April 2012

Broken World

Ethan Collins Lawrence
Worcester Polytechnic Institute

Mark P. Troutt
Worcester Polytechnic Institute

Nathanael Clay Thorn
Worcester Polytechnic Institute

Project Number. MLC1112

Broken World: A Massively Multiplayer Online Game

A Major Qualifying Project Report
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degrees of Computer Science and Interactive Media and Game Development
by

Nathanael Thorn

Mark Troutt

and the
Degree of Interactive Media and Game Development
by

Ethan Lawrence

Date: April 26, 2012

Approved:

Professor Mark Claypool, Major Advisor

Professor Joshua Rosenstock, Major Advisor

Abstract

Broken World is a non-violent, non-competitive MMO set in an expansive post-fuel world with an emphasis on building a community. The art style is drawn from the real world, using low-poly models. The game is built upon a unique peer-to-peer networking model, which allows for a low-resource server. It also features a terrain rendering system that utilizes PNGs to store landscape data efficiently. Along with the game, a suite of custom development tools were built to allow developers to insert game content quickly. Post-development, the game and its related technologies were evaluated for their effectiveness.

Table of Contents

1 Introduction

The popularity of video games has been growing steadily. It is now a billion dollar industry: a recent first-person shooter game, *Call of Duty: Black Ops*, sold 5.6 million copies on launch day.[1] According to the Entertainment Software Association, seventy-two percent of American households play games, and the average gamer is 37 years old.[2] More and more people are spending increasing amounts of time playing video games, making the video game industry an important facet of modern entertainment.

One popular genre of games is the massively multiplayer online game, or MMO for short. In an MMO, many players—sometimes even hundreds or thousands of players—play a game together in a persistent world. Some MMOs, like *World of Warcraft*, have reached over 12 million players.[3] Players have collectively spent over 5.93 million years playing.[4] It is undoubtedly the most successful MMO on the market.[5] Other MMOs, like *Runescape*, allow players to play a limited version for free and can unlock the full experience by paying a monthly subscription. Since its creation in 2001, *Runescape* has had 156 million registered accounts with 10 million of them currently.[6]

Despite the widespread popularity of MMOs, they are largely homogenous in theme, gameplay, and technology. A significant portion of the MMO market is high-fantasy or space-themed games. Most MMOs are based around combat mechanics, requiring the player to kill enemies in order to get ahead of the other players. In addition, the networking models for these games tend to be very similar.

Broken World is an MMO. However, *Broken World* makes several attempts to deviate from the archetypical MMO. In fact, *Broken World* is more than simply its MMO gameplay mechanics, but it also encompasses an expansive MMO framework and associated toolset that are innovative in their own right.

The goals of *Broken World* are as follows:

- **Alternative Gameplay Style** - *Broken World* aims to provide a non-violent, non-competitive alternative to the current trend of violent MMOs.
- **Consistent Style** - The art style in *Broken World* should be immersive and cohesive.
- **Low-Poly Art** - The models used in *Broken World* must have a low polygon count.

- **Scalability** - The game should be able to handle 30 players at any given time, preferably more.
- **Lightweight Server** - The server needs to run on a weaker machine than conventional MMO servers.
- **Expansive World** - The game world should be infinite or sufficiently large enough to seem infinite to the players.
- **Persistent World** - The game world should persist across all players and all play sessions.
- **Development Tools** - *Broken World* should be accompanied with a set of development tools to allow new content to be added to the game.

The gameplay goal behind *Broken World* is to provide an alternative to the common gameplay trope of competition, especially competition expressed as combat. Instead, *Broken World* invites the players to step into the shoes of an explorer of a post-fuel world, where the modern comforts of everyday life no longer exist. Players are tasked with rebuilding civilization, using the resources that they gather throughout the world to build cities and towns in a social environment with their fellow players. The core mechanics of *Broken World* involve resource gathering, tool crafting, and building placement.

The look and feel is an important part of *Broken World*. If the world is not convincing to the player, then it ruins the sense of immersion. The overarching goal of the art design is to create a visually appealing style that takes inspiration from reality. Concepts take reference from real world items, characters look and move in similar ways to anatomically correct humans and assets have realistic proportions. Other artistic goals are to create the assets with low polygon counts and to create assets in such a way that they can be placed modularly and still look like they are part of the environment.

In addition to the gameplay, *Broken World* has several technological goals. First and foremost, the game is an MMO, and as such supports many simultaneous player connections in a persistent world. This is necessarily supported by a strong networking backbone that allows information to be exchanged quickly and efficiently between all of the affected players. *Broken World* accomplishes all of this while still maintaining a low-resource server. This is made possible by *Broken World's* networking model, which has game clients connect directly to one another to share certain types of information. This allows each client to offload some of the work that is typically done by the server. When multiplied across hundreds of clients, this results in a significant decrease in memory, CPU, and bandwidth used by the server.

Another major goal of *Broken World* is that the world itself is expansive and persistent. This is accomplished through *Broken World*'s unique terrain generation system. The world is divided into "tiles", each of which is hand-crafted in a custom tool to be aesthetically pleasing with appropriate distribution of gameplay elements. These tiles are then arranged seamlessly by the server to form the world, and as players push out against the boundaries of the known world the server uses the tiles to generate a near-infinite space to explore. Tiles themselves are stored in an innovative manner, using image files to represent the necessary data, saving enormous quantities of disk space and allowing tiles to be easily transmitted over the network.

All of these elements are complemented by a set of development tools used to insert gameplay-related content into *Broken World*. These tools are custom-built to allow designers to add game content quickly, without having to involve a programmer. Because of the modular nature of the game's framework, content added through these tools can be automatically downloaded by the clients on the players' machines. This allows developers to change the gameplay over time—even while players are playing the game—without having to redeploy or patch the client software.

To evaluate the success of the project and its goals several tests were performed. Captures of network traffic were used to assess the efficiency and scalability of the network code. Through the use of computer performance monitors the client and server were both evaluated for their overall performance under different operating conditions. A user study and accompanying survey were later used to get player feedback and opinions on the gameplay, art style, and overall feel of the game.

Through the testing it was found that *Broken World* does in fact meet many of the goals set out for it. Players found the gameplay concept to be promising, but felt that the game lacked a level of polish, which detracted from their experience. Players also felt that the environment felt immersive and expansive. The server was able to handle 100 connected clients with ease, even when run on low-end hardware.

The following sections explain in detail the technologies involved in the creation of *Broken World*, as well as the motivations and impact of *Broken World*. Section 2 gives some general information about other works that are similar to *Broken World*. Section 3 details the gameplay aspects of *Broken World*, including its setting, theme, and player actions within the game. Section 4 describes the MMO framework upon which *Broken World* is built from a

technological perspective. Section 5 explains both the methodology involved in producing the art found in *Broken World* and the artistic motivations behind it. Section 6 describes the tests that were run to evaluate *Broken World* and the results of those tests. Section 7 analyzes the ways in which *Broken World* fits into the larger context of video games and software and contains a discussion of the successes and failures of *Broken World*'s development process. Section 8 lists the team members and their roles in the production of *Broken World*.

2 Background Information

Broken World is a fairly unique concept, but is not without precedent. As a video game, *Broken World* draws on many concepts from a widely varied set of fields. As with most video games, the gameplay mechanics are a combination of mechanics drawn from other games, with novelty arising from the context in which they are used. *Broken World's* theme and setting are a twist on existing themes in pop culture movies and video games. The technologies used in the construction of *Broken World* are based on existing software paradigms. In order to fully understand the areas in which *Broken World* innovates, one needs to first understand the larger context of existing video games and other software. The following sections explain this context.

2.1 Setting and Theme

The setting of *Broken World* is similar in many ways to the post-apocalyptic science fiction sub-genre. Works in this sub-genre focus on civilization-ending events, and on the way society looks after a worldwide disaster. One example work in games is the *Fallout* series, which takes place in a world devastated by nuclear war.[7] Characters' primary focus is survival. However, the *Fallout* series of games portray a cutthroat, wild-west type world where the law of the land is to kill or be killed. Figure 1 shows a character walking through a barren wasteland, the gun strapped to his back a symbol of the violence prevalent in the *Fallout* series. This interpretation of a devastated society devolving into complete barbarism is very common in post-apocalyptic fiction. *Broken World* suggests an alternative: that humankind could instead focus on reconstructing society. This type of theme is much less common.

Figure 1: Fallout 3[8]

Broken World does not take place in a barren nuclear wasteland like the *Fallout* (1997) series. Its setting is much closer to that of the movie in Figure 2: *I Am Legend*[9] (2007). *I Am Legend* takes place in a world where a disease turned humans into zombies, removing their influence on the world, but left the land unscathed. Grass and trees grow throughout the streets of New York, slowly reclaiming the metropolis into the landscape. This creates an environment that better lends itself to rebuilding.

Figure 2: I Am Legend

Much of post-apocalyptic fiction includes highly fantastical elements, such as *Fallout*'s super mutants and *I Am Legend*'s zombies. The catastrophic event in *Broken World* is that the world ran out of fossil fuels, so threats are more closely grounded in reality. In this sense, it is

similar to works such as the movie *Mad Max*[10] (1979), which is also set in a post-fuel world. *Mad Max*, however, focuses on a lone wolf fighting his way through a world of humans treating each other brutally. Self-reliance and an uncaring attitude are shown as necessary for survival in the wastes. These traits are not the type of traits that *Broken World* emphasizes. *Broken World* provides an alternative solution: that by working together, people can rebuild society and achieve great things.

2.2 Gameplay Inspiration

Post-apocalyptic games like the *Fallout*, *S.T.A.L.K.E.R.*[11] (2007), or *Metro 2033*[12] (2010) series all focus on combat as the primary mechanic. Many of these games have non-combat based sub-mechanics, however, which make for interesting gameplay. The *Fallout* games, for instance, encourage players to gather mundane everyday items—scraps scavenged from the past civilization—and use them to build useful items. Items degrade over time with use, and must be repaired using parts scavenged throughout the world. *Broken World* uses scavenging as a core mechanic.

There are many massively multiplayer online role-playing games (MMORPGs) that involve non-combative play such as resource collection and crafting. *Wurm Online*[13] (2006) spends the first hour of the tutorial teaching players how to judge the type and age of various kinds of trees, and the ways in which players can cut them down and use them to build wooden objects and fires. *World of Warcraft*[14] (2004) includes a basic crafting system. As seen in Figure 3, *Runescape*[15] (2001) allows the player to engage in many non-combat skills such as fishing and woodcutting. In all of these examples, however, the primary reason to engage in these activities is so they can gather resources to better their combat skills. *Broken World* elevates these roles to primary mechanics, eliminating combat altogether.

Figure 3: Runescape

An example of another game whose focus is not combat is *Minecraft*[16] (2009). The main gameplay element in *Minecraft* is going out into a procedurally generated world and gathering resources. These resources are used to construct a home base. Resources can be crafted into other, more selectively useful items like torches for light and tools for faster resource gathering. Players are required to explore as resources closer to home become scarce. *Broken World* draws inspiration from a number of these gameplay elements, but *Broken World*'s execution is very different than that of *Minecraft*. Where *Minecraft* uses an abstract building block world as seen in Figure 4 to implement many of these ideas, *Broken World* uses representations closer to that of games from other genres.

Figure 4: Minecraft

Resource collection and construction in *Broken World* draws more from real-time strategy games such as *Empire Earth*[17] (2001) than from *Minecraft*. In *Empire Earth*, players must explore a procedurally generated map in search of resources such as iron, gold, stone, wood, and food. These resources are used to build infrastructure, such as roads, harbors, barracks, and other buildings. *Broken World* also has many different kinds of resources, and building construction works similarly, requiring players to amass a certain number of resources before they can complete construction.

The closest existing game to *Broken World* is a MMORPG called *A Tale in the Desert*[18] (2003). *A Tale in the Desert* has no combat system at all. Instead, players progress through the game by acquiring new skills. Players start by learning how to make bricks and plant flax, and learn more and more complex skills until they can build everything from pearl necklaces to obelisks. Tasks are built on top of one another such that the process of spinning silk might require that the player learn stonecutting along the way. The game is fun as a result of its complexity. There are many different areas to explore, and many different tasks to complete in the game, which drives players to want to continue acquiring new skills and trying different aspects.

2.3 Existing Technologies

Like all video games, MMOs can be built in a multitude of different ways. Many video games are built from scratch, but more and more frequently video games are being built on top of

existing pieces of software, collectively called "game engines". By building a game in a game engine, developers can avoid rewriting the same code that is common to many video games, and instead focus on the game logic that is unique to their game. There are a multitude of game engines that developers can choose from, ranging from engines specializing in a particular type of game to more generic game engines.

There are only a few game engines built specifically for MMO development. The Esenthel Engine[19] is one example, however it has not to date been used for any high-profile projects. More commonly, MMOs that use engines build them on top of more general engines. Recent MMO *Star Wars: The Old Republic*[20] (2011) was built in the HeroEngine[21], an engine used for any sort of online game development. Another popular engine is the Unity Engine[22]. There are thousands of games built in the Unity Engine, many of which are MMOs.[23] Unlike many other game engines, the Unity Engine has a free version, and is affordable for independently-published projects and large-scale projects alike.

By default, the Unity engine does not have many tools to address features specific to the MMO genre. There do exist a few solutions in the Unity Asset Store, such as the Ratspell MMO Toolkit[24], but they are not all free, and there are not many choices available to developers. Most developers using the Unity engine need to create their own MMO frameworks from scratch, especially if they want to make a game that differs from the typical MMO model.

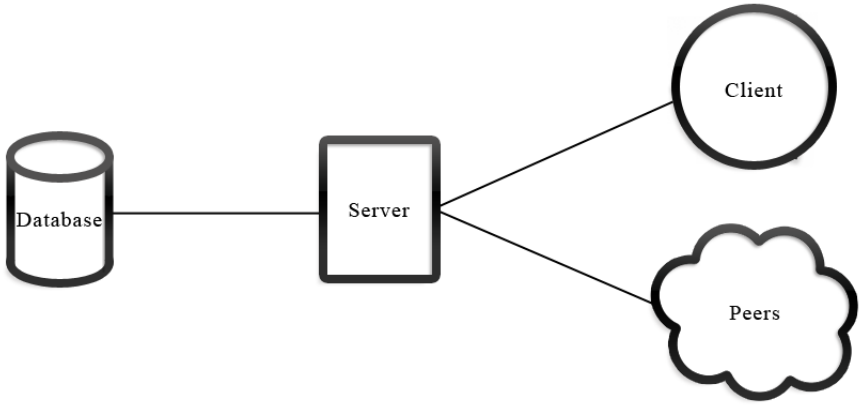


Figure 5: The client-server networking paradigm used by most MMOs

The networking model most used in MMOs—and arguably most networking in general—is the client-server paradigm, seen in Figure 5. For MMOs this model allows clients to connect to a static server whose IP or network address remains constant, and communicate solely with that server for the duration of their play session. The server is responsible for client authentication,

processing packets from the client which describe what the client is doing, letting the client know the world state and any changes to that state, as well as sending the proper packets to the client which describe what the other players are doing. What the client is responsible for varies from game to game, but is generally limited to the player's actions, local physics, particles, and other graphical representations. This requires the server to keep track of and manage every other aspect of the game. All player chat and interactions are relayed through the server and sometimes evaluated by the server for correctness. The server is also responsible for any generation that is needed, like item drops or creature spawns.

This model continues to be very successful for many companies so there is often little to no need to try something new. The model also lends itself to being fairly secure as anything that needs to be hidden from the player can simply be handled by the server or checked to make sure the player is operating under the game's normal operating constraints. The client-server model is also very well documented, including its use in games, making it easier in some ways to implement and troubleshoot.

The downsides to this traditional model become apparent for studios that have a small production budget or lack the overall resources needed to host a server. All the calculations and management required of the server means it must be fairly powerful. Most MMO servers, while appearing to the client as being one computer, are actually a cluster of computers all working together to provide the services of the game to all their players in a timely fashion.[25] The bandwidth necessary, while not as expensive a problem, is still something that must be factored in when deciding how to build a server. For small companies or student groups, these costs make hosting an MMO server unsustainable.

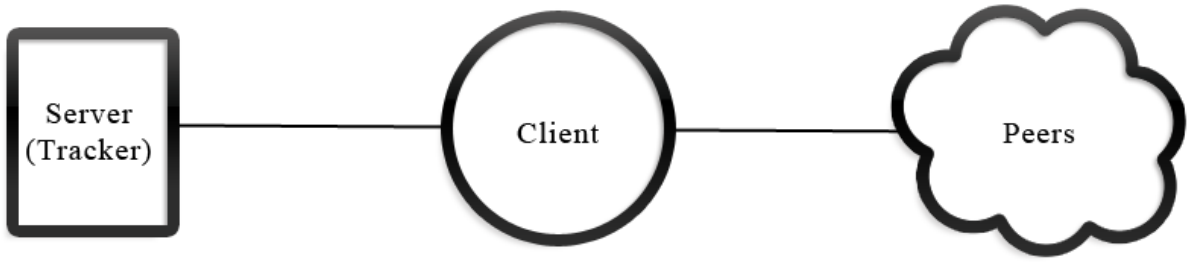


Figure 6: The BitTorrent networking paradigm

Another networking paradigm is that of BitTorrent. BitTorrent allows clients to connect to each other in a more distributed fashion which takes most of the network traffic load off the server by divvying it up among the clients. Clients still initially connect to a server—often

referred to as a tracker—but are then given a list of peers (other clients) to whom they directly connect. In this model the clients do all of the heavy lifting and the server merely keeps track of the locations of the clients. The key benefit is that the load is distributed throughout the system, allowing it to be run off of a much weaker server.[26] There are not many games—and no known MMOs at the time of this writing—that use the BitTorrent model for networking. This model doesn't immediately lend itself to use in MMOs. There are a myriad of security problems with allowing players to directly tell other players what they are currently doing. Also, it would be unclear who is keeping track of the world state, as well as what happens when all your players go offline, or when the only player who had an up-to-date world state suddenly disconnects.

Another important technology in MMO development is the method through which the world is created. There are many different available models, each with their own benefits and detriments. The most common method of world generation, as seen in games like *World of Warcraft*, is to have level designers generate the entire world statically.[27] By using either in-house tools or tools provided by the game engine, level designers can generate the landscape and place objects by hand. The benefit of this method is that it allows each area of the game to feel unique, and gives designers full control over the way the world is laid out. Unfortunately, it can be a time-consuming process requiring many level designers to generate an entire world, especially for an MMO—which is expected, by its definition, to have a massive world.

On the other end of the spectrum are games like *Love*, which generate the entire MMO world procedurally.[28] The world is generated using a set of rules, rather than being predefined by a level designer. The benefit of this method is that it allows the creation of a gigantic world with a small development team. *Love*, for example, was built by a single person. The downside of using procedural generation to create an MMO world is that the designer loses control over what the game world will look like. If done poorly, procedurally-generated worlds can feel like they have nothing unique in them.

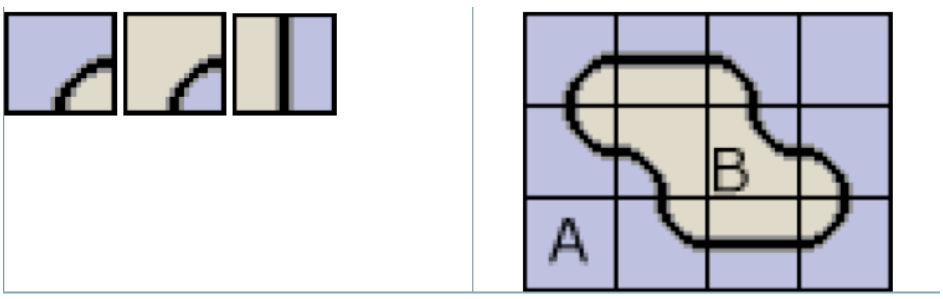


Figure 7: The tile system used in Ligo

Ryzom uses a tiling system called "Ligo" to help their level designers make a fun and interesting world.[29] They have a system of 2D tiles, seen on the left side of Figure 7, each of which represent a 3D landscape piece that was made by hand by a level designer, which they can place together to form the world, as seen on the right side of Figure 7. *Broken World* uses a similar system which allows for a procedural world to be quickly generated in 2D and then seen by the players in 3D. Developers can craft each tile and then allow the generation system to place them in interesting ways as defined by an algorithm.

Images are frequently used in games for a variety of purposes: textures, user interface elements, fonts, and more. Image formats can be compressed or uncompressed. Uncompressed images, like the BMP format[30], merely represent the raw image data without any effort to shrink it in size. Compressed formats, however, use some trick to represent the same data in a smaller filesize. There are two types of compression: lossless and lossy.[31] Lossy compression does not guarantee that the image will look exactly the same when it is decompressed, while lossless compression does. Some types of images, such as photographs, often look fine using lossy compression, but other types of images require pixel-perfect precision.

One of the major losslessly compressed image formats is the Portable Network Graphics (PNG) format.[32] PNGs have five different types of filters that can be applied to each row of pixels that change the way they are stored.[33] When chosen well, these filters can be used to optimize the PNG on a row-by-row basis, allowing the compression algorithm used to shrink the size of PNGs to reduce them in size by many orders of magnitude. Most programs do not take the time to fully optimize their use of PNG row filters when they save out PNGs, but there exist many different PNG optimizer programs which, through a fairly time-intensive process, pick each row filter in such a way as to shrink the image to its smallest possible size. *Broken World* makes extensive use of PNGs in both traditional and novel ways, taking advantage of their exceptional compression algorithms. See Section 4.3 for more details.

3 Game Design

Broken World is a non-violent, non-competitive MMO focused on community building and resource gathering. Players build buildings and items as they explore an expansive, procedurally generated world. This game is an attempt to create a multiplayer experience based on primarily constructive gameplay mechanics rather than the current trend of destructive gameplay mechanics. *Broken World* suggests an alternative to the belief that a catastrophic event would create widespread selfishness and violence. Instead, *Broken World* suggests the more optimistic view that society could rebuild itself in a more civilized manner.

3.1 Setting

Figure 8: Concept art showing how the "old world" still exists, but is subsumed by nature

Broken World takes place in a potential future United States (particularly the New England area) where fossil fuels and many other natural resources have been used up. Power plants have ceased functioning and gas pumps remain empty. No one has easy access to electricity for lights or gas for cars. Many technological advances no longer work. Cities and towns begin to erode as there is no easy way to maintain them. Nature has slowly begun growing over what were once heavily populated areas.

Figure 9: Concept art showing open fields and forests

The open world which the player explores consists mostly of open fields, with landscape features such as forests, mountains, abandoned towns, and campsites. Pioneers are able to search these areas and salvage goods needed to survive. The world is unkempt and overgrown, as the human influence over the land has decreased.

3.2 Gameplay

Broken World intentionally has a slower pace than most modern MMOs. Focusing on exploration and town-building instead of the overused action tropes common in other games, it invites players to set their own goals. The actions available to players involve searching for and collecting raw materials, crafting tools using those raw materials, and then placing buildings throughout the persistent world. *Broken World* is meant as a social experience, as players can see and communicate with one another as they play.

Player goals in *Broken World* are not as explicit as the goals in other MMORPGs. Many MMORPGs require players to complete quests, which drives the gameplay forward. Broken World is fairly open-ended and player-driven, similar to *Second Life* (2003). *Second Life* is a persistent virtual space which does not lay out any explicit goals for players to achieve. Instead,

it serves as a platform in which players can socialize and create their own goals. Players do whatever they find most interesting and play at their own pace.

