

Sprint 3 Report

*Team Report*

*Group: Team B*

*Project: Crossing Streams*

*Course: Cosc 470 - Software Engineering*

*Report version 1.0*

*Submitted by: Billy Spelchan for Team B (Nov 30th)*

[**1. Project Vision**](#_wgets6xsshir)[**3**](#_wgets6xsshir)

[**2. Problem statements:**](#_hvdyvr22iov9)[**4**](#_hvdyvr22iov9)

[2.1 Sprint 1 - Problem statement](#_badij81gn1pw) [4](#_badij81gn1pw)

[2.2 Sprint 2 - Problem statement](#_krnlagynzyzl) [4](#_krnlagynzyzl)

[2.3 Sprint 3 - Problem statement](#_47t2ialb9krs) [4](#_47t2ialb9krs)

[2.4 Review and compare your original problem statement](#_53a5ecahwfzz) [5](#_53a5ecahwfzz)

[**3. Team Members**](#_icvtisbanfov)[**5**](#_icvtisbanfov)

[**4. Development Process**](#_owr9c7rwnxh)[**7**](#_owr9c7rwnxh)

[4.1 Description of the Agile Scrum process](#_ra0ly15bzv7v) [7](#_ra0ly15bzv7v)

[4.1.1 Product Backlog](#_4kzhbvbq9nr5) [7](#_4kzhbvbq9nr5)

[4.1.2 Sprint Planning](#_2rzuqupifhi5) [7](#_2rzuqupifhi5)

[4.1.3 Sprint Backlog](#_8jryhingm53) [8](#_8jryhingm53)

[4.1.4 Sprints](#_53md7516g76o) [8](#_53md7516g76o)

[4.1.5 Scrum](#_y2jpf6f2h4b1) [8](#_y2jpf6f2h4b1)

[4.1.6 Demos](#_k4c9dveyx9zu) [8](#_k4c9dveyx9zu)

[4.2 Project development and management platform (JIRA)](#_grusmv52kxny) [8](#_grusmv52kxny)

[4.3 Preliminary Communication Method](#_3rkalbe6hfej) [9](#_3rkalbe6hfej)

[**5. Glossary of terms**](#_eqt14bp9rfu1)[**10**](#_eqt14bp9rfu1)

[**6. Results**](#_7avv6mi254jw)[**12**](#_7avv6mi254jw)

[6.1 Requirement gathering and specifications:](#_moxry83hbb7t) [12](#_moxry83hbb7t)

[6.1.1 Functional requirement / Specification:](#_uhqb6gfi8uri) [13](#_uhqb6gfi8uri)

[6.1.2 Non Functional requirement / specification](#_51f59m3agi0u) [15](#_51f59m3agi0u)

[Datastore](#_ekcoftw0w0dd) [17](#_ekcoftw0w0dd)

[Website](#_ltf8fiq0jge) [17](#_ltf8fiq0jge)

[Patcher](#_8lwc5ukaenbg) [18](#_8lwc5ukaenbg)

[Game Client](#_tifkl2rx848t) [18](#_tifkl2rx848t)

[Game Server](#_5ltxl4z8asru) [19](#_5ltxl4z8asru)

[6.1.3 Specific Requirements](#_8xe4qva5lc0i) [20](#_8xe4qva5lc0i)

[6.2 Engineering Execution](#_d81xlv9umwnr) [21](#_d81xlv9umwnr)

[6.2.1 Team Software Integration Plan (best for continuous integration)](#_nhwp4764789s) [21](#_nhwp4764789s)

[6.2.2 Builds (procedure of code integration and building the final product, version control, software change control)](#_tuyx2csn1i61) [21](#_tuyx2csn1i61)

[6.2.3 Installation (procedures, user manual or user guide, source code or repository)](#_epa58hrb8mt8) [22](#_epa58hrb8mt8)

[6.3 Project development results](#_zfen6ckjqcf2) [22](#_zfen6ckjqcf2)

[6.3.1 Sprint planning / roadmap summary](#_r6a5ht3161ex) [22](#_r6a5ht3161ex)

[6.3.2 Burn down chart history from Sprint 0 to final](#_w7lxcy9vwsmv) [23](#_w7lxcy9vwsmv)

[6.3.3 Sprint velocity and hours](#_xzwimizdi5bl) [27](#_xzwimizdi5bl)

[6.3.4 Risk Tracking from Sprint 0 to final](#_afyvvylvwys0) [29](#_afyvvylvwys0)

[6.3.5 Summary of Retrospectives](#_ioedvdgeaivt) [29](#_ioedvdgeaivt)

[6.3.6 Summary of Scrum Meetings](#_7khgj2j31yk7) [31](#_7khgj2j31yk7)

[**7. Maintenance**](#_q8b0512gt8x5)[**37**](#_q8b0512gt8x5)

[7.1 Source Code locations (git repository)](#_nwl2zbd3kxgy) [37](#_nwl2zbd3kxgy)

[7.2 Build Script and Setup](#_fwru4flc3gnr) [37](#_fwru4flc3gnr)

[**8. Conclusion**](#_nmak2eekwpic)[**42**](#_nmak2eekwpic)

[8.1 Summary of your project results](#_1d86yozra8x) [42](#_1d86yozra8x)

[Sprint 1 - The Streams](#_6tn8e4hmtcj5) [42](#_6tn8e4hmtcj5)

[Sprint 2 - The Crossing](#_ljzn2b6rkbez) [42](#_ljzn2b6rkbez)

[Sprint 3 - The Finale](#_ksel3yyb5jwh) [42](#_ksel3yyb5jwh)

[8.2 Suggested list of improvements](#_lxoo58ez7spu) [43](#_lxoo58ez7spu)

[Sprint 1 - The Streams](#_8t5vwnkll5a7) [43](#_8t5vwnkll5a7)

[Sprint 2 - The Crossing](#_ctvng3ke7835) [43](#_ctvng3ke7835)

[Sprint 3 - The Finale](#_k1hs89gnc072) [43](#_k1hs89gnc072)

[8.3 Lessons learned](#_3nkgdnagas82) [43](#_3nkgdnagas82)

[Sprint 1 - The Streams](#_5jqirye5xk20) [43](#_5jqirye5xk20)

[Sprint 2 - The Crossing](#_vjnoxqdxm7e6) [44](#_vjnoxqdxm7e6)

[Sprint 3 - The Finale](#_4eo5vaglauul) [45](#_4eo5vaglauul)

[**9. References**](#_cmyi0e50iu)[**46**](#_cmyi0e50iu)

[**Appendix A: Product backlog**](#_vao9xi755f6)[**47**](#_vao9xi755f6)

FINAL DOCUMENTATION

Website: <http://10.1.144.90> (need vpn access)

Game Server: <http://10.1.144.91> (need vpn access)

# 1. Project Vision

The overall objective of this project is to develop and design a comprehensive game platform that allows users to seamlessly connect with other players, databases, and websites. The Crossing streams game is an online solution that allows users to play over the internet with each other, upload their stats to a website and communicate either in game or online.

Deliverables

* A standalone game that is both multiplayer (local/online) and single player
* A dynamic game server that connects game clients to allow for online gameplay
* A database that contains user data (user account, user stats, forum posts, etc)
* A basic website where players can register, pay and download the game
* Documentation which provides understanding of the software for future developers.

What we are trying to accomplish with this project is creating a game that fits the niche of other similar style games, but with a multiplayer aspect to the game. Currently all of the other games of this genre only work as single-player games. When the system is done a user will be able to download our client, update to the most recent build of the client and play the game. The game flow consists of clearing rooms of enemies in a dungeon floor, and descending the dungeon; killing bosses on every floor until the final floor has been reached. Along the way players will be able to upgrade their players with random power-ups, and the more they player plays the game the more power-ups they will unlock for future runs through the dungeon. Below is a happy-path diagram of how the user will go from first installation to winning the game.



# 2. Problem statements:

## 2.1 Sprint 1 - Problem statement

Here is our initial version of the problem statement as used in the first sprint:

The client wants to create a Multiplayer Online Rogue-like Top-Down Shooter (MORTDS) that leverages concepts from games such as Binding of Isaac, Enter the Gungeon and Gauntlet. This project’s timeline consists of two semesters. The first semesters will consist of proof of concept and the second will be focused on adding and refining game features such as procedural map generation. The game should be distributable to clients in an online environment.

