Browser-based Online Multiplayer Roleplaying Game

Final Report for CS39440 Major Project

Author: David Field (dvf9@aber.ac.uk)

Supervisor: Dr. Hannah Dee (hmd1@aber.ac.uk)

April 18, 2014

Version: 0.1.1 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature .....

Date .....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature .....

Date .....

Acknowledgements

I am grateful to...

I'd like to thank...

Abstract

Include an abstract for your project. This should be no more than 300 words.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

Games are interesting projects to take on; they have a history of being difficult to make and pushing technology to its limits. There are a great many systems and features that can be included in them—AI, physics, graphics, audio, UI, multiplayer and more—and a great many ways to implement each, from simple to very complicated depending on the needs of the project.

Games also have a history of aiming for too many of these features in too little time. In most cases, every one of the features thought up would enhance the final product in some way, from a significant improvement that changes the way the game is played to a minor enhancement that makes things just a little more pleasant for the user. Many of these features may be considered mandatory for the game to be worth making at all. For example, a single-player chess game would probably not be very good if there were no AI to play against.

It is not just players who require features, however; developers of games need tools to implement the game design and a good engine to hang the design off of. A lot of game projects build custom tools that let developers and designers implement things quickly. Many game engines even provide methods for scripting and modding them after their release to players.

It is clear that, given the sheer enormity of the possible things that can be put into any one game, there is not enough time in this project to implement even half of them without a great deal of previous experience and skill. Every one of the major systems mentioned can be extremely complicated, requiring a lot of research, time and effort to make them work.

This chapter will discuss what the project is, why it was worth taking on and how a minimal system was devised that would satisfy enough of the game design requirements to be playable but also be implementable in the time given.

1.1 Background

1.1.1 The Project

Before discussing the decisions made about what was doable and why it was interesting, it's useful to know what the project actually is.

The name of the project—*Browser-based Online Multiplayer Roleplaying Game*—gives a relatively good hint of the nature of the game. "Browser-based" and "multiplayer" are fairly self-evident in meaning: multiple people play together in a game hosted in the browser. "Roleplaying game" is more ambiguous. In this case, it refers to a game in the style of the classic tabletop roleplaying game *Dungeons & Dragons*.

In the context of the project that meant the following things: Firstly, there needed to be two types of players—regular players and a Game Master. A regular player plays the game as a character inhabiting the world they happen to be in. For example, they may be a dwarf in a fantasy kingdom, or a space marine on a futuristic space station.

The Game Master is a player responsible for building the world, telling the story and controlling characters that aren't controlled by the other players (known as Non-Player Characters or NPCs). Traditionally, the Game Master would also be responsible for enforcing the rules of the world. However, in this project the game was to be responsible for that instead, with the Game Master given the option of overriding or changing the rules if he or she wished to do so.

Combat is the most obvious area where the game enforcing the rules comes into effect. Combat is turn-based, with players put onto a grid and given limits on the distance they can move and number of actions they can perform in each turn. When they attempt to do something—such as attack another character or creature or escape from a trap—they have to roll dice, the result of which decides whether they were successful or not, and how well they succeeded or failed. As an example, a player failing to attack a creature with their sword could simply miss, or they could throw the sword away accidentally, depending on the result of the dice roll.

In the above scenario the dice rolls would be simulated by the game, rather than physical dice being used by the players. The game will also decide whether or not an action was valid in the first place. A player hoping to attack a creature with their sword would be unable to do so if the creature was too far away from them.

The design also called for interactions outside of combat. For instance, a player might be faced with a locked door. To get through a player could attempt to use a key they found or, alternatively, they could attempt to bash the door open with an item, such as an axe, or even their bare hands.

There are a few specific implementation details as well. Apart from the game being played via a browser, there needed to be graphics and those graphics needed to be 2D isometric tiles. Further, the server was to be written in Python with the goal of gaining experience in the language.

With this overview of the game—a full account of the original game design can be found in Appendix C—it is now important to answer why the project was worth doing.

1.1.2 Why Make The Game?

The first answer to this is that games are interesting in general. Most obviously, the final product of a game is (hopefully) something fun to play with appeal to a wide range of people. More relevant to the context of a project, however, is that games are interesting from a software perspective.

Games are made up of a lot of different parts, each one potentially being difficult to implement by itself. In this game, the most challenging individual parts are graphics and multiplayer. More important than just the individual parts, however, is making sure they integrate properly. In most cases the game needs to share data between these different pieces—game logic needs to know what an object is doing so it can perform game functions on it; the renderer needs to know what the object is doing so that it can be drawn to the screen correctly; the networking part needs to know what the object is doing so that it can forward on any relevant information to the server.