Like many Massively Multiplayer Online Role-Playing Games, there is no "win condition" for Broken World. Instead, the game is driven by self-imposed goals. Players can choose to focus on whichever activities they most enjoy, be that exploring the world in search of new frontiers, socializing with other players, or rebuilding towns and cities. Broken World is a social sandbox game.

Much of the players' time is spent exploring the expansive, procedurally-generated world looking for resources. The further players press out in the wilderness, the more likely they will be to discover resource caches that nobody else has found before. Players can find many different caches of resources such as trees, old abandoned oil barrels, or mineral deposits. With the proper tools (axes for cutting down trees, or a pickaxe for mining, as a few examples) players can collect these resources and use them to build other tools and buildings.

The primary way for players to get tools is to craft them out of raw materials. Players are able to build items from the various materials that they scavenge throughout the world. Most of these items grant them some ability: Axes can be used to cut down trees, picks can be used to mine rocks, and saws can be used to turn felled trees into wood, as just a few examples.

Figure 10: Ryzom's crafting system

The crafting system in *Broken World* is inspired largely by the crafting system of an MMORPG called *Ryzom*[34] (2004). In *Ryzom* crafting is done with recipes that are comprised of two parts: amounts of resources, and the resources themselves. For example, in Figure 10 the player is crafting a piece of armor. The recipe calls for 4 armor shell resources, 4 lining resources, 2 stuffing resources, and 2 armor clip resources. The player then chooses the specific resources they wish to use from their inventory. The crafting system in *Broken World* works roughly the same way.

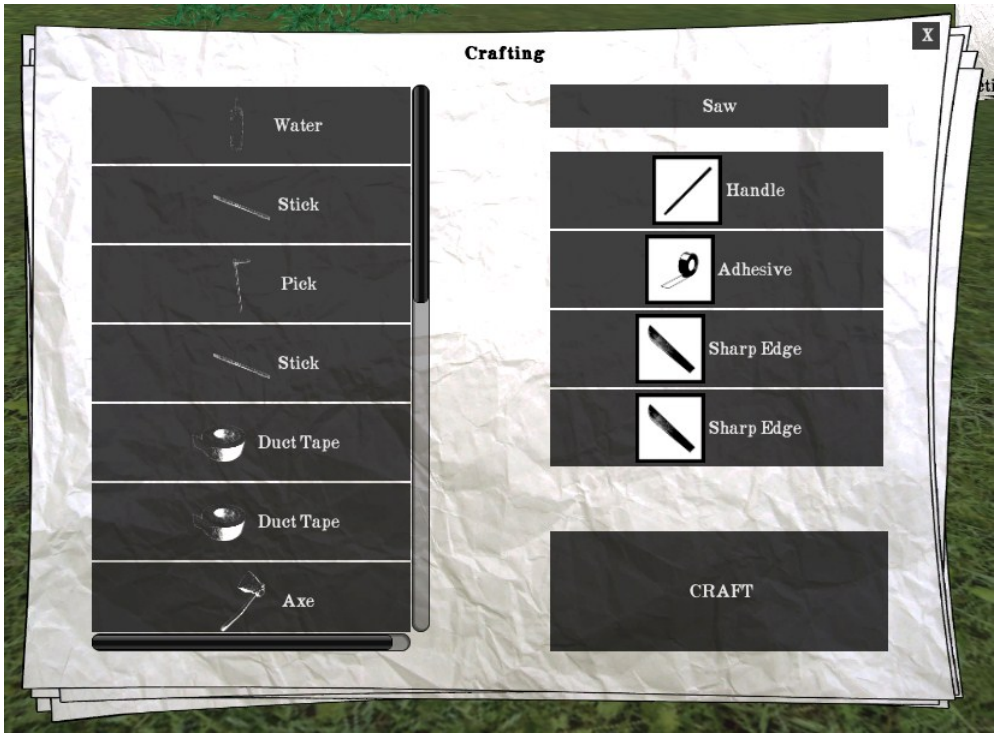


Figure 11: The crafting menu in Broken World

Items are crafted using recipes, which are essentially a list of "item roles" required to build that item. On the right side of the crafting menu shown in Figure 11, the recipe for a saw is listed as requiring a handle, an adhesive, and a sharp edge. The player can pick items from his or her inventory (shown on the left) that fill these roles. Once all these roles have been filled, the item can be crafted. An item may potentially be used to fill many different roles, and most items can themselves be used to fill certain roles in crafting other objects. See Section 4.2.5B for more details on how items and item roles are added to the game.

Figure 12: Early concept art for a building

With enough materials, players are also able to craft buildings. Buildings are crafted in the same way as other items, but they have a significantly larger number of constituent parts. Once crafted, the player is given a blueprint that can be set as the active item and then placed anywhere in the world. In this way, players can change the look of their environment. Players can even work together to build conglomerations of buildings into towns.

4 Technology

Games straddle many different technical fields within computer science. *Broken World* must render a 3D world, provide communication between players, store player data, and give players a means to manipulate the world state, to say nothing of the gameplay logic itself. As such, it draws heavily on the fields of computer graphics, networking, data storage, and human-computer interaction. In addition to the software comprising the game itself, a host of tools had to be constructed in order to manage the game content. The following sections detail the technologies used in the implementation of the various parts of *Broken World*, as well as the tools constructed to manage the gameplay content.

4.1 Game Client

The *Broken World* game client is implemented in the Unity game engine, using primarily C# scripts. Unity 3D was chosen over other potential game engines because it already implemented a lot of what was needed for the game: it handles rendering of the 3D world, user input, management of the hierarchical scene graph, and the game loop. It is also easy to extend in C# using the power of the Mono Framework, which provides a lot of basic functionality ranging from list management to networking capabilities.[35] Another benefit of using Unity as a game engine is the ease with which new art assets can be added to the game—files can simply be dragged from the filesystem into the engine without having to deal with the complicated import procedures commonly found in other engines.

While Unity does provide a built-in physics engine and rudimentary networking support, neither of these components gave the efficiency or level of depth that was needed to implement *Broken World*, so they were replaced with custom components written as extensions in C#.

4.2 Network Layer

As with any MMO, the networking component makes up a significant portion of *Broken World*'s infrastructure. Section 2.3 describes two networking models: the client-server model and the BitTorrent model. *Broken World* uses a hybrid solution between these two paradigms, where

clients connect to both a static server which keeps track of the world state as well as directly to some of the other clients to exchange information considered non-crucial to gameplay. MMO servers are typically resource-intensive programs that require powerful machines to run. The reduced load on the server through the *Broken World* model allows for a much lighter server. With a lighter server, fewer machines are needed to be able to service a large number of players, and the machines themselves need not be as strong. This becomes a net savings in both the initial cost of machines purchased to run the MMO server as well as the maintenance cost of running the machines. As a practical matter, it also makes it feasible for college students to develop and test an MMO with limited access to powerful hardware.

Broken World is structured as several separate programs, each working in tandem to provide the complete experience. The five parts that make up a single player's *Broken World* experience are the MySQL database[36], the login server, the state server, the client, and the client's peers.

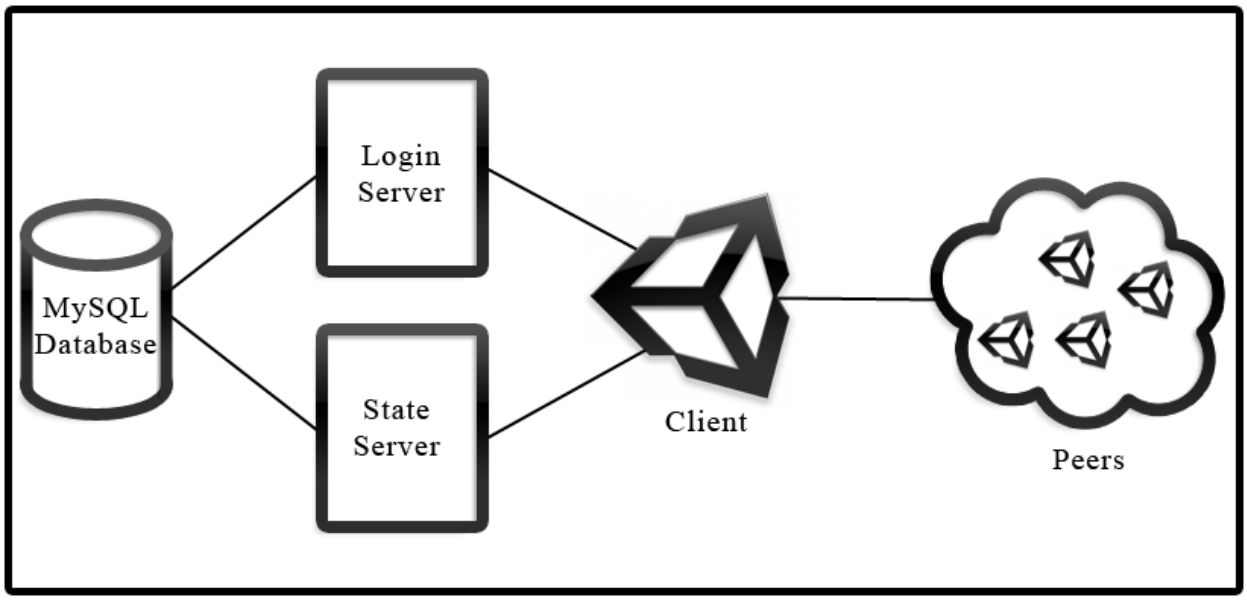


Figure 13: The program structure of Broken World

These five pieces of software exchange information as shown in Figure 13. In both the development environment and the live environment the MySQL database, login server, and state server all run on the same machine together, but there is no reason they cannot each be run on separate machines. This modular design means that were *Broken World* to be run in a high-load environment, more powerful machines could be used to host the MySQL database and state server, with a lower-quality machine running the less CPU and memory-intensive login server.

Each element has a specific role to play. The MySQL database stores all of the persistent state information of the game world. Any changes that the players make to the world are ultimately stored in the database. The login server, implemented in C++, is responsible for authenticating users and logging them in for a given play session. The state server, also implemented in C++, is the trusted server-side program with which the client interacts in order to retrieve and change the world state. The client program is the game program with which the user directly interacts, and it is responsible for both rendering the game world and notifying the other parts of the system of changes that the user wants to make. The client also interacts directly with other game clients of other players running the game to exchange certain low-impact information about the game state.

4.2.1 State Server to Login Server Communication

As can be seen in Figure 13, the state server and login server do not directly communicate. They do, however, pass information between one another by changing rows in the MySQL database. This is a simple form of communication, but it makes sense for the servers because they merely need to pass the latest data between one another. They never need to send a message that needs to be addressed immediately by the other.

The first example of this sort of communication is that the state server, upon starting up, places its IP address and the port it is running on in a table in the database. The login server can later query this table to find the latest location of the state server to which it can direct players. Implementing the server in this manner allows the state server to be moved from one machine to another (perhaps in the event of a hardware failure, or the purchase of a new machine) without having to shut down the login server too.

The second piece of information exchanged between the login server and the state server is the session identification number which is generated by the login server and then read by the state server when users take certain actions. This number is used to verify that the user who logged in is indeed the same user who is contacting the state server—if they know the generated session identification number, then they must be the same user.

4.2.2 Client to Login Server Communication

The first server with which the client interacts is the login server. This communication is done entirely over the Transmission Control Protocol (TCP). TCP is a transport-layer networking protocol, which means it is responsible for ensuring that messages from one machine reach the destination machine. TCP itself is connection-based: it establishes a connection with the destination machine and then holds that connection open for easy back-and-forth communication. One of the primary advantages of TCP over other transport-layer protocols is that it guarantees that data sent over the network will reach the other side if it is possible. This does require some extra processing overhead, however.

The login server's hostname is hard-coded into the client. This hostname needs to be resolved into an IP address using the domain name system (DNS) which effectively turns a string into an IP address. Because the login server's location is stored as a hostname instead of an IP address, the location of the login server can be changed by updating its location in the DNS. This means that, in the event of a hardware failure or system upgrade, the location of the login server can be changed without having to rebuild and redistribute the clients.

The two actions that the login server handles are logging in and creating a new account. The first time that a user tries to initiate one of these actions, a TCP connection is established between the login server and the client. That TCP connection is then held open until either the client program is terminated, the user successfully logs in, or the TCP connection is idle long enough for it to time out. Each individual action has a unique integer which is sent before any other information so that the login server can identify which action the player wants to use. Any other relevant data is transmitted after the action identification code.

When the player creates an account, a row is added to a table in the MySQL database. When the player tries to log in, the password they enter is checked against the one stored in the database. If it is correct, the login server generates a unique number to represent the current play session for that user. This session number is stored in the database and sent back to the client. The login server then sends the IP address and port at which the state server can be located. Upon receiving this information, the client stores the session number in memory, terminates the TCP connection with the login server, and establishes a connection with the state server.

4.2.3 Client to State Server Communication

Like the login server, the client communicates with the state server via TCP. Also like the login server, different actions are differentiated using an action identification code, which is always the first integer sent to initiate communication between the client and state server. There are many different actions that get passed between the client and state server, some of which request data and others which request that changes be made in the database. The client initiates most of these requests, but there are a few messages that the server sends to the client as well.

As soon as the client establishes a connection with the state server, it requests the landscape tiles surrounding the player's current position. This means nine tiles in total—the one the player is currently on, as well as the eight tiles that border it. In addition, the client requests the list of all other logged in players (along with their IP addresses and ports on which they are listening) that are currently occupying one of those nine tiles.

Objects in the world as well as items from the inventory are loaded in a just-in-time manner and stored in a cache on the client's local machine. Objects are stored as Unity assetbundle files, which are essentially the collection of 3D models, scripts, and whatever else makes up an object. These files can be stored on the local machine and loaded into the client at runtime. By storing the objects as cached assetbundle files, the client is able to check the version number of an object (as sent by the state server) against the version it has cached. If the cached version is lower than the version number sent, or if the object has not been cached yet, the client requests the latest version of the assetbundle and downloads it from the state server. Inventory items work precisely the same way, except instead of being stored as assetbundle files, they are simply stored as PNG files representing the icon of the item. By loading items and world objects at runtime like this, changes can be made to the game content while the players are playing without having to patch or redistribute the client. Models, icons, and even game logic in the form of scripts can be changed in the database and automatically downloaded by all of the clients without having to restart any part of the *Broken World* system.

Any time the player switches tiles, the client notifies the state server of the change and requests the new bordering tiles and players. The state server also marks this change in the database. Any time the player wants to make changes to the world state—be that adding items to the inventory from a container in the world, crafting an item, placing a building, or making any other sort of change to the world—the client sends the action identification number to the state

server followed by its session identification number. The state server then can check that the user performing the action is logged in, and then make the necessary changes to the database.

Most actions affect only a single player's state, and therefore can be performed without the need to synchronize with other players. A few actions—such as moving an item from a container in the world to a player's inventory—do depend on the order in which player actions occur. If two players try to retrieve the same item at the same time, one needs to be allowed to retrieve the item while the other needs to be denied. Implemented improperly, this situation could lead to what is known as a race condition: a bug introduced into the code as a result of time-dependant factors. The state server avoids this by placing a mutex—a low-level lock on a particular area of code—around operations changing the contents of containers.

When players do make changes to the world state that can be seen by other players—moving from one tile to another, placing a building, and so on—the state server notifies all players within one tile of the affected area about the change. Each client can then update its representation of the world.

4.2.4 Client to Peer Communication

Clients maintain a list of all players logged in within a one-tile radius of their current location. In a traditional client-server game model, clients would send regular updates to the server containing small changes that the player makes to his or her state, such as the character's location and rotation as he or she moves around. *Broken World* takes a somewhat novel approach to these small state changes. Rather than burdening the server with constant updates from all logged in players, the clients merely notify other nearby clients of these small changes. Clients notify each other of position and rotation changes, as well as chat messages that the player sends; none of these communications involve the server at all.

These changes are sent using the User Datagram Protocol (UDP) instead of TCP. UDP has less overhead than TCP does because it does not try to establish and hold open a connection. Instead, it sends a packet over the network and does not bother to check whether it arrived at its destination or not. This means that some packets may end up being dropped, but because of the frequently-changing nature of these small changes to player state, it is not a large loss. Positional data is sent 10 times a second to provide smooth movement on the respective clients. If a packet gets dropped the client will merely interpolate movement based upon the packets it has received.

When packets arrive out of order the older packets are ignored to prevent unnecessary jitter from occurring.

Bypassing the server for small, frequent changes of a localized nature cuts out a lot of communication that traditional servers would have to deal with. The only time that any given client contacts the server is when it makes a major change to the world state. This means that the server can be run on a much weaker machine than traditional MMO servers. See Section 6.1: Analysis for more details on the ramifications of this model.

4.3 Dynamic Tiles

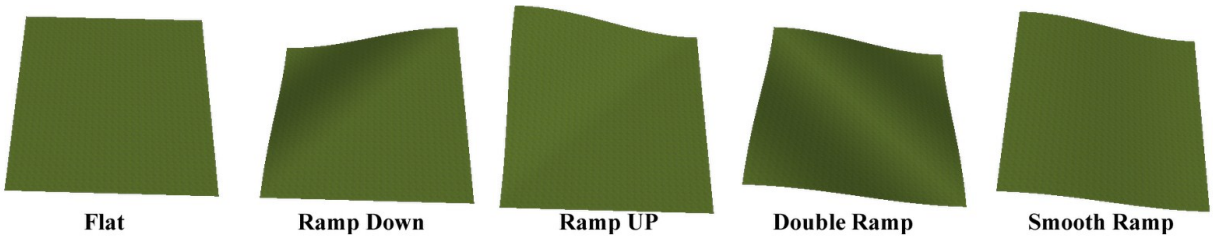


Figure 14: The five different possible types of tiles

The world of *Broken World* is divided up into 500 by 500 game unit chunks, called tiles, which are fit together based on simple rules so that their sides all fit together seamlessly. The sides of a tile can be classified into one of three types; flat, sloped upwards, and sloped downwards. These sides form five different tile types as seen in Figure 14. Each tile type was named based upon its appearance; flat, smooth ramp, ramp up, ramp down, and double ramp. With these five tile types it becomes fairly straightforward to both generate a seamless world as well as allowing for an interesting world with variable heights.

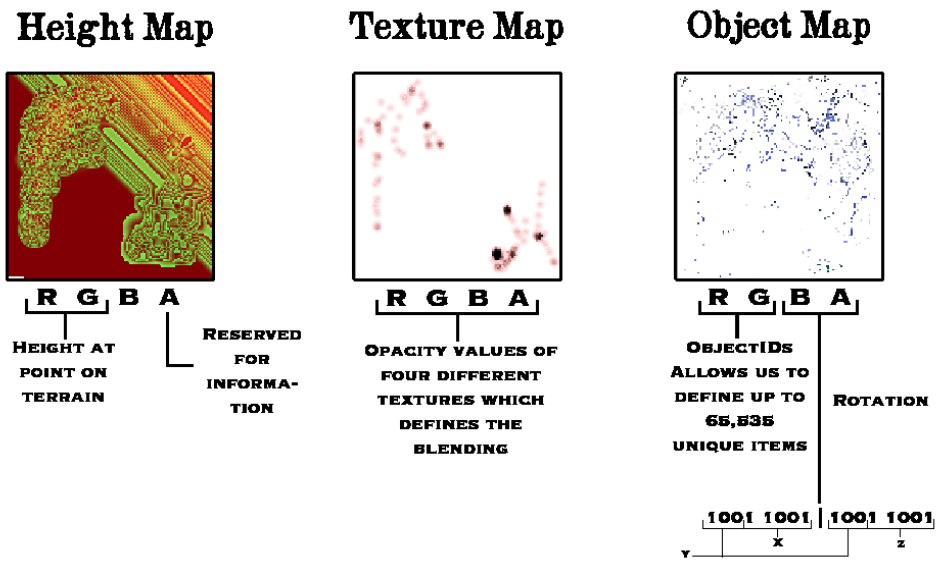


Figure 15: What each channel is used for in the pixel maps

Terrain information is saved and transmitted as a series of tiles consisting of three images; a heightmap, texturemap, and objectmap. Each tile is 500 by 500 units in the game space represented by 128 by 128 pixel images. The PNG image format is used to store the images due to its lossless compression, which results in small files without losing any data. Clients load the images from the server as needed and dynamically build the world using the three images as lookup tables for various values.

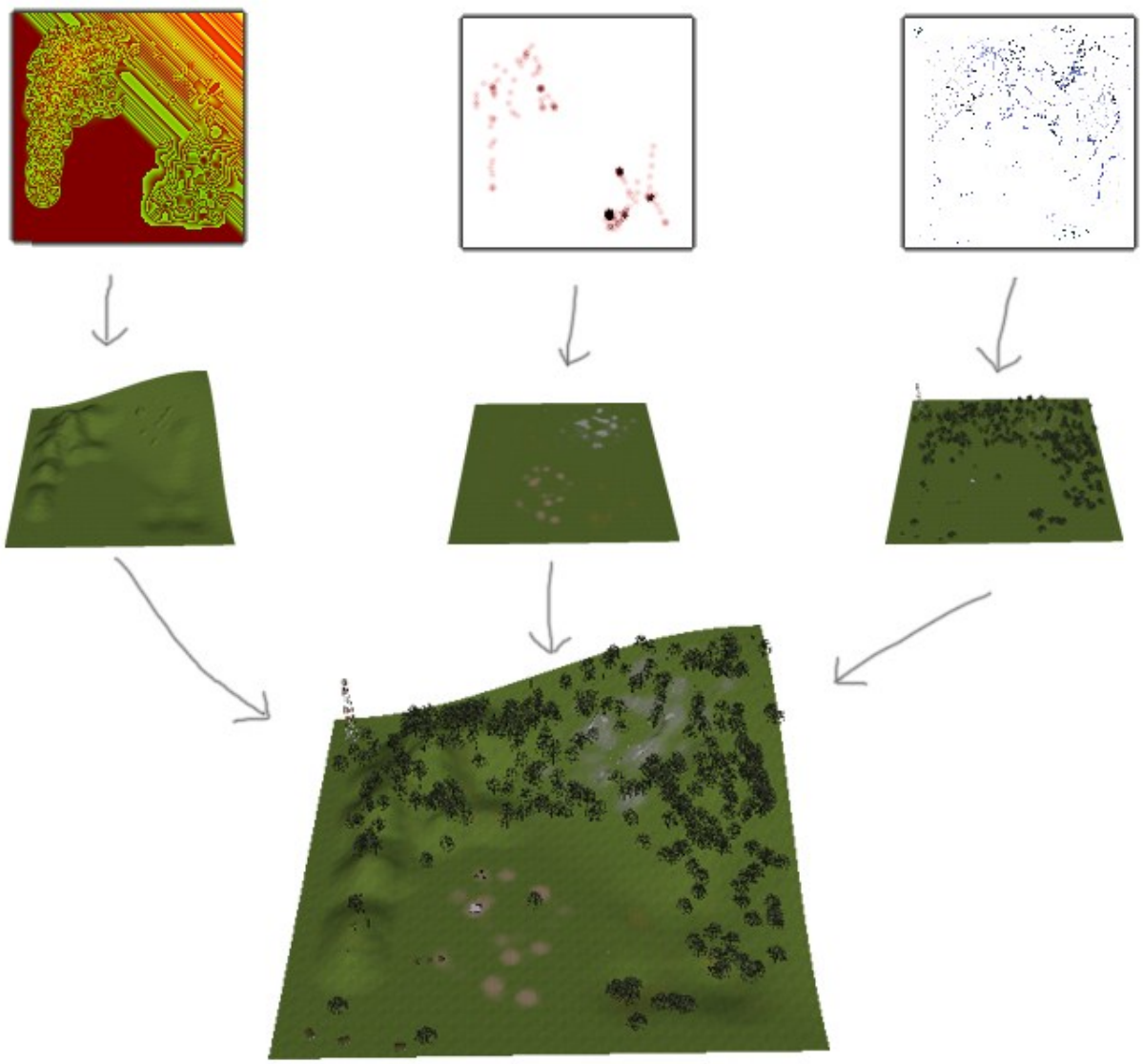


Figure 16: Tile generation using PNGs

As seen in Figure 15, in the heightmaps the red and green channels are used to store the height data for the terrain. The red channel is used as the first 8 bits and the green channel as the second 8 bits resulting in 16 bits of resolution for elevation detail. The height of each tile ranges from -128 to 128 game units with 65,536 steps of resolution. Each pixel on the map corresponds to a single vertex on the resulting mesh where the X, Z location is determined by the location of the pixel on the image and the y location is defined by the red and green channels. The blue channel is unused. The alpha channel is used to store metadata for the tile. For example, the height level of a tile, defined in 62.5 unit increments, is defined using the alpha values of four pixels along the bottom of the image. The rotation of a tile, the base type of the tile, and the version info are all stored.