The problem involves the creation of a set of interconnected systems that combine to form our game experience. The team will be challenged by the interlinking of the different technologies, organizing group tasks, and finding relevant assets for the project. A second problem will be in discovering what makes the genre compelling and making sure our game builds upon the compelling elements while adding multi-player features. Our constraints will be time, communication, scheduling/availability and the difficulty curve of learning the utilized technology.

When you combine the team enthusiasm for the project with our client's observation that many Isaac players want such a game, there is certainly potential with the project. Looking at the numbers for Binding of Isaac, (http://store.steampowered.com/app/113200/ accessed September 23) The game has been given 37,882 positive reviews while only having 1,508 negative reviews with 39,235 steam purchases. Enter the Gungeon (http://store.steampowered.com/app/311690/ last accessed September 23) had 5,413 positive reviews and 625 negative reviews with 5709 Steam purchases. These numbers indicate that there is a potential market for such a game and that a well created game would be able to recoup its development costs.

The users of this game platform should be able to interact with other users in a variety of ways. The backend should be robust and expandable to leverage the unique integration of the platform allowing for future development. Administrators should have the ability to review player behavior and make corrective actions to maintain the overall user experience.

## 2.2 Sprint 2 - Problem statement

No changes have been made to the problem statement this Sprint.

## 2.3 Sprint 3 - Problem statement

No changes have been made to the problem statement this Sprint.

## 

## 2.4 Review and compare your original problem statement

|  |  |  |  |
| --- | --- | --- | --- |
|  | Modifications | Exclusions | Additions |
| **Sprint 0** | Test sprint - No changes made | | |
| **Sprint 1** | No changes to deliverables made this sprint | | |
| **Sprint 2** | No changes to deliverables made this sprint | | |
| **Sprint 3** | No changes to deliverables made this sprint | | |

# 3. Team Members

Team roles. Daniel is our unity expert. Ben is Go and web framework experienced. Billy is web developer and game developer. Corey is our database expert. Marc has user interface design experience. All members hope to gain knowledge of procedural generation, and working on larger teams.

**Benjamin Ward: *Scrum Master / Developer***

As a developer, the main responsibility I hold is maintaining the server infrastructure and designing the layout and execution of server-side net code that will allow for game clients to play together online. As a Scrum Master, I facilitate all scrum meetings (standup, sprint planning, review and retrospective) as well I keep track of what the other team Members are currently doing, providing assistance if any other developers run into roadblocks. I take notes at all the meetings and maintain the repository for documentation.

**Billy Spelchan: *Team Lead / Developer***

As a team lead, I will be communicating with James, handling the submission (and assembly if necessary) of the final document, trying to keep the team thinking about the larger goals of the project, managing our three strike policy, and acting as a devil’s advocate if other team members are not already taking on that role. As a developer I tend to be a jack of all trades so will probably have my fingers in everything. With my real-world experience, the key areas that I will be focusing on are the website and game development.

**Corey Frank: *Product Owner / Developer***

As a product owner, I will be in contact with our client Ben Heggie, I will keep him updated on our progress during development and will communicate any changes that the client wishes to implement. I am also in charge of setting up our meeting times and booking rooms for the group to meet. As a developer, I am responsible for several parts of the system, I will be programming several game aspects including (but not limited to) level design, in game chat, the game patcher, as well as some web programming.

**Daniel Atkinson: *Developer***

As a developer, I will be focusing on the development of the game using my experience with game design, unity game engine, and C# programming language. I will ensure that our implementation of various game functions makes the best use of the unity game engine and conforms to best practices as far as is useful to us as a team. Because I have the most experience of the team members with game development and the use of unity I will be focusing primarily on some of the more technical and involved aspects of the game development including but not limited to enemy intelligence and game networking.

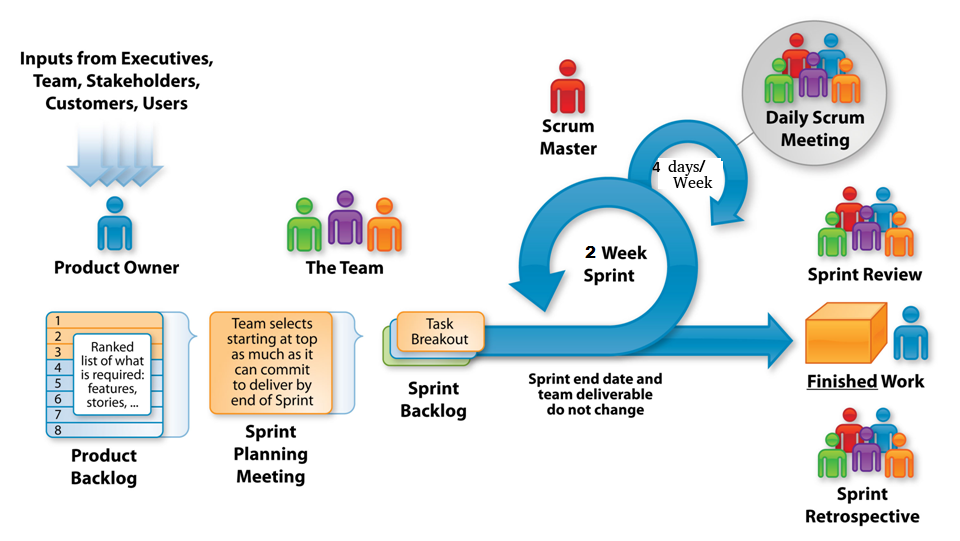
**Marc-Andrew Dunwell: *Developer***

As a developer, I will be focusing on learning and utilizing the Unity game engine to the best of my ability to develop several aspects that are core to gameplay, and with my experience in User Interface design, I will be ensuring that our game follows through with the standard that is expected to make playing easy, and simple for all users. I will be utilizing my experience in design and to keep the game attractive and following the theme that we have chosen.

# 4. Development Process

## 4.1 Description of the Agile Scrum process

(that your team followed through the project development.)



*Source: http://www.etechpulse.com/2015/02/agile-scrum-process-sdlc.html*

### 4.1.1 Product Backlog

The Product backlog is a prioritized list of work for the development team, containing short descriptions of the functionality that will be expected throughout the project. Our requirements obtained from the client provide the foundation for our current product backlog. The user story map then arranges user stories from the product backlog into a useful model to help understand the functionality of the system, identify holes and omissions in your backlog, and effectively plan holistic releases that deliver value to users and business with each release.

### 4.1.2 Sprint Planning

Sprint planning is a group effort involving every individual involved in the team (product owner, scrum master, and the entire development team). Our scrum master Ben runs the meeting; the product owner Corey describes the highest priority features to the team from the user story map which was generated by the product backlog, and all of the team defines work and effort by asking questions to turn the high-level user stories into the detailed tasks of the sprint backlog. These items once defined are run through a process known as planning poker which estimates the story points of each task. The cards 1,2,3,5,8 and king are given out to each team member which then chooses a card that represents the size complexity in completing the story. Once all members have chosen a card they are revealed and if they are different a discussion on the differences of opinions occurs followed by a re vote until everyone agrees. If a task is 8 or higher we discuss breaking down the task further as it is too large for the project. Finally each team member assigns themselves a task that suits their ability and time available. These team members then flesh out the individual sub-tasks and estimate the time of each task. Once the process has finished, Corey starts the sprint. This planning method has been implemented in our project which has resulted in our Sprint 1, 2 and 3 backlogs.

### 4.1.3 Sprint Backlog

The Sprint backlog is a list of tasks defined by the team that is to be completed during the Sprint. It is maintained by the product owner Corey, who sorted it by order of priority and developers pulled tasks by order or priority and their own individual expertise or interest. An example would be a developer that is interested in web server development would pull a task to do with server/client communication. Tasks are at points within the sprint added or removed from the backlog in response to risks occurring (for example, members of the group becoming sick), bugs popping up in the code base, or tasks that were estimated as low effort but found to be high in required work hours.

### 4.1.4 Sprints

Sprints are timeboxed to 2-weeks per sprint over the course of this project. We have several Scrum meetings throughout the week during our Sprints (Tuesdays, Wednesdays, Thursdays and Saturdays). We would discuss what we planned on completing, and ensure that our tasks are completed before the end of the sprint.

### 4.1.5 Scrum

We have several Scrum meetings throughout the week (Tuesdays, Wednesdays, Thursdays and Saturdays), during this time, we discuss what has been done, what will be done next and any problems that we face. After our Scrum meetings, we would continue working on our tasks which will be updated in the following scrum.

### 4.1.6 Demos

At the end of each sprint a demo will be given on each successful feature completed where they are displayed to the project client.