The game also offers a lot of extendibility. Given more time, more features can be added. Each feature, and fitting it together, offers a lot of potential for learning as well. For example, an extra feature could be AI, which is an interesting area in itself that offers a lot of opportunity to learn something new.

1.1.3 Research

With the previous questions answered some research needed to be done in order to gain an understanding of what was plausible in the time allowed for the project.

The first research step performed was to find information on games written in JavaScipt for the browser. The first thing to come up here was Mozilla's *BrowserQuest* [8, 16], a multiplayer role-playing game created in 2012 to show off the capabilities of modern browsers. This initially looked like a good find because it is open source [7]. However, it unfortunately proved to be fairly useless as the source code is large and, without technical documentation or a previous knowledge of game design, it is difficult to comprehend.

Moving on from that, another game was found called *Roll20* [1]. This was an interesting find, showing that it was plausible to create the game in the first place but also that the game type has some audience. However, there are key differences between *Roll20* and this project: namely, *Roll20* is a game engine, supporting whatever game rules you care to enter into it; this project is an actual game. Beyond this there was little to gain except ideas for more extra features, none of which would be realistic to include in the project time frame.

Attempts were made to find books, tutorials and guides covering the creation of HTML5 browser games. This proved to be harder than expected, primarily because most information on browser games is related to creating them in Adobe Flash with the ActionScript language. The materials that do exist for HTML5 games are generally lacking. The best resources found—within budget, at least—consisted of two books [12, 14], which proved to be either too long to read in their entirety to get the useful information, or too simplistic in game design model to be used as reference.

There were some online resources which proved somewhat useful. Initial development in the project was started by following a tutorial [11] that allowed initial understanding of how HTML5

games are hooked into the browser and the basics of creating tile-maps (which will be discussed in more detail in Chapter 3). The code was all replaced but it was a useful start.

Another online resource which proved to be very useful was a tutorial on how to render isometric tiles in games in a simple way [4]. This tutorial was referenced heavily in early development to get the renderer up and running.

Finally, an online book detailing common game design patterns [10]—as well as how to use some traditional design patterns in games—proved to be extremely useful for creating the overall structure of the game engine.

1.2 Analysis

With the knowledge that the scope of games can expand to encompass a ridiculous area, and that everything within that scope is likely to be time consuming to implement, it is important to define what features are absolutely mandatory for the game to be worth making, and figure out how long it might take to implement them. This section provides a discussion of how the minimal system was decided upon, with the list of features in the minimal system being available in Appendix D.

1.2.1 Necessary Features

The first of the necessary features is graphics. Graphics are the player's view into the world, letting them see the state of the game and figure out what their input needs to be. Without this view the only way to see into the world would be to debug the code as it was playing, which is not a very efficient way to play a game and the constant pausing would also break the game's timing, which is vital for games to operate correctly, particularly in a multiplayer setting. Of course, it is reasonable to operate a game with a textual interface. However, the game design specified that graphics be implemented and that those graphics be isometric tiles.

Players will need a way to interact with the game. There are two things that affect how this works: the Game Master and the isometric graphics. The Game Master needs to be able to select different characters to control and, in theory, create maps for the other players to play the game in. Using simple keyboard controls probably wouldn't work very well in this case. Isometric graphics also cause issues with keyboard controls. Players attempting to move their character using, for example, the arrow keys on the keyboard might find themselves confused as to which direction their character will move.

The solution to both of these is mouse-based interaction. The Game Master will be able to click to select characters he wants to control and place tiles and items into the world, and movement will be a case of clicking on the desired location and letting the character move there by itself, so that there is no confusion about direction.

The disadvantage to this is that mouse interaction is more complicated to handle than keyboard interaction. With a keyboard you check whether a key is pressed and perform an action based on that. With a mouse you have to work out the location of the cursor and what else is in that

location. Further, mouse-based movement requires pathfinding, which ultimately adds another required feature.

The game also requires multiplayer. All interactions in the game happen between players, be they regular players or the Game Master. Without any way for them to interact the game is relatively pointless to play—all you'd get as a player is being able to walk around a map with your character.

Multiplayer implies a few things. The first is that there needs to be a way of clients communicating with some remote version of the game. This could, in theory, be a peer to peer connection. However, there are problems with this. The first is that peer to peer is not such an easy thing to support technologically with a browser, as the web operates on a client-server model.