As explained in Figure 15, the texturemap acts as a mixing mask for four other textures. Each channel reflects the opacity of a particular map. As implemented, the red channel controls a grass texture, the green channel controls a stone texture, the blue channel controls a dirt texture, and the alpha channel controls a second grass texture which is less dense. The texture map is rendered by a shader which uses the map as a lookup table to mix the four different textures.

The object map defines where and how objects are placed on a given terrain tile. Objects include trees, buildings, lock-boxes, and other static meshes found throughout the world. As can be seen in Figure 15, the red and green channels are used to define the ID of the object. When an object is created by the developers it gets placed into a MySQL table where the ID of the row in the table corresponds to the ID stored in the image map. The blue and alpha channels are then used to define the X, Y, Z rotation of each object in Cartesian space. Because it is used more commonly than other axes, the Y rotation requires greater fidelity than X or Z rotation. For that reason 6 bits are dedicated to Y rotation, which provides 64 steps of resolution, and only 5 bits are used for the X and Z rotations providing 32 steps of resolution. The Y rotation allows for 360 degree rotation while the X and Z are only allowed 180 degree rotation (from -90 to 90 degrees), meaning that the same level of rotation resolution was provided for each axis. As such, objects can be rotated all the way around the Y axis, but cannot be flipped upside-down.

When players come close to parts of the world which do not yet exist the server generates an appropriate tile by taking a premade tile from the database of the appropriate type and adjusting the rotation or height level of the tile as necessary. It is able to do this because each of the tile types have specifically-defined edges which can be guaranteed to fit together. The flat edges of tiles are put together and the upward sloped edges are paired with downward sloped edges. Tiles are chosen based on the one to four already generated tiles they will find themselves adjacent to. To add variation to the landscape the generation code figures out all the possible tiles that can be in a particular place and then chooses one at random for placement.

Certain actions in the game, such as when players place or change objects on the terrain, required the modification of pixels in the PNG maps. When these modifications are done on the client side, they are done through C# functions native to the Mono framework. When these modifications are done on the server side, they use the libpng++ library[37], which is a C++ wrapper over the C libpng library[38].

4.4 Database

Broken World uses a MySQL database to store world and player state. MySQL was chosen over other possible database types because it is free to use (even for commercial development) and has a wide base of support. Several different technologies have direct access to the database. The state and login servers manipulate the database using the MySQL++ open-source driver, chosen because it easily supports multi-threaded programming.[39] Several development tools also access the MySQL database using PHP's built-in MySQL functions.

The database contains four major classes of tables: networking, items, world state, and users. These are informal classes useful for organizing and describing tables, but are not formalized anywhere in the code of *Broken World*.

4.4.1 Networking

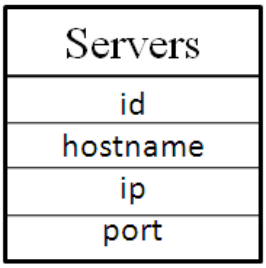


Figure 17: Entity relationship diagram for the networking-related tables

The only table in the networking category is the Servers table, which is used to store the IP address and port on which the state server is currently running. Each time the state server is started up it places its IP and port into this table, which the login server can retrieve when users log in to direct them to the state server.

4.4.2 Items

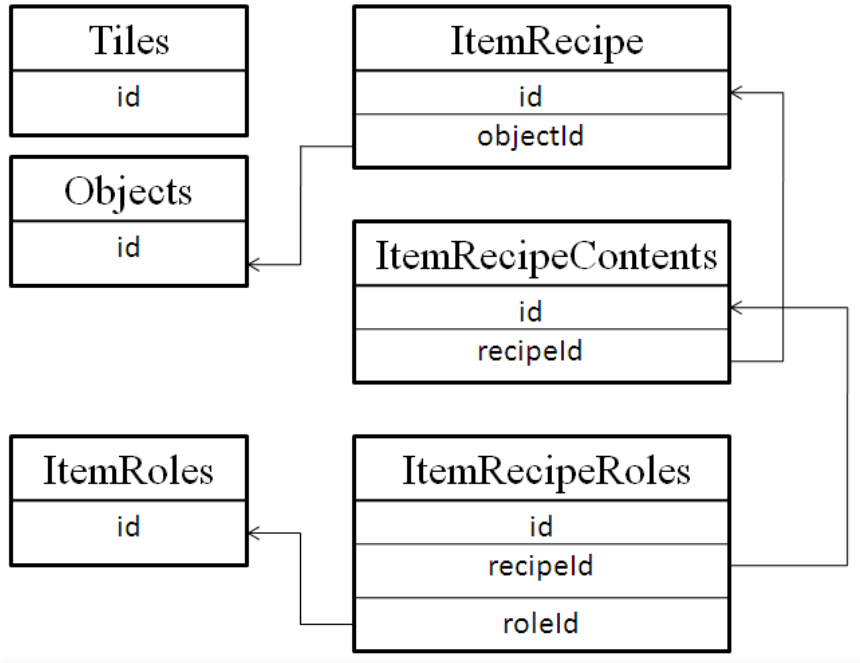


Figure 18: Entity relationship diagram for the item-related tables

There are several tables relating to items. Item representations are fairly complex as a result of *Broken World*'s crafting system. An item recipe, as defined by a row in the ItemRecipes table, represents a single type of object that can be crafted. In order for a recipe to be completed, several items must be brought together that fill certain roles in the recipe. For example, in a hammer recipe, these roles might be that the player needs a handle, a weight, and an adhesive to stick them together. These roles ("handle", "weight", and "adhesive") are defined in the ItemRoles table, while the particular roles needed to fill a recipe are stored in the ItemRecipeContents table. Because a recipe might itself be used to fill a role in another recipe, the roles that a given recipe can be used to fill are stored in the ItemRecipeRoles table.

Recipes represent the different kinds of items that can be crafted, but specific instances of items—a hammer built out of a stick, a rock, and some duct tape, for example—are stored in the ItemTypes and ItemTypeContents tables. These tables are filled on an as-needed basis—when the player crafts a particular item, if it has not yet been added to the ItemTypes table, it is added at that moment as a new item type.

The icons used to represent item recipes as well as item roles and specific item types are all stored in the Icons table. These icons are downloaded by the client from the database any time

the client does not have the icon in its cache or when the icon in the database has a higher version number than the one the client has stored.

See Section 4.5.3 for more details on how items are added to the database.

4.4.3 World State

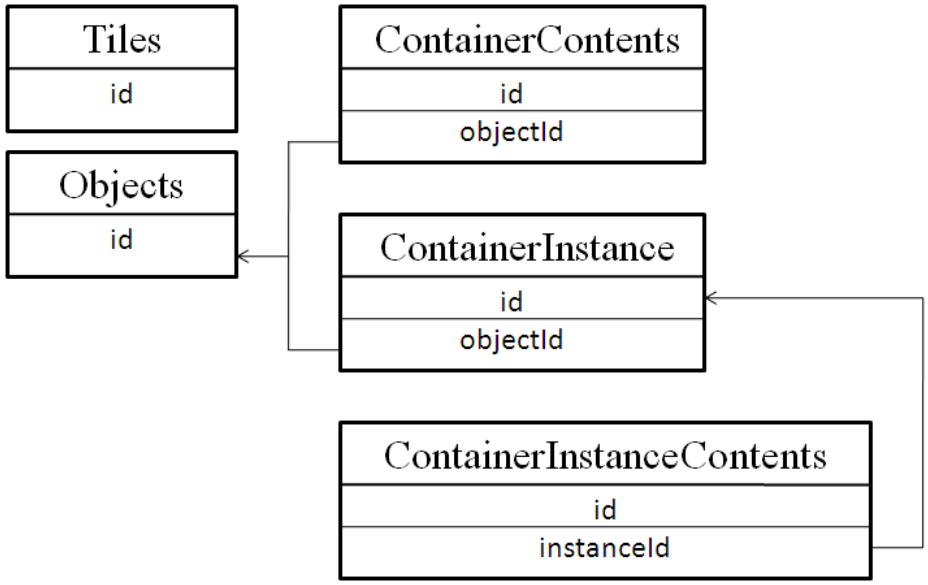


Figure 19: Entity relationship diagram for the world state-related tables

The Tiles table stores base tiles that have been created in the tile editor. Both the image data and tile type is stored which allows the server to easily search through the available tiles. See Section 4.5.1 for more details on the creation of tiles.

The static objects in the world are drawn from the information stored in the Objects table. This table contains Unity assetbundle files, which are essentially collections of meshes and scripts that make up a single object in the world. These assetbundle files are downloaded by the client whenever it encounters an object that it has not already cached, or when the cached version is older than the version stored in the database. See Section 4.5.2 for more details on the creation of objects.

If an assetbundle contains a Container script, it is marked as a container in the database. Containers are special objects that can hold items in them for players to pick up. The types of objects that a given type of container can hold are stored in the ContainerContents table. When a player tries to open a container in the world, the state server checks to see if it has an entry in the ContainerInstance table. If not, it creates one and generates the items that are stored in it, placing

them in the ContainerInstanceContents table. These two tables are kept up-to-date to keep track of exactly which items are in each container in the world as players remove items from them. See Section 4.5.3 ItemForge for more details on how containers are defined.

4.4.4 Users

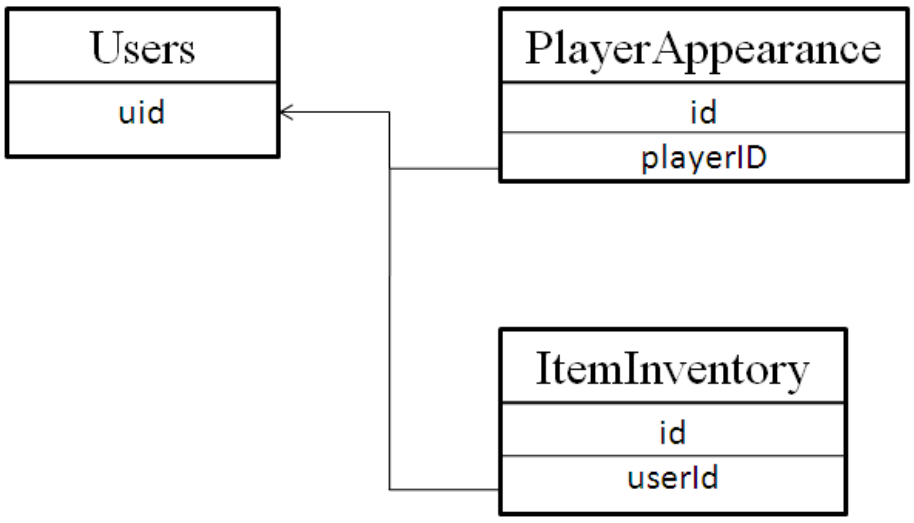


Figure 20: Entity relationship diagram for the user-related tables

There are three tables which, together, define the state of a single player of *Broken World*. The Users table is used to define the player's login information, including their username, password, current session ID, and last known IP address. It also stores the player's current tile and X,Y location within that tile. The ItemInventory table stores a list of all items that a player has in his or her inventory. The PlayerAppearance table contains information regarding the player model that gets loaded to represent the player's avatar. The PlayerAppearance table keeps track of what players look like. Information stored there is the obejctId of their hat if they have one, their base mesh ID (male or female), and the mesh of their shoes. This info is requested by the client as needed from the server. It is important to note that while all of these appearance-related features have been implemented and tested in the game framework itself, some of them—namely the shoes and base mesh—lack associated complete art assets and therefore have been omitted from the final build of the game.

4.5 Development Tools

Because of the large amount of game content needed in *Broken World*, it is essential that new game content is easy to add and change. Landscape tiles, static objects, and items are all important pieces of art (with associated gameplay logic) that need to be updated regularly in order for the game to stay fresh. The process of creating new content needs to be easy for a designer. As such, several tools were created for use with *Broken World* to enable designers to add and change game content quickly.

4.5.1 Tile Editor

To create a visually interesting world it was decided that the individual tiles should be hand crafted to guarantee meaningful tiles for the player to explore. To accommodate this customization a toolset was created for use by level designers. The editor consists of three separate modules which all work together. The modules each correspond to a particular image map of the tile: the object module, texture module, and height module. Standard across each module is the ability to save and load the file, save and apply an undo/redo history, and display a selection overlay to show the user where the current tool will affect the tile.

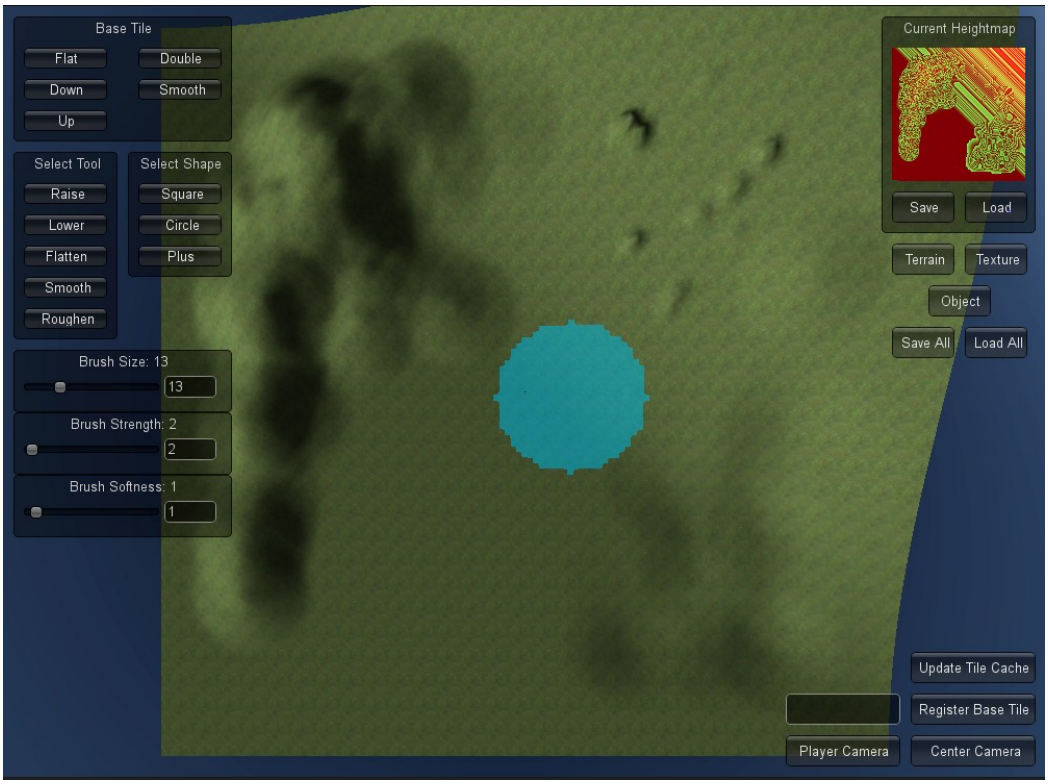


Figure 21: The terrain editor height module

The height module allows the user to change the topology of the tile. As seen in Figure 21 the user is presented with four main areas of control: tile type, tool type, tool shape, and tool info. By selecting a tile type the terrain and resulting heightmap are reset to a base state. These base states guarantee that the creation of a new tile of a particular type will result in sides which line up seamlessly when the server places the tile in the world. The height module then allows for modification of every vertex except edge vertices.

The tools available to the user are raise, lower, flatten, roughen, and smooth. As the names imply raise and lower add to or subtract from the height at a particular point, respectively. Flatten allows a level designer to set a height and then moves all affected vertices to the specified height. The roughen tool was never implemented and has no effect on the tile. The smooth tool works by averaging the heights of the affected vertices to remove the rough edges from the map.

The tool shapes define which vertices get affected by the currently selected tool. The available shapes are a square, circle, and plus sign. The size parameter relates to the square as being the length of each side, the circle as being a radius, and the plus sign as being the length of the plus. The different shapes, along with the tool types, allow the user to add interesting features to the landscape.

The tile information is fairly consistent among the different tools, allowing the user to specify the size and strength of the brush. The name above each slider tells the user the effect of the slider and changes appropriately depending upon the selected tool.

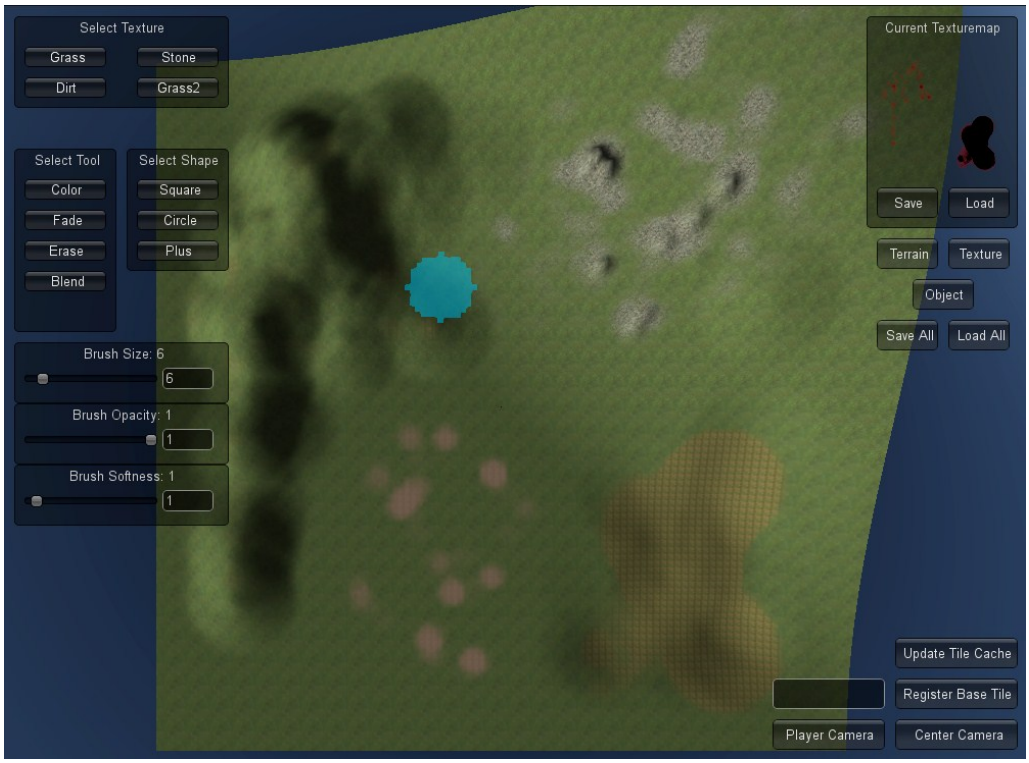


Figure 22: The terrain editor texture module

The texture module, as seen in Figure 22, allows the user to paint the terrain with one of four textures; grass, stone, dirt, and dirty grass. While painting on the terrain, the user can control the opacity and shape of the brush, allowing for both broad and subtle detail. Adjusting the brush opacity and softness defines the blending and responsiveness of a brush.

Figure 23: The terrain editor object module

The object module allows the placement and manipulation of objects on the terrain. The module starts off with a default set of five environmental assets, trees and rocks, with a sixth option to load any asset bundle from the object cache. Asset bundles, created in Unity in a separate part of the pipeline (see Section 4.5.2) can contain models, scripts, particle effects, and animations. The object editor treats each object indiscriminately, allowing level designers to place them in the world as long as they have been registered with the server.

The tools available to users of the object module are add, remove, select, and variation. The add and variation brushes both add objects to the world. The add brush will place objects on every pixel of the brush shape with the same rotation as set by the sliders. The variation brush will randomly vary the rotation of placed objects by an amount specified by the user. The variation brush also allows the user to space objects out so they are not all placed adjacent to each other. The remove brush simply removes any objects where the user clicks. The select brush allows the user to select one or more objects and edit their rotation in the world post-placement. When objects are selected a selection ring or rectangle appears at their base based upon their associated physics collider.

4.5.2 Object Registration

The static objects that fill the landscape of *Broken World* must be registered in the database so that clients can download them from the server if they do not yet have them cached. The process of adding an object to the database is straightforward for the designer, but actually involves several steps.

The first step in the process is for the designer to create the object in Unity. This is done using processes native to the Unity game engine itself: the designer creates a new prefab and places whatever meshes, scripts, or other Unity objects he or she wants on it. When the designer is finished with the object, he or she selects it and goes to the "GameObject" menu, and then selects "Register With Server". This menu option was added to Unity through the use of a custom C# script.

The designer then enters the name under which the object is to be registered into a pop-up window and clicks the "Register" button. This begins the process of converting the prefab into an assetbundle file. That file is then posted to a PHP page on the server, which adds it to the database. Once this process is complete, the designer can add the new object to tiles, and any clients encountering the new object will download it from the database. If an object is registered with the same name as a previous object, the old one is replaced with the new one, its version number is incremented, and any clients encountering the object will re-download it from the server.

4.5.3 Item Forge

The Item Forge is a web-based tool that designers can use to create and manage item recipes and containers in *Broken World*. The front-end is built in HTML, CSS, and Javascript, using the JQuery Javascript library[40] to speed up the process of development. The back-end is built in PHP. The Item Forge is hosted on the same server as the MySQL database, but there is no reason it has to be: the Item Forge can access the MySQL database on any machine by simply setting its IP address in a configuration file.

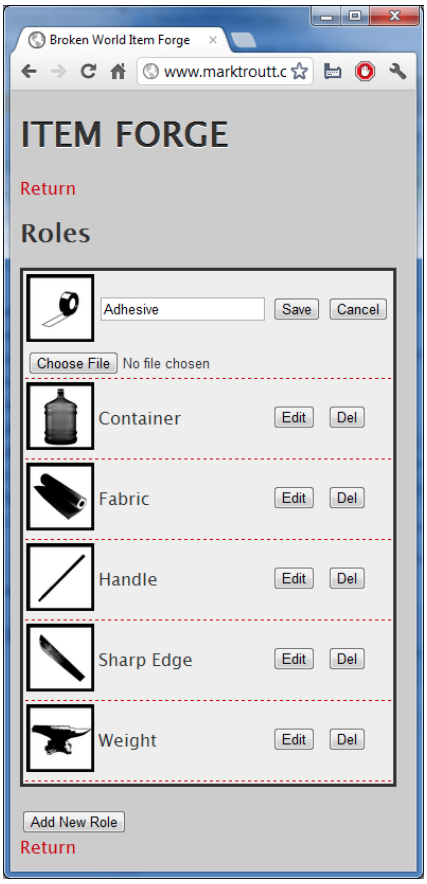


Figure 24: Editing roles in the Item Forge

The Item Forge allows designers to define new roles for items to fulfill, as seen in Figure 24. Whenever new roles are added to the database, the designer is also asked to upload an image to be used as an icon to represent that role. Whenever a client starts the game, the list of roles is checked and any new icons are downloaded and cached.

