

# Requirements and Analysis Document for Project “Programmers United Networking Game”

**Version: 1.0**

**Date: 2016-03-24**

**Author: Martin Sixtensson**

## **Contents:**

### **1. Introduction**

1.1 Purpose of application.....	2
1.2 General characteristics of application.....	2
1.3 Scope of application.....	2
1.4 Objectives and success criteria of the project.....	2
1.5 Definitions, acronyms and abbreviations.....	2

### **2. Requirements**

2.1 Functional requirements.....	3
2.2 Non-functional requirements.....	3
2.2.1 Usability.....	4
2.2.2 Reliability.....	4
2.2.3 Performance.....	4
2.2.4 Supportability.....	4
2.2.5 Implementation.....	4
2.2.6 Packaging and installation.....	4
2.2.7 Legal.....	4
2.3 Application models	
2.3.1 Use case model.....	4
2.3.2 Use cases priority.....	4
2.3.3 Domain model.....	5
2.3.4 User interface.....	5
2.4 References.....	5

# 1 Introduction

## 1.1 Purpose of application

The purpose of the project is to create a so called “zombie-shooter” game. The game should feature multiplayer over a network with up to four players.

## 1.2 General characteristics of application

The application will be a PC application consisting of one server and one client portion. The application will have a simple GUI for things done outside of the actual game. The game itself will consist of a 2D gamespace. A player can either join or host a game. Up to four players should be able to join a game on different computers. The application will score the player depending on how well the player does according to the game. The application will end or restart itself when the players lose according to the game rules or the application itself is canceled. There is no situation a player can win, a score is only given when the players have lost. The game will feature a so called map editor allowing players to create their own playable maps. The mapeditor will use a simple GUI.

## 1.3 Scope of application

“Programming United Network Game” is game in which players must work together to defend a house from hordes of zombies or enemies controlled by the computer. The game takes place in a 2D gamespace to which four players in total can connect and play a game. The game starts when all players are ready and the enemies will start appearing. The players must then aim and shoot at the enemies to destroy them. The wave is ended when all enemies of a wave are destroyed. The game itself, however, only ends when all players are killed by the enemies. The enemies kill the players by getting close to them and attacking them. The players are scored depending on how long they survived and how many enemies they killed. The application will not save any data from play-sessions.

## 1.4 Objectives and success criteria of the project

The following objectives determine the success of the project:

- The development of a working client and server for the game
- The design of a working GUI to host and/or join a game
- The design of a 2D gamespace
- The creation of a controllable player
- The creation of enemies controlled by the computer
- The creation of a tool to create maps for the game

## 1.5 Definitions, acronyms and abbreviations

**NPC** Non-player character. Game characters not controlled by the players.

**AI** Artificial Intelligence. The functions that control the enemies.

**Host** The person or computer who hosts the game. Hosting in this instance means having the server running on the selected computer.

**Client** A client is the part of the program that connects and talks to the server. A player could be considered a client.

**Server** Controls most of the of the game's different states. Players can create a server to host their own game or a player can join an existing server to play.

**Frame Rate** How fast the client updates what it sees on the screen. I.e if the the frame rate is 30 frames per second, the client will redraw the position of the enemies and players thirty time per second.

**Sprite** The skin of the moving enemy or player.

**Enemy** Controlled in this case by the computer. The goal in this particular game is to destroy/remove enemies.

**Map** The board or space where the player and enemies can move around.

**Mapeditor** A tool for players to create their own maps for the game.

**Tile** A map is built from tiles. There are different tiles;

- Wall tile; where players cannot move.
- Floor tile; where players are free to move around.
- Spawner tile; where zombies are spawned/created.

**Rate of fire** How fast bullets are created from a certain gun.

## **2 Requirements**

### **2.1 Functional requirements**

**See also:**

*The player(s) should be able to;*

1. Host a new game and select map.
  - a) Select map for the game
2. Join a new or ongoing game.
3. Move around with the character on the gameboard.
4. Shoot their weapon.
5. Buy a new weapon

### **2.2 Non-functional requirements**

**Possible NA (not applicable).**

#### **2.2.1 Usability**

The application should be easy to use. A normal user should be able to use the application with some or little computer knowledge.

There should be an english tutorial for both setting up a game, joining a game, how to play the game and how to use the map editor.

#### **2.2.2 Reliability**

It is very important that the application is reliable. There should not be regular disconnects from the host/server. A game with all four players should be able to play the game reliably for at least 30 minutes.