## 4.2 Project development and management platform (JIRA)

The project team used Jira and Google Docs to document, develop user stories and sub-tasks. Git was used for version control. Furthermore we are using the following tools:

JIRA for handling the backlog and sprints

* <http://cs-oracle.okanagan.bc.ca:8088/>
* Board (CrossingStreams)

GitHub for hosting our repository

* Documentation (https://github.com/CoreyFrank/CrossingDocuments)
* Project (<https://github.com/CoreyFrank/CrossingStreams>)

Unity game engine

* Gameplay and game features

Jenkins

* Versioning and CI

## 4.3 Preliminary Communication Method

Our method of communication consists of 4 meetings a week as follows:

Tuesday 11:00 am - Weekly in person (Stand up) - To be changed

Wednesday 8:30 am - Weekly Lab class SCRUM and planning meeting for 3 hours

Thursday 6:30 pm, Evening SCRUM and additional planning

Saturday 1:00 pm - (Online) Skype meeting for SCRUM with additional planning

In addition to our meetings, we have Slack and Skype (for scrum meetings) setup. Slack is an all in one embedded communication platform which allows for text, file sharing and other features available with plugins.

# 5. Glossary of terms

|  |  |
| --- | --- |
| **Agile** | A group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. |
| **Binding of Isaac** | A popular game featuring procedurally generated levels and top-down shooting. While the theme of the game is controversial, the game play mechanics is the aspect of this game that we are focusing on. |
| **C#** | Microsoft’s alternative to the Java programming language. Like Java, it is an attempt to tame the C++ beast while adding garbage collection. It is the Scripting language used by Unity. |
| **Continuous Integration** | The process of regularly merging working copies from multiple developers into one up-to-date, comprehensive build. |
| **Enter the Gungeon** | A popular, top-down shooter wherein players journey through procedurally generated dungeons in search of a legendary gun. This game features a number of mechanics that we will be drawing inspiration from. |
| **GoLang** | Google’s attempt to tame the C++ beast while adding concurrency. |
| **Gauntlet** | A classic, top-down, hack and slash fantasy game noted for being one of the first multiplayer dungeon games. Featured four player multiplayer gaming on a single arcade machine. |
| **Jenkins** | An open source automation server that can handle build processes, continuous integration services, and testing. Jenkins conveniently can be submitted files or pull from a repository and run all services on a constant schedule and return a comprehensive result alerting developers of any issues. |
| **Multiplayer** | A game in which more than one player is playing in the same game world at the same time with the ability to interact with the other players who are playing the game. |
| **Patcher** | The application that is run in order to start the game. It ensures that the client has the latest published version of the core application. If there is a newer version than the one installed, then it will download the new version for the user. |
| **Permadeath or Permanent Death** | When a player dies in the game and reaches a game over state then their progress is wiped or reset to some base value and they must start the game anew. There may be outside progress meters that remain but all match progress is reset. |
| **PHP** | PHP Hypertext Preprocessor. A server-side scripting language that merges HTML tags with a C-like programming language for generating dynamic HTML pages. |
| **Procedural Generation** | The technique where aspects of a game, such as the layout of the levels, is generated using a seed value resulting in vast replay-ability as each seed will in a different experience. |
| **Rogue-Like** | A game sub-genre defined by the key traits of procedurally generated dungeons, item collection, and permanent death. |
| **Scrum** | An Agile framework(subset of Agile) used to manage software development projects. |
| **Slack** | A private, free chat room style instant messaging service. Private team rooms can be created to facilitate team member communication and rooms can be broken into sub-rooms for more topic oriented discussion as well as one on one private messages between users. |
| **Sprint** | A regular, repeatable work cycle during which development/work is done and made ready to be reviewed. |
| **Top-Down Shooter** | A game with an overhead perspective where the predominant activity is shooting at enemies. |
| **Unity** | A popular game engine with no licensing fees until $100,000/year has been earned from the products produced with the product. The engine handles the basic functionality of the game allowing developers to focus on creating the content using C# as the scripting language. |

# 

# 

# 6. Results

## 6.1 Requirement gathering and specifications:

**User Story Map:**

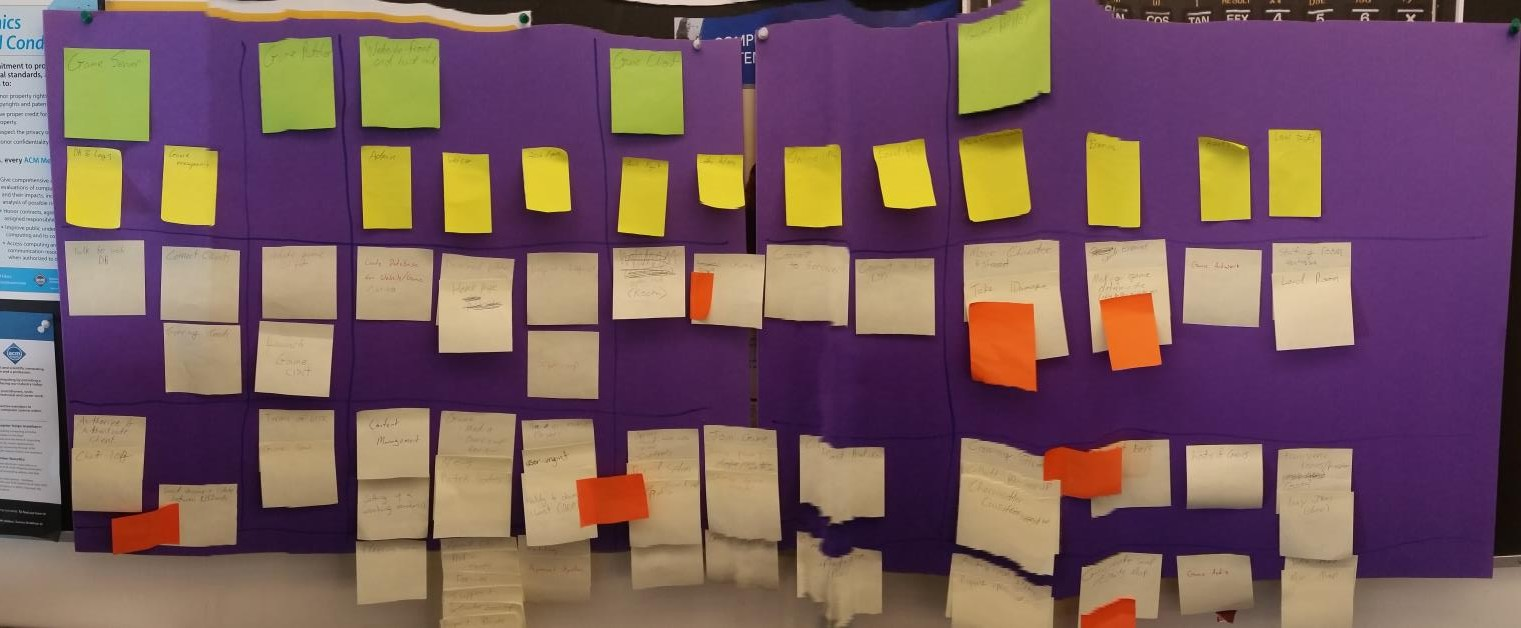


Fig. - User Story Map

Figure - User Story Map breakdown:

The beige sticky notes are user stories.

The Yellow sticky notes are Epics.

The Green sticky notes are components.

The Orange sticky notes indicate high uncertainty issues.

The user story map is divided horizontally by components and epics. Vertically the user story map is divided by sprints. The story map is split amongst base functionality in sprint 1 (highest business value and risk) to sprint 2 and 3 which is lower risk but high business value.