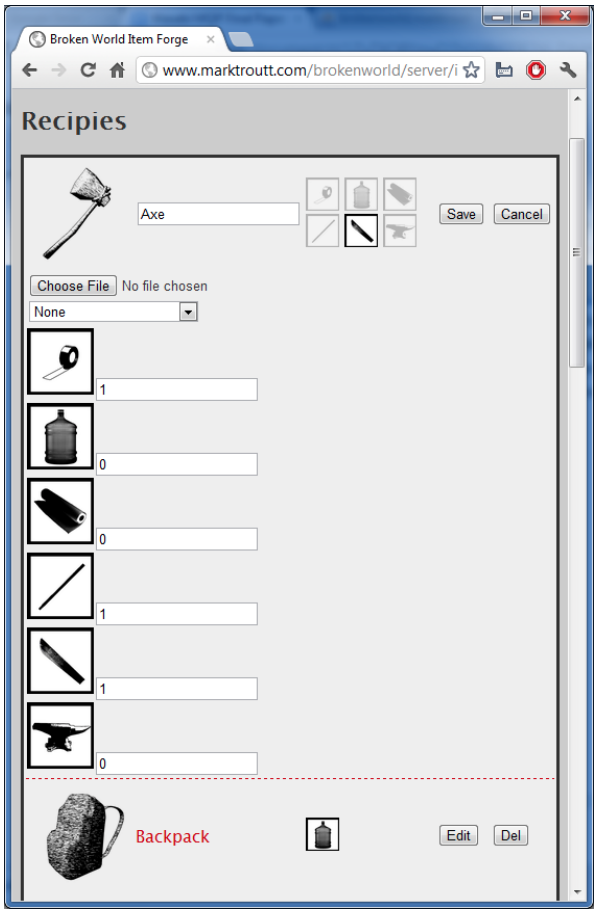


Figure 25: Editing recipes in the Item Forge

The Item Forge also allows designers to define new recipes for crafting items. A recipe consists of an icon, a name, a set of roles that the recipe can fulfill if used to craft another item (see the small squares at the top of Figure 25) and a set of roles that are needed to craft the item (the larger squares along the side of Figure 25.) Some items, called “base items”, cannot be crafted and so they have a value of 0 for every role. Base items are eligible for random generation in incompletely-specified containers (see below).

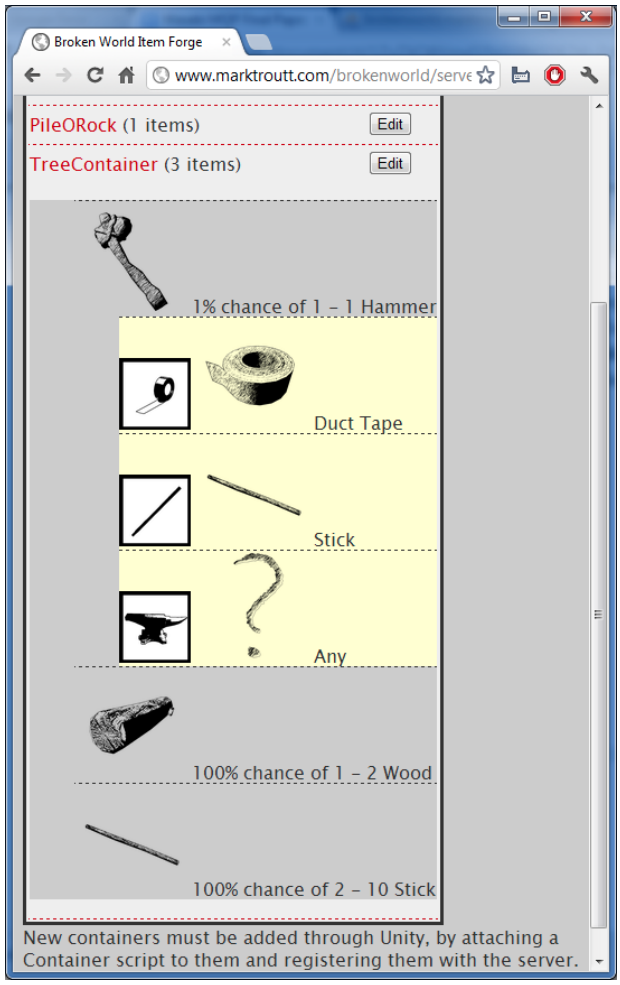


Figure 26: Viewing containers in the Item Forge

Objects registered with the server (see Section 4.5.2 Object Registration for details) with a Container script attached to them are marked in the database as containers, which causes them to appear in the Item Forge. Containers are special objects which provide items to players who interact with them. The exact contents of any given container are determined at runtime the first time a player checks what they contain. The contents are constrained by the details entered by the designer in Item Forge, as shown in Figure 26.

The designer is able to specify the different types of objects that might appear in a container, selecting them from the list of recipes registered in the database. Recipes which contain other items give the designer even deeper control over what might appear. In the example shown in Figure 26, the designer has specified that all of the hammers found in this particular container are to be constructed of duct tape, a stick, and any item that can fulfill the weight role. To prevent infinite recursion, roles marked as being fulfilled by "any" object will only select from the list of base items.

In addition to the types of items extant in a container, designers can specify the probability with which each item will occur, and, if it does occur, how many items are to be generated. The quantity of items generated is listed as a range, giving an equal chance of any number in the provided range being placed in the container.

5 Art

A large portion of a game is the visuals. Having something to view while playing aids in the player's immersion. An important part of the design process is making sure that the look and feel of the art assets is consistent. The art content for *Broken World* can be split into three categories: environment, buildings and characters. Each one of these requires planning to make sure they complement each other. The following sections describe the art style used in *Broken World*, as well as the technical constraints and methods used to create the game's visuals.

5.1 Style & Design

Broken World takes place many years after human influence is largely removed from the world, so having a game that feels deserted is important. Much of what is around from the past has degraded into nothing, and what remains looks old and abandoned. The initial look includes open fields, dense forests, mountains, and abandoned towns and cities. The wide open fields are used mostly to put distance between objects, but also to provide ample ground for starting communities. There are many objects such as decrepit buildings and old boxes scattered about for the player to rummage through for resources. The forests provide much of the natural resources the players need, like wood and stone. Mountains are mostly just environmental obstacles. They provide resources, but not nearly the same amount as a forest or a town. Abandoned towns and cities are where most of the salvaging takes place.

The players spend most of their time gathering resources to build up a community out of buildings. These buildings are not in pristine condition; they are rough and thrown together. Buildings are made from scraps of sheet metal, stones that were just picked off the ground and wood scavenged from telephone poles. As time goes on and the players obtain better means of collecting resources, the quality of the buildings goes up. A log cabin is much better looking than an old wooden shed and a brick house is better than one made of sod.

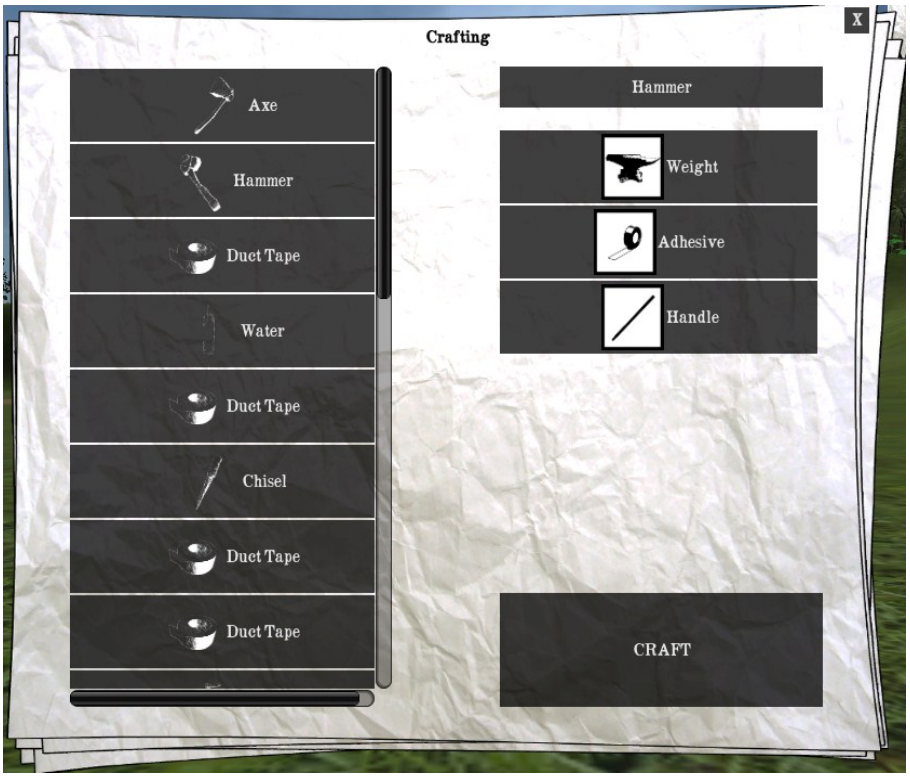


Figure 27: GUI elements in Broken World

The GUI for *Broken World* is kept to a minimum, but it also has to fit with the feel of the game. The game menus are the only interface elements, so it is important that they be visually interesting as well as functional. In many post-apocalyptic settings, floating papers are a common symbol representing a place that used to be inhabited, but has been abandoned. Newspapers contain plenty of information that can be arranged in many different ways. Newspapers also contain a mix of text and photographs, making them ideal for menu interfaces. All of the graphics, such as text and pictures, are in black and white. Having crumpled paper creases and a yellowish tint gives the newspaper an old look.

Figure 28: The login screen

As the first interface element that players will encounter in the game, the login screen is designed to look like the front page of a newspaper. It contains a black and white rotating image of the world, introducing players to the idea of exploring a massive world even before they begin playing. Two articles are visible, their titles giving the player a little information about the story of the game.

5.2 Technical Constraints

All of the objects are essentially placed on a point in a grid, with only one object per point. While the grid is very large, this does prevent a certain level of detailing. Cities and towns are much harder to create because there is less control when placing buildings, while forests end up looking very grid-like and the forest floor loses the undergrowth one would usually find underneath a tree. That is the tradeoff of the tile editor. While it makes that part harder, it is very easy to create a tile with objects scattered around the world. Importing the objects into the editor is also easy. It is the same as bringing any object into a Unity game, but with the extra step of uploading the object, textures and item information included, to the game's central server. The

object can be re-uploaded at a later time, and any instance of that object on any tile would be changed as well.

5.3 Implementation

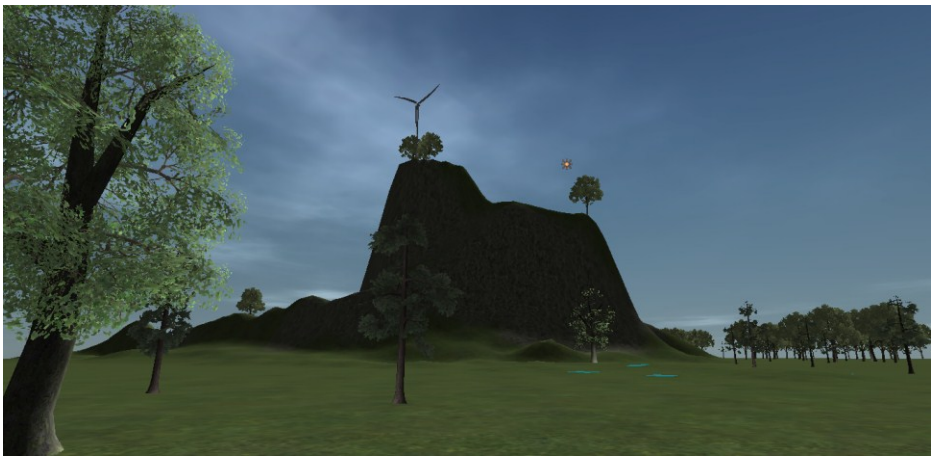


Figure 29: The environment populated with assets

The landscape is made up of tiles, each of which is created using the custom tile editor. Each tile is created in three passes. The first pass is terrain elevation. Landscapes usually have some amount of topography and are not completely flat. The terrain can be molded into different elevations, making it easy to create mountains or valleys. The second pass in the tile editor is the texturing. In the current incarnation of the tile editor, there are only four different textures available for use. This does not allow for a lot of variation in detailing the landscape, but it does provide just enough to define major features (a rocky cliff or a grassy meadow). Elevation and textures are applied to the terrain much like Unity's terrain editor. The third pass is the object placement phase. One asset can be placed on each point. Randomization brushes were used to create variation in the forests.

Figure 30: Environment Assets

All of the environment assets are made to look either old, ruined or overgrown. The two most abundant items in the world are trees and rocks. These were taken from Unity's default environment assets, to save development time. Trees can be painted on the landscape to make large forests. Some of the trees are rotated to look like they have fallen. The other objects that litter the world are man-made structures that have fallen into disrepair, as seen in Figure 30. There are ruined concrete buildings as well as other structures like wind turbines and transmission towers. The objective was to keep the polygon count in the models low. The bases of the buildings have to be detailed enough since the players can walk right up next to them, while the tops do not need as much detail. Other, smaller man-made items, like stop signs, telephone poles and oil cans are placed around the world, giving the appearance that there was once a civilization there. Some buildings have accompanying mailboxes. The box is made of plastic, and the flag signifying that there is mail is a ruler. They sit on a bent post. Many of these objects can be searched or harvested to obtain resources that the player can use to build and upgrade.

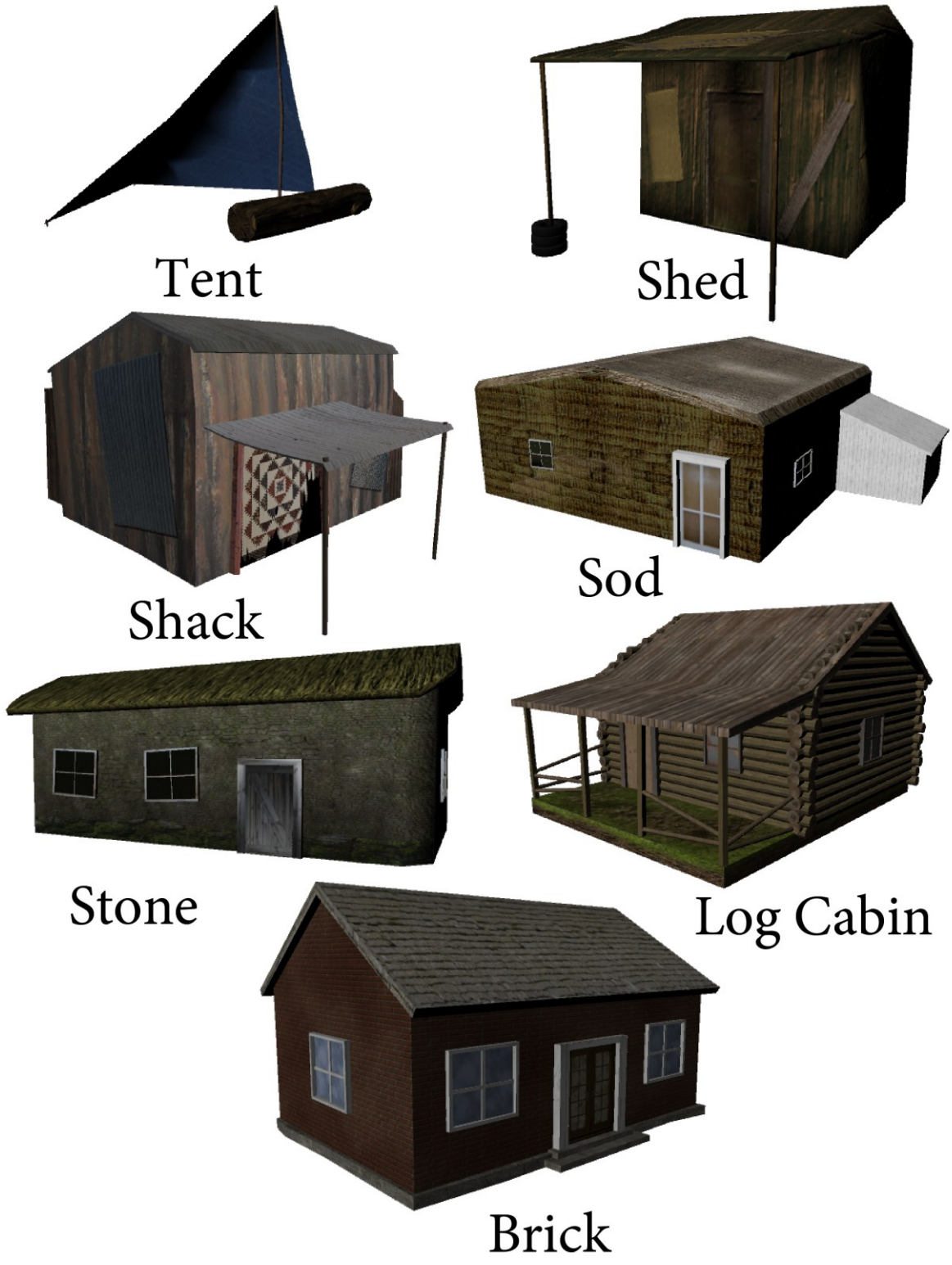


Figure 31: The different dwellings that can be built

The player-built buildings are the centerpiece models. These buildings are split into two different categories: houses and community buildings. There are seven different houses that the players can create. The first house is the tent. It is made from a tarp and some sticks, with a stone fire-pit in front. This is the first dwelling that the player owns. The second tier of housing has two different buildings; the shed and the shack. The shed is a house made out of scrap wood. The resource used to upgrade to the shed is primarily wood, with some stone and scrap metal. The shack is made up of scrap metal sheets, so the resources used for upgrading are mostly metal pieces. There is a quilt as the front door for aesthetics. The third tier of housing has three types of houses. These houses are bigger than the tier two buildings. The first type of tier-three house is the log cabin. The log cabin is mainly made out of wood. The cabin has a traditional New England log cabin style. It has a fairly thrown-together style, seeing as the characters are not very experienced in house building. Another tier-three house is the sod house. A sod house is made up of mud bricks and some scavenged wood for the doorways and windows. There is also usually some sort of storage shed sticking off the side. The final tier-three building is the stone house. It consists of mossy stone walls with old wood for the doorways and windows (much like the sod house), while the roof is thatched. The ultimate housing level is the brick house. It is a well put together dwelling that offers plenty of space, but also requires the most resources.

Figure 32: The different community buildings that can be built

Community buildings are all made to look pieced together from many different parts, since early designs of *Broken World* allowed many different players to work on the same building. There are five types of community buildings. The workshop was originally meant to be used as a marketplace, and as such is modeled after a vendor stall. The stalls are constructed using scavenged wood, metal poles and a tarp covering. The contents of the stalls are tools that the player can use (a texture on a single polygonal face). The well is the main source of water for players. It is made up of unevenly cut stone, two branches and a hockey stick. The community board is constructed out of plywood and two square posts. There are pieces of paper stuck to the sides. The barn, originally intended for players to use as a warehouse (a feature cut due to time constraints) is modeled after a traditional barn with red wood paneling and two large doors. The forge is made up of rusty metal pipes.

All of these assets are modeled using Autodesk Maya[41] and are designed to have a low polygon count. Rendering too many polygons will lower the game's framerate, so the lower the number of polygons the better. Textures for each model are mapped in Maya so that each asset only needs one image file. While having multiple images would have increased the detail, one image is many times smaller and thus saves valuable storage space. Textures are comprised mostly of images and photographs put together in Adobe Photoshop[42] in such a way that they are both seamless and convincing. To give all these low poly objects their detail, the textures are brought into CrazyBump[43] to generate normal maps. Normal maps give the appearance of detail by calculating shadows based on the light source. CrazyBump takes images and predicts where the details are based on the lights and darks of the image, then creates a normal map based on those predictions.

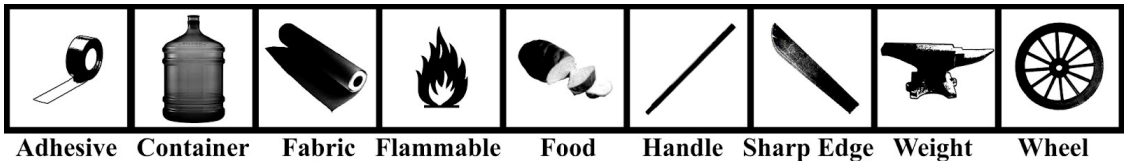


Figure 33: Material icons, representing the different roles an item might play in item crafting

Figure 34: Item icons, representing the items a player might find or craft in Broken World

Many of the items that the player collects throughout the game are not physical objects (the tile editor does not make placing small objects optimal) but rather part of a list that players can see in their inventory. Though functional, it would be rather boring to only see a player's accomplishments as text. It is better to have some form of visual indicator. Not only is it better looking, but it also helps the player to identify what the item is. There are two types of icons in the inventory menu. There are material icons and there are item icons. Each item has a material type, and the icon helps signify this. The material icons are solid black pictures that resemble the material (such as a stick for the handle material or a lawnmower blade for the sharp edge material). Item icons are seen when an item is selected in the inventory. These icons are representations of the items the player picks up or crafts. Each of these pictures are first modeled in Maya and textured if necessary (most of the models would not be seen in game, so texturing is generally not needed). After the model is rendered out, it is brought into Photoshop and the graphic pen filter is applied to it. This makes the item look like a drawing rather than a

photograph (Figure 34). The icon serves no other purpose than to give the player something to look at.

Figure 35: Male character model

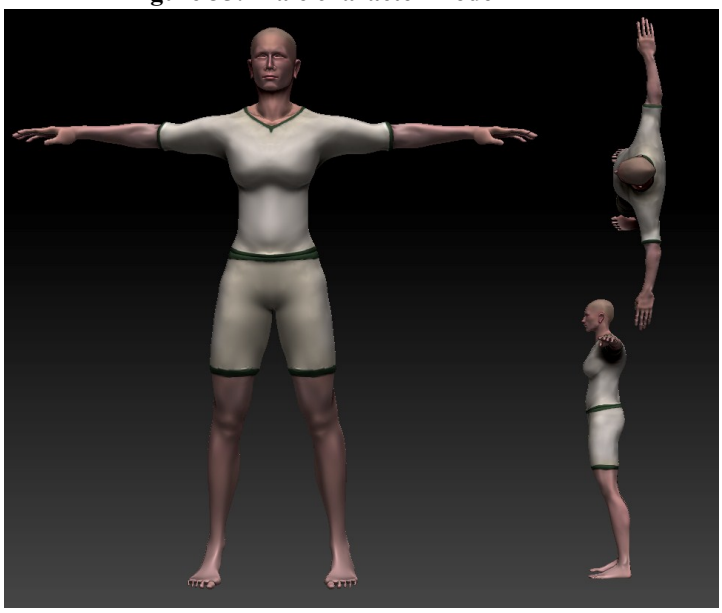


Figure 36: Female character model

The character models were created using Pixologic's ZBrush[44]. A free human model with a low polygon count was downloaded to use as a base in order to provide an accurate representation of a human body. The model was modified into two different characters; a male (Figure 35) and a female (Figure 36) both with their respective anatomical features. Using the UVMaster plugin in ZBrush,[45] the textures and normal maps were applied to the lowest polygon count version of the model. This low polygon count model was then brought into Maya where it could be rigged. Using the ERig tool[46] to create an easy to use skeleton and controllers, the

character could then be animated. Idle, walk, run, crouch, and use animations were all created, although only the idle and run animations made it into the final build of *Broken World* due to time constraints. In the final weeks of the project, the shared skeleton used to animate both models was changed to handle more complex animations. Unfortunately, there were problems with the female model using the new skeleton that were never resolved, and as such she does not appear in the final build of the game.

