Requirements and Analysis Document for NNN

Table of Contents

Version: 1.0

Date 2014-05-25

Author Daniel Jansson, Erik Pihl, Jacob Genander, David Michaëlsson

This version overrides all previous versions.

1 Introduction

1.1 Purpose of application

The purpose of this project is to create a tower defense desktop application. Our primary goal is to create a very basic version of tower defense, and implement more advanced features if time or difficulty are not obstructing our workflow.

1.2 General characteristics of application

The project goal is to develop a 2D Tower defense game - a type of game in which the player tries to kill all enemies moving along a given path. If an enemy reaches the end of the path without being killed, the player loses lives. The player places various towers on the game map, and has the possibility to upgrade them if he/she has enough money, which is earned by killing enemies. The game is over when all the player’s lives are lost.

1.3 Scope of application

The progress of the game will not be saved if one decides to quit the program during a session.
Due to the nature of tower defense games, there will only be a single player mode.
The graphics will be two dimensional.

1.4 Objectives and success criteria of the project

The success criteria is to have a playable game at the project’s deadline. This includes the ability to play at least five waves of enemy attacks in a row. Also, having at least two different kinds towers is necessary. One or more of these towers has an effect associated with it, which may be applied to the enemy it shoots. At least two different kinds of enemies are required.

1.5 Definitions, acronyms and abbreviations

Enemy - Monster that needs to be killed before it reaches the end of the path.

Game over - The player’s game session ends if the player has no more lives. Prompts the user to play the same map again or to select a new map.

Map - A top down representation of a map. Contains a path and an area to build towers at.

Money - Money is used to upgrade or buy towers. The player gets money from killing enemies.

Path - The path that the enemies follow across the map.

Player - The person that plays the game.

Status effect - an effect that changes the status of an enemy, e.g. decrease the speed of the enemy.
Tower - An automated sentry-like object which shoots and damages enemies. Is bought by the player and can be upgraded, these actions require money. Can only be placed in the build area of the map.

2 Requirements

2.1 Functional requirement

Here are the high level functions of the game:

Start game - starting the game by loading a map with a path and enemies.

Exit game - terminating the application.

Choose map - choosing among a number of maps to play on.

Build tower - building a tower in an unoccupied space (not on a path or a tower).

Select tower - making a tower selected and thereby displaying information about it and enabling the player to upgrade it or sell it for money.

Upgrade tower - making a selected tower more powerful for a given amount of money, i.e. by raising its attack damage.

Sell tower - removing a tower form the map. The player receives money.

Next wave - sending in a number of enemies to the path, moving along it.

Pause game - makes the game freeze in its current state.

Player dies - the player loses all its lives and the game is over.

Tower shoots enemy - a tower shoots an enemy and thereby damage it and possibly gives it a status effect.

Enemy dies - an enemy loses all its lives by being shot by towers. The player receives money.

Enemy reaches end of path - an enemy has travelled to the end of the path and the player loses some lives.

2.2 Non-functional requirements

2.2.1 Usability

Since the game is very similar to other games in the Tower defense genre, a user that has played a Tower defense game before will most likely not have any difficulties understanding how to interact with the components that are presented on the screen. The game is designed in such a way that a novice user may safely explore the various interactions possible without the risk of getting stuck in a game state, a restart will reset the whole game.

A novice user should not have to lay down much time on learning how to play the game. Since there are only a few actions available, the game should be somewhat self-explanatory.

2.2.2 Reliability

The system is an offline application and because of that it is always available for the user and does not depend on a server. The user will therefore not be affected by future downtime, like maintenance or updating.

No specific screen resolution is required to play the game, since the application window is scalable. However,the aspect ratio of the resolution is 16:9 and the application window initially starts at a size of 1280x720 pixels, which also is the size the visuals were created for.
If the screen size or resolution is different the application will be scaled and black space will fill any unused screen area.