#### **2.2.3 Performance**

The game should run at a stable framerate and at least over 30 frames per second. The delay between the different clients and server should be kept as low as possible.

#### **2.2.4 Supportability**

The application needs to be able to run on any Java-running Operating System. The application has to be usable on a screen with a resolution of 800x600 or more.

#### **2.2.5 Implementation**

The application will use the Java environment. All hosts and clients must have JRE installed and configured.

#### **2.2.6 Packaging and installation**

We plan to package the game in a jar file. We will supply a readme file with instructions on how to start the application and how to join or host a game within the application.

### **2.2.7 Legal**

NA

## **2.3 Application models**

### **2.3.1 Use case model**

See appendix for use cases.

### **2.3.2 Use cases priority**

1. Host game
2. Join game
3. Move
4. Shoot
5. Buy weapon

### **2.3.3 Domain model**

See appendix.

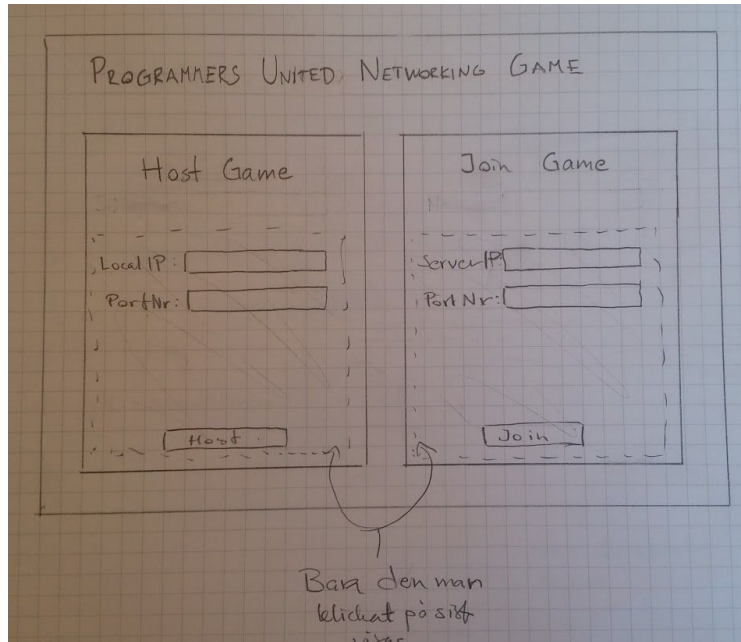
### **2.3.4 User interface**

Application will use separate GUI to launch into game. See appendix for screens.

