* lobby appears on the lobby list screen.

Use Case Name	UC #2: Configure Settings
Participating Actors	Initiated by *Player [Lobby Host]*
Flow of Events	1. *Player [Lobby Host]* clicks on the "configure game" button on the lobby interface. 2. Several options appear from which the *Player [Lobby Host]* can change: a. Game Type b. Time Limit c. Max Lives d. Stage Selection 3. Once game options have been configured, *Player [Lobby Host]* selects "confirm" 4. The new game configuration is now displayed on the lobby interface so all players know what game will be played.
Entry Condition	• *Player [Lobby Host]* has an existing lobby.
Exit Condition	• Game settings have been changed.

Use Case Name	UC #3: Close Lobby
Participating Actors	Initiated by *Player [Lobby Host]*
Flow of Events	1. *Player [Lobby Host]* decides he/she wishes to quit the game. *Player [Lobby Host]* clicks on the "close lobby" button. 2. Hyper Bout! provides a confirmation warning informing that the lobby will be closed and all players will be kicked out. *Player [Lobby Host]* clicks "Yes" to close the lobby. 3. All present *Player [Lobby Participant*] are kicked from the lobby and the lobby is closed. 4. The lobby is removed from the lobby list server.
Entry Condition	• *Player [Lobby Host]* has an existing lobby.
Exit Condition	• *Player [Lobby Host]'s* lobby is removed from the server lobby list and players are kicked.

Use Case Name	UC #4: Start Game
Participating Actors	Initiated by *Player [Lobby Host]*
Flow of Events	1. At least 1 *Player [Lobby Participant]* joins *Player [Lobby Host]'s* game. 2. The "Start Game" button becomes clickable. *Player [Lobby Host]* clicks on the "Start Game" button. 3. The lobby is removed from the server lobby list as new players cannot join the lobby in the middle of a game. 4. *Players [General]* are now at the game interface screen and ready to play.
Entry Condition	• *Player [Lobby Host]* has an existing lobby. • There is at least one *Player [Lobby Participant]* in the lobby.
Exit Condition	• *Players [General]* are brought into the game with their own HUDs. • *Player [Lobby Host]'s* lobby is removed from the server lobby list.

Use Case Name	UC #5: Join Lobby
Participating Actors	Initiated by *Player [Lobby Participant]*
Flow of Events	1. *Player [Lobby Participant]* logs into the Hyper Bout! System and clicks on the "Join Lobby" button. 2. *Player [Lobby Participant]* is brought to the lobby list interface and selects an available lobby (i.e. one with < 4/4 players). 3. *Player [Lobby Participant]* joins the lobby and has his/her name appear in the lobby player list. 4. The lobby list is updated so that the number of players in the joined lobby is raised by 1. 5. *Player [Lobby Participant]* can select character / projectile to use in game.
Entry Condition	• *Player [Lobby Participant]* is logged in. • The lobby list has at least one open lobby. • The lobby to be joined has less than 4 people.
Exit Condition	• *Player [Lobby Participant]* is added to the lobby player list. • +1 is added to the number of players of that lobby in the lobby list.

Use Case Name	UC #6: Leave Lobby
Participating Actors	Initiated by *Player [Lobby Participant]*
Flow of Events	1. *Player [Lobby Participant]* decides to leave the lobby and clicks on the "Leave Lobby" button. 2. A confirmation message is brought up and the *Player [Lobby Participant]* confirms that he/she wishes to leave. 3. *Player [Lobby Participant]* has his/her name disappear from the lobby player list. 4. The lobby list is updated so that the number of players in the joined lobby is reduced by 1.
Entry Condition	• *Player [Lobby Participant]* is currently in a lobby
Exit Condition	• *Player [Lobby Participant]* is removed from the lobby player list. • -1 is subtracted from the number of players of the lobby in the lobby list.