**Product Backlog:**

[<See Appendix A>](#_vao9xi755f6)

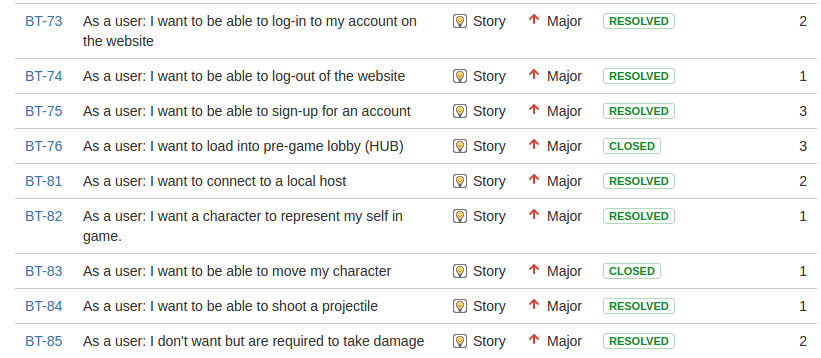
### 6.1.1 Functional requirement / Specification:

The Functional requirements are split into the sprints as followed:

**Sprint 1:**









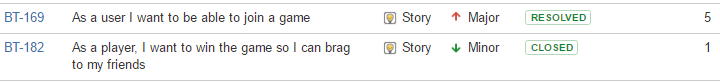
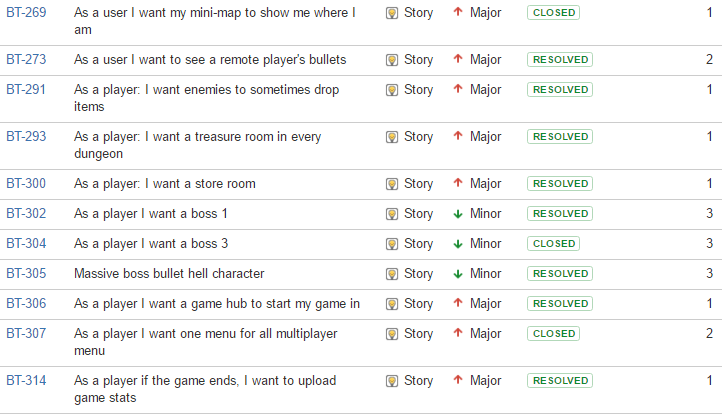


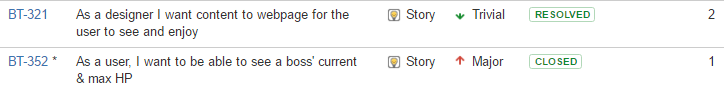
**Sprint 2:**





**Sprint 3:**

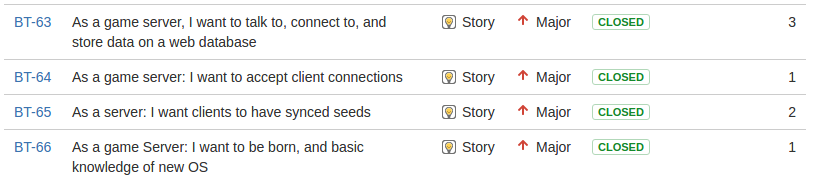


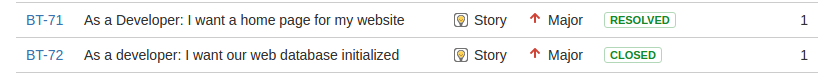


### 

### 6.1.2 Non Functional requirement / specification

**Sprint 1:**





**Sprint 2:**





**Sprint 3:**





Provided below is the system architecture that is used to help explain how the system’s technology works with each other.



Fig. - System Architecture v 2.0

 Fig. - System Architecture v 1.0

The project can be broken into five basic systems. The datastore, web server, patcher , game client, and game server (which has local and online versions). Below is an explanation with diagrams of the 5 components that make up our architecture

#### Datastore

The datastore is our PostgreSQL database which is shared by the web server and the game server. User information is managed by the web server while the game server manages the statistics which the player generates.

The web site uses the database for three tables as shown in the following entity relationship diagram.



The users database holds our user information, the user\_stats holds the statistics for each of the 3 allowed saves and a global save for the game, and news holds patcher news and news for the site (with plans to add image gallery).

On the server side, logging into the game uses the users table, while saving game stats uses the user\_stats table.

#### Website

The web server aka website, would be where users of the game would login or register, learn about the game, download the game client, view news, participate in forums, and possibly participate in tournaments. This would have to be easy to navigate and a pleasant experience for the user as it has an impact on the game experience as a whole.



The website uses php to generate the basic structure of a page using the template. As this generates the menu, it is easy to add or remove main level pages. Pages do not directly access the database but instead use include files for any database work they have to do. This allows pages to easily share database code as well as making it possible to change the underlying database without much impact on the pages as new database code would handle the transition work.

#### Patcher

The patcher is what the user runs to start the game. It makes sure that the player has the latest version of the game client before launching the game client. As a multi-player game, it is important that the players of the game all have the same version of the client otherwise there could be issues due to slightly different ways that different versions of the game work.

#### Game Client

The game client has to be able to also log into the site as it shares login information with the web site. The login is done through the game server which shares the user database with the web site. As the game is multiplayer, it is important that all players are using the same version of the client so the patcher should be what launches the client. The client has to communicate with the game server to get map information as well as know the location of other players when they are in the same room. Basic text chat would probably have to be in the game.

The game client can be thought of as consisting of four layers. At the bottom is the hosting layer which is the operating system that the user is using and Unity game engine which runs on top of it. The server may be ran locally (for LAN games) or on our server (internet play).

Built on top of Unity and connecting with the server, the base game layer has the game save/statistics management, procedural level generator, and physics/prediction. These systems are used by the scenes to control the game. There are a number of different scenes that either control setup related selections or actual gameplay.

Finally, the elements of the game are created as part of the particular scenes.



Unity uses a component model where everything in the game is a GameObject which has components (inherited from MonoBehavior) attached to them. Components can communicate with other components by obtaining a reference to the desired component using a variety of different methods for finding components. Below is a class diagram of this relationship with many of the event handlers that MonoBehavior contains omitted.

#### Game Server

The game server would handle the world and positions of the players and items within the world. As the game would have procedurally generated dungeons, a large amount of effort will be required to generate these in such a way that the game preserves a consistent difficulty curve. The server would also have to track the player's behaviour so that it would be possible to block players for bad behaviour. Some cheat detection, such as making sure actions are within normal ranges, would also be a nice option to prevent cheating. While the game server would run on our server, for better performance it is likely that we will also want the ability for clients to act as a server for LAN games, though the LAN version would not have any of the cheat monitoring otherwise hackers would be able to make undetectable cheat bots.

The server architecture is outlined in the specific requirements section and covers the datastore, website, and game server components. The patcher is a stand alone utility so has no underlying architecture that it is built upon.

The Non Functional requirements are split into the sprints as followed:

### 

### 6.1.3 Specific Requirements

(interface, hardware, communication interface):

The specific requirements are split into the sprints as followed:

**Sprint 1:**





**Sprint 2:**

**<**No Specific requirements this sprint>

**Sprint 3:**

**<**No Specific requirements this sprint>

**Server Stacks:**

Server_Stack_Diagram.png

**Web Server:**

IP: 10.1.144.90

DNS: COSC470DCentOS-2016.cis.okanagan.bc.ca

OS: CentOS Linux release 7.2.1511 (Core)

Web Server software - Apache (http://10.1.144.90/)

/var/www/html (directory root for website)

- index.php is main landing page

Database - Postgresql (10.1.144.90:5432)

Web Panel - Webmin (<http://10.1.144.90:10000/>)

CI - Jenkins(10.1.144.90:8080)

**Game Application Server:**

IP: 10.1.144.91

DNS: COSC470ACentOS-2016.cis.okanagan.bc.ca

OS: CentOS Linux release 7.2.1511 (Core)

Crossing Streams Game Server:

IP/Port: 10.1.144.91:25565

go version go1.7 linux/amd64

Crossing Streams Game Lobby Process:

IP/Port: 10.1.144.91:[dynamically created]

Go version go1.7 linux/amd64

## 6.2 Engineering Execution

### 6.2.1 Team Software Integration Plan (best for continuous integration)

For the first sprint we opted to go with manual builds thinking that as our tasks were spread out enough over getting 5 separate components working together we should have very little issues so delaying the setup of Continuous Integration (CI) software could be delayed until the second sprint. This turned out to be a mistake as things did not go smoothly when it came to working with Unity, which four of the five developers had to do.

For the second sprint we are going to be setting up a Jenkins server. This CI software was chosen as it has support for Unity. This was able to be setup on a poll basis where it checks commits to the master branch in our two repositories (CrossingServer, CrossingStreams repos). Once the repo has been pulled it runs a script that parsed through all tests specified. If these tests pass the script then builds the project and deploys the build code in a separate repository which can be downloaded for any manual testing that is required for behavioural code.

### 6.2.2 Builds (procedure of code integration and building the final product, version control, software change control)

For the first build, the software is manually built. The server software is compiled using go.

For Sprint 1 we did not have any Continuous Integration set up for building our Unity project, so manual building of the project was done. This requires setting up unity on your system.

Download unity from the Unity store (<https://store.unity.com/> ). We are using the personal edition version 5.4.2f2 which is free up to the first $100,000.