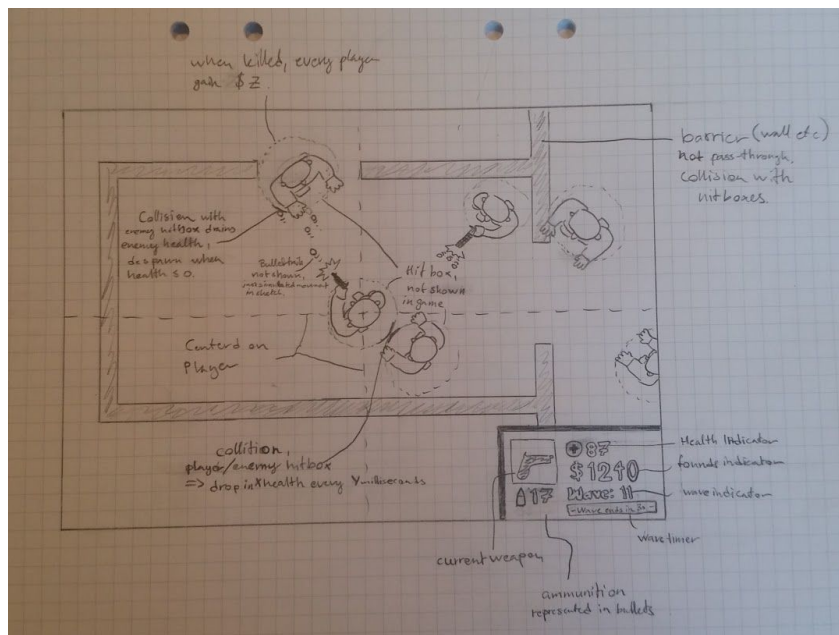
## **2.4 References**

## **APPENDIX**

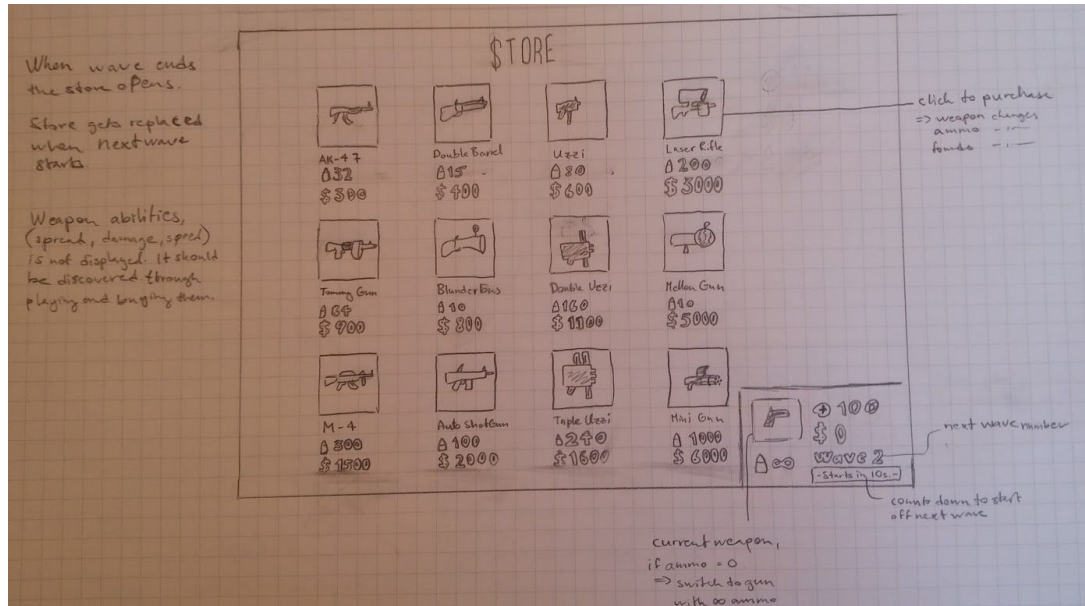
### **GUI**



Initial starting screen

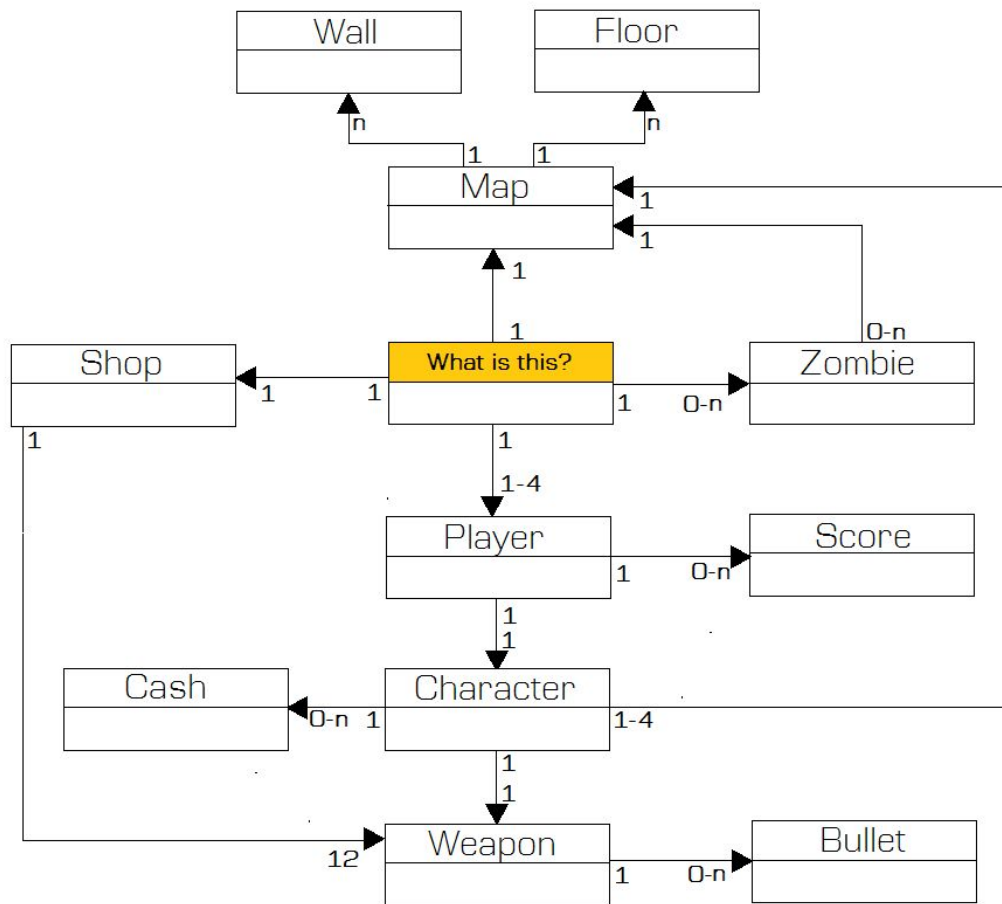


In game view



Store view

## Domain model



## Use cases

### Use Case: Host Game

**Summary:** Player hosts a game which other players can join.

**Priority:** High

**Extends:** -

**Includes:** -

**Participants:** Actual Players

Normal flow of events:

	Actor	System
1	Fills in necessary textfields.	
2	Clicks "Host" button.	
3		Main menu is closed.



		Server is started. Game client is started.
--	--	---

**Alternate flow:**

No alternate flows

**Exceptional flow:**

Flow 3.1 Necessary information is not entered or is wrong.

	Actor	System
3.1		Prompts text is displayed. Prompts player to enter necessary information.

**Use Case: Movement**

**Summary:** Movement of the player character. A game must have started for movement to be enabled.

**Priority:** High

**Extends:**

**Includes:** Collision

**Participators:** Actual Players

**Normal flow of events:**

	Actor	System
1	Moves character with movement keys	
2		Player character is moved in specified direction.

**Alternate flows:**

Flow 2.1 Player moves into zombie:

	Actor	System
2.1.1		Player takes damage. Player health is reduced. Damage animation is played.

Flow 2.1.2 Player health is reduced below zero.

	Actor	System
--	-------	--------

2.1.2.1		Player character is disabled/killed until current wave ends.
---------	--	--

Flow 2.1.2.2 All other players are dead

	Actor	System
2.1.2.2		Game ends. Prompt to restart or exit is shown.

### Exceptional flows:

Flow 2.2 Player moves into wall or player:

	Actor	System
2.2.1		Player doesn't move in specified direction.