Figure 37: Early character concepts envisioned motley outfits

Originally, clothing differences were going to be a significant feature of *Broken World*. Early concepts of characters had them wearing a hodgepodge of various types of clothing, as seen in Figure 37. A second artist was brought on the team to complete various types of clothing for *Broken World*, but he never produced any results and eventually dropped off the team. As such, the method of creating clothing had to be quick and easy to make up for lost time.

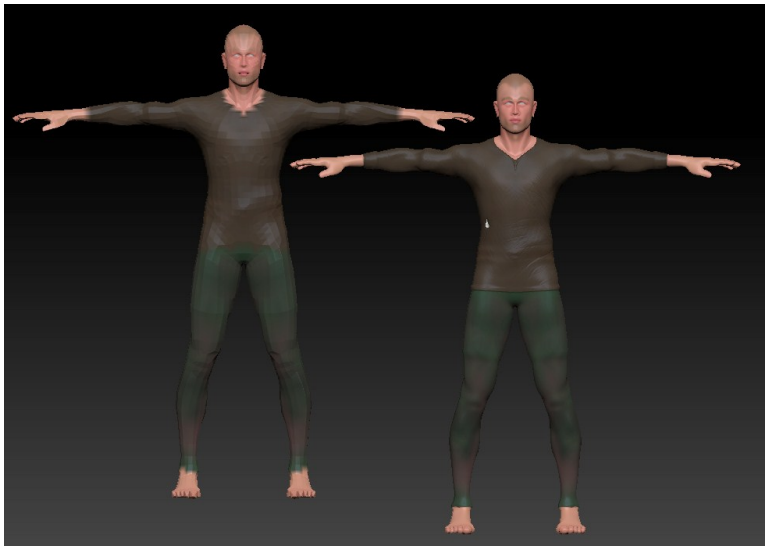


Figure 38: Character with clothing

The way that the clothing was created underwent a few iterations (clothes as separate objects that are bound to the skeleton, clothes that are dynamic cloth that are calculated based on animation) but the final technique—chosen because of its rapid results—was to make the clothes a texture file with a normal map, giving the appearance of clothing while still only using one model. Details on the clothes were sculpted and painted in ZBrush then exported as a normal map onto the lowest subdivision. This method proved to be a quick and easy way to make decent clothing as long as the clothes are skin-tight. It could not be used for larger clothes like jackets or hooded sweatshirts.

Figure 39: Hats that can be used as a method of player customization

The primary way players can make their avatars look more unique is hats. Hats are simple, low-poly objects that can be parented to the head, and are very easy to create. Three hats were created, as seen in Figure 39: A rusty bucket, a Wasabi Studios baseball cap, and a top hat.

All of the art comes together to create a world that can be pieced together to make the expansive landscape. Using their characters, players are able to explore the various environmental features and gather resources from a multitude of assets. The realistic nature of

the environments, characters and UI make *Broken World* immersive and a good playground for discovering new places.

6 Evaluation

After the completion of *Broken World*, several tests had to be performed to determine its effectiveness along several different vectors. *Broken World* was tested for technical robustness, efficiency, and functionality, as well as gameplay. Section 6.1 describes the different tests that were run, what they are trying to test, and the ways in which the data was collected. Section 6.2 analyzes the results of these tests.

6.1 Testing Methodology

There were two distinct types of data that needed to be collected on *Broken World*. The first type of data is detailed in Section 6.1.1, which describes tests that were run to determine the effectiveness of *Broken World's* programming. Section 6.1.2 describes the type of data that was collected on the gameplay, art, and theme of *Broken World*.

6.1.1 Technical Testing

Because a significant portion of *Broken World's* programming is dedicated to network-related functions, significant effort had to be put into evaluating its network efficiency. It was important to measure both the strain created on the server with having a large number of clients as well as the strain on the clients when many clients are connected together. It would be intractable to try to run hundreds of clients on multiple machines. Instead, a network-testing tool was developed to simulate multiple logged-in clients. The tool automatically logs in a specifiable number of users (creating accounts for them if an account does not yet exist) using the same network protocols as a regular client. Once logged in, the tool contacts the state server and downloads the network tiles and requests a list of neighbors for each client. Each client then runs around at a constant speed, changing its rotation vector every frame as it moves. Clients occasionally move far enough to switch tiles. Each headless client has approximately a five percent chance of switching tiles in any given minute. When this occurs, the client notifies the server as such and downloads new tiles as appropriate. The end result, on a network level, looks identical to having hundreds of real clients logged in, except that all of the clients originate from the same IP address.

Operating System	Windows 7 64-bit	Macintosh OS 10.6.8
CPU	Intel Core 2 Duo 2.53GHz	Intel Core i7 2.3 GHz
RAM	6.0 GB DDR3	8 GB DDR3
Graphics Card	NVIDIA GeForce GTX 260M	AMD Radeon HD 6750M

Figure 40: The specifications of the machines used to test the Broken World client

Wireshark, a packet capturing and analysis tool,[47] was used to collect data on the packets being sent between different parts of *Broken World*. Wireshark is able to record all incoming and outgoing traffic on a specified network interface. Using a capture filter, only packets relating to *Broken World* were recorded, reducing noise in the data. Packets were captured with varying numbers of simulated clients (0, 1, 60, and 100) and one actual client. Tests were run while the client performed several different actions: logging in, loading a tile, crafting an item, placing a building, and "normal play", defined as a player logging in, retrieving an item from a container, crafting an item, and placing a building. Two different machines were used to test the actual client; their specifications are listed in Figure 40. Once collected, the data was split into upload and download for both TCP and UDP traffic. This allowed analysis of where the bottlenecks were and analyze what benefits, if any, *Broken World's* network model has on the bandwidth usage of all parties.

Operating System	Ubuntu 11.10
CPU	2.2 GHz Athlon 2800+
RAM	512 MB DDR2
Location	Remote Access: Waltham, MA

Figure 41: The specifications of the machine used to host the Broken World server during testing

In addition to the network bandwidth, the machine resources used by the server was an important metric. A big goal for the project was that the server could handle hundreds of clients on a relatively weak machine. Several tests were performed to ascertain server efficiency. Using tools built into Ubuntu, the operating system of our server, the CPU and memory usage over time were measured with 101 connected clients (100 headless and 1 normal).

The limitations of the client also needed to be tested. Every object map for specific tiles can define the locations of up to 16,384 objects. Bench tests were performed using the tile editor to see just how many objects could be on a tile before the framerate dropped below 60 and 30 frames per second. Using a combination of Unity's profiler and PerfMon,[48] the framerate, CPU usage, and memory usage of the client were measured throughout several tests. These tests were also conducted during "normal play", as defined above.

The final test run was a subjective one: two clients were connected, with both machines sitting right next to one another. One player moved around the world, and the time it took to propagate to the other client was estimated visually. The same test was then repeated with chat messages.

6.1.2 Gameplay Testing

To test the gameplay, eight testing participants were asked to play *Broken World* for 10-15 minutes and then fill out a short survey about their experience. Before the playthrough each participant was presented with a word document explaining the controls for the game. After playing for the allotted time users were asked survey questions about what they did or did not like about the art style and the game overall. They were also asked to report any bugs they found. See Appendix 10.5 for the exact text of the survey. This test helped to determine if *Broken World* succeeded in its gameplay goals. The reported bugs were also a good indicator as to the severity and frequency of noticeable programming flaws.

6.2 Data Analysis

This section contains analyses of the data collected through the tests described in Section 6.1. These analyses contain both quantitative and qualitative data, and will be analyzed as such. The data collected on *Broken World* fits into two categories: technical data, and gameplay data. Section 6.2.1 details the findings of the former, while Section 6.2.2 details the findings of the latter.

6.2.1 Technical Analysis

One of the design challenges of *Broken World* was to build an MMO server that could be run on a low-end machine. One of the ways this was accomplished was to separate account maintenance actions from gameplay actions by having separate login and state servers. The tests that were run on the *Broken World* server, however, were not able to tax it enough for this feature to be proven necessary.

The most taxing test run was with 100 simulated clients connected. During this test, the maximum CPU usage reached was 4% of a 2.2 GHz single core Athlon processor. The maximum memory usage was 12.3MB. The average bandwidth usage over two runs was 37 kb/sec with a peak usage of 87 kb/sec. The server uploaded, on average, 16 times more data than it downloaded. See Appendix 10.4 for the full data set. The remote host responsible for spawning the headless clients ended up failing long before the server hit its limits. Usually a thorough test of server load would require the use of several machines running the testing tool. Given the resources available to this project, however, such a rigorous test was not feasible. The machine used for testing the server was itself a low-end machine, and it was nowhere near being taxed. In fact, it would be feasible to run the *Broken World* server on a netbook without fear of it reaching its limits very quickly. This is largely due to *Broken World*'s unique peer-to-peer networking model (see Section 4.2.2 for details) which reduces the amount of communication between clients and the server to a bare minimum.

FBX	PNG	OPTIMIZED PNG
211 kb	610 bytes	97 bytes
234 kb	3.37 kb	613 bytes
368 kb	1.65 kb	689 bytes
368 kb	2.79 kb	2.867 kb
300 kb	1.64 kb	686 bytes

Figure 42: Size comparison of five basic terrain tiles in FBX, PNG, and optimized PNG formats

Another important innovation in *Broken World* is the use of images—specifically PNGs—to store terrain data. Traditionally, 3D meshes are stored in a 3D mesh format. One

popular 3D mesh file format is the Autodesk FBX format, commonly used for its interoperability with many rendering engines.[49] As seen in Figure 42, over the five different kinds of base tiles, the PNG height maps output by *Broken World*'s tile editor ranged from 69 to 345 times smaller than the same tile in FBX format. If these PNG files are then run through the PNG optimizer[50] (see Section 2.3 for more details about PNG optimization) the average optimized PNG file ranged from 128 to 2175 times smaller than the FBX file. Not only is this a huge gain in the amount of data that can be stored on a drive, but it also makes it feasible for terrain tiles to be sent over the network to clients and for clients to store a large number of terrain tiles on their machines in a cache.

On the client side, during normal play CPU usage averaged 28% on a dual-core 2.53 GHz Intel processor, with a peak usage of 67%. The client used an average of 525 MB of memory with a peak usage of 612MB. Although there was occasional lag, the framerate generally stayed above 60 frames per second, often soaring well above 200 frames per second. Interestingly enough, this means that it takes a stronger machine to run *Broken World*'s client than it does to run its server. This is largely due to the overhead of having to render the entire world in addition to all of its networking responsibilities. While this behavior is expected in the average networked video game, it is unusual for an MMO because MMOs typically require the use of server farms.[51]

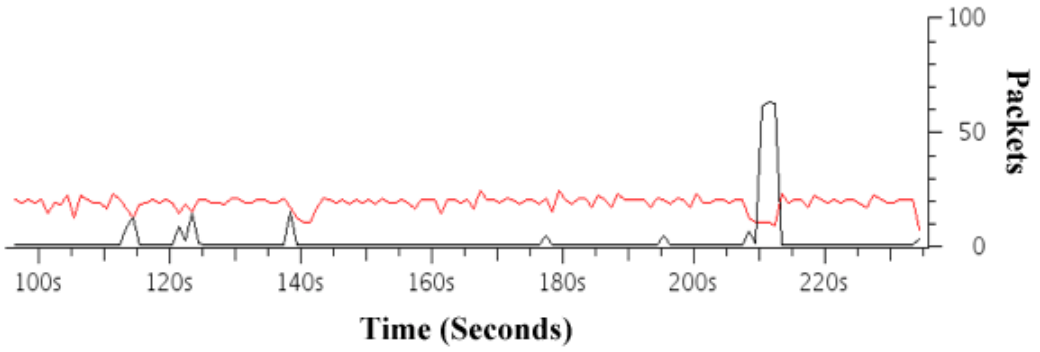


Figure 43: Packets sent over UDP (red) and TCP (black) with one other client during normal play

Network usage for the *Broken World* client consists of two parts: TCP and UDP packets (for more on TCP and UDP, see Section 4.2.2). As seen in in Figure 43, UDP packets are sent to other clients in a fairly steady stream. TCP packets, on the other hand, are sent in bursts to the server whenever the client performs an action that changes the world state. The highest-bandwidth TCP process that can occur is tile loading, during which the highest bandwidth recorded was 89 Kb/sec. Since the average DSL modem is capable of handling 128 Kb/sec, this

is well within reason.[52] It is important to note that visual inspection was unable to detect any noticeable network lag during testing; to the naked eye, player actions seemed to propagate instantly on other clients.

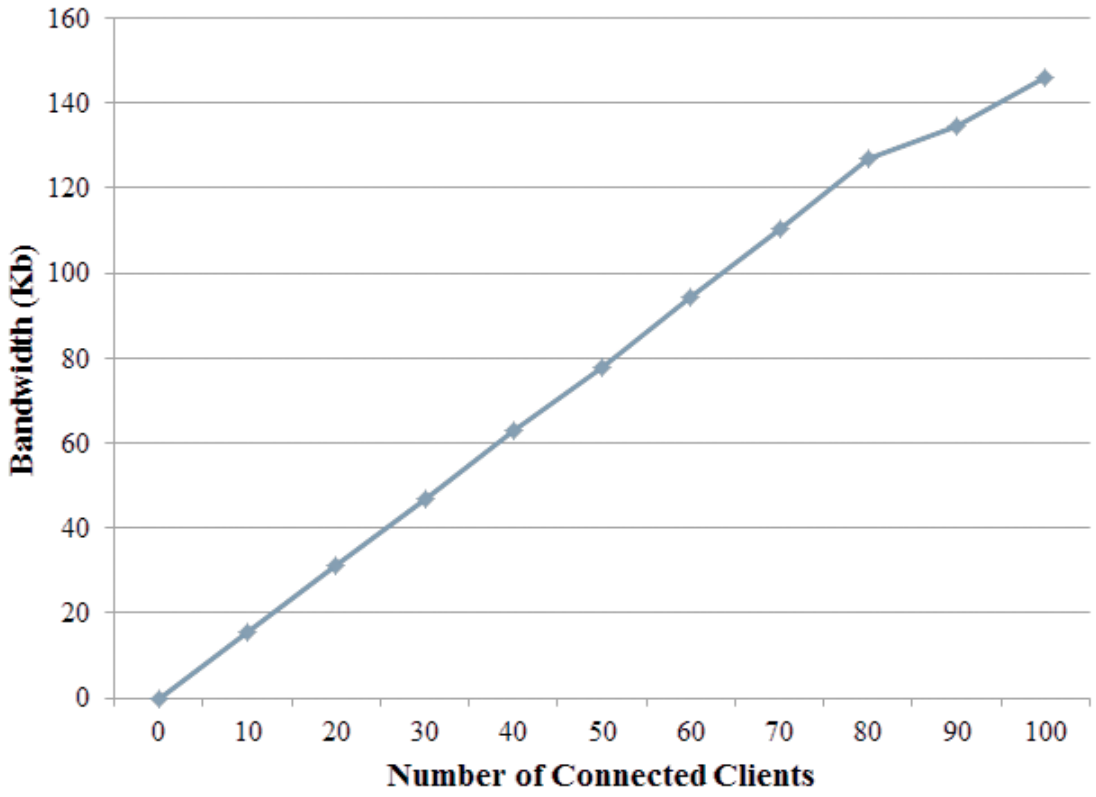


Figure 44: Combined up/down average bandwidth from a client rises linearly with the number of clients in neighboring tiles

As seen in Figure 44, the amount of UDP traffic experienced by the client scales linearly with the number of players logged in on tiles adjacent to the player's current tile. In the unlikely event that there are 100 clients in the same general geographic location, the client's bandwidth climbs to approximately 150 Kb/sec, which is a little high for the average DSL modem to handle. Looking at solely UDP traffic, the average DSL should be able to handle between 70 and 80 connected clients before beginning to lag. If the tile-loading TCP spike from above is also accounted for, the number drops to closer to 20 to 30 clients, which is also closer to the number of anticipated users on any given cluster of adjacent tiles. As a point of comparison, popular MMO *World of Warcraft* uses between 50 and 80 Kb/sec on average during a 25 player raid,[53] which is more than *Broken World* uses on average for the same number of players. As such,

despite its peer-to-peer networking model, the *Broken World* client still uses comparable amounts of bandwidth to other modern MMOs.

6.2.2 Gameplay Analysis

The user study surveys provided great feedback about the current gameplay and where the development should head in the near future if development were to proceed. The first question asked of users was about goals and if they developed one since the game provides no real goals of its own. Several players quickly formed a goal of making a hammer when one of the first buildings they tried to interact with required one. One player tried to find the edge of the world and learned many interesting things about the terrain generation in the process. The biggest takeaway from this question was that players easily developed their own goals, so the open sandbox worked well.

When asked what players would have liked to know before playing, they all wished for either an explicit goal and/or a better warning of the unpolished state of the game. While it was noted previously that players did create their own goals eventually it seems that the experience would have been more enjoyable to many players if they had been given some initial direction. This need was seen by the developers while building the game, as there is no guidance or tutorial while in-game, but was abandoned towards the end of production due to time constraints.

When asked about the art many players said they enjoyed the landscape and environment. They all felt the world seemed fairly large and visually appealing to explore. What players disliked most was the run animation of the main player avatar. This was due to a known bug caused by Unity not reading in the animation the same way it appeared in Maya. There was also a graphical bug where hats would go through the avatar's head a small amount while they were running. This was also an issue related to the way Unity read in the skin weighting of the mesh as the head was not moving with the head bone like it was meant to, which is what the hats were attached to. While player reactions to the avatar were primarily negative, overall impressions of the rest of the art were positive.

What was learned about the gameplay overall is that players saw great potential in the gameplay but felt that it lacked direction and could have benefitted from features we dropped during development. As seen in Section 7.3.2, the real benefit from this study was in seeing how gameplay could be improved given more development time.

In addition to comments on the gameplay, players found a few bugs in the test build. Most of these bugs involved the rendering of terrain tiles, and the vast majority have been fixed in the final version of *Broken World*.

7 Conclusions

MMOs are an incredibly popular genre of video game. The vast majority of MMOs, however, follow the same basic formula. Gameplay mechanics are uniformly combat-based, storylines are homogeneous, and the game worlds are of a finite scope. The underlying technologies adhere to similar network models.

Broken World, however, attempts to innovate along several different vectors. The gameplay revolves around exploration and resource gathering rather than combat, the storyline is player-driven rather than scripted, and the game world is of near-infinite size. In order to complete these objectives and create a functional MMO experience, certain goals had to be met:

- **Alternative Gameplay Style** - *Broken World* aims to provide a non-violent, non-competitive alternative to the current trend of violent MMOs.
- **Consistent Style** - The art style in *Broken World* should be immersive and cohesive.
- **Low-Poly Art** - The models used in *Broken World* must have a low polygon count.
- **Scalability** - The game should be able to handle 30 players at any given time, preferably more.
- **Lightweight Server** - The server needs to run on a weaker machine than conventional MMO servers.
- **Expansive World** - The game world should be infinite or sufficiently large enough to seem infinite to the players.
- **Persistent World** - The game world should persist across all players and all play sessions.
- **Development Tools** - *Broken World* should be accompanied with a set of development tools to allow new content to be added to the game.

The gameplay of *Broken World* is constructive rather than destructive. Players are tasked with rebuilding the world using resources that they scavenge. While crafting and gathering mechanics are common in other MMOs, they are typically a sub mechanic supporting a larger combat system. In *Broken World*, these mechanics are the core gameplay experience. During the

user study, players saw great potential in these mechanics, noting that they were interesting. The unpolished state of the game, however, prevented players from fully engaging with these mechanics. With more testing and polish, the gameplay would likely provide a more satisfying experience.

The art style of *Broken World* models an expansive rural landscape from a world in which humanity has lost its influence over the land. The user interfaces use imagery drawn from old, discarded newspapers to suggest a world that was once a civilization. The user study revealed that players found the world immersive, and that the art was evocative of an expansive landscape. Common complaints during the user study were related to the player's avatar, universally mentioning a problem with the main character's run animation. Models were also designed to have low polygon counts for efficiency reasons. Tests were run where models were placed on a tile until framerates dropped below acceptable levels. During these tests, several hundred models could be placed without any detrimental effects on performance.

Broken World's inclusion of peer-to-peer networking paradigms allowed the server to be run on a low-end machine. When stress-tested using 100 simulated clients, it came nowhere close to taxing the limits of the machine. The ability of clients to share information with one another allows them to share the burden of many of the responsibilities typically allocated to the server, allowing the server to support hundreds of clients on much weaker hardware than conventional MMO servers.

The custom terrain generation system used in *Broken World* creates the world space out of a series of seamless environment tiles. As players explore, new tiles are chosen by the server, allowing the world to extend in all directions, giving the appearance of an expansive landscape. During the user study, players did indeed describe the world as "expansive", and frequently listed it as something they liked about *Broken World*. Players can make changes to the environment by placing buildings and interacting with certain objects. These changes persist for all players.

To assist in the creation of *Broken World*, several development tools were created. A web-based tool called "Item Forge" was created to allow designers to add and modify items, as well as place items in the world. The world itself is built in the tile editor, which allows level designers to paint elevation, textures, and objects onto terrain tiles. These tiles are stored as a series of PNG files, saving space on the server, and allowing terrain data to be efficiently

transferred over the network. Both of these tools allow designers to add content to the game quickly and easily, and do so in a modular way such that the game content can be changed even while the game is in progress.

The following sections evaluate both the final version of *Broken World* and the methods by which it was developed. Section 7.1 places *Broken World* in the larger context of existing software and games, explaining the areas in which it innovates. Section 7.2 evaluates the methodology used in developing *Broken World*. Section 7.3 explains ways in which *Broken World* could be improved.

7.1 Impact

Broken World is unique both as a game concept and as a piece of software. *Broken World* contributes new ideas for potential gameplay themes through its focus on non-violent play. It also features several technological innovations that could easily be adapted for use in other software, especially other video games. Its software innovations are somewhat better implemented than its gameplay innovations, but it shows positive results in both arenas.

As seen in Section 6.2.2, users who played *Broken World* thought that the game concept was promising. They even suggested implementing some of the features that were originally planned for inclusion in *Broken World* but were cut due to limited development time (see Section 7.3.2 for details on these features). Despite the frequency of games based on competitive and violent game mechanics, there is opportunity for non-violent and non-competitive games like *Broken World*, even in settings like post-apocalyptic fiction in which violence is a common theme.

One of the technical features that *Broken World* does especially well is its storage of terrain tiles as PNG files, as described in Section 4.3. Because of the massive space savings over traditional mesh file formats (see Section 6.2.1) transfer of terrain data over the network is easy. Written almost entirely in the C# Mono framework, using Unity almost exclusively as a renderer, the entire terrain system of *Broken World* could be ported to another game with minimal effort. In addition, the idea of using PNGs to represent mesh data should be further explored to see if other meshes could be compressed using advanced image compression techniques.