2.2.3 Performance

Due to the nature of the program, performance is important. Not only should the game be runnable without any prominent bugs or crashes, but it should also run with a generally high frame rate on modern laptops.
Each use case should not take too long since initiation by the user or the system. The game should accommodate one user only. There is really no limit on the maximum amount of

rendered objects but the memory and processing limitations set by java and LibGDX.

2.2.4 Supportability

The project follows the java code convention standard set by Oracle, but also other conventions. The graphics library LibGDX which this project uses, does also affect names and other standards for this project.

The source code is required for maintenance, and maintenance of the project will therefore be limited to those who have access to the source code. There will be no tools developed for maintenance of the project.

2.2.5 Implementation

The Java Runtime Environment needs to be installed on the computer in order for the game to be runnable. Run the TeeDee.jar file to start the application.

2.2.6 Packaging and installation

The application comes as a runnable .jar file. Requires Java Runtime Environment.

2.2.7 Legal

Everything used in system is self made, freeware or have been authorised by owner.

2.3 Application models

The game model is composed of these components:

Money - the currency used throughout the game.

Lives - the variable holding the state of a creature's health.

Path - a number of checkpoints that describe a way to be travelled.

Map - a map which includes a path.

Enemy - a creature that travels on a map along a path. When shot by a tower, its lives are reduced and a status effect may be applied.

Tower - can be bought for money and placed in an unoccupied space on the map (not on a path or another tower). The tower shoots at enemies.

Status effect - an invisible object that towers provide and applies on a shot enemy. A status may affect the enemy i.e. by changing its speed and health over time.

2.3.1 Use case model

UML and a list of UC names (text for all in appendix)

2.3.2 Use cases priority

Here is presented a table of the use cases and their priority:

High Priority	**Middle Priority**	**Low Priority**
Start game	Select map	Pause game
Exit game	Select tower	Toggle speed
Build tower	Upgrade tower	Toggle sound
Next wave	Sell tower	
Player dies	Select difficulty	
Tower shoots enemy		
Enemy dies		

2.3.3 Domain model

UML, possibly some text.

2.3.4 User interface

The user interface consist of mainly three different screens.

The first screen the user would see is an intro or start screen that would only consist of a few buttons, like start and exit game, and would look like any similar application or game.

After the user decides to start the game she would see a screen that gives the user the option to change some settings for the game, for example what map to play and a difficulty.