### Use Case: Shoot

**Summary:** Player shoots a weapon which creates a moving projectile.

**Priority:** High

**Extends:**

**Includes:** Collision

**Participants:** Actual Players

### Normal flow of events:

	Actor	System
1	Player clicks the shoot button.	
2		Projectile is created. Projectile is given velocity. Shooting sound is played. Player ammo is reduced.
3		Projectile collides with something. Projectile is removed

### Alternate flow:

Flow 3.1 Projectile collides with wall or boundary.

	Actor	System
3.1.1		Projectile is removed.

Flow 3.2 Projectile collides with zombie.

	Actor	System
3.2.1		Projectile is removed. Enemy takes damage.

		Damage animation is played
--	--	----------------------------

Flow 3.2.1 Enemy's health is below zero

	Actor	System
3.2.1.1		Enemy is removed. Score is updated.

Flow 3.2.1.1 Enemy is last enemy on map.

	Actor	System
3.2.1.1		Wave is ended. Funds are distributed to players.
3.2.1.2		Weapon shop prompt is displayed. 30 second timer is started. See Buy Weapon Use Case

### Exceptional flow:

Flow 2.1 Player has no ammo for current gun.

	Actor	System
2.1.1		Player's current weapon is switched to pistol. No projectile is created.

Flow 2.2 Not enough time has gone by since last shot.

	Actor	System
2.1.1		No projectile is created.

### Use Case: Buy Weapon

**Summary:** Player buys a weapon from the weapon shop

**Priority:** Medium

**Extends:** Shoot

**Includes:** -

**Participators:** Actual Players

Normal flow of events:

	Actor	System
1	Player selects desired weapon from list.	
2	Player clicks "Buy Weapon" button.	
3		Player is given weapon and ammo.
4		30 seconds pass. Game continues.

Alternate flow:

Flow 1.1 Player does not buy weapon.

	Actor	System
1.1		Players is not given a weapon. 30 seconds pass. Game continues.

Exceptional flow:

Flow 2.1 Weapon