Another novel technical feature of *Broken World* is its networking model, described in Section 4.2. By using a hybrid of the peer-to-peer networking model and the client-server

networking models, the *Broken World* server can be run on a very low resource machine, as explained in Section 6.2.1. This model could be adapted for use in other game servers to significantly reduce the cost of purchasing and maintaining machines.

7.2 Postmortem

In the software industry (and by extension, the video games industry) "postmortem" refers to a document created after the launch and release of the product that evaluates the development process. A form of risk management, the postmortem allows developers to isolate both what went right with the previous project and what went wrong, with the hopes of better streamlining and perfecting the process for future projects.[54] Postmortems can be useful tools not only for the team that participated in the software creation process, but also for other teams to look at to get an idea of what practices might help while also pointing out pitfalls to avoid. The following two sections form a postmortem for *Broken World*.

7.2.1 What Went Right

Throughout the project, the team maintained a timeline. This timeline was updated on a weekly basis in accordance with the amount of work accomplished in previous weeks. This produced a quick iteration cycle, and pushed the developers to show progress each week. Even when the weekly goals were not able to be met, the timeline made it easy to reassess the project's progress and plan for the future.

The programmers frequently met to do coding together. Even when tasks had been divided and did not necessarily need interaction, having both programmers in the same room helped to keep them focused on the project and made sure that there was always someone nearby to help out with any particularly obstinate bugs. Technical design challenges can be easier when there is someone else nearby to help talk through solutions.

Another helpful programming practice was the use of virtual machines to simulate the server environment up until the very end of the project. In the final weeks of the project, the server was moved to a remote machine, making updates a hassle, and introducing countless other difficulties outside of the realm of easy influence of the programmers. Because the server was run on an easily-controlled, stable virtual machine environment up until the last second, development on the *Broken World* server was able to progress smoothly. Had a remote machine

been used during development, the frequent network quirks combined with the lack of control over the machine would have hindered progress.

All of the core art assets that needed to be finished were completed to a polished state. There were many assets that would have added quite a bit to the game, but they were not part of the core or required category. All of the assets followed a similar theme and generally looked like they belonged in the same environment. Having complete game models that fit together is the major goal of the art section of the project.

7.2.2 What Went Wrong

Making an MMO requires a large tech backbone that everything else must be built upon. On the networking side not only does the server need to be coded in a robust way, the client must also be created in tandem to properly support and test all the networking and player interaction that is required. Early on in the project, too much time was spent refining gameplay details rather than beginning work on the technological backbone of the game. So much time was spent on planning documents (see Appendix 10.3) that by the time the coding was started there was already only enough time to get the core MMO framework completed, leaving no time for any of the gameplay features that had been meticulously planned. As none of the team members have a focus on pure game design as a profession—all members are seeking technical or artistic degrees—the original focus on gameplay placed undue emphasis on areas outside the team members' areas of expertise.

As a further attempt to refine the gameplay experience, once coding had begun much emphasis was placed on quickly completing a gameplay prototype which resulted in the rushed coding of features. This created issues later as features which appeared to be finished from the user's perspective had to be recoded to truly work as intended. The time consumed in reconnecting all these features to be synchronized across both client and server took away from much needed testing opportunities. Many of the problems found later on would have been much easier to fix if they had been coded properly in the first place.

The biggest mistake made on the client side was with the server-to-client communication. It would have been beneficial if all the network communication had been put into its own thread. This would avoid lag when loading tiles or objects. The reason separate threads was initially avoided was due to the limitations in the Unity engine, which does not provide support for

multithreading. A work-around was devised and later implemented to handle the peer-to-peer communication but by the time it had been discovered the server-to-client networking framework was already too fundamental to the code to be changed.

Besides the increase in testing the tech development would have benefitted greatly from closer collaboration with the artist. The most glaring example of this was in the development of the tile editor. During the early stages of the editor features were added in and arranged based upon the viewpoints of the coder. It was not until everyone was in the same room using the tool that the more important features were recognized, and bugs discovered, that were unanticipated during the development. More progress could have been made faster if the entire team had worked side-by-side. Both sections could be worked on independently, so there was no apparent need to meet very often during each week for work sessions.

For such a large artistic scope, having more than one artist would have been preferable. Having one artist work on environmental assets and another work on characters and animations would have produced better quality assets overall. Only having one artist meant that everything had to get done in some form (characters, levels and environment) so the quality suffered in favor of quantity. Specifically, more work should have gone into the character rig and animations, but they were low priority compared to the other assets (ones that affected gameplay).

Another benefit of having multiple artists is that they can keep each other in check. One can always make sure the other is working. In a sense, having the artists compete against each other is a good way to produce quality work. When there is only one artist he has to self-manage, which can be unreliable. Though time management documents were created (such as a color coded asset list) they should have been in place closer to the beginning of the project and were only effective for the second half of the project. The team attempted to contract an outside artist to help with character clothing, but that artist produced no results before dropping out of the project entirely. Since the initial design was developed with the intention of having someone else doing the character art assets, none of it happened until the very end of the project, and it was done in a rushed fashion.

While the scope of the game required a more unique approach to environment creation and the terrain editor made it very easy to make many different terrain tiles, it is also limiting. The primary problem is that all assets must be confined to a grid, and it does not allow for as

much control as programs like Unity or Maya. The tiles lacked a level of detail that would have made the environment much more interesting. In particular, forests have much more than trees; they also have underbrush that lines the floor. Due to the limitations of the editor only one object can be placed per point. The way terrain is generated also does not allow shadows with Unity's default lighting. Shadows would have made everything look much nicer. Unity's lighting also affects geometry in odd ways, sometimes casting shadows on half of a face.

Many of the assets were not started until late. Though concept art was created, many of the early art assets had to be either scrapped or recreated because of new tech implementation or just that the models were just not good enough. Due to the shortened timeframe, a lot of the assets had to be redesigned, so many of the assets do not reflect what was in the concept art.

Ultimately, the biggest issue was that the project scope was a little large for a three person team. Simply creating an MMO of any kind in a year would have been ambitious for a three person team. On top of this, *Broken World* planned to innovate on many areas of gameplay and software design. Little time was allotted for planning, experimentation, and for things to go wrong. If *Broken World* had had fewer areas of innovation, it could depend more on tried-and-true gameplay mechanics or software paradigms, and would have taken less time to implement overall. Alternatively, a greater number of team members would have distributed the load more evenly and allowed the team to accomplish a greater number of the project's initial design goals.

7.3 Future Work

No piece of software—and especially no game—is ever truly finished. There are always new features which can be added. This section describes a few potential improvements that could be made to *Broken World*, be they improvements to already-existing features or completely new unimplemented features that were planned for inclusion at one point and eventually removed in the interest of time.

7.3.1 Areas of Improvement

As pointed out in Section 6.2.2, *Broken World* is not bug-free. Several fairly minor bugs were found throughout the program, ranging anywhere from floating objects to tiles not quite lining up. The interpolation of networked players' movement is somewhat jittery, and with a little work could be smoothed out to something less jarring. In addition to the bugs reported by

users, technical testing revealed some stability issues related to multiple users logging on or off simultaneously. Occasionally, this would cause a race condition that would cause different types of messages being sent from the server to a single client to get mixed up, causing the client to lock up or the server to crash.

In addition, very little attention was paid to security during the development of *Broken World*. Due to the limited amount of development time, it was decided that security features were beyond the scope of the project. Passwords, for example, are stored in plain text, which is commonly considered to be a security faux pas due to the sensitive and personal nature of passwords. The commonly accepted method of handling passwords is to store not the password itself, but the result of hashing the password combined with a randomly-chosen salt. When the user logs in, their input is also hashed with the same salt and the result is checked against the database. This way, should the database become compromised, the user's password is not available for an attacker to easily read.

Other security concerns involve cheating. Most player actions are not verified, or if they are, they are not verified very thoroughly. By carefully constructing the correct packets to send to the server, players would be able to add items to their inventory, build buildings at any location, and modify the terrain without any problem. This could be corrected by checking that an action that the player claims to be taking makes sense given his or her current context.

Another security problem is that all game messages are sent in plain, unencrypted text. Even though clients are verified via a session identification number, this number could easily be stolen by an attacker using a man-in-the-middle attack, performed by looking at the packets travelling through some intermediary computer and using that information to construct false packets to send to the server. In this way, a well-positioned hacker could steal the identity of any logged-in user and issue commands as if he or she were any other player. This could be mitigated by using any one of the many different forms of encryption, such as sending information over an SSH tunnel instead of in plaintext.

Other desirable improvements represent limitations in *Broken World*'s underlying Unity engine. Unity's support for lighting on dynamic objects—the only type of object used in *Broken World*—is quite poor. To fix this problem, however, completely new shaders would have to be written to handle the lighting, which is a daunting task. In addition, Unity is single-threaded, which means it cannot take advantage of more than a single core in the user's CPU. With multi-

core machines becoming ubiquitous, this is a large waste of potential efficiency. It would be difficult to change this without replacing the game's entire Unity core with something else.

There are other small touches that would make *Broken World* feel much more polished as well. Extra animations for characters to show what they are doing, such as when they open a container or cut down a tree, would go a long way to sell the world to players. In addition, music and sound effects would make the game feel much richer. Many of these features were dropped in the interest of time; it was decided that additional work on the game's core technical features was more valuable in the long run than spending time on detail work.

7.3.2 Scrapped Features

Broken World was initially conceived as an MMO with heavily cooperative gameplay. Buildings originally needed multiple players to work together to be able to complete them, and every player sought to improve a "reputation score" which was awarded by peers in acknowledgement of the player's positive effects on the community. Players receiving enough upvotes were to be given more and more abilities (anything from the ability to start community projects all the way up to moderation tools.) Players with negative reputation scores were to have abilities revoked, such as no longer being able to use text chat and instead being forced to pick phrases from a prebuilt list until their reputation increases. These features would add an additional layer of depth to the *Broken World* gameplay experience, encouraging players to work together to build communities.

A feature scrapped very late in the process was the idea that each player would have his or her own house. This house would be identifiable as belonging to that player, and could be upgraded from the initial starting tent to a more substantial house, and eventually a brick house. Houses were also intended to give players in-game benefits such as the ability to store particular types of immobile items which would themselves add elements to the gameplay. This mechanic was intended to balance out the cooperative game mechanics as something players could focus on when nobody else was around to play with.

Items in the game were originally much more complex. The number of items or resources that a player could bring along was to be limited by that items' "bulk rating". This means that if a player finds a particularly large cache of valuable resources, he or she might need to make multiple trips to get it all if he or she goes alone. Another, more reliable method of gathering

would be to do so in parties along with other players. This way, players can work together to gather large caches of resources at a time and divvy it up amongst their community members when they return. Players would also be able to craft items such as backpacks, wheelbarrows, and even wagons which could aid in the carrying of resources.

Items originally also had a durability rating and would break down over time depending upon their use. The initial durability of an object, as well as its rate of decay, were based upon the materials with which it was crafted. For example using a crowbar as a handle would be much more durable than using a stick as a handle. Broken items could then be either repaired at a workshop or reused as material to craft other items.

Crafting and building in *Broken World* was originally meant to involve an element of time—specifically that it would take a certain nontrivial amount of real-world time in order for certain tasks to be completed. Progress bars would continue to fill while the player is logged off. The waiting mechanic was meant to add some interest to the game, because even when not playing the player's character would continue to make progress, compelling the player to want to check back in later and continue playing. The decisions a player made would feel like they had weight and value because of the real time it took to complete them.

The original versions of the game also included the use of food as a game currency. Players were to have to keep track of the amount of food they have, which would slowly decrease over time as they eat it. Food was to be used both to keep the players' stats in tip-top shape and to buy and sell items. Barter was intended as an easy, although sometimes expensive way to get tools and resources.

All of these features would make excellent additions to the *Broken World* experience. Together, they would provide a more rounded gameplay experience, and provide more meaning to player goals in the game.

8 Authorship

During the early stages of the project, all three team members split the game design process evenly. Design sessions typically involved gathering around a whiteboard and talking through game details together, followed by everyone splitting up and writing sections of the appropriate design document. When it came time for early prototypes to be produced, Nate focused a little more heavily on keeping the documents up-to-date while Mark did more coding, and Ethan produced concept art. As the project matured, the team members' roles diversified more.

Ethan did all of the original art for the game, including concept art, character models, animations, and most of the static environment assets. Some of the environment assets (such as trees and rocks, as well as the terrain textures) were stock art available as part of the Unity engine. Ethan also did the User Interface design and all of the art associated with the user interface.

Mark and Nate shared the programming burden equally. Testing and debugging was frequently a joint venture. Whenever a new technology was needed to fill some sort of gap, research and experimentation with potential frameworks was divided between both programmers. There were, however, certain aspects of the program that were worked on more heavily by one programmer than the other.

Mark did most of the work on the tile editor and its surrounding technologies, including the code used to render and store terrain PNGs. A few tools were developed early on using the Microsoft XNA Framework[55] to convert the original base tiles, made in Maya, into *Broken World's* custom PNG format. Mark also worked on the parts of the server that selected and updated tiles.

Nate laid most of the framework for *Broken World*'s core networked communications, on both the client and the server side. In addition, the headless client networking test tool and the Item Forge were initially built by Nate. He also built the framework that handles blended player animations.

9 References

1 Schreier, Jason. "'Call of Duty: Black Ops' Smashes One-day Sales Record." CNN Tech. CNN, 11 Nov. 2010. Web. 25 Apr. 2012. <http://articles.cnn.com/2010-11-11/tech/call.of.duty.sales_1_modern-warfare-activision-ceo-bobby-kotick-black-ops?_s=PM:TECH>.

2 Entertainment Software Association. "Industry Facts." The Entertainment Software Association. Web. 25 Apr. 2012. <http://www.theesa.com/facts/index.asp>.

3 Blizzard Entertainment. "WORLD OF WARCRAFT® SUBSCRIBER BASE REACHES 12 MILLION WORLDWIDE." Blizzard.com. Blizzard Entertainment, 07 Oct. 2010. Web. 25 Apr. 2012. <http://eu.blizzard.com/en-gb/company/press/pressreleases.html?id=2443926>.

4 Perry, David. "Are Games Better than Life?" TED Online. TED, Oct. 2008. Web. 25 Apr. 2012. <http://www.ted.com/talks/david_perry_on_videogames.html>.

5 Droniac. "Top 10 Most Successful MMOGs of 2010." Game Drone. 30 June 2010. Web. 25 Apr. 2012. <http://gamedrone.net/2010/06/30/top-10-most-successful-mmogs-of-2010/>.

6 Hartley, Adam. "MMO Developer Jagex Outlines 'MechScape'" TechRadar. 17 July 2009. Web. 25 Apr. 2012. <http://www.techradar.com/news/gaming/mmo-developers-jagex-outline-mechscape--617551?artc_pg=1>.

7 Interplay Entertainment. Fallout. Computer software. Interplay Entertainment Edusoft.

8 Bethesda Game Studios. Fallout 3. Computer software. Bethesda Softworks ZeniMax Media.

9 I Am Legend. Dir. Francis Lawrence. Perf. Will Smith, Alice Braga and Charlie Tahan. Warner Bros. Pictures, 2007.

10 Mad Max. Dir. George Miller. Perf. Mel Gibson, Joanne Samuel and Hugh Keays-Byrne. MGM Studios, 1979.

11 GSC Game World. S.T.A.L.K.E.R. Computer software. THQ.

12 4A Games. Metro 2033. Computer software. THQ.

13 Code Club AB. Wurm Online. Computer software.

14 Blizzard Entertainment. World of Warcraft. Computer software. Blizzard Entertainment.

15 Jagex. RuneScape. Computer software. Jagex.

16 Mojang. Minecraft. Computer software. Mojang.

17 Stainless Steel Studios. Empire Earth. Computer software. Sierra Entertainment.

18 EGenesis. A Tale in the Desert. Computer software. EGenesis.

19 Ślaziński, Grzegorz. Esenthel Engine. Computer software. Esenthel Engine. Web. 25 Apr. 2012. <http://www.esenthel.com/>.

20 BioWare. Star Wars: The Old Republic. Computer software. Electronic Arts.

21 Idea Fabrik Plc. HeroEngine. Computer software. HeroEngine. Web. 25 Apr. 2012. <http://www.heroengine.com/>.

22 Unity Technologies. Unity. Computer software. Unity Game Engine. Web. 25 Apr. 2012. <http://unity3d.com>.

23 Unity Technologies. "Made With Unity: Game List." Unity. Web. 25 Apr. 2012. <http://unity3d.com/gallery/made-with-unity/game-list>.

24 Aydin, Eren. Ratspell MMO Toolkit for Unity. Computer software. Ratspell. Web. 25 Apr. 2012. <http://www.ratspell.com/>.

25 Spartikus3. "The Inner Workings of an MMO Server Farm." The Lord of the Rings Online Vault. IGN, 28 Mar. 2007. Web. 25 Apr. 2012. <http://lotrovault.ign.com/View.php?view=CommunityArticles.Detail>.

26 Carmack, Carmen. "How BitTorrent Works." HowStuffWorks. Web. 25 Apr. 2012. <http://computer.howstuffworks.com/bittorrent.htm>.

27 Blizzard Entertainment. "Blizzard Insider #39 -- Art of the Firelands." Battle.net. Blizzard Entertainment, 28 June 2011. Web. 25 Apr. 2012. <http://us.battle.net/wow/en/blog/2993675>.

28 Eskil. "Love Development." Love. Quel Solaar. Web. 25 Apr. 2012. <http://www.quelsolaar.com/love/development.html>.

29 Rayowski, Matt. "General Overview of NLLIGO." Open NeL. 17 Sept. 2007. Web. 25 Apr. 2012. <http://www.opennel.org/confluence/display/NEL/NeLLigo>.

30 Bourke, Paul. "BMP Image Format." BMP Files. July 1998. Web. 25 Apr. 2012. <http://paulbourke.net/dataformats/bmp/>.

31 "Image Compression." Wikipedia. Wikimedia Foundation, 18 Apr. 2012. Web. 25 Apr. 2012. <http://en.wikipedia.org/wiki/Image_compression>.

32 Roelofs, Greg. "A Basic Introduction to PNG Features." Libpng.org. 14 Mar. 2009. Web. 25 Apr. 2012. <http://www.libpng.org/pub/png/pngintro.html>.

33 Roelofs, Greg. "Chapter 9. Compression and Filtering." PNG The Definitive Guide. San Jose, CA: O'Reilly & Associates, 1999. Compression and Filtering (PNG: The Definitive Guide). 2002. Web. 25 Apr. 2012. <http://www.libpng.org/pub/png/book/chapter09.html>.

34 Winch Gate. Ryzom. Computer software. Winch Gate.

35 "Mono." Web. 25 Apr. 2012. <http://www.mono-project.com/Main_Page>.

36 "MySQL :: The World's Most Popular Open Source Database." MySQL. Web. 25 Apr. 2012. <http://www.mysql.com/>.

37 "Libpng++." Libpng++ Documentation. Web. 25 Apr. 2012. <http://www.nongnu.org/pngpp/doc/0.2.1/>.

38 "Libpng Home Page." Libpng.org. Web. 25 Apr. 2012. <http://www.libpng.org/pub/png/libpng.html>.

39 MySQL++. Web. 25 Apr. 2012. <http://tangentsoft.net/mysql++/>.

40 "JQuery: The Write Less, Do More, JavaScript Library." JQuery. Web. 25 Apr. 2012. <http://jquery.com/>.

41 "Maya - 3D Animation Software." Autodesk. Web. 25 Apr. 2012. <http://usa.autodesk.com/maya/>.

42 Photo Editing, Photo Sharing. Adobe. Web. 25 Apr. 2012. <http://www.photoshop.com/>.

43 "CrazyBump." CrazyBump. Web. 25 Apr. 2012. <http://www.crazybump.com/>.

44 "Pixologic :: ZBrush." Pixologic. Web. 25 Apr. 2012. <http://www.pixologic.com/zbrush/>.

45 "UV Master." Pixologic. Web. 25 Apr. 2012. <http://www.pixologic.com/zbrush/features/UV-Master/>.

46 Borenstein, Elliot. "ERig." Elliot Borenstein. 2011. Web. 25 Apr. 2012. <http://www.elliotborenstein.com/Tools/eRig.html>.

47 "Wireshark." Wireshark. Web. 25 Apr. 2012. <http://www.wireshark.org/>.

48 "PerfMon." PerfMon. Web. 25 Apr. 2012. <http://perfmon.sourceforge.net/>.

49 Autodesk. "FBX - Platform-Independent 3D Data Interchange Technology." Autodesk. Web. 25 Apr. 2012. <http://usa.autodesk.com/fbx/>.

50 Nilsson, Hadrien. PngOptimizer. Computer software. PSYDK.ORG. Web. 25 Apr. 2012. <http://psydk.org/PngOptimizer>.

51 Radoff, Jon. "Anatomy of an MMORPG." Web log post. Jon Radoff's Internet Wonderland. 22 Aug. 2008. Web. 25 Apr. 2012. <http://radoff.com/blog/2008/08/22/anatomy-of-an-mmorpg/>.

52 Bode, Karl. "Average DSL Speeds By Country." Broadband DSL Reports.com. 26 Nov. 2007. Web. 25 Apr. 2012. <http://www.dslreports.com/shownews/Average-DSL-Speeds-By-Country-89739>.

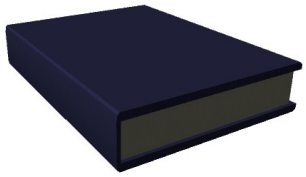
53 Jackson, Charles. "World of Warcraft Bandwidth Usage." Entertainment Guide. Demand Media. Web. 25 Apr. 2012. <http://entertainmentguide.local.com/world-warcraft-bandwidth-usage-3499.html>.

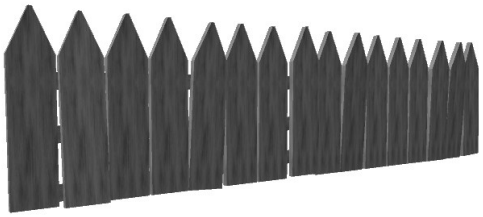
54 Shirinian, Ara. "Dissecting The Postmortem: Lessons Learned From Two Years Of Game Development Self-Reportage." Gamasutra. 8 July 2011. Web. 25 Apr. 2012. <http://www.gamasutra.com/view/feature/134679/dissecting_the_postmortem_lessons_.php>.

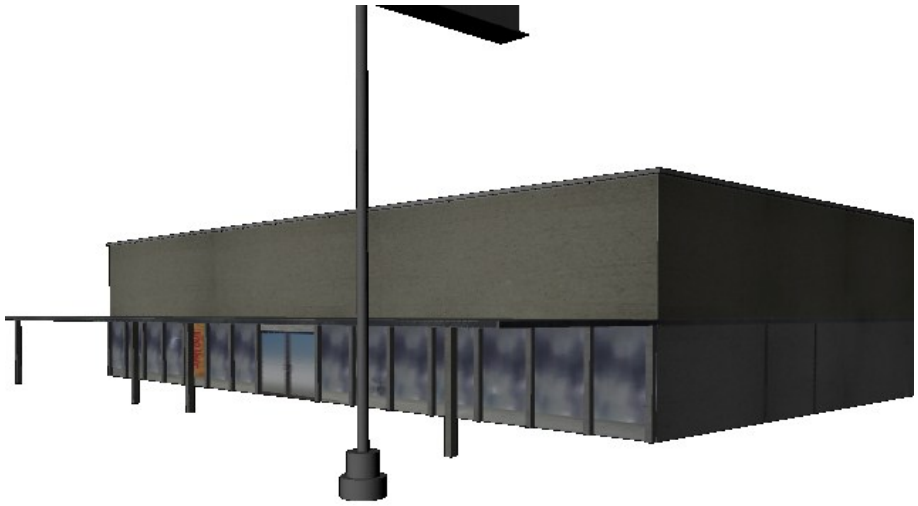
55 "XNA Framework Class Library." XNA API. Web. 25 Apr. 2012. <http://msdn.microsoft.com/en-us/library/bb203940.aspx>.