The second problem with peer to peer is that there is no trustworthy version of the game. All the players need to have a view into the same world and the underlying data needs to be consistent between them or problems would occur. There is nothing to stop any one of the players from modifying their client to perform actions they shouldn't be able to do, and it being run in a browser using JavaScript makes this even more of an issue as all browsers come with easy access to JavaScript debuggers. A single client could be made the authoritative version (with the Game Master being the most logical choice here). However, if that client is tampered with it would affect the game for everyone.

A server solves both of the issues that peer to peer represents. It is, of course, more natural for a browser to operate in a client-server manner. A server is also far more trustworthy than any individual client, and can be used as an authoritative base for the game so that, even if a player tampers with their client, the other players won't be affected.

The disadvantage to a server is that it requires resources and time to host and keep running. The more people who decide to play the game, the more server power is required. If the servers went down, no one would be able to play at all.

These things represent technical requirements as much as game design requirements. Their existence requires consideration in designing the structure of the game code. However, there are some other features required which could be considered purely game design issues.

The first of these is combat. In this case, combat is what makes the game a game. Without it players really only get to walk around a world together and look at how pretty it may or may not be, depending on the quality of the graphics used.

The full design of combat is fairly comprehensive and involves a lot of possible things. In the game design overview the idea of levels of success or failure was introduced, with things like throwing away a weapon if a character fails really badly on a dice roll. The game checking for valid options was also required, with the example given being whether a player was close enough to hit another character with a weapon. However, in the full game design there is also the option for ranged attacks using projectiles.

These options all make the game a lot more interesting. However, things like being able to throw away a weapon, or even just having levels of failure, require extra implementation time. It was decided that the project only needed to implement a very basic version of combat with other things considered extras to be done given more time. This basic version of combat was binary

success or failure based on dice rolls; only 'mêlée' combat, requiring characters to be next to each other; and no items would be implemented in the base game, so no weapons and all combat would essentially be unarmed.

This would allow players to fight in the game, which is an important aspect of the game design, without creating too much work adding extra features that need a lot of development time and testing, such as projectiles or the use of items.

The existence of combat also implies the existence of attributes in characters and possibly other entities that allow them to interact in some way. In this case, the design specification calls explicitly for attributes called Hit Points (HP) and Mana, which represent the number of points of damage an entity can take before it is destroyed and the number of points it has to spend on magical spells respectively. Because magic would not be implemented in the minimal system, only HP and a way to affect it need to be created.

There should also be some way of interacting with the world outside of combat. Ideally these interactions would be almost limitless and allow the Game Master to specify even more based on items they add to the game. However, this is also a huge task. Because of the time consuming nature of adding interactions to the game only a few very basic ones should be included into the minimal system.

Finally, the last feature that is considered to be required in a minimal version of the game is a chat system. This is important because the game is multiplayer and operates through players interacting with each other. It is unreasonable to rely on players to have a third party tool available for communication, so some way for them to communicate must be provided by the game. In this case, a simple chat system is the most logical choice as it is not too difficult to implement. The basic version of the chat will be global for the game but an enhanced version could allow local chatting between players who are in a specific area or private messaging between the Game Master and a player if the GM wants to do something special.

1.2.2 Unnecessary Extra Features

Given either extra time in the project or unexpected time available at the end of the project it is useful to have a list of features that could be added. Of course, ultimately it would be nice to include everything that the original design specification calls for, but it is also unreasonable to expect that enough time would ever be found to implement all of them. Therefore, the extra features should be given priority based on how worthwhile they are to implement on top of the minimal system.

Improving combat gains the highest priority because it is the most important part of the game outside of letting the players communicate with each other. The addition of items would come first, as players generally expect to be able to equip weaponry and defences like armour. After that, ranged attacks, such as using a bow. Magic would be the final part added to the game because it requires extra attributes to operate correctly and could have a wide range of effects, such as teleporting a user or causing a fire.

The next most important thing is to have a much wider range of interactions outside of combat. Supporting the ability for players to knock down doors or use a key on them, as was given as an

example in the game design overview, enhances the game considerably from a player's point of view, opening up a lot of gameplay options. It may also be reasonable to include the use of weapons and magic outside of a combat context to interact with things, such as perhaps using magic to set a door on fire.

There are plenty of other features in the full design that could be implemented, such as voice chat or the ability for players to upload custom graphics. However, the extra features listed here represent a large enough amount of work that it isn't necessary to prioritise everything else.

1.3 Process

With a minimal system decided on the task becomes to decide how the time available is to be divided up between each feature and the systems necessary to support them.

