Requirement analysis

Ben Dangelmayr, Jonas Roos

April 2022

1 Introduction

1.1 Purpose of the system

The foundation of the developed system is to implement a tower defense game. Players should be able to spend time on different runs of defending an objective from incoming danger.

1.2 Scope of the system

The scope of the system will contain a map with which the player/user interacts via mouse and keyboard. The user base is directed towards computer scientists and enthusiasts enjoin the background story of the game and the parallels to hardware management.

1.3 Objectives and success criteria of the project

The main goal is to entertain the player/user. Therefore the game should be entertaining and differ from the standard tower defense game as there are many available. To stand out we have to implement unique features and mechanics. Some of these are the possibility to alter the terrain itself beside the building of the towers. Also it will be possible to place different towers with different attack types to increase the strategic requirements. Another way to increase the excitement for the game is do diversify the enemies. They will obtain different resistances to certain towers, gain different behaviour patterns such as different movement speeds or attacking towers.

1.4 Definitions, acronyms and abbreviations

In the following we will use certain abbreviations.
TD as tower defense, **PGTD** as Potato Graphics Tower Defense.
Tower defense games themselves reside in the genre of real-time strategy games and consists out of a map, some sorts of defense mechanisms and upcoming enemies which try to break through the defense. This sort of game is usually played as a single player game or some times as a cooperative game.

Level: As TDs are turn based a level will start at a wave 1 and end with the defeat of a preset last wave x or with a loss for the player/user.
Normal game mode: The player/user is victorious after reaching a preset number of turns of increasingly stronger enemies. As the player/user progresses the levels will consist out of more turns and stronger enemies but will also be able to build better towers.
Endless game mode: The endless game mode is an infinite level with the aim to conquer as many waves as possible.
Wave: A wave is defined as one turn of a level. It starts with the spawning of enemies and ends with the defeat of all enemies. There should be a small time step between the waves to let the player/user prepare himself/herself for the next one.
Loss: The player/user will lose if he can´t defend the incoming enemies and a certain amount of enemies reached the end point.

1.5 References

Popular examples for tower defense games are plants vs zombies, Bloons TD and gemcraft. Also the world builder of Warcraft III lead to a great number of different TDs which finally were the foundation of further TD game development. All these games have some things in common: It comes down to diversify the towers and adding challenging enemies with unique abilities which require own solving strategies and to find the overall balance of towers in the game. Common shared game modes are the story mode to progress in the game and to find new enemies, achievements and technologies. Another one would be the endless mode so the players can indirectly compete with each other how far they can come in a turn based game.

1.6 Overview

2 Proposed system

2.1 Overview

2.2 Functional requirements

A yet incomplete list of functional requirements is:

Enemies:

- Enemies should have the following properties: Movement speed, health, armor, resistance, payload and bounty
- Enemies should be automatically and continuously spawned at the beginning of each wave
- Each enemy killed increases the asset counter of the player by the enemy specific bounty

- Enemies killed are defined by them having zero HP and they should vanish immediately from the map

Towers:

- Towers should have the following properties: Damage type, attack modifier, attack damage
- The software should let towers automatically attack enemies if in range and reduce their HP according to the damage calculations
- The damage should be calculated by tower_damage * effectivity_multiplier/armor
- The $effectivity\_multiplier$ is defined by a lookup table determined by $Damage\_type$ and $enemy_resistance$
- The system should allow the player to select a tower and place it on valid spots on the map
- The system should allow the player to select placed towers to upgrade them
- Towers can´t be placed with insufficient funds, if the funds are sufficient they will be reduced by the cost of the tower after the tower has been placed on the map
- Towers can be sold receiving 75% of the initial costs

Spells:

- The player is able to click the spell button and a menu opens, if he click it again, it shows again the tower
- The player can click the spell he wants to use in the new menu

System

- The system shall allow the user to start the game
- The system shall allow the user to select the game mode
- The player is able to Pause/Continue the Game via a button in the right corner of the Game
- The player should be able to pause or close the game by pressing escape or the options button and a menu should be shown:
  Being in the menu the user should be able to select "Settings","Back to main menu" or "Leave Game". Continue should continue the game, settings should open a sub-menu to regulate settings and leave game should end the game, "Back to main menu" should take the player back to the main menu.

- The map should be loading after selecting the game mode
- The map should be able to represent towers, enemies, the core and the scenery
- The system should display the current HP of the core

2.3 Nonfunctional requirements

2.3.1 User interface and human factors

The user interface should be easy to use and understand. A prior is not to make it too complex and clear to every new user.