Use Case Name	UC #7: Chat
Participating Actors	Initiated by/Communicates with: *Player [Lobby Participant]* / *Player [Lobby Host]*
Flow of Events	1. *Player [General]* wishes to convey a message to everyone in the lobby so he/she types in the text area the message he/she wants to send. 2. *Player [General]* clicks "Submit" 3. *Player's [General]* message is uploaded onto the lobby text area for all users in the lobby to read.
Entry Condition	• *Player [General]* is in an existing lobby.
Exit Condition	• *Player's [General]* message is displayed on the lobby text area.

Use Case Name	UC #8: Create Account
Participating Actors	Initiated by *Player [General]*
Flow of Events	1. *Player [General]* supplies a username to which other people on Hyper Bout! will see 2. *Player [General]* supplies a password and retypes it again. 3. *Player [General]* clicks on "Submit". 4. The system adds *Player [General]'s* account to the database.
Entry Condition	• *Player [General]* does not have an account / does not wish to use a temporary guest account.
Exit Condition	• *Players [General]* log in information is stored in the database.

Use Case Name	UC #9: Damage Enemy
Participating Actors	Initiated by *Player* Communicates with *Player [Enemy]*
Flow of Events	1. *Player* throws a projectile at *Enemy*. 2. *Player's* hit on *enemy* is detected and registered on the authoritative server. 3. *Enemy* loses 1 bar of health (possibly more depending on if the *Player* has an active power up). 4. *Enemy* goes into a 2 second invincibility period so he/she cannot get hit for a period of time.
Entry Condition	• *Player* is currently in an active game with another active *Player [Enemy]* • The hit is detected on the authoritative server.
Exit Condition	• *Enemy* loses some amount of health according to the object/power up that the user hit him by.

Use Case Name	UC #10: Deplete Enemy Health
Participating Actors	Initiated by *Player* Communicates with *Player [Enemy]*
Flow of Events	1. *Player* throws a projectile at *Enemy*. 2. *Player's* hit on *enemy* is detected and registered on the authoritative server. 3. *Enemy* loses the rest of his/her health and is flashed off the screen. a. If the *Enemy* has more lives or game type is set to infinite lives, the *Enemy* reappears a few seconds later with full health. *Enemy* goes into a 2 second invincibility period so he/she cannot get hit for a period of time. 4. *Player* gets 1 point for eliminating *Enemy*. 5. *Enemy* loses 1 point for getting eliminated.
Entry Condition	• *Player* is currently in an active game with another active *Player [Enemy]* • The hit is detected on the authoritative server. • *Enemy* has low enough health to be eliminated in one hit.
Exit Condition	• *Enemy* loses a life and either stays out the match or rejoins with a new life according to game lobby setup. • *Player* scores 1 point for eliminating *Enemy*. • *Enemy* loses 1 point for being eliminated by *Player*.

Use Case Name	UC #11: Take Damage
Participating Actors	Initiated by *Enemy* Communicates with *Player*
Flow of Events	1. *Enemy* throws a projectile at *Player*. 2. *Enemy's* hit on *Player* is detected and registered on the authoritative server. a. *Players* can also lose health by landing on environmental hazards. 3. *Player* loses 1 bar of health (possibly more depending on if the *Enemy* has an active power up). 4. *Player* goes into a 2 second invincibility period so he/she cannot get hit for a period of time.
Entry Condition	• *Player* is currently in an active game with another active *Player [Enemy]* • The hit is detected on the authoritative server.
Exit Condition	• *Player* loses some amount of health according to the object/power up that the user hit him by.

Use Case Name	UC #12: Lose Life
Participating Actors	Initiated by *Enemy* Communicates with *Player*
Flow of Events	1. *Enemy* throws a projectile at *Player*. 2. *Enemy's* hit on *Player* is detected and registered on the authoritative server. 3. *Player* loses the rest of his/her health and is flashed off the screen. *a.* If the *Player* has more lives or game type is set to infinite lives, the *Player* reappears a few seconds later with full health. *Player* goes into a 2 second invincibility period so he/she cannot get hit for a period of time. 4. *Enemy* gets 1 point for eliminating *Player*. 5. *Player* loses 1 point for becoming eliminated.
Entry Condition	• *Player* is currently in an active game with another active *Player [Enemy]* • The hit is detected on the authoritative server. • *Enemy* has low enough health to be eliminated in one hit.
Exit Condition	• *Player* loses a life and either stays out the match or rejoins with a new life according to game lobby setup. • *Enemy* scores 1 point. *Player* loses 1 point.