The original process chosen was a custom variation of SCRUM, with many of the roles and tools taken out due to the project having just a one-person team. The primary reason for this was familiarity with the process but it also potentially had advantages beyond this.

The project was started with a clear idea of what was necessary to complete it. More importantly, there was a belief that a clear idea was had of the complexity of each feature and how difficult it would be to implement.

The time was divided up into week-long "sprints". Each sprint would be assigned a certain number of stories, such as the implementation of graphics stating something along the lines of "As a user I want to be able to see the state of the game from an isometric 2D viewpoint." These stories would be broken up into the tasks necessary to complete the story.

For discrete features like this, SCRUM proved to be a decent choice. However, problems became apparent after the first few sprints were completed. The most obvious problem was that implementation time was often longer than expected. This is not entirely uncommon when using an agile methodology like SCRUM—further experience in the task usually brings the estimations closer into line with reality—so the problem was ignored at first. The second problem was that no real consideration had been given to how the various features fit together. Previous research had given some insight into how features like isometric rendering worked but no clear picture of how to link the rendering with the other parts of the game.

Further, as implementation proceeded, new and better ways of doing certain things emerged to solve problems that had no been foreseen. For example, rendering was originally very simple and was implemented almost entirely as the sprint had laid it out. However, it actually proved to be too simple and so modifications had to be made. This issue continued and eventually too much time was spent changing the tasks in the sprint to match up with what reality demanded than being able to follow them as they were planned.

To solve this issue the process was switched to something closer to Feature Driven Development (FDD).

Chapter 2

Design

You should concentrate on the more important aspects of the design. It is essential that an overview is presented before going into detail. As well as describing the design adopted it must also explain what other designs were considered and why they were rejected.

The design should describe what you expected to do, and might also explain areas that you had to revise after some investigation.

Typically, for an object-oriented design, the discussion will focus on the choice of objects and classes and the allocation of methods to classes. The use made of reusable components should be described and their source referenced. Particularly important decisions concerning data structures usually affect the architecture of a system and so should be described here.

How much material you include on detailed design and implementation will depend very much on the nature of the project. It should not be padded out. Think about the significant aspects of your system. For example, describe the design of the user interface if it is a critical aspect of your system, or provide detail about methods and data structures that are not trivial. Do not spend time on long lists of trivial items and repetitive descriptions. If in doubt about what is appropriate, speak to your supervisor.

You should also identify any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.

Some example sub-sections may be as follows, but the specific sections are for you to define.

2.1 Overall Architecture

2.2 Some detailed design

2.2.1 Even more detail

2.3 User Interface

2.4 Other relevant sections

Chapter 3

Implementation

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on ☒real users☒? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1 Overall Approach to Testing

4.2 Automated Testing

4.2.1 Unit Tests

4.2.2 User Interface Testing

4.2.3 Stress Testing

4.2.4 Other types of testing

4.3 Integration Testing

4.4 User Testing

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [3]. The library is released using the Apache License [2]. This library was used without modification.

Appendix B

Code samples

2.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [13].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
  /*-------------------------------------------------*/
  /* Minimum Standard Random Number Generator        */
  /* Taken from Numerical recipies in C              */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity   */
  /* Always call with negative number to initialise  */
  /*-------------------------------------------------*/
```

```
int j;
long k;
static long idum2=123456789;
static long iy=0;
static long iv[NTAB];
double temp;

if (*idum <=0)
{
  if (-(*idum) < 1)
  {
    *idum = 1;
  }else
  {
    *idum = -(*idum);
  }
  idum2=(*idum);
  for (j=NTAB+7;j>=0;j--)
  {
    k = (*idum)/IQ1;
    *idum = IA1 *(*idum-k*IQ1) - IR1*k;
    if (*idum < 0)
    {
      *idum += IM1;
    }
    if (j < NTAB)
    {
      iv[j] = *idum;
    }
  }
  iy = iv[0];
}
k = (*idum)/IQ1;
*idum = IA1*(*idum-k*IQ1) - IR1*k;
if (*idum < 0)
{
  *idum += IM1;
}
k = (idum2)/IQ2;
idum2 = IA2*(idum2-k*IQ2) - IR2*k;
if (idum2 < 0)
{
```