Install and set up your unity account. When you start unity, it will prompt you to create a new project. Select a name and location to store the project and name the project appropriately. Make sure that the project is set up as a 2D project. Note that 2D mode is not vital, but when you start in 2D mode the environment is set up for the creation of 2D games.

Make sure that meta files are visible so that you can use a repository such as git. This is done using Edit->Project Settings->Editor Settings. Set version control mode to visible meta files.

Use File->Save Project to make sure the project is created.

Copy the files from the git repository’s Unity/Asset directory into the assets directory within the project.

Refresh the assets folder in Unity (right click Assets folder in project folder, select refresh).

If the repository is up to date and no issues should be able to run. Keeping the repository up to date was one of our issues for Sprint 1 but we hope to get this under control for sprint 2.

For website pages, CI is simply the process of copying the pages to the web server. At the moment this is done manually, but it should be possible, if desired, to automate this process by pulling changes to the web pages from the repository and placing them in the html folder that contains the web site.

As of Sprint 2 we now have Jenkins poll and build/deploy our project. The output for the website is just a script that pulls the repo, The server is compiled on the server and run in headless mode, and the game client is tested, compiled and pushed to a repository.

### 6.2.3 Installation (procedures, user manual or user guide, source code or repository)

The user obtains the software by going to the web site. Once they have created an account they can download the patcher which is used to download, update, and run the game.

From a developer’s perspective, once the servers have been set up with the suite of software we are using, run the server software, and copy the website pages to the appropriate directory so apache can serve up the pages. The patcher grabs the latest version of the game client from a repository so a copy of the latest version of the build will have to be placed there.

## 6.3 Project development results

### 6.3.1 Sprint planning / roadmap summary

|  |  |
| --- | --- |
| **Sprint 0** | **The Jira Tutorial**  Test user stories were associated with this sprint. Members of the team went through a Jira Tutorial and added stories, tasks, epics, estimations, etc and logged hours into open tasks which were then set to done once completed. All members contributed and the tutorial is considered to be a resounding success as our utilization of Jira has gone seamlessly. |
| **Sprint 1** | **The Streams**  The system being developed is made up of a number of interconnected components which at the end of this sprint are going to be working at the most basic level possible. The client will be able to download the patcher from the game site, update to the current (first) build of the game, and run through a very simplified version of the game with just rudimentary server connections. |
| **Sprint 2** | **The Crossing**  Building on the system built in the first sprint, our focus now is to add support for multiple players to exist in the same world. To make the world compelling for multiple players we are fleshing out the experience by adding power ups, more monsters, more rooms, and an expanded website. |
| **Sprint 3** | **The Finale**  The focus of the 3rd sprint was to get the basic happy path completed from start to finish. Implementing everything from the website to the multiplayer focus. The multiplayer while there in sprint 2 was far from complete and needed the additional functionality of multiple lobbies and support for 4 players. In addition, we endeavor to make the world compelling for multiple players and are continuing to more add power ups, more monsters, and more rooms. |

### 6.3.2 Burn down chart history from Sprint 0 to final

|  |  |
| --- | --- |
| **Sprint 0** | **The Jira tutorial**    The burndown chart shows that there is inconsistency. The first half shows that not much work was logged for Wed-Fri, someone was a keener on the weekend and in the last half we started to log hours. In the end it’s shown that much of the work wasn’t being logged till the very end. All in all, the takeaway is we should be more consistent in logging hours. |
| **Sprint 1** | **The Streams**  burndown.PNG  The burndown for sprint 1, (at 9:26PM tuesday Nov.1)  We *are off to a slow start due to starting a day late as well the tasks assigned to my other team members require a considerable amount of infrastructure setup before they can start. This was resolved over the course of the first weekend.*  We finished most of the tasks (37 out of 44 story points) in the overall sprint, but some tasks in JIRA were not defined properly, and there were some underestimates made for some tasks, as well as time not logged while working with research and tutorials. These reasons lead to the remaining time estimate not being as close to the bottom.    The Burndown chart of our completed story points reflects a more accurate representation of how many tasks we had yet to complete. |
| **Sprint 2** | **The Crossing**    The burndown for sprint 2, (at 11:06PM tuesday Nov.1)  We *are off to a slow start due to starting a day late.*  We finished most of the tasks (50 out of 54 story points) in the overall sprint, but we bit of a bit bigger of a chunk than we were able to accomplish this sprint, as well as time not logged while working with research and tutorials. Another reasoning for the massive downward spike is due to tasks not being broken down enough which is more clearly seen in the burndown chart by story points. These reasons lead to the remaining time estimate not being as close to the bottom as well as large cliffs where work is being committed all at once.    The burndown chart of our completed story points shows the story of stagnancy followed by a mass commitment of work not logged as well as tasks that could have been broken down further. We will work on fine tuning our stories for Sprint 3. |
| **Sprint 3** | **The Finale**    The burndown for sprint 3, (at 5:39PM wednesday Nov.30)  We decided to try something a bit different this time. During the sprint planning meeting for Sprint 3 we all worked on fleshing out the individual stories, story estimation and tasks. To save time we decided to estimate the hours for each subtask separately by the user who accepted the task.This explains the jump in hours from 0 to almost 80. Afterwards we see a steady stream of logged hours which indicates a focused movement towards goal completion and work on the actual tasks within the project.    The burndown chart of our completed story points shows the continued story of stagnancy followed by a steady commitment of work, as well as tasks that could have been broken down further. |

### 

### 

### 6.3.3 Sprint velocity and hours

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Hours Estimated** | **Hours Completed** | **Story Points estimated** | **Story Points completed** |
| **Sprint 0** | 99w 4d 7h 10m | 99w 4d 7h 10m | 44 | 44 |
| **Comment:**  This was a test sprint. The values entered above are not realistic or accurate at all. Some of the user stories and tasks that were estimated were using experimental values to determine what Jira could handle which is reflected in the estimated hours. | | | |
| **Sprint 1** | 1w 1d 4h 30m | 1w 2d 3h 40m | 44 | 37 |
| **Comment:**  Our sprint Velocity for sprint 1 can be seen in the chart below, it shows we committed to 44 story points and we completed 37 story points, though there is still more to do that hasn’t been captured in JIRA, For the most part our user story estimations have been quite accurate. Though we need to refine our criteria more. | | | |
| **Sprint 2** | 2w 4h 15m | 2w 1d 7h 45m | 54 | 50 |
| **Comment:**  Our sprint velocity for sprint two can be found below, We committed to 54 story point worth of work, and completed 50. We committed to doing more work than the previous sprint (goal of 44) and we managed to complete more tasks this sprint; though we did not finish a few minor tasks. We plan on having fewer story points in our next sprint,(40-45) as this was a heavy workload. | | | |
| **Sprint 3** | 2w 1d 5h | 2w 1d 4h | 56 | 38 |
| **Comment:**  Our sprint velocity for sprint three can be found below, We committed to 56 story points work of work, and completed 38. We committed to doing more work than our previous sprints (54 and 44) and managed to complete 38 which was more than sprint 1 but fallen short of sprint 2. Sprint two left many of us exhausted after the work though, which showed how we were unable to commit to as many tasks. With these 3 initial sprints completed we can see our velocity being close to 42 story points per sprint. We will adjust all future sprints to this velocity and adjust as necessary. | | | |
| **Velocity Chart** |  | | | |
| **Time Spent over Sprints** |  | | | |

### 6.3.4 Risk Tracking from Sprint 0 to final

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprints** | **Risk** | **Probability of Risk** | **Size of Loss (Days)** | **Risk Exposure (Days)** |
| **Sprint 1 - The Streams** | Team member falls ill or otherwise unavailable. | 40% | 5 | 2 |
| Repository unavailable (github goes down) | 5% | 1 | .05 |
| The school’s Servers go down and we can’t work push to production servers or test online infrastructure. | 20% | 5 | 1 |
| Team member drops out of course | 40% | 14 | 5.6 |
| **Sprint 2 - The Crossing** | Too many tasks committed for the sprint | 30% | 3 | 1 |
| Someone get’s sick | 20% | 6 | 1.2 |
| **Sprint 3 - The Finale** | Too many tasks committed for the sprint | 50% | 3 | 1.5 |
| Higher priority tasks reveal themselves in the middle of the sprint | 20% | 2 | 0.4 |
|  | Exposure: | | |  |