Use Case Name	UC #13: Leave Game
Participating Actors	Initiated by *Player*
Flow of Events	1. *Player* decides to quit in the middle of an active game and clicks the "Quit Game" button. 2. A confirmation message appears to confirm *Player's* departure from the game. *Player* clicks "Yes". 3. *Player* is disconnected and all associated models are removed from the game.
Entry Condition	• *Player* is currently in an active game.
Exit Condition	• *Player* disconnects from game. • *Player's* in game model and thrown objects are removed.

Use Case Name	UC #14: End Game
Participating Actors	Initiated by System Communicates with *Players*
Flow of Events	*1.* Game terminates due to the game configurations when either (a) There is one user left standing in the "Stock" game type or (b) When time limit is reached. *2.* A winner is determined to be as (a) The last *player* standing in the Stock game type or (b) As the *player* with the most points by time limit. *3.* Players are brought to a postmortem report about how well they performed in the match *4.* After 20 seconds, *Players* are brought into the lobby and can play another match or leave if they wish to.
Entry Condition	• The match has ended due to specified game conditions.
Exit Condition	• A winner for the match is determined. • *Players* are brought from the game interface to the lobby interface.

Use Case Name	UC #15: Get Help/Info
Participating Actors	Initiated by *Player*
Flow of Events	1. *Player* visits the game website 2. *Player* clicks on the "Need Help" link on the bottom left portion of the screen. 3. An interface pops up with a pictorial description of how the game works.
Entry Condition	• *Player* has access to Hyper Bout! (no registration needed).
Exit Condition	• Help interface is displayed to *Player*

Use Case Matrix

*** Defined by Requirement (S-F-#) and Requirement (UC#)**

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	U14	U15
1								X							
2	X														
3					X										
4									X	X	X	X			
5									X	X	X	X			
6									X		X				
7										X		X			
8									X		X				
9									X		X				
10													X		
11													X		
12					X										
13					X										
14		X													
15									X	X	X	X			
16									X	X	X	X			
17									X	X	X	X			
18									X	X	X	X			
19	X		X	X	X	X									
20			X												
21	X				X										
22					X										
23				X											
24		X													
25															X
26							X								
27														X	

Object Oriented Design Models

The Object Oriented Design model in Hyper Bout can be seen in the classes inside the Hyper Bout project. There are 5 classes inside Hyper Bout. Three of those classes are client codes, which are the HyperBout.js, HyperPlayer.js, and HyperPowerUp.js. And the other two are server code, which are ServerBout.js and ServerPlayer.js.

In the Hyper Power up class of Hyper Bout inside HyperPowerUp.js file, generalization is applied. HyperPowerUp is essentially a parent class of all the power ups that are implemented inside Hyper Bout. For example, the health power up is an extension of Hyper Power Up.

All classes in Hyper Bout also communicate to one another in order to be able to work which depicts associations between this classes. For example, in order to be able to draw power ups and assign logic to these power ups, for example, increase player health when power up is picked up, the Hyper Bout class needs to communicate with the Hyper Power Up class.

The Hyper Bout project also implements the singleton design pattern where there is only one class with a single instance, HyperBout. The HyperBout class contains all the core engine, mechanic, and also logic that happens in the game. HyperBout is the one who handles and put everything together in the game. In other words, HyperBout depicts the singleton pattern because HyperBout is restricted to only have one instance and it is the class that coordinate actions across Hyper Bout.

The class diagram shown in Section 3.1.3 of this document details how the classes in Hyper Bout interact with one another.