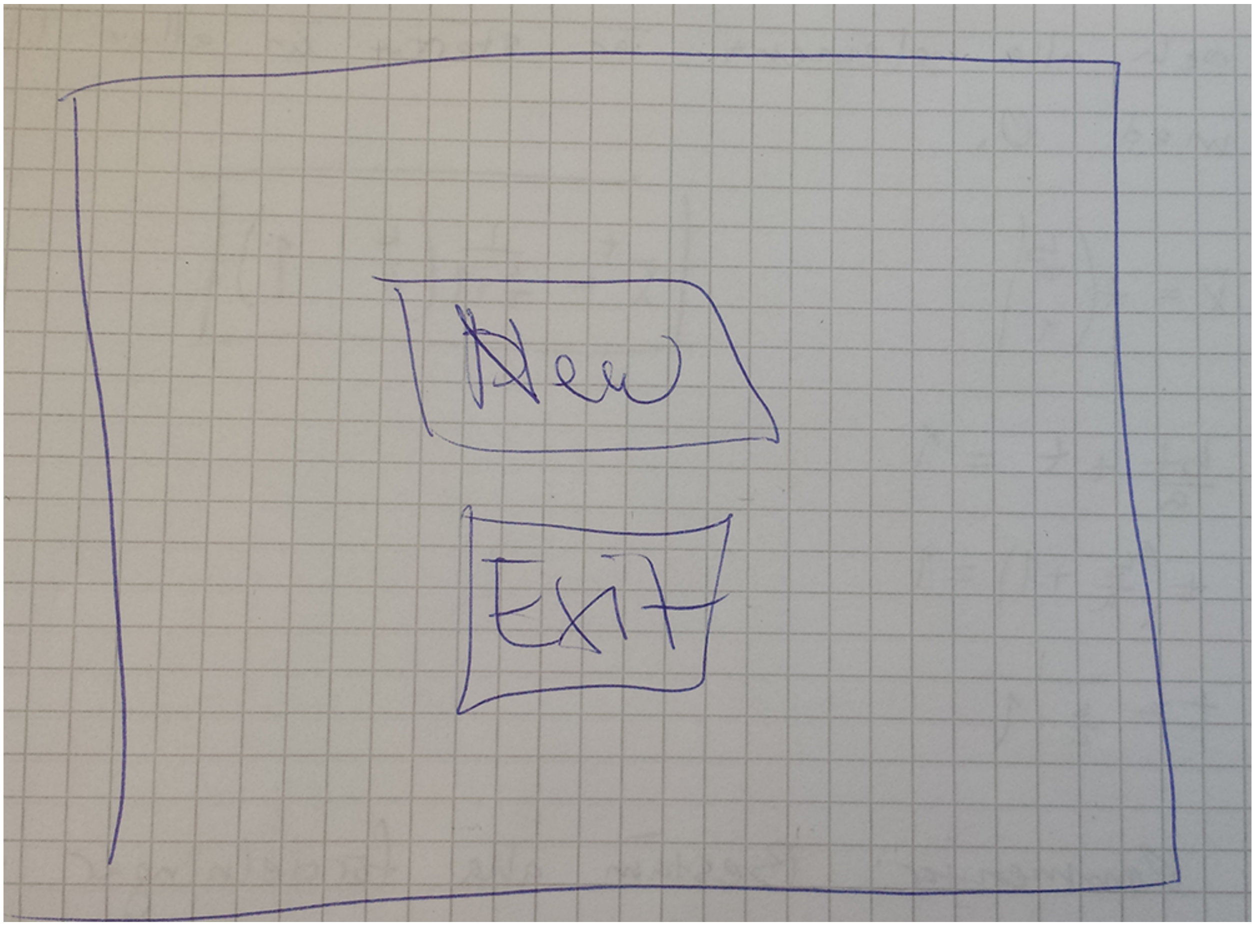
The user would after these steps come to the game screen. This screen will be easy to understand. A user that already used a similar application would recognize the design and also understand how the game works. For a first time user this would still be a clean and easy to understand interface and as described in Usability, the user would be able to try out the different features without the risk of getting stuck.