### 6.3.5 Summary of Retrospectives

|  |  |
| --- | --- |
| **Sprint 0** | This was a test sprint that was utilized to help members of the group understand the Jira tool that will be used. |
| **Sprint 1** | 1. **Continue:**    1. Good vision of what we wanted for the end of sprint 1 (Theme)       1. Good grasp of scope    2. Communication between members in the last week went surprisingly well.    3. All committed to the project and the tasks assigned.    4. Trust (everybody did what they said they would do) 2. **Stop:**    1. Integration (we didn’t do it and suffered for it)       1. Pushing to master (more frequently) when you finish the story.          1. Uml diagram (minimal at least)          2. Clear test plan (unit test) and test plan to describe how to do the testing.       2. Automatically build/test/deploy    2. Lateness to meetings.    3. Acceptance criteria could use work    4. User stories could be better defined (more specific) 3. **Start:**    1. Set up Jenkins CI server (get up and running and document how it works)    2. More diagrams to explain architecture    3. Final documentation and presentation start now.    4. Presentation needs more slides       1. Task breakdown for each user story per team member |
| **Sprint 2** | 1. **Continue:**    1. Final presentation and documentation better    2. Everyone was better at attending meetings this Sprint    3. Good vision of what we wanted for the end of sprint 2 (Theme)       1. Good grasp of scope    4. Integration made working on the code considerably easier       1. We pushed to master at a higher rate which helped communication       2. Automatically build/test/deploy works!    5. Communication between members in the last week went surprisingly well.    6. All committed to the project and the tasks assigned.    7. Trust (everybody did what they said they would do)    8. Final presentation looks more fleshed out 2. **Stop:**    1. Uml diagram (minimal at least) better, but still needs work    2. Too much work (story points) this sprint for the time available       1. We all had more work/exams    3. Acceptance criteria/ user stories better written but still need more work    4. Clear test plan (unit test) and test plan to describe how to do the testing. We need to have more test classes to debug/code. 3. **Start:**    1. Logging hours earlier, we dropped the ball this Sprint at the beginning.    2. More diagrams to explain architecture       1. We started this but we need to work on it still    3. Final documentation and presentation start now. |
| **Sprint 3** | 1. **Continue:**    1. Everyone was better at attending meetings for this Sprint    2. Good vision of what we wanted for the end of sprint 3 (Theme)       1. Good grasp of scope    3. Integration made working on the code considerably easier       1. We pushed to master at a higher rate which helped communication       2. Automatically build/test/deploy works!    4. Communication between members the entire sprint went well    5. Trust (everybody did what they said they would do)    6. Final presentation looks more fleshed out 2. **Stop:**    1. Say something if you can’t complete all the tasks assigned    2. Final documentation needs continual improvement       1. Don’t let it get stagnant.    3. Uml diagram (minimal at least) better, but still needs work    4. Stop focusing on terminology and accept documented norms    5. Too much work (story points) this sprint for the time available       1. We all had more work/exams       2. We wanted the happy path to be done but ended up not doing everything we had planned.    6. Acceptance criteria/ user stories better written but still need more work 3. **Start:**    1. Timeboxing subtasks the day when we accept them.    2. More diagrams to explain architecture (Continue doing this)       1. We started this but we need to work on it still    3. Final documentation and presentation start now. |

### 6.3.6 Summary of Scrum Meetings

|  |  |
| --- | --- |
| **Sprint 0** | **The Jira tutorial**  *No Scrums were held for this sprint tutorial.* |
| **Sprint 1** | **The Streams**  **Saturday Oct. 29 Meeting**  ***What we've done:***  Corey: Patcher work, not finished (will upload tonight)  Was going to do JIRA Tasks (JIRA not working)  Marc: Committed code, from unity  Daniel: Front end work for client LAN  Billy: Finished AI components (haven't moved to done, JIRA issues)  Started website, (cannot get onto OC VPN)  Ben: Worked on setting up game servers, how to send and get data  (not here, was moving boxes, excused)  ***What we're working on (for Tuesday):***  Corey: Finish Patcher  Somewhat of a game demo for tuesday  JIRA tasks (Checking to see if it updates)  Marc: Starting on documents (team lead)  Template for MS Powerpoint  Daniel: POST and GET request  Documentation  Billy: Working on website, getting working on local machine, but needs VPN access  Ben: (Not here, but he is done almost all his tasks)  ***Issues:***  Marc: JIRA  Billy: Cannot access OC - VPN  All: JIRA not working  **Tuesday, Nov. 1 Meeting:**  ***What we've done:***  Corey - Patcher, working in unity, documentation  Ben - created game server and client server, request vpn access, helping team members commit,  working with Dan on how game server API works  Billy - website, download, all except password hash, final report  Marc - <sick>  Dan - hooking up servers (launching from client)  ***What we're working on (for Wednesday):***  Corey - Demo, documentation & reports  Ben - individual report, team report  Billy - password hashing in php  Marc - individual/team report, powerpoint  Dan - individual report, team reports  ***Issues:***  JIRA offline (weekend)  Billy - No VPN access to college  Ben - Moving on the weekend |
| **Sprint 2** | The Crossing  **COSC 470 - SCRUM Meeting - Nov. 9 2016**  ***What we've done:***  Marc :  - working on team report  - starting on JIRA tasks (monster functionality)  - team report  Billy: - Final report  re-wrote several sections & conclusion  Ben: -Game server connection stuff  accepts connections from game client  send messages to server  server can send instructions to client  - Team Report  Corey: - Worked on patcher  - game audio files gathered and code to integrate sounds added  - Team report  Dan: - Bosses  - uploaded idea docs for bosses  ***What we're working on: (for Thursday)***  Marc : - Monster functionality  Billy: - Save state functional  Ben: - Game server connection stuff  - getting jenkins to pull from repo  - create standard for communication  Corey: - Patcher bug fix  - store scene  - applying audio files to required events  Dan: - Bosses  ***Problems:***  Ben: need a dedicated jenkins user account that has push/pull and admin access to all 3 repositories  **COSC 470 - SCRUM Meeting - Nov. 10 2016 *What we've done:***  Marc : - completed all but arc monsters  - arc monster started but needs work (60-70% done) Billy: - Menu to select save file done (wants code review)  - setup levels had to be done to switch between save file scene and hub  - this needs to be reviewed to see if it breaks anything  - 5 power ups finished (task done) Ben: - created standard for communication  - have multiple clients connecting to each other  - started work on media web page Corey: - Patcher bug fix done  - Working on store scene (almost done)  - Git organization (for jenkins)   - applying audio files to required events (done) Daniel:- Finishing bosses (mostly finished)  - some issues (bugs)  ***What we're working on: (for Tuesday)*** Marc : - Work on webpage (about us)  - Start on options screen  - Work on final report and presentation  - review and retrospective  - Add sprint 3 user stories  - Add tasks, descriptions, tests, acceptance criteria Billy:- more testing for power ups  - finishing up prediction engine  - Work on final report and presentation  - review and retrospective  - Add sprint 3 user stories  - Add tasks, descriptions, tests, acceptance criteria Ben:- finalize server connection for multiplayer  - finish website media page  - getting jenkins to pull from repo (need help from Corey)  - Work on final report and presentation  - review and retrospective  - Add sprint 3 user stories  - Add tasks, descriptions, tests, acceptance criteria Corey: - Patcher done  - store scene done  - website  - Work on final report and presentation  - get demo working (on Tuesday)  - review and retrospective  - Add sprint 3 user stories  - Add tasks, descriptions, tests, acceptance criteria Daniel: - Finish bosses  - Game news web page  - Level editor (transition to next level aka Win!)  - Work on final report and presentation  - review and retrospective  - Add sprint 3 user stories  - Add tasks, descriptions, tests, acceptance criteria **Issues:**  Billy - ui learning curve (not that bad once you get it)  Ben - Jenkins (Corey is finishing today) |
| **Sprint 3** | **The Finale**  **COSC 470 - SCRUM Meeting - Nov. 23 2016**  ***What we've done:***  Marc : - added sub tasks to boss  Billy: - almost finished login task  Ben: - refactoring serverside code (combine crossingLobby and  crossingServer code base)  - create go function for lobby instantiation  - creating a list of lobbies  Corey: - started working on level generation algorithm  - created new happy path  Daniel: - started working on enemy refactoring  ***What we're working on: (for Thursday)***  Marc : - start on boss user story  Billy: - finish login task  - starting on monster drops  Ben: - Create Network protocol for client communication  Corey: - working on room prefabs  - work on dungeon generation  Daniel: - fix unity issues  - continue working on enemy refactoring  ***Problems:***  Daniel: - Unity editor broke  **COSC 470 - SCRUM Meeting - Nov. 26 2016 *What we've done:***  Marc : - Boss  -minor kinks to work out  Billy: - (Done) Sending stats to server  - (Done) Turtle boss  - figured out unit test failing issue.  Ben: - 4 players at once in a lobby (done)  - Movement of 4 players (continue some bugs)  - player instantiation done  - joining players instantiate all current players  - current players instantiates joining player  - Continue some more on Game Network protocol  - Refactored Billy's code into crossingServer code base  Corey: - (done) room prefabs  - continue dungeon generation  - started on boss character  Daniel: - enemy refactoring (almost done)  - factory works  - not all enemy types work yet  - stat collection (continue)  - Network menu (continue)  ***What we're working on: (for Monday)***  Marc : - Started on the health bar (finish by tonight)  - Begin the clear a room task.  Billy: - Finish saving stats to database  - UML charts  Ben: - Movement of 4 players (continue some bugs)  - Finish shooting, enemy instantiation  - host state sending  - Finish on boss  - Continue some more on Game Network protocol  Corey: - finish dungeon generation  - finish on boss character  - update patcher  Daniel: - enemy refactoring (continue)  - factory works  - not all enemy types work yet  - stat collection (continue)  - Network menu (continue)  ***Problems:***  *Ben: - Game network protocol may be too big of a task*  *- (very big)*  *Daniel: - Needs to know how to insert stats to database*  *- from billy* |