```
    idum2 += IM2;
  }
  j = iy/NDIV;
  iy=iv[j] - idum2;
  iv[j] = *idum;
  if (iy < 1)
  {
    iy += IMM1;
  }
  if ((temp=AM*iy) > RNMX)
  {
    return RNMX;
  }else
  {
    return temp;
  }
}
```

Appendix C

Game Design Specification

An online, multiplayer roleplaying game similar to *Dungeons & Dragons* hosted in the browser. The game will have a Game Master and several players in a group. The players move around the world and interact with it and the people/creatures in it and the Game Master guides the players, sets rules and manages the experience. They will be able to manipulate the game world and do things players can't do. For example, they could teleport a player to another location, spawn new monsters or change the weather.

It will include isometric graphics and aim to be as permissive as possible in what players are allowed to do (for example, interact with the environment using items they have, such as burning down a house with a torch). This means that there will be several systems, including combat, movement, character/creature creation, interacting with other characters (players and non-players, with chat as well as game system rules) and the environment (things like objects having heat and a burning point, in the fire example).

The nature of the design means that the project is open-ended, allowing for a basic game with a few simple systems and a single dungeon and character type to a game with several systems, full environmental interaction, a map and character editor and the ability to upload your own artwork.

It will require a server to manage the clients and let them interact, relay chat and act as an authority in the game world to keep clients in sync. It will also have to keep track of where players are in the world and whether they can see and interact with each other. There will be a graphical front end client that the players use and, in theory, multiple different clients could be made, such as a mobile app.

Appendix D

Minimal System

BUTTS BUTTS BUTTS BUTTS BUTTS BUTTS BUTTS

Annotated Bibliography

[1] "Roll20, online tabletop game engine." [Online]. Available: http://roll20.net/

Roll20, an online game engine for playing tabletop games on your computer.

[2] Apache Software Foundation, "Apache License, Version 2.0," http://www.apache.org/licenses/LICENSE-2.0, 2004.

This is my annotation. I should add in a description here.

[3] ——, "Apache POI - the Java API for Microsoft Documents," http://poi.apache.org, 2014.

This is my annotation. I should add in a description here.

[4] J. Bose, *Creating Isometric Worlds: A Primer for Game Developers*, May 2013. [Online]. Available: http://gamedevelopment.tutsplus.com/tutorials/creating-isometric-worlds-a-primer-for-game-developers--gamedev-6511

Tutorial series used to understand the basics of rendering isometric graphics.

[5] H. M. Dee and D. C. Hogg, "Navigational strategies in behaviour modelling," *Artificial Intelligence*, vol. 173(2), pp. 329–342, 2009.

This is my annotation. I should add in a description here.

[6] S. Duckworth, "A picture of a kitten at Hellifield Peel," http://www.geograph.org.uk/photo/640959, 2007, copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.

This is my annotation. I should add in a description here.

[7] Mozilla, L. Workshop, F. Lecollinet, and G. Lecollinet, "BrowserQuest Source Code," GitHub, 2012. [Online]. Available: http://github.com/mozilla/BrowserQuest

BrowserQuest source code.

[8] ——, "Mozilla's BrowserQuest Game," 2012. [Online]. Available: http://browserquest.mozilla.org/

The BrowserQuest game itself.

[9] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, "Don't touch me, I'm fine: Robot autonomy using an artificial innate immune system," in *Proceedings of the 5th International Conference on Artificial Immune Systems*. Springer, 2006, pp. 349–361.

This paper...

[10] B. Nystrom. Game Programming Patterns. Online. [Online]. Available: http://gameprogrammingpatterns.com/index.html

Online book detailing common game design patterns, as well as examples on how to use more general design patterns in the context of games.

[11] N. G. Obbink, *Javascript tile engine tutorials*, 2012. [Online]. Available: http://nielsgrootobbink.com/wokflok/jte/

Series of tutorials going through the creation of a simple RPG in JavaScript.

[12] M. A. Pagella, *Making Isometric Social Real-Time Games with HTML5, textsccss3, and Javascript*. O'Reilly Media, Sept. 2011. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/1449304753 1449304753.

A book about creating isometric games in the browser using textschtml5 and JavaScript

[13] W. Press *et al.*, *Numerical recipes in C*. Cambridge University Press Cambridge, 1992, pp. 349–361.

This is my annotation. I can add in comments that are in **bold** and *italics and then other content.*

[14] P. Rettig, *Professional HTML5 Mobile Game Development*, 1st ed. Wrox, Aug. 2012. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/B0094P2TU6

Book on creating browser games using HTML5 and JavaScript

[15] Various, "Fail blog," http://www.failblog.org/, Aug. 2011, accessed August 2011.

This is my annotation. I should add in a description here.

[16] L. Workshop. (2012) Little Workshop Information on BrowserQuest. [Online]. Available: http://www.littleworkshop.fr/browserquest.html

Web page by the authors of the BrowserQuest game giving some information on it.