10Appendices

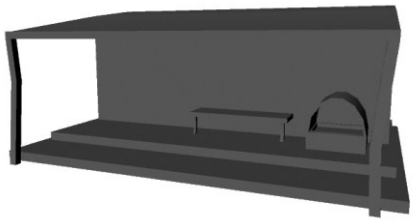
10.1 Unused Models

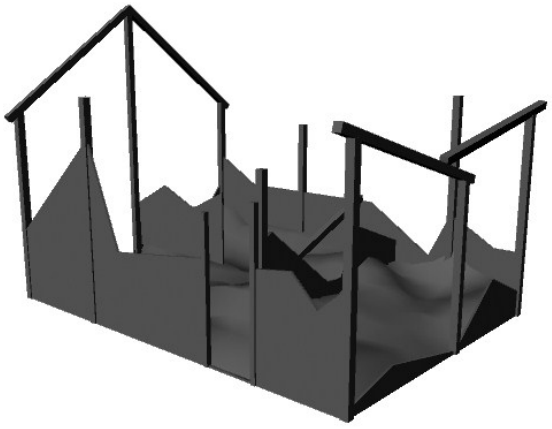


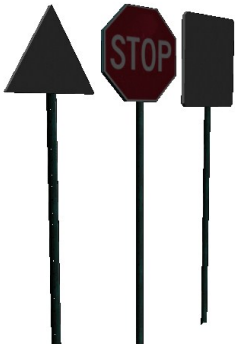




10.2 Incomplete Models







10.3 Original Design Document

Summary

Broken World is a massively multiplayer online game focused on community building and resource gathering. Players will work together to build buildings and items as they explore an expansive, procedurally generated world.

Philosophy

This game is an attempt to create a fun multiplayer experience based on primarily constructive gameplay mechanics rather than destructive gameplay mechanics, which is the current trend. We want to provide an alternative to the belief that a catastrophic event would create widespread selfishness and violence. Instead, we want to suggest the more optimistic view that society could rebuild itself, given enough cooperation.

Setting

Broken World takes place in a future America (New England) where fossil fuels and many other natural resources have been used up. Power plants have ceased functioning and gas pumps remain empty. No one has easy access to electricity for lights or gas for cars. Many technological advances no longer work. Without the production of medicinal products, many people who depended on medical care and those prone to sicknesses have passed away, leaving only those who are able to survive. Small scale conflicts arose around the globe, but as resources dried up, the armies disbanded and turned to the simple task of survival. Cities and towns begin to erode as there is no easy way to maintain them. Nature has slowly begun growing over what was once heavily populated areas.

Broken World is a fairly unique concept, but is not without precedent. The setting of *Broken World* is similar in many ways to the post-apocalyptic science fiction sub-genre. Works in this sub-genre focus on civilization-ending events, and on the way society looks after a worldwide disaster. One example work in games is the *Fallout* series, which takes place in a world devastated by nuclear war. Characters' primary focus is survival. However, the *Fallout* series of games portray a cutthroat, wild-west type world where the law of the land is to kill or be killed. This interpretation of a devastated society devolving into complete barbarism is very

common in post-apocalyptic fiction. In *Broken World*, we want to suggest an alternative: that humankind could instead band together in the face of adversity. This type of theme is much less common.

Broken World does not take place in a barren nuclear wasteland like the *Fallout* (1997) series. Its setting is much closer to that of the movie in Figure 2: *I Am Legend* (2007). *I Am Legend* takes place in a world where a disease turned humans into zombies, removing their influence on the world, but left the land unscathed. Grass and trees started to grow throughout the streets of New York, slowly reclaiming the metropolis into the landscape. This creates an environment that better lends itself to rebuilding.

Much of post-apocalyptic fiction includes highly fantastical elements, such as *Fallout*'s super mutants and *I Am Legend*'s zombies. The catastrophic event in *Broken World* is that the world ran out of fossil fuels, so threats are more closely grounded in reality, pitting players against wild animals, lack of resources, and starvation. In this sense, it is similar to works such as the movie *Mad Max* (1979), which is also set in a post-fuel world. *Mad Max*, however, focuses on a lone wolf fighting his way through a world of humans treating each other brutally. Self-reliance and an uncaring attitude are shown as necessary for survival in the wastes. These traits are not the type of traits that we want to emphasize, however. *Broken World* provides an alternative solution: that by working together, people can rebuild society and achieve great things.

The Pioneers:

A pioneer (player character) is a human character dressed in a hodgepodge of various scavenged clothes. What the pioneer looks like is decided by the player before entering the world.

NPCs:

After building up a town, merchants and doctors will move in when certain requirements are met. They will offer assistance, like the option to buy food or medicine. Other NPCs might include thieves and slightly dangerous animals. They will add a degree of danger to the Pioneer's journey.

Housing:

The player's housing starts off as a make-shift tent (made from tarps, sticks and rocks), and progresses towards more traditional house styles (colonial, saltbox, etc...). Housing can be

upgraded when the player amasses enough resources of specific types (such as wood and metal). Houses will act as a safe haven, storage location, and a spawn point for when things go wrong. The color can be customized to the player's liking, but the houses are otherwise unchangeable.

The World:

The open world in which the player explores consists mostly of open fields, with landscape features such as forests, lakes, mountains, abandoned towns, junkyards and campsites. Pioneers are able to search these areas and salvage goods needed to survive. The world is unkempt and overgrown, as the human influence over the land has decreased.

Items:

There are many items available to the player, all serving their own purpose. Certain subsets of items include: Furnishing for housing, food and medicine for survival, scrap resources for crafting and construction, tools, and trinkets for notoriety.

Gameplay

Overview

Post-apocalyptic games like the *Fallout*, *S.T.A.L.K.E.R.* (2007), or *Metro 2033* (2010) series almost always focus on combat as the primary mechanic. Many of these games have non-combat based sub-mechanics, however, which make for interesting gameplay. The *Fallout* games, for instance, encourage players to gather mundane everyday items—scraps scavenged from the past civilization—and use them to build useful items. Items degrade over time with use, and must be repaired using parts scavenged throughout the world. *Broken World* uses scavenging as a core mechanic.

There are many massively multiplayer online role-playing games (MMORPGs) that involve non-combative play such as resource collection and crafting. *Wurm Online* (2006) spends the first hour of the tutorial teaching players how to judge the type and age of various kinds of trees, and the ways in which players can cut them down and use them to build wooden objects and fires. *World of Warcraft* (2004) includes a basic crafting system. As seen in Figure 3, *Runescape* (2001) allows the player to engage in many non-combat skills such as fishing and woodcutting. In all of these examples, however, the primary reason to engage in these activities

is so they can gather resources to better their combat skills. *Broken World* reverses these roles such that combat is not a primary focus to the gameplay.

An example of another game whose focus is not combat is *Minecraft* (2009). The main gameplay element in *Minecraft* is going out into a procedurally generated world and gathering resources. These resources are used to construct a home base. Resources can be crafted into other, more selectively useful items like torches for light and tools for faster resource gathering. Players are encouraged and required to explore as resources closer to home become scarce. The world is dangerous, especially at night, and there are many ways for the player to die. When death occurs, all items the player has on them at the time are dropped. *Broken World* draws inspiration from a number of these gameplay elements, but *Broken World*'s execution is very different than that of *Minecraft*. Where *Minecraft* uses a building block mechanic to implement many of these ideas, *Broken World* uses implementations closer to that of games from other genres.

For example, resource collection and construction in *Broken World* draws more from real-time strategy games such as *Empire Earth* (2001) than from *Minecraft*. In *Empire Earth*, players must explore a procedurally generated map in search of resources such as iron, gold, stone, wood, and food. These resources are used to build infrastructure, such as roads, harbors, barracks, and other buildings. *Broken World* also has many different kinds of resources, and building construction works similarly, requiring players to amass a certain number of resources before they can complete construction.

Broken World also draws gameplay elements from simulation games such as *The Sims* (2000). The main idea behind *The Sims* is building a life for a virtual character. What the player's house looks and feels like, including things like color and furniture placement, is all decided by the player. Occasionally, other characters from the neighborhood show up at the player's house and socialize. Individual player progression in *Broken World* works like it does in *The Sims*—players will gather resources that they can use to improve their own house.

Player goals in *Broken World* are not as explicit as the goals in other MMORPGs. Many MMORPGs require players to complete quests, which drives the gameplay forward. *Broken World* is fairly open-ended and player-driven, similar to *Second Life* (2003). *Second Life* is a persistent virtual space which does not lay out any explicit goals for players to achieve. Instead, it serves as a platform in which players can socialize and create their own goals. Players do

whatever they find most interesting and play at their own pace. Much of *Second Life* focuses on community building, with players working together to build interesting world features. *Broken World* is not as extreme as *Second Life* in these respects, however, and falls somewhat closer to more traditional video games, in terms of limiting a player's influence over their environment to what the player's character is able to perform..

The closest existing game to *Broken World* is a MMORPG called *A Tale in the Desert* (2003). *A Tale in the Desert* has no combat system at all. Instead, players progress through the game by acquiring new skills. Players start by learning how to make bricks and plant flax, and learn more and more complex skills until they can build everything from pearl necklaces to obelisks. Tasks are built on top of one another such that the process of spinning silk might require that the player learn stonecutting along the way. The game is fun as a result of its complexity. There are many different areas to explore, and many different tasks to complete in the game, which drives players to want to continue acquiring new skills and trying different aspects. Players can see the quests that others have completed, which spurs them on to want to get as many badges by their name as possible. Despite being an MMO, however, the game is largely a solitary experience. This is partly due to the scarcity of players, but also due to the fact that there are few incentives for players to work together. Players typically learn everything they need themselves, and trading is not strictly necessary because most players can just as easily build everything they need. *Broken World* hopes to achieve a game built around similar gameplay principles, but with a stronger emphasis on player cooperation and community building.

Player-Driven Goals

Like many Massively Multiplayer Online Role-Playing Games, there is no "win condition" for *Broken World*. Instead, the game is driven by self-imposed goals. Players can choose to focus on whichever activities they most enjoy, be that exploring the world in search of new frontiers, socializing with other players, rebuilding towns and cities, or constructing and upgrading their own personal shelters. *Broken World* is a sort of social sandbox game.

Exploration

Much of the players' time will be spent exploring the expansive, procedurally-generated world looking for resources. The further players press out in the wilderness, the more likely they will be to discover resource caches that nobody else has found before. Players can find natural resources, such as edible plants, trees that can be cut down for wood, or mineral deposits. With the proper tools (axes for cutting down trees, a pickaxe for mining) players can collect these resources and use them to build interesting devices.

Gathering resources is a vital part of *Broken World*. Some resources, such as berries, trees, and other natural things are renewable. Others come in the form of non-renewable sources, such as metals, plastics, and even items scavenged from "the time before". Players will need to explore further and further to gather the resources that they need. Trading with other players can be a good way to obtain scarce items, or items from far away.

Players might also scavenge man-made resources from the time before The Crash. Old cars might be stripped for scrap metal, rubber, and small amounts of oil. Old refrigerators might be scavenged for parts to build new, makeshift ones, and batteries could be pulled from old electronic devices to make new ones. If players want to take advantage of large caches of resources, they would need to bring their friends along to help them strip things down and divvy up the rewards later.

The number of items and resources that a player can bring along is limited by the items' bulk. This means that if a player finds a particularly large cache of valuable resources, he or she might need to make multiple trips to get it all if he or she goes alone. Another, more reliable method of gathering would be to do so in parties along with other players. This way, players can work together to gather large caches of resources at a time and divvy it up amongst their community members when they return. Players would also be able to craft items such as backpacks and even wagons which could aid in the carrying of resources.

In addition to gathering resources from the world, players can collect seeds which will allow them to farm, creating a renewable source of food. Plants can be planted on soil anywhere, but have slightly different effects depending on where they are planted. Plants planted on a player's personal soil near their house can only be used by the player who owns the house. Plants planted in the wild can be used by anyone who comes across them. Players can also work together to build a plantation community project which limits the number of resources that players gather from the plantation in a given day (but does not limit the number of plants a player

can plant there.) Trees can be harvested for wood only once, but food-bearing plants will continue to produce food over time.

Crafting Economy

In a world where survival is the key objective, food becomes the primary currency. Players will have to keep track of the amount of food they have, which will slowly decrease over time as they eat it. Food is used both to keep the players' stats in tip-top shape and to buy and sell items. NPC merchants will trade food for items (or items for food) and players will be able to include it in trades amongst one another. Barter is an easy, although sometimes expensive way to get tools and resources.

Another way to get tools is to craft them oneself. Players will be able to build items from the various materials that they scavenge throughout the world. Each of these items will grant them some ability. Armor will grant a bonus to health, axes can be used to cut down trees, and bows and arrows can be used to hunt animals for food. Items degrade over time as they are used based upon their durability, and players will have to scavenge resources in order to repair them.

The crafting system in *Broken World* is inspired largely by the crafting system of an MMORPG called *Ryzom* (2004). In *Ryzom* crafting is done with recipes that are compromised of two parts: amounts of resources, and the resources themselves. For example, in Figure 7 the player is crafting a piece of armor. The recipe calls for 4 armor shell resources, 4 lining resources, 2 stuffing resources, and 2 armor clip resources. The player then chooses the specific resources they wish to use from their inventory which is what determines the quality and effectiveness of the item. The crafting system in *Broken World* works roughly the same way.

Crafting and building in *Broken World* involves an element of time—specifically that it takes a certain nontrivial amount of real-world time in order for certain objects to be completed. This can be a dangerous element to try to incorporate, because waiting in itself is not a fun gameplay mechanic. Games like *Wurm Online* use it poorly—every action that a player takes in *Wurm Online* takes a certain amount of time (usually a few seconds) to complete. While the player is waiting for the action to complete, however, he or she cannot do anything else but watch the progress bar fill up. Other games like *A Tale In the Desert* force the player to watch an animation every time that an action is taken as a way of regulating the frequency of player

actions. Neither of these make for a fun mechanic because it does nothing more than slow down the gameplay.

There are games that implement realtime mechanics effectively, however. For example, *EVE Online*'s (2003) method of skill progression is time. A player selects their skill and the bar fills (the first tier takes 5-10 minutes). Each level takes longer to complete. *Tiny Tower* (2011) is an iPad game that uses the amount of time to build things as its primary game mechanic. Even though it seems like waiting for something to complete is boring, the number of choices and resources that the player is given is limited in ways such that the player always has something to do. In both *Eve Online* and *Tiny Tower*, what makes this mechanic effective is that the player is not prevented from taking other actions while they are waiting for the progress bar to fill. Progress bars also continue to fill while the player is logged off. The waiting mechanic actually adds some interest to the game, because even when the player is not playing the player's character continues making progress which compels the player to want to check back in later and continue playing. The decisions a player makes feel like they have weight and value because of the real time it takes to complete them.

The Player's House

Every player starts out with a simple tent on a plot of land. This serves as the point to which players can return, rest, and regroup after exploring the wilderness and scavenging for resources. Using resources that they scavenge from the world, players can build items and upgrades for their houses. There are several items which can only be crafted at the player's home. These items might include small power generators, ethanol distilleries, or refrigeration units. Home-bound items grant the player special abilities when they are at their home.

The tent is only capable of supporting a small number of home-bound items. In order to support a greater number of items, players will have to upgrade their homes. To do this, they will have to collect a pre-specified number of raw resources (such as wood, metal, and stone.) Upgrading a tent will give the player a shack. Upgrading a shack will give the player a small cabin, and so forth. Each upgrade allows the player to build a greater number of home-bound items.

Ideally through paint players will be able to change the color schemes of their houses by picking a new main color and a new accent color.

Defense

The world is not perfectly safe. Bands of thieves, packs of wolves, bears, and other dangers exist in the wilderness. Players will need to defend both themselves and their homes from threats. The simplest way to do so is with the "safety in numbers" principle. Wild animals and thieves are less likely to appear in places where there are a greater number of players. Players are encouraged to work together for safety reasons.

The player has two status bars to keep track of: health and stamina. When a player's health reaches zero, he or she faints and is dragged back to his or her house and restored to full health. A player's health is reduced by attacks from wild animals, thieves, or natural forces. Stamina is a general measure of fatigue, and is reduced slowly over time when the player does not have any food, or when the player performs particularly strenuous tasks. The higher a character's stamina is, the more effective he or she will be at performing any task in the game. Health regeneration rate is also affected by the player's stamina.

In addition, players can craft items to defend both their homes and their characters. Bows and arrows and armor will all help to keep a character safe while he or she is out gathering resources. If a player is attacked (and brought to zero health) whilst in the wilderness, he or she will be transported back home, but all carried items will be lost. This makes safety during exploration imperative.

Players can defend their homes when they are away in various ways as well. There are several home-bound items that will make them more difficult for thieves and animals to break into: fences, locks, and alarms might be a few examples. In addition to building items for defense, players can rely on community support to keep their homes safe. Community buildings such as lookout towers provide some level of protection for all homes within a certain radius of them.

Players will not be allowed to attack each other, or steal items from one another. Those actions are reserved for the NPCs. The reason for this is because we are trying to shape a game focused around community building. Just as other MMORPGs limit player actions to attacking things instead of diplomacy, we too are limiting the number of actions that players can take in order to shape a particular gameplay experience.

Collaboration

There are many tasks in *Broken World* that cannot be accomplished (or cannot be accomplished at a reasonable rate) without the cooperation of multiple players. These tasks designed to promote a gameplay experience that encourages community building, in both the figurative and literal senses.

Players are given a single house that they alone are responsible for maintaining, but they are encouraged to help others out with their houses. Players can post messages to passers-by requesting help completing construction on various tasks. They can post rewards in return for assistance and are notified of any help other players have given them.

In addition to building items and upgrades for their own personal benefit, players will also be encouraged to work together on buildings that will benefit the community. A player with enough reputation can lay the foundation for a community building, such as a marketplace, town hall, or power station. Each community building will have its own unique effects on the gameplay, benefiting all players in the vicinity. The amount of resources required to build a marketplace (or any community building) would be prohibitively large for a single player to collect on his or her own, but multiple people working together would be able to build one. Communities would be responsible for defending and supporting one another.

Community buildings are built in phases, with each phase requiring a certain number of resources to complete. Once the foundation has been built, the statistics for which resources are needed to finish the phase will be displayed for all passer-bys to see. Anyone who wants to contribute can drop off resources into the community project. After enough resources have been amassed, players can work together to craft the building in the same way that items are crafted. After enough time has been spent, the building's appearance will change to look "more complete" and a new phase will begin, with different resource requirements. Once all phases have been completed, the building will be finished and its gameplay elements will begin to take effect. For example, as soon as a marketplace is built, NPC vendors will move in and be willing to trade items with players.

The primary method of encouraging positive community involvement is based on a system used by the programming question-and-answer website Stackoverflow (2000). Stackoverflow uses a reputation system to regulate and moderate the entire website. Each user is given a reputation score, which starts at zero. As users post good questions or give good answers,

other users can give them upvotes which increase their reputation score. Particularly bad, irrelevant, or dangerously incorrect answers can be given downvotes, which decrease a user's reputation score. A user's reputation score is used to determine what he or she can do on the site. Users with lots of upvotes are given access to editing tools, commenting on posts, and even moderation tools while users with low reputation scores are not. This way, the community polices itself, and posting good questions and answers (activities beneficial for creating a positive and helpful community) become a game which encourages benevolent behavior. *Broken World* has a reputation system intentionally built to emulate Stackoverflow's, with the hopes of creating the same effect—namely that of encouraging helpful behavior while discouraging nonconstructive behavior. Players receiving enough upvotes will be given more and more abilities (anything from the ability to start community projects all the way up to moderation tools.) Players with negative reputation scores will have abilities revoked, such as no longer being able to use text chat and instead being forced to pick phrases from a prebuilt list until their reputation increases.

Technology

The technical details behind generating a massive procedural world are loosely based on *Ryzom*'s world generation system. Under the hood *Ryzom* uses a tiling system which they call Ligo to help their designers and artists make a fun and interesting world. They have a system of 2d tiles, each of which represent a 3d landscape piece that was made by hand by an artist, which they can place together to form the world. *Broken World* uses a similar system which allows for a procedural world to be quickly generated in 2d and then seen by the players in 3d. Developers can craft each tile and then allow the generation system to place them in interesting ways as defined by an algorithm.

The networking for *Broken World* is based on the BitTorrent (2001) peer-to-peer model. In the BitTorrent paradigm users start off by announcing their presence to a tracker who pairs them up with other peers who have the same files. Seeders—clients who have the entirety of a file—give freely to clients in an attempt to get the largest variety of packets to as many people as they can. The leechers—clients who are still downloading the file—are sharing packets with other leechers as well as requesting packets from the other seeders and leechers they are

connected to. This distributed model allows for files to be shared or downloaded to a large number of people without totally draining the bandwidth of a single server.

In *Broken World*, the sever acts like the tracker, which tells clients who they should connect to, as well as the "true" seed, which maintains an up-to-date state of the whole world via a database. When a client approaches a new tile it asks their peers if the tile has been generated in the past. If the tile has been generated, they get the current state from whichever client knows it or from the server if there are no clients who know about that tile. Clients also validate the player states of other players which helps keep the load off the server. Whenever there are conflicts with either tile states or player states the server will step in and resolve the conflict. If a particular client causes too many conflicts it will be assumed that they are trying to cheat and they will be disconnected or dealt with separately.

As far as we can tell, the use of a BitTorrent model for networked games has yet to be applied in a game. This model allows for *Broken World* servers to be run on weaker machines than are typically required for MMO games which reduces the cost required to host them and increases the scaling potential.

Gameplay Stories

The following is a short user story detailing an experience a player might have while playing *Broken World*.

May has been playing Broken World *for a few weeks now. The first few days were spent getting used to the game and amassing some of the basic things her character needed for survival—food, an axe for cutting down trees, and a few other basic resources. She had upgraded her house from a simple tent to a small shack by herself, and had begun to meet a few of her neighbors.*

As she walks through town, she passes GeneralFluffy's house and notices that it is partway through an upgrade from a shack to a small cabin. Since she's not working on any projects right now, and she could use a reputation boost, she decides to help GeneralFluffy out. She clicks on the house and makes it her current project, and with her added help the amount of time to complete the project drops from an hour to a half hour.

She makes it to the center of town and sees the community project that somebody had started a few days ago. The town was starting to get a little bigger, so somebody had decided to

start building a marketplace. May is looking forward to its completion, because that means that she could have easier access to some of the more difficult items to find. She checks the building's progress and sees that all that is left to find is a significant amount of scrap metal.

Another player named ScottGregor walks by and says that he found a large store of scrap metal, but couldn't bring it all back by himself. May says she'll come with him and they wait a few minutes for a small group to gather. Once they have enough people, they follow Scott out of town through the forest and then to an old junkyard. Scott had already used his prized bolt-cutters to cut through the locked fence earlier, so the group can go through freely. They enter the junkyard and scavenge all of the scrap metal that they can carry, and bring it back to town.

When they get back to town, they add their newly-found resources to the marketplace project. Now that they have enough resources, they can begin building. May has a few minutes left until GeneralFluffy's house is complete, though, so she decides to wait until that is done before switching her project. In the meantime, she opens the reputation screen, finds ScottGregor, and gives him a thumbs up for helping everyone out. As she's looking at his page, a few of the other members of the group thumbs-up him too, and his reputation score jumps by a lot.