# 

# 

# 7. Maintenance

## 7.1 Source Code locations (git repository)

**Documentation:** https://github.com/CrossingStreams/CrossingDocuments

**Server Application:** https://github.com/CrossingStreams/CrossingServer  
 **Web and Game Application:** https://github.com/CrossingStreams/CrossingStreams

## 7.2 Build Script and SetupImage result for jenkins diagram

***Fig: Jenkins CI diagram***

The Server setup has been fully documented for both the web server and the game application server. Jenkins is fully automated to poll and pull from our repositories mentioned above that has had a commit to master made. This will then go through a process where tests will be executed, built and deployed to their proper places.

**Deployment:**

1. Web server files are copied to their ultimate file locations.
2. Patcher is built and deployed to website database for download.
3. Game server application files are copied from local to remote server.
4. Game client is built and deployed to consumer repository

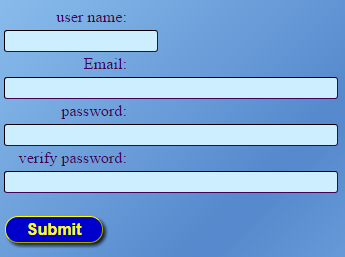
7.3 User Manual / Guides

**Step 1: Register an account**

Go to the web addres of 10.1.144.90 and click on this button

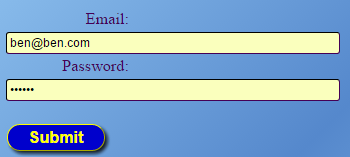


Fill in your unique user details



**Step 2: Download the Game Patcher**

Go to the main web page and login

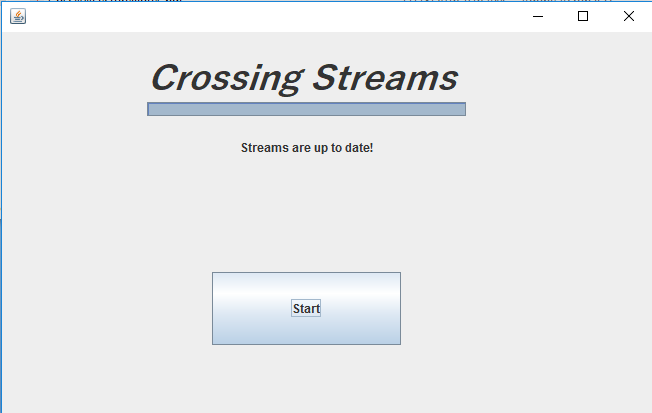


Click the download button after you are redirected back to the home page



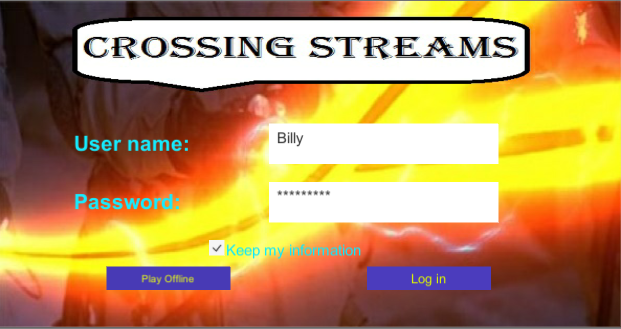
**Step 3: Run the Patcher to get the game updates**

Run the CrossingStreams.jar file to launch the game patcher and press the start button



**Step 4: Launch the game**

Login to your account on the game



**Select your save file**

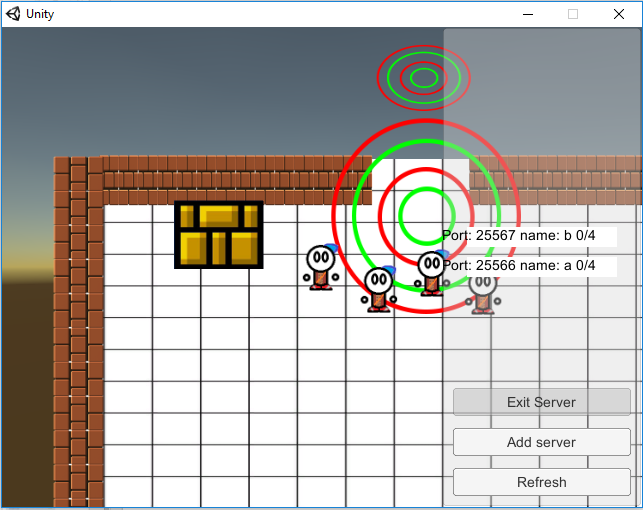


**Step 5: Join/Host a game**

Add server button to add new server

Refresh button to refresh server list

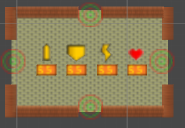
Exit server to leave current server



Make your way through our randomly generated dungeon (those non grey things are rooms)



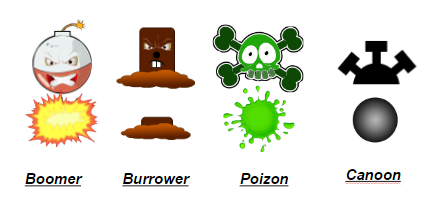
Buy Items for game money



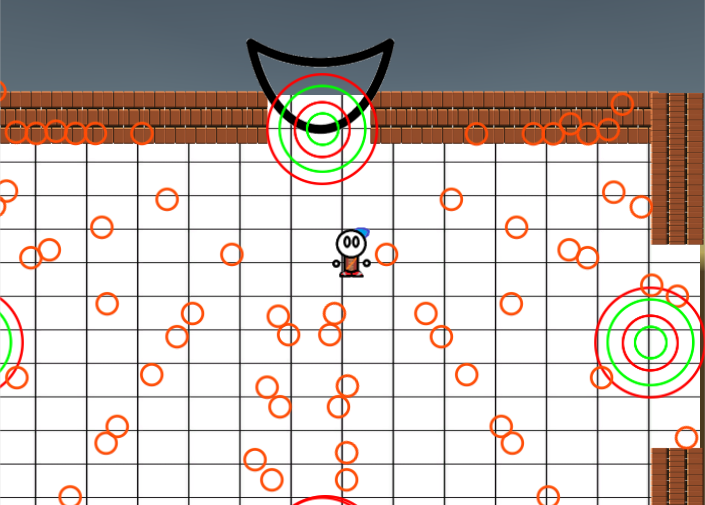
Use your minimap to traverse the dungeon



Fight Unique monsters and bosses



Beat the Final Boss



**Step 6: Repeat the game as many times to beat your scores!**

# 8. Conclusion

## 8.1 Summary of your project results

### Sprint 1 - The Streams

We were able to complete most of the stories in the first sprint and as a result have a foundation for building off of in future sprints. The servers have been set up hosting our database, web-server and game-server. The game server is able to accept connections and share the game seed with the players. The website allows the creation and validation of users with registered users who are logged in able to download the patcher. The patcher will grab the latest version of the game from the repository but launching the game once updated was not implemented before the end of this sprint. The game runs with the ability to transition between rooms. Player’s can move and shoot in different directions, with monsters having different behaviors and attributes.