2.3.2 Documentation

Should be done in a Trello Board and the version control is done through GitHub. The different documents are written in LaTeX and linked in the GitHub repository.

2.3.3 Hardware considerations

Minimum requirements:
Operating system: Windows 10 (64bit)
Processor: 1.5 GHz or better (x86-64)
Memory: 4096MB Ram
Graphics: OpenGL 2.0 compatible, ATI, Nvidia or Intel HD
Disk space: 4096MB available disk space

Recommended requirements:
Operating system: Windows 10 (64bit)
Processor: 2 GHz or better (x86-64)
Memory: 8192MB Ram
Graphics: OpenGL 2.0 compatible, ATI, Nvidia or Intel HD
Disk space: 4096MB available disk space
Sound card: Windows compatible sound card

2.3.4 Performance characteristics

The game should at least run 60fps on average and never below 40FPS for recommended requirements.
For the minimum requirement it should run never below 30FPS.
The response time from inputs should be below 50ms on average.

2.3.5 Error handling and extreme conditions

Depending on the Error the game should be closed and stop all his processes for not overloading the computer. If it's a minor error the game should be continue in the hope that the error is not important and bad.

2.3.6 System modifications

You don't need to change anything on your system except that you need Unity to run the game.
The game should only change data in his own folder.

2.3.7 Physical environment

For the physical environment it's planned to get the game working on a computer with little power. So even a computer with a non-dedicated graphics card can run the game without any problem. Only Windows support is planned.

2.3.8 Security issues

Since the game is initially intended to be a solo player, there is no need to take security into account with regards to multiplayer. Overall the game shouldn't collect any data that is not important for the game.

2.3.9 Resource issues

Different Assets are needed for enemy's and own structures/buildings. They should be served as a high quality but at the same time should not consume to much data.

2.3.10 Coding style

We will be bound to the coding standarts and will ensure an increased quality of code by regularly conducting code refactoring. As a two person group we will either rely on code reviews based on the person who didn´t participate writing the code snippet or use pair programming techniques.

2.4 Pseudo requirements

The game should run on at least windows 10. It should be written in Unity with C# as it's main programming language.

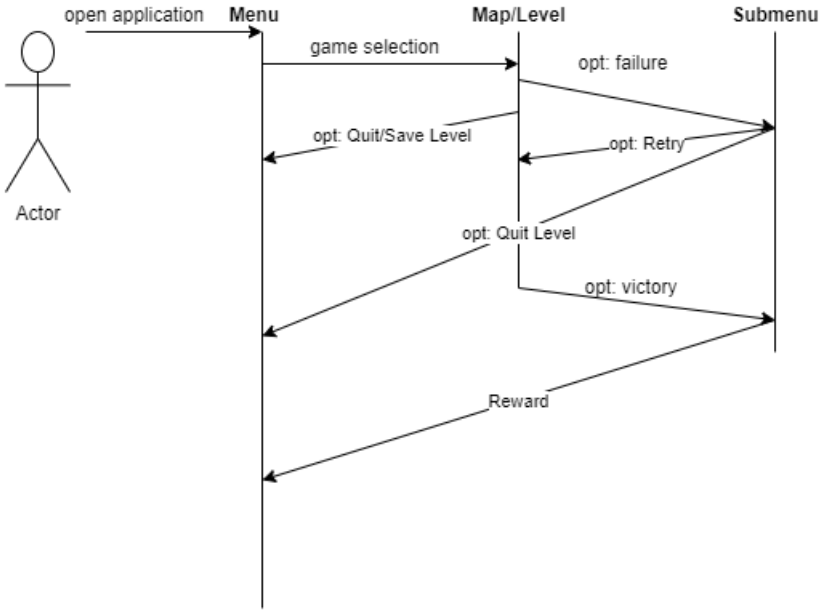
2.5 System models

2.5.1 Scenarios

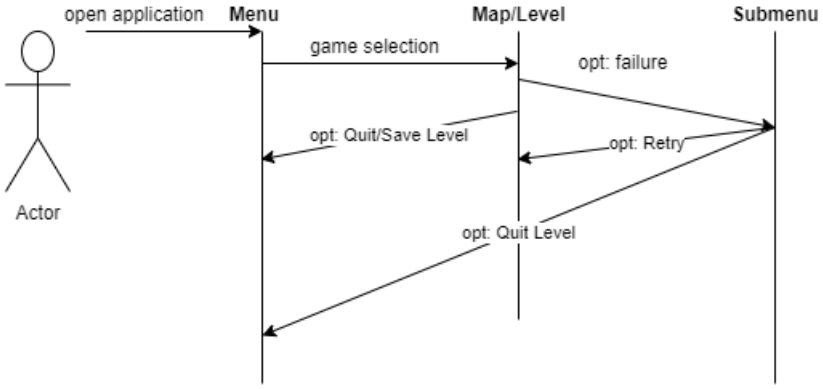
The two main ways to play the game consist out of the two game modes: Normal/Story mode and endless mode. The endless mode is not winnable and

therefore always results of a defeat notification in the submenue:

Story mode



Endless mode



2.5.2 Use case model

There will be only one type of user: The player. The main goal of the game will be to entertain the player for a decent time and to make him come back to

the game. Therefore there has to be some incentives to continue playing such as increasing diversity in towers and enemies. We want the player to keep playing so the player has to be awarded for clearing levels.

2.5.3 Object model

2.5.3.1 Data dictionary

- **Building**: building, cost, HP
- **Tower**: range, damage, attack rate, attack modifier, damage type
- **Game environment**: map, height, width
- **Game element**: xy-position, sprite
- **Enemy**: HP, movement speed, resistance, armor, bounty
- **Map**: map
- **Projectiles**: Optional
- **Path**: Optional, part of map?

2.5.3.2 Class diagrams

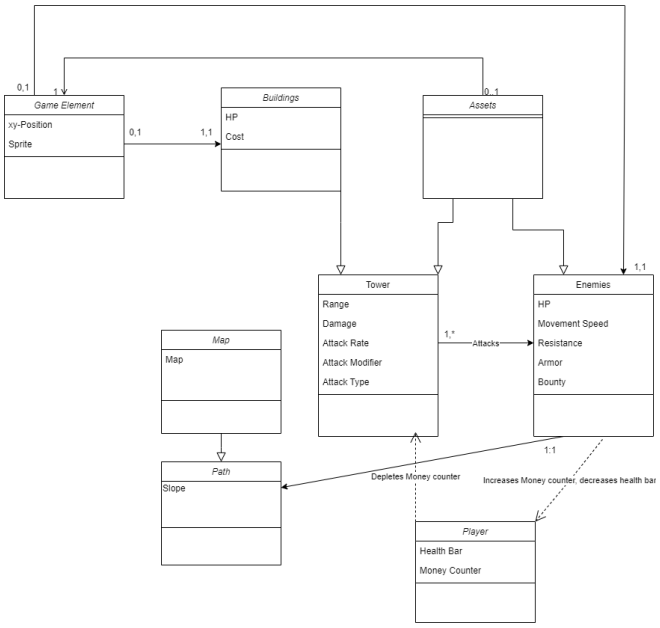


Figure 1: Class diagram

2.5.4 Dynamic models

See figure 1 for the enemy activity diagram

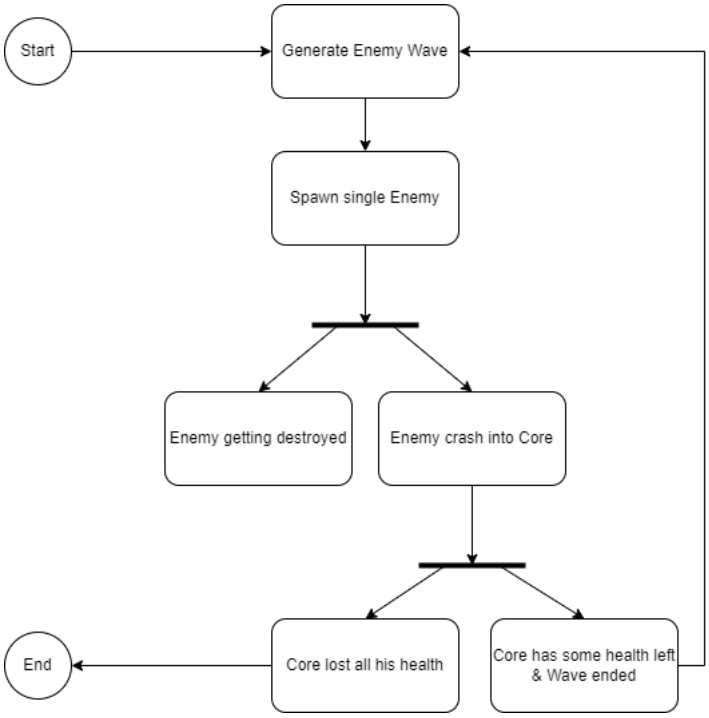


Figure 2: Diagram for "workflow" for enemy's generation

2.5.5 User interface – navigational paths and screen mock-ups

See figure 3-6 for the screen Mock-ups and User interface – navigational paths
Main Page (start menu) —> game selection —> in game —> ingame-menu to leave to the main page —> main page