A notification pops up on May's screen informing her that GeneralFluffly's house has been completed. Now that it's done, she can switch her current project to the marketplace. Even though there are ten people working on the project, it will still take a whole two days to complete if they can't find more players. The good news is that even if she logs out, her character will continue to contribute to the project.

As she's contemplating what to do next, May receives a notification informing her that GeneralFluffy has given her a thumbs-up. Excited, she opens up her reputation window and finds that that thumbs-up was all she needed to push her over the next landmark. With five hundred reputation, now she is allowed to start community projects herself! She looks through the new menu she has for something exciting to build, but eventually decides against it because the town doesn't need to be distracted with two projects at once.

May realizes that she has something to do, so she logs off.

Broken World does not use traditional quests, but rather uses player driven tasks. Many, if not all, tasks have requirements that must be filled before they can be completed. These tasks can be categorized into three different sections: resource gathering, crafting, and building. Each

of these can be completed by oneself or cooperatively, but players can accomplish a great deal more if they work together. The following descriptions are an in game play-by-play of specific tasks. Each description takes place in a different section of the game, so the timeline is not consistent.

Scavenging/Resource Gathering

- *The player May has just entered the world for the first time. All she has placed is the starter house (a tarp tent). She has some food (a loaf of bread, pouch of rice and a canteen of water) and some supplies (rope, a small mallet, a knife and a fire starter). In order to progress any further, May must explore the area for some starting resources. Since it grows naturally, wood is the most abundant resource. Unfortunately for May, she cannot chop down trees, because she does not have an axe. She must forage for sticks and stones to make one (the requirements for making an axe are one piece of wood, one rock and one rope). Maybe if she's lucky, she'll find a piece of sharp scrap metal that will make a better axe. May has to keep track of where she's going, however, for one wrong turn can mean getting lost. If May starves, she'll lose all of her inventory and be sent back to her house. Fortunately, she is able to find the necessary resources, and even gets lucky by finding an abandoned car that has duct tape and radio parts in the glove compartment and a crowbar in the trunk. She makes her way back to her house, where she can keep any items she found safe.*
- *May has done some exploring and has stumbled upon an abandoned town. The town is abundant in scrap metal. There is far too much for May to carry home by herself in one trip, so she enlists the help of another player, ScottGregor. May was able to work out a deal with ScottGregor: she would give him some food and metal if he helped out. ScottGregor agrees because his food supplies are low at the moment. They search through the town and find plenty of resources in the various buildings. (ScottGregor finds a high quality axe that has an improved wood gathering yield). Both May and ScottGregor make it back to their respective houses. May gives ScottGregor a "thumbs up" on the community reputation board, increasing his standing with other players.*
- *The player GeneralFluffy has posted a task on the community board. He wants to build a well for the town, but he doesn't have the required amount of stone. He is willing to give*

a good amount of food to anyone who brings him some. May happens to know of a place where there's a large amount of stone, but it's too far north and there's far too much stone for a single trip to be feasible. Fortunately for May, she has recently acquired the necessary resources for an outpost (a movable, temporary housing unit) and push cart. When she arrives as her destination, she gathers enough stone using a pickaxe and is ready to return home. The push cart allows her to transport the stone long distances, but she cannot carry her outpost back at the same time. She leaves it there for the time being, hoping no one comes by and takes it for themselves. May delivers the stone to GeneralFluffy and collects her reward.

Crafting

- *May wants to be more effective at resource gathering. Since wood is an important resource, she will make an axe. By opening up the crafting menu (something that is available while at home) she is able to create what she wants. A list appears showing what is available to make with the resources at hand. Because she has enough resources to create an axe, the option appears. Craftable items that she doesn't have the resources for are greyed out, and show what resources are needed in order to craft. Crafting something big, like a chainsaw, is very expensive and requires many different pieces. Seeing as May doesn't have very many resources at this point, she crafts a simple axe.*
- *Having a radio would greatly increase the distance at which May could communicate with others. The problem is, radios are very complicated and take many parts to craft. Things like batteries are hard to find, and some parts need to be crafted out of other resources first. Luckily for May, GeneralFluffy found some batteries in one of the towns he rummaged through. He's willing to trade them away for some food and that nice pickaxe May has been using. May agrees and takes the batteries. She has plenty of scrap metal and copper wiring. Using multiple tools such as the mallet and duct tape, she is able to hack together a radio. While it's not 100% effective (finding an intact radio would be the most desirable), it will get the job done.*

Building

- *May wants to upgrade her house to the second level. There's more space for storing items and food, as well as bigger and better technology, like beds and solar-powered generators (some of these are cosmetic, while others fulfill certain requirements). By opening up the building properties, May can see what the requirements for upgrading her house are. 30 wooden planks are required (obtained by chopping down trees with an axe). May has already gathered over 30 planks and is ready to start upgrading. She clicks the 'upgrade' button in the properties menu and a progress bar starts filling. After 10 minutes the building will have finished upgrading and May will have a level 2 house. May is free to walk around while the house is upgrading, but she cannot build anything else while it is still in progress.*
- *May has started a town with players ScottGregor and GeneralFluffy. They have done so by building their houses close to each other. GeneralFluffy's house is already at the third level, and he's recently obtained all of the necessary resources to upgrade to the fourth. Unfortunately, it's going to take an hour to complete by himself. GeneralFluffly is a little bit impatient, so he puts a task on the town's community board. This task is to help him build his house. Any player can open up the building properties of other player's buildings and help them upgrade. May sees that GeneralFluffy wants some help and goes over to lend a hand. By clicking the upgrade button in the properties menu for GenearalFluffy's house, May has contributed her time towards upgrading another player's house. This reduces the amount of time required to finish the project. It is a bit of a sacrifice for May (she cannot work on any other projects while this is going on) but hopefully GeneralFluffy will repay the favor in the near future, and possibly give her a reputation point for her trouble.*
- *May wants to build a clinic for the town, just in case someone suffers a grievous injury. However, she lacks the necessary resources and the project is going to take multiple hours. She asks around town and, as it turns out, ScottGregor has the resources that she's missing. ScottGregor offers his resources and time as long as May will help him find parts for a radio that he is crafting. May agrees and construction of the town clinic begins.*

Item Build List

The following is a list of all materials and items developed for the first pass of the game.

Material Types:

1. Handle
2. Sharp edge
3. Adhesive
4. Weight
5. Fabric
6. Body
7. Container
8. Food
9. Wheel
10. Firestarter
11. Flat
12. Flammable

Base Items: These items cannot be broken down any more, and are not specific tools

- Sticks (1, 12)
- Wood (1, 4, 6, 12)
- Food (8)
- Water
- Stone (4, 11)
- Dirt
- Scrap Metal (1, 2, 4, 6, 11)
- Rope (3, 12)
- Duct Tape (3)
- Tires (9)
- Paper (12)
- Cloth (5)
- Matches (10)

Tools: * denotes an immobile tool

- Hammer: Required to build buildings
    - 1 weight

    - 2 handle
    - 1 adhesive
- Crowbar (1): Turns items into scrap metal
- Axe: Turns trees into wood
    - 2 handle
    - 1 adhesive
    - 1 sharp edge
- Shovel: collects dirt; makes holes
    - 3 handle
    - Adhesive
    - 2 flat
- Pick: collects stone from deposits
    - 2 sharp edge
    - 1 adhesive
    - 2 handle
- Saw: turns wood into planks
    - 1 handle
    - 2 adhesive
    - 3 sharp edge
- Chisel (2): turns stone into cut stone
    - 1 sharp edge
    - 1 handle
- Bucket (7): turns dirt & water into mortar; allows you to transport water
- Kiln*: turns fire, dirt & water into bricks
    - 5 mortar
    - 25 weight
- Firestarter: turns flammable objects into fire
    - 1 flint
    - 1 steel
    - (OR 1 box matches)
- Backpack: increases carrying capacity

    - 5 fabric
- Wheel Barrow: greatly increases carrying capacity
    - 1 wheel
    - 10 body
    - 4 handle
- Mailbox*: allows players to leave you messages
    - 1 container
    - 2 handle
- Signpost*: leaves stationary messages for other players
    - 3 handle
    - 2 flat

Houses:

- Shack (Tier 1)
    - 100 scrap metal
    - 10 stone
- Shed (Tier 1)
    - 100 wood
    - 10 scrap metal
- Sod House (Tier 1)
    - 100 dirt
    - 10 wood
- Shanty (Tier 2)
    - 1000 scrap metal
    - 50 wood
    - 10 stone
- Stone House (Tier 2)
    - 1000 cut stones
    - 100 wood
    - 50 metal
    - 50 mortar
- Cabin (Tier 2)

    - 1000 planks
    - 100 stone
    - 50 scrap metal
- Brick House (Tier 2)
    - 1000 bricks
    - 100 stone
    - 50 wood
    - 50 scrap metal
    - 50 mortar

Community Buildings: Require a certain amount of reputation to initiate, and a lot of labor

- Marketplace (75 rep): allows players to barter with NPCs for tools/food and resources; allows players to sell items to players visiting nearby marketplaces, auction-house style
    - 1000 stone
    - 100 cut stone
    - 2000 food
    - 200 cloth
    - 100 stick
    - 500 scrap metal
    - 2000 wood
- Well (20 rep): source of water [must have shovel & pick in inventory to contribute labor!]
    - 50 cut stone
    - 10 mortar
    - 10 rope
    - 1 bucket
    - 20 wood
- Community Board (30 rep): large-scale messaging forum
    - 20 wood
    - 50 paper
- Forge (60 rep): turns scrap metal into useful tools, for a fee (of food)
    - 30 rope

    - 50 weight
    - 10 flat
    - 50 fabric
    - 500 stone
    - 2000 scrap metal
    - 1000 plank
    - 3000 wood
    - 10 firestarter
    - 5 hammer
    - 100 water
- Workshop (50 rep): when in a workshop, players are considered to have a hammer, saw, axe, crowbar, pick, chisel, and bucket (regardless of whether they actually do or not)
    - 1000 plank
    - 300 wood
    - 500 stone
    - 5 hammer
    - 5 crowbar
    - 5 axe
    - 5 pick
    - 5 chisel
    - 5 saw
- Warehouse (55 rep): allows you to publicly donate or take items; good source of reputation flow
    - 800 stone
    - 1000 food
    - 2000 plank
    - 2000 wood
    - 1000 scrap metal

Reputation Benefits:

- 5 – mini map
- 10 – downvoting

- 15 – veto tear down operations
- 20 – extra upvotes per day
- 25 – veto build operations
- 35 – upvotes are worth more
- 100 – ability to place reputation bounties on projects
- 150 – initiate tear down operations
- 250 – can reallocate resources between community projects
- 500 – fast travel
- 1000 – moderator abilities (ban users)

Feature List

Core

- You have a house
    - It is upgradable
- Infinite world [Prototyped]
- Premade Tiles (500x500 units)
- Client / Server Networking
- Client / Client Networking
- Storage of player changes
- Resource placement / generation
- Resource gathering
- Inventory
- Crafting system
    - Craftable Items
- Player Character & Animations
- Account Setup & Initial Player Position
- Login Handling
- Logout Handling

Primary

- Character Creation Screen
- Water & Player Interactions

- Procedurally-Generated Filler Space on Tiles
- Entering your House
- Resource Regeneration
- Resource Gathering Hierarchy (Cutting Down Trees, Mining, etc)
- Upgrading Crafted Items
- Building/Labor System
- Reputation System
- Community Building Projects
- Chat System
- Item Condition/Durability
    - Repairing Items
- Food and Hunger
- Trading System
- Events that happen to your house when you're not around
- Workbenches
- Character Creation / Customization
- Title Screen
    - Intro Sequence (Including Wasabi Studios Logo)
- NPC Dialog System
- TShirts
- Editable Character Skin Color

Secondary

- Organizing Your House (Moving Stuff Around)
- Hunting
- Fishing
- Defending oneself with weapons
- Restricting Chat Based on Reputation
- Other Restrictions Based on Reputation
- Thieves that attack you physically (AI)
    - Player Death
- Gardening

- Narrative
- Electricity-Related Items
    - Batteries, Power Plants, Generators, Etc
- Option to Start Near People You Know
- Detailed Resources (Copper, Iron, Steel, etc)
- Character Accessories
- Compass / Minimap Style Thing to Orient Players
- Wagons
- Editable Character Proportions

Reach

- Animals
- Mounts
- Illnesses
    - Doctors, Clinics, Condition Effects
- Day/Night Cycle [Prototyped]
- Seasons
- Weather
- Jumping
- Missions
- Friends List
- Rebinding Controls
- Patching / Update System
- Support Forum
- Super Detailed Resources (Balsa Wood, Pine Wood, Oak Wood, etc)
    - Realistic Statistics
- Wandering Friendly AI
- Housing Trees
- Finite / Controlled NPC Populations
- Natural Disasters / World Events
- Oceans, Boats, and Swimming
- Achievement System

10.4 Technical Testing Data

Broken World	Technical Testing Data
Client	
CPU	Intel Core 2 Duo 2.53GHz
RAM	6.0 GB
OS	Windows 7 64-bit
Graphics Card	NVIDIA GeForce GTX 260M

Login Screen — Login & Level Load

FPS	Total TCP Up (Bytes)	Total TCP Down (Bytes)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
86.62087099	18,772.	259,753.	1841	60048	431.26	5,967.45
85.99421015	18,837.	259,923.	2058	66065	334.35	4,613.48
75.87177914	19,036.	260,041.	1896	60220	484.95	6,624.65
91.5442757	19,088.	259,816.	1724	57120	556.73	7,577.92
161.8282439	19,386.	260,449.	1941	66066	529.34	7,111.66

Tile Loading

Total TCP Up (Bytes)	Total TCP Down (Bytes)	Lag Time (ms)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
3,891.	128,132.	2995.34	1940	66347	1,599.32	52,666.30
4,485.	67,587.	3007.2	2092	35627	1,441.08	21,716.51
6,963.	112,913.	2834.58	3388	62979	1,082.17	17,548.56
6,051.	109,111.	2232.23	3993	77372	2,647.23	47,734.50
3,513.	112,345.	2651.73	1886	66043	1,297.66	41,498.83

Normal Play

# Headless Clients	FPS	Total TCP Up (Bytes)	Total TCP Down (Bytes)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)	Total UDP Up (Bytes)	Total UDP Down (Bytes)	Max UDP Up (Bytes/sec)	Max UDP Down (Bytes/sec)	Avg UDP Up (Bytes/sec)	Avg UDP Down (Bytes/sec)
0	53.00557229	0	0	0	0	0	0	0	0	0	0
0	77.9734785	0	0	0	0	0	0	0	0	0	0
0	72.88456758	0	0	0	0	0	0	0	0	0	0
0	63.14921603	0	0	0	0	0	0	0	0	0	0
0	70.52894318	0	0	0	0	0	0	0	0	0	0
1	153.2565889	0	0	0	0	46847	46136	1027	948	789.4628857	777.4811555
1	146.9964571	0	0	0	0	46768	45188	869	869	790.243881	763.5464526
1	115.4236688	0	0	0	0	46768	45899	869	869	790.1271411	775.4457246
1	147.2574939	0	0	0	0	46610	45741	869	790	791.3472754	776.5933432
1	202.1314783	0	0	0	0	46531	46294	869	869	783.8909739	779.8983204
60	-	-	-	-	-	-	-	-	-	-	-
60	-	-	-	-	-	-	-	-	-	-	-
100	-	-	-	-	-	-	-	-	-	-	-
100	-	-	-	-	-	-	-	-	-	-	-

Item Crafting

Total TCP Up (Bytes)	Total TCP Down (Bytes)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
669.	780.	507	720	124.00	144.57
1,746.	23,246.	828	18156	157.82	2,101.24
561.	720.	507	720	559.78	718.44
936.	15,782.	882	15782	785.04	13,236.71
569.	728.	515	728	566.54	724.85

Building Placement

Total TCP Up (Bytes)	Total TCP Down (Bytes)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
416	513	416	513	465.99	574.65
416	513	416	513	435.98	537.64
416	513	416	513	466.54	575.33
416	513	416	513	478.46	590.03
416	513	416	513	450.52	555.57

"Normal Play"

Avg Mem (Bytes)	Max Mem (Bytes)	Avg CPU	Max CPU		
536234193	604901376	38.7855	66.876		550747847.5
517694695	602324992	19.3585	65.7705		
589709529	659361792	29.5115	62.2395		
559352973	641482752	25.632	64.4725		

Polygon Test (Revision 415 B-Term Demo)

Mesh	Verts	< 60 FPS	< 30 FPS
Tent	2402	375	800
Female Player	15556	31	59
Rock	40	1900	4400
Alder	1733	1500	2475
Transmission Tower	1099	1800	2900

Terrain Size

Terrain Type	Mesh (fbx) Size (KB)	PNG Size (KB)	Optimized PNG Size (KB)	Terrain Size Improvement Ratio: (Mesh over png)	Ave Size Improvement Ratio:	Compressed size Ratios:	Ave Compressed Improvement Ratio:
Flat	211	0.61	0.097	345.9016393	190.638923	2175.257732	731.3538573
Smooth	234	3.37	0.613	69.43620178		381.7292007	
Down	368	1.65	0.689	223.030303		534.107402	
Double	368	2.79	2.867	131.8996416		128.3571678	
Up	300	1.64	0.686	182.9268293		437.3177843	

Synchronization

Chat	Instant to the naked eye
Movement	Instant to the naked eye

Broken World Technical Testing Data

State Server

CPU	2.2 GHz Athlon 2800+
RAM	512 mb
OS	Ubuntu 11.10

"Normal Play"

# Headless Clients	Total TCP Up (Bytes)	Total TCP Down (Bytes)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
0	0	0				
1	258,780.	10,419.				
1	258,510.	10,419.				
1	258,348.	10,299.				
1	258,348.	10,299.				
1	258,298.	10,186.				
100	26,750,735.	1,637,249.	12,396.	86,515.	32,136.51	1,966.88
100	26,712,443.	1,641,828.	13,236.	84,347.	37,685.00	2,316.23

Mem Usage	Max CPU	MySQL Max CPU	MySQL Avg Mem
0.007	0.02		
0.006	0.04	0.02	0.012
0.024	0.02	0.02	0.015

Broken World	Technical Testing Data					
Client						
CPU	Intel Core i7 2.3 GHz					
RAM	8 GB 1333 MHz DDR3					
OS	Mac OSX 10.6.8					
Graphics Card	AMD Radeon HD 6750M					
Login Screen	Login & Level Load					
FPS	Total TCP Up (Bytes)	Total TCP Down (Bytes)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
188.4927254	27,178.	285,623.	3106	70,172.	2,339.25	24,584.04
181.1516932	27,970.	285,689.	3370	69,817.	2,525.03	25,790.92
186.9559099	28,115.	285,635.	3657	68,582.	1,699.33	17,264.37
186.1736746	27,224.	285,689.	2965	60,920.	2,407.29	25,262.08
179.2647163	27,377.	285,635.	3172	68,861.	2,174.86	22,691.12
Tile Loading						
Total TCP Up (Bytes)	Total TCP Down (Bytes)	Lag Time (ms)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
5,260.	129,372.	2,653.	2,557.00	65,024.00	1,982.64	48,763.98
4,773.	87,682.	1,927.	3,269.00	64,023.00	2,476.37	45,491.97
5,325.	129,426.	1,775.	3,359.00	86,127.00	3,000.75	72,934.22
4,401.	87,682.	1,794.	2,822.00	63,957.00	2,453.23	48,876.20
6,303.	129,462.	1,850.	3,998.00	86,061.00	3,406.94	69,977.61
Normal Play						
# Headless Clients	FPS	Total TCP Up (Bytes)	Total TCP Down (Bytes)	Total UDP Up (Bytes)	Avg TCP Up (Bytes/sec)	Total UDP Up (Bytes)
0	136.0001075	0	0	0	0	0
0	66.33146691	0	0	0	0	0
0	241.3856605	0	0	0	0	0
0	79.50284536	0	0	0	0	0
0	243.2217266	0	0	0	0	0
1	130.2702	0.	0.	0	0	46,136.
1	163.3581	0.	0.	0	0	45,188.
1	120.4662	0.	0.	0	0	45,899.
1	105.8993	0.	0.	0	0	45,741.
1	70.30034	0.	0.	0	0	46,294.
60		0.	0.	0	0	2,853,480.
60		0.	0.	0	0	2,834,520.
100		0.	0.	0	0	3,807,405.
100		0.	0.	0	0	4,764,700.
	Total UDP Down (Bytes)	Max UDP Up (Bytes/sec)	Max UDP Down (Bytes/sec)	Avg UDP Up (Bytes/sec)	Avg UDP Down (Bytes/sec)	
0	0	0	0	0	0	
0	0	0	0	0	0	
0	0	0	0	0	0	
0	0	0	0	0	0	
0	0	0	0	0	0	
1	46,847.	948	1,027.	777.48	789.46	
1	46,768.	869	869	763.55	790.24	
1	46,768.	869	869	775.45	790.13	
1	46,610.	790	869	776.59	791.35	
1	46,531.	869	869	779.90	783.89	
60	2,830,728.	52140	81528	47,350.66	46,973.11	
60	2,805,290.	52140	48585	47,477.82	46,988.22	
100	3,812,303.	76788	66123	63,600.31	63,682.13	
100	4,741,185.	94800	81212	79,078.13	78,687.86	

Item Crafting

Total TCP Up (Bytes)	Total TCP Down (Bytes)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
801.	810.	801	810	2,257.30	2,282.66
809.	818.	809	818	2,279.84	2,305.21
561.	720.	561	720	559.78	718.44
936.	15,782.	936.	15,782.	785.04	13,236.71
801.	810.	801	810	2,270.79	2,296.30

Building Placement

Total TCP Up (Bytes)	Total TCP Down (Bytes)	Max TCP Up (Bytes/sec)	Max TCP Down (Bytes/sec)	Avg TCP Up (Bytes/sec)	Avg TCP Down (Bytes/sec)
566	579	566	579	2,287.95	2,340.50
566	579	566	579	2,255.24	2,307.04
566	579	566	579	2,286.63	2,339.15
416	513	416	513	478.46	590.03
566	579	566	579	2,339.05	2,392.77

Polygon Test

Mesh	Verts	< 60 FPS	< 30 FPS
Tent	2402	350.	750.
Female Player	15556	50.	100.
Rock	40	2,100.	4,500.
Alder	1733	1,500.	2,650.
Transmission Tower	1099	2,650.	5,500.

10.5 User Study Survey

Broken World User Study

Instructions

Please play Broken World for 15 minutes, and then answer the questions on the questionnaire. If you wish to play more after that, feel free to do so!

NOTE: We make no guarantees about the security of the system. When creating an account, please do not use a personal password!

Controls:

W,A,S,D & Mouse – Move

Right Click – Interact

P – Use Active Object

C – Crafting Menu

I – Inventory Menu

X – Chat Window

E – Character Modification (Must Restart to have effect)

Esc – Quit

Broken World User Study

Questionnaire

Demographic Information:

This section is optional. If you don't want to complete it, leave it blank.

Age:_____ Gender: _____ Major (or profession): _____

Questions:

1. What did you do during your play session? Did you develop a goal?

2. What, if anything, would have been nice to know before starting the game?

3. What did you like about the art style of the game?

4. What did you dislike about the art style of the game?

5. What did you like about the game overall?

6. What did you dislike about the game overall?

8. Did you find any bugs? If so, what were they?