Our client seemed happy with the team’s demonstration but would really like to see multiple players playing the game as that is, for him, one of the biggest selling points of this project. While the team does agree that multiplayer is important, this required the necessary infrastructure to be in place first, which was the goal of this first sprint.

### Sprint 2 - The Crossing

Again we were able to complete a majority of the tasks assigned but this was largely due to putting in extra hours and not due to actually taking on a reasonable amount of work. With exam and other projects coming up in the next couple of weeks, we are going to have to make sure that our final sprint this year is more reasonable when it comes to the workload.

We are going to be able to present to our client (this report is due before our presentation) with a game that supports multiple players playing in the same world. To make the game more exciting, there are now power-ups which give the player extra abilities, a store where players can take the money they find and spend it on power-ups, a larger variety of monsters to challenge the player, a larger world to play in, and a boss. The website has also been expanded, however the pages are in a very rudimentary state.

### Sprint 3 - The Finale

Not learning from our previous mistake of taking on too many tasks, we put the lofty goal of completing the happy path ahead of having a realistic workload and paid the price by only finishing two-thirds of our tasks. While this was demoralizing we did manage to accomplish quite a few of the tasks and still have a very solid foundation to take with us into COSC 471.

The project has come along quite a ways with the different components that make up the game actually working pretty well together so our concern of putting together such diverse components has been put to rest. Next semester, however, we are going to have to make sure that our multi-player is functioning much smoother than it currently is but the team is confident we can make this work.

Considering the huge scope of this project, we have actually managed to accomplish a very significant amount of work. The team should be quite proud of how much we have accomplished even if we did not accomplish everything that we wanted to.

## 8.2 Suggested list of improvements

### Sprint 1 - The Streams

The three key things that we need to focus on are CI, diagrams, and test plans. CI was clearly an issue that we ran into as not seeing the overall changes that a contribution has to the build clearly was detrimental. Documenting and creating simple diagrams of the work that we have completed is going to have to be a bigger priority so that other members of the team can get up to speed on what other members are doing especially as tasks start having larger overlap. Finally, having a formalized test procedure for the stories we have completed would be a good idea to make sure that we can always verify that what we have done is not causing other problems and that the program is working the way it is suppose to.

### Sprint 2 - The Crossing

The biggest issue we have is taking on too much. When you consider the large scope of this project it is quite understandable that everybody is worried about getting everything we want to put into the game finished within the paltry six months that we have to work on this project. It is, however, important to remember that the members of the team have other things (jobs, other classes, other commitments) to do beyond the project and some members would like to have enough time to one day watch one of those “movie” things that James keeps talking about.

Testing is the other major issue that we have to deal with. We really need to make sure that we have enough tests for our code. Automated tests are preferable as it makes it easy to quickly determine when a change in code has broken existing functionality. The interactive nature of our project does limit the amount of automated testing that we can do making manual testing an unfortunate requirement. Manual testing, however, only really works well if the tests that need to be done to validate functionality are documented so focusing on a formal testing document may be something that we will have to look into.

We have made some inroads towards proper CI and diagrams, but still have a ways to go before these are at the levels that they need to be. As long as we continue to make inroads towards these issues, we will be on a good footing going forward.

### Sprint 3 - The Finale

While it is understandable that not all tasks get completed, it would be really helpful if people with tasks that they don’t think they can compete communicate this with the team. This way if there is a task some else needs done they can either trade tasks or take on the blocking task. Unfortunately, it appears that our team likes to race deadlines a bit too much so it quite likely that people don’t realize that they are not going to finish their tasks until it is too late to communicate the problem. The solution to this is to start working on stories right away.

We still need to work on our testing, documentation, and diagrams but we have made pretty good inroads in all three of these. The presentation clearly showed how diagrams can make things more understandable.

The story problem isn’t too bad, but each sprint we run into several stories that are just not rated properly. Breaking down the stories more is not really the issue, which seems to be the solution we keep trying to apply. Instead I think that we need to start making sure that we are detailed enough in the requirements so that we will know exactly what a task will involve. Still, planning poker is not a panacea so we have to expect this to continue to happen to some extent so need to start making sure that we work on tasks from highest to lowest priority whenever possible so a surprise in difficulty or time will only result in less-important tasks being pushed back.

## 8.3 Lessons learned

### Sprint 1 - The Streams

We had trouble with some tasks that had vague or ill-defined acceptance criteria that made it hard to judge when a task was completed or what was required to finish any given task. Some of our acceptance criteria also conflicted with what the task itself was requesting and either should have been applied to a whole new task or the task should have been rewritten to better define what was expected of the developer. We will take this knowledge of trying and struggling with acceptance criteria and better inspect that the criteria that we apply to tasks in sprint two are quantifiably defined, clear, and match up with the tasks they are assigned to.

Because in part due to poorly defined tasks and partly due to trying to plan ahead for tasks in future sprints a considerable amount of time was spent working on things that weren’t tasks. The project we are creating is very complicated so we are still working out stories that will need to be added to the backlog for the next eight sprints. While this is and isn’t a bad thing, as we are working out details that will affect the future of this project, that time could have been better spent on current tasks and not on work that wasn’t in this sprint. For future sprints we will spend more time defining the issue in JIRA and hopefully stay closer to the work that needs to be done and not unrelated work. Additionally checking in more often with JIRA and using the comments section will hopefully keep developers on task and let others keep up with committed work and provide guidance if necessary.

The networking issue could be larger and more complicated than we had anticipated. This is something that must be planned for otherwise it could lead to major issues in future sprints. Some tasks or stories might need to be reprioritized or moved to a sooner sprint than might have otherwise been intended. Additionally we will use this found knowledge to distribute tasks differently and take it into account for our planning sessions. New stories or tasks might be needed for investigation or different implementation methods but this will be discussed at our sprint planning and backlog grooming meetings.

Unity physics is not consistent between different platforms. We need determinism for our project so may need to write our own physics engine. As our needs are simple, this may not be as bad as it sounds but will have to be something we consider.

### Sprint 2 - The Crossing

The biggest lesson we learned is that time is finite. We have a bad habit of trying to take on far more work than we should. While this is partially a result of having an ambitious project, part of the issue is taking on too many tasks that are not directly related to the theme of the sprint. Our future sprints are going to have to be more focused as we go forward with this project.

We all committed to updating Jira and committing to the master branch of the repository more frequently yet our updating of Jira was done at roughly the same frequency, if not less, than the first sprint. Our merging into master improved a bit but not as much as it should have. This makes continuous integration difficult as there is very little happening on the master branch until the end of the sprint where everyone is in a rush to try and get all their code finished and working together. This is a tricky issue to fix as I believe that one of the main reasons behind not wanting to commit to master is not being finished with the code. Nobody wants to break master so nobody commits until they are finished. We are hoping that having the Jenkins server running will act as motivation to merge into master.

Related to the repository is too much overlap in the tasks being assigned. This is a concern as it causes dependencies and bottlenecks in the development process. Waiting for someone to finish their task before you can properly start on your task is not productive and when one has only certain periods of time that they can devote to working on the project can be a big issue. We need to start taking overlap into account when assigning tasks and group the tasks a bit better so that any bottleneck tasks are given to the same person whenever possible. To aid this, the team should listen to James and start assigning ownership to areas of the project. One other possible solution is instead of having individual branches, we structure the branches into major subsystems. This will then allow people working on related tasks to see changes that the other person is doing.

### Sprint 3 - The Finale

The biggest lesson that we learned is that it is far too easy to fall back on bad habits. We let the desire to complete the happy path lure us into making the mistake of taking on far too many tasks we then made the situation worse by having stories that could have been omitted in the sprint because they belonged on the path even though they were not vital to completing the happy path. This is an issue we need to solve as it is not a sustainable practice.

As much as the team hates creating diagrams, the presentation was far smoother as a result of the diagrams and they made understanding other member’s tasks much easier. The key here is to make sure that we are only creating diagrams to clarify how things work. The principles of just-in-time and just-what-you-need has to be applied here. After each story we need to be asking ourselves, what would make this easy to understand and do the one or two diagrams that explain the task.

# 

# 9. References

Binding of Isaac, (<http://store.steampowered.com/> app/113200/ accessed September 23)

*Enter the Gungeon* (<http://store.steampowered.com/> app/311690/ last accessed September 23)

PHP (http://php.net)

Golang (https://golang.org/)

Unity (https://unity3d.com)

# 

# 

# Appendix A: Product backlog