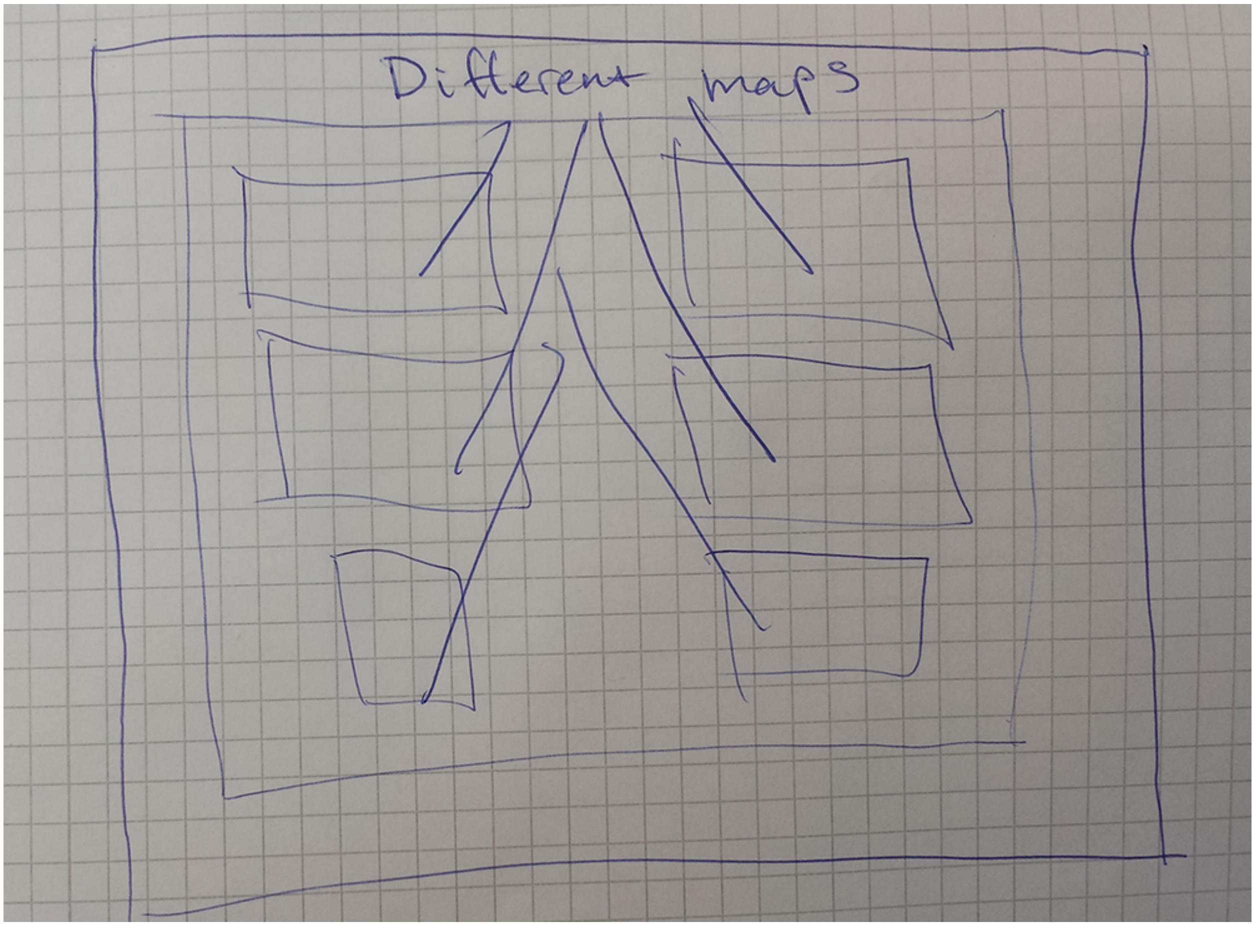
2.4 References

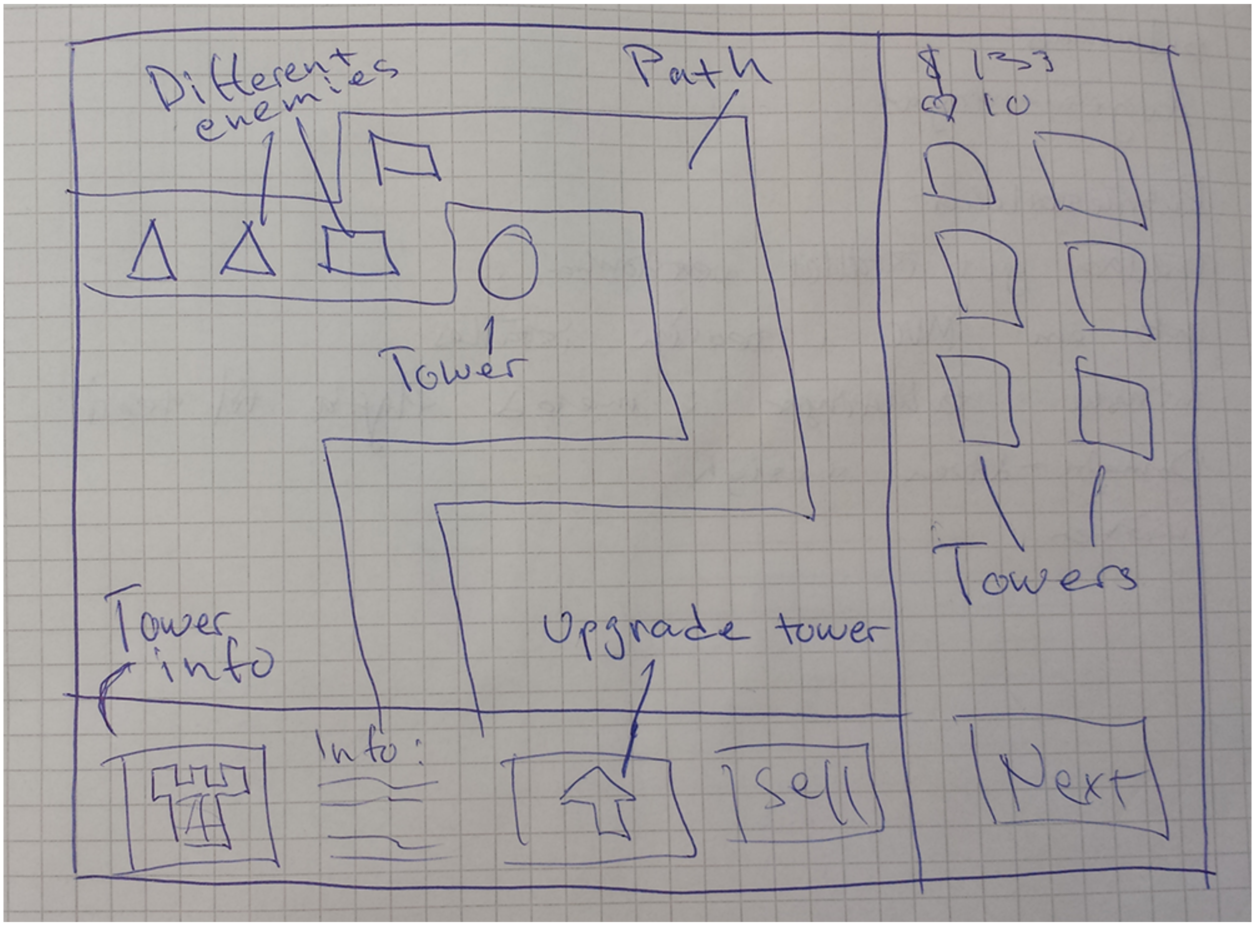
NA

APPENDIX

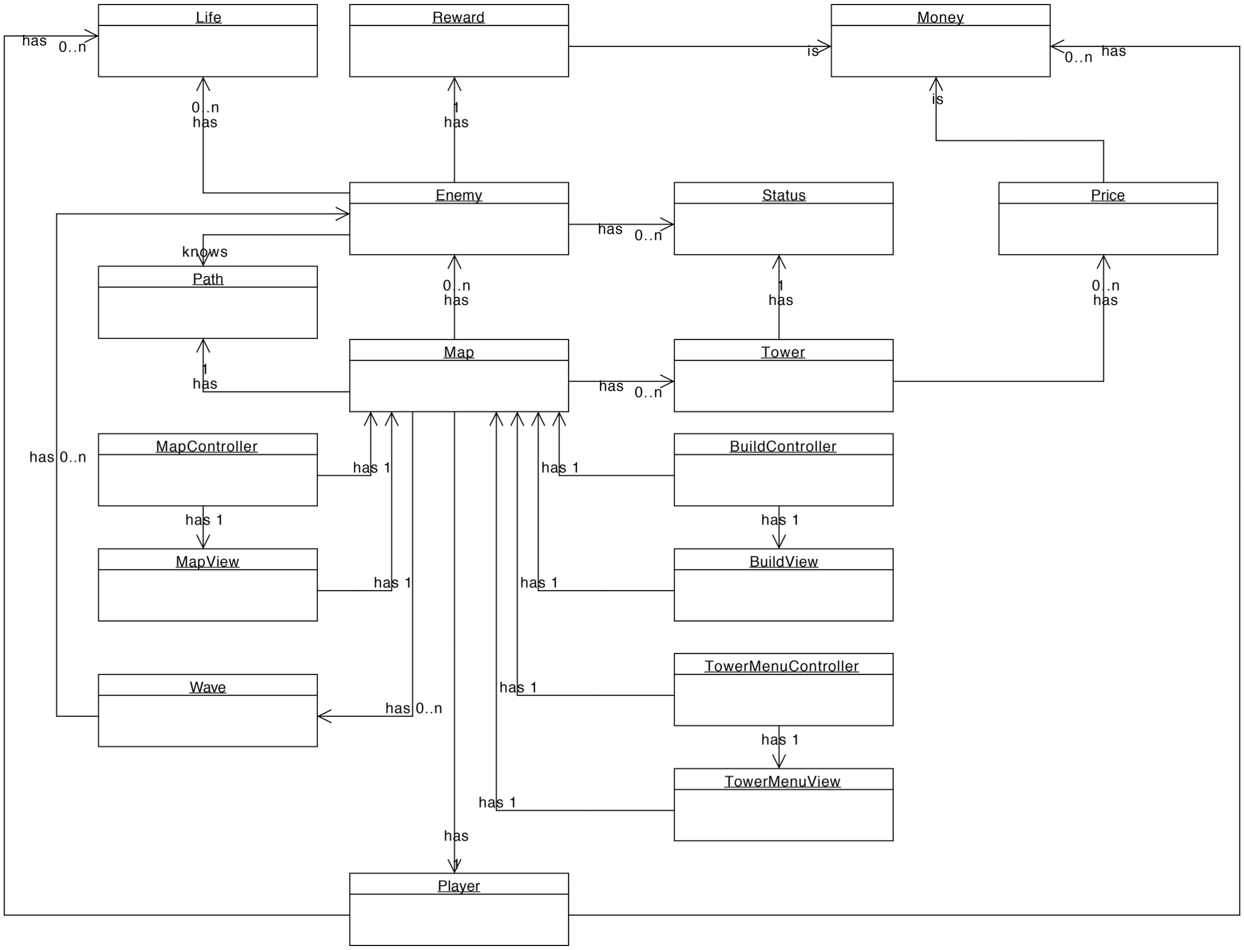
GUI:




Different maps


Different enemies
Path
$ 133
♡ 10
Tower
Towers
Tower info
Info:
Upgrade tower
Sell
Next

Domain model:



Use cases:

Use case: Start game

Summary: The user starts a new game with the previously selected map and begins playing.

Priority: High

Extends: None

Includes: Build tower, Sell towerUpgrade tower, , Pause

Participators: The player.

Normal flow of events

	Actor	**System**
1	Clicks the “Start Game”-button	
2		Displays the map and initializes the game.

Use case: Exit game

Summary: Shuts down the program

Priority: High

Extends: None

Includes: None

Participators: The player.

	Actor	**System**
1	The user press exit button	
2		Shutdown program

Use case: Choose map

Summary: The user chooses a map for the game.

Priority: Mid

Extends: None

Includes: Start game

Participators: The player.

	Actor	**System**
1	The user clicks on a representation of a map	
2		The system loads the selected map

Use case: Build tower

Summary: The player purchase and places a new tower. Deducts money.

Priority: High

Extends: None

Includes: None

Participators: The player

	Actor	**System**
1	User selects a tower from an actionpanel	
2	User clicks on the map	
3		Constructs the tower on the clicked coordinate

4		Deducts player’s money
5		Sets the area tower’s area on the map to unbuildable

Alternate case: Player’s funds are insufficient.

	Actor	**System**
2.1		Displays a message informing the player, that he/her has not enough money

Alternate case: The player places the tower on an unbuildable area

	Actor	**System**
2.1		Informs the user that he/she can’t place a tower on the clicked area

Use case: Select tower

Summary: Selects tower and brings up the context menu

Priority: Mid

Extends: None

Includes: Upgrade tower, Sell tower

Participators: The player.

	Actor	**System**
1	The user presses on the tower	
2		Presents a popup menu

Use case: Upgrade tower

Summary: Improves the towers characteristics

Priority: Mid

Extends: None

Includes: None

Participators: The player.

	Actor	**System**
1	The user selects upgrade from the context menu	
2		Upgrades the selected tower
3		Deducts the user’s money

Alternate case: Player’s funds are insufficient.

	Actor	**System**
2.1		Displays a message informing the player, that he/her has not enough money
2.2		Context menu closes

Use case: Sell tower

Summary: The user sells a tower and earns money.

Priority: Mid

Extends: None

Includes: None

Participators: The player.

	Actor	**System**
1	Selects “Sell” from the context menu	
2		Removes the tower
3		Gives the player money

Use case: Next wave

Summary: The user starts a new round

Priority: High

Extends:

Includes:

Participators: The player.

	Actor	**System**
1	The player presses the “Next wave” button	
2		Updates GUI based on the information of the incoming wave
3		Starts spawning enemies

Use case: Pause game

Summary: The user pauses the game.

Priority: Low
Extends: None

Includes: Reset, resume, toggle sound, exit map

Participators: The player.

	Actor	System
1	Clicks the “Pause”-button	
2		Holds the game’s state

Use case: Player dies

Summary: The player loses all his/her lives and the game is over. The system prompts the user to play the same map again or to choose a new.

Priority: High

Extends: None

Includes: Reset, Exit map

Participators: The player.

	Actor	System
1	Loses all lives	
2		Prompts the user to play the same map again or to choose a new
3	Chooses alternative by clicking	
4		Loads the selected map

Use case: Tower shoots enemy

Summary: An enemy is within the shooting radius of a tower and the tower shoots the enemy.

Priority: High

Extends: None

Includes: None

Participators: The player, enemies

	Actor	System
1	Detects an enemy within the shooting radius.	
2	Begins shooting at the enemy.	
3		Enemy takes damage

Alternate case: The tower applies an effect to the shot enemy, e.g. slows the enemy down.

	Actor	System
3.1		The towers specified effect is applied to the tower.

Use case: Enemy dies

Summary: When an enemy loses all its hitpoints, it dies and disappears from the map.

Priority: High

Extends: None

Includes: None

Participators: Enemies

	Actor	System
1	The enemy disappears.	
2		The player gets money.

Alternate case: Last enemy is killed

	Actor	**System**
2.1		Wave ends
2		System spawns no more enemies until nextWave is invoked.

Use case: Enemy reaches end of path.

Summary: When an enemy reaches the end of the path, the player loses lives.

Priority: High

Extends: None

Includes: Player dies

Participators: Enemies

	Actor	**System**
1	The enemy disappears.	
2		The player loses lives.

Alternate case: Last enemy reaches end of path

	Actor	**System**
2.1		The current wave ends.

Use case: Toggle speed.

Summary: Makes the game speed increase if the speed has the default value or reset it if it’s increased.

Priority: Low

Extends: None

Includes: None

Participators: The player

	Actor	**System**
1	Presses the speed button	
2		The speed is increased

Alternate case: Speed is already increased

	Actor	**System**
2.1		The speed is set to normal

Use case: Toggle sound.

Summary: Mutes the sound if unmuted, unmutes the sound if muted.

Priority: Low

Extends: None

Includes: None

Participators: The player

	Actor	**System**
1	Player presses toggle sound	
2		Mute sound

Alternate case : Sound is muted

	Actor	**System**
2.1		Unmute sound