3 Glossary

Dynamic objects: Moving objects, mainly enemies and probably tower missiles
Static objects: Stationary objects, mainly towers
Environment: Map background and scenery
Spawn: Point where the enemies are spawned each wave
Victory: The player will win after clearing the last wave of a level

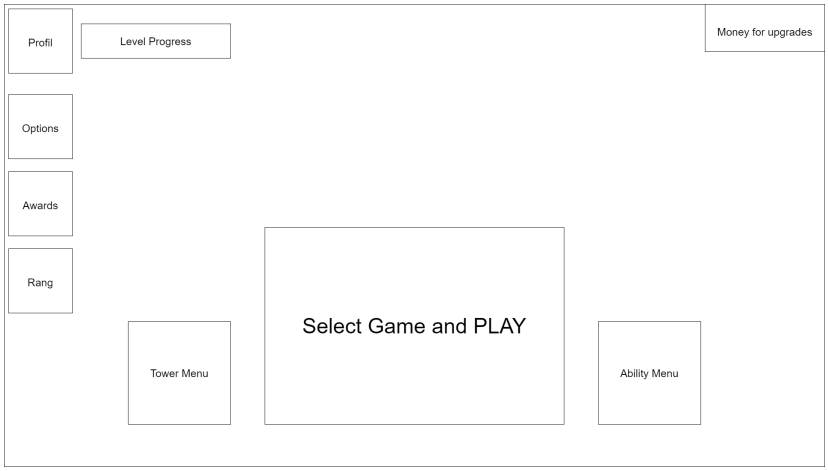


Figure 3: First Menu after starting the game

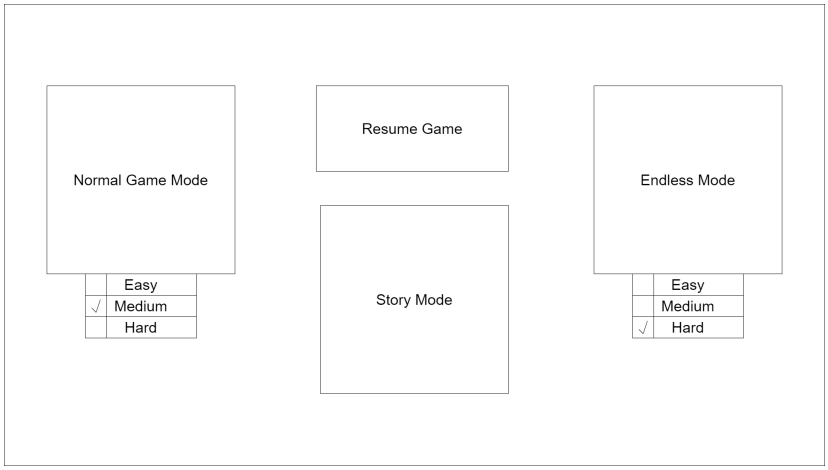


Figure 4: The player can choose between the game modes with their difficulty or can resume the last saved game

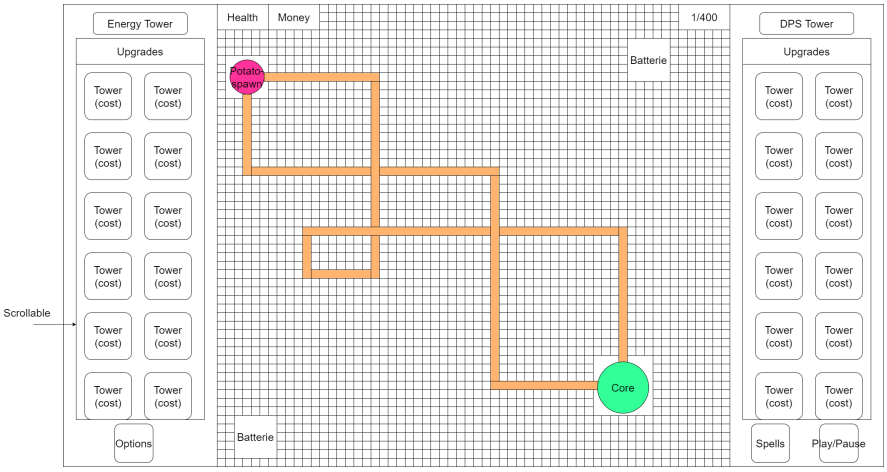


Figure 5: The User interface in the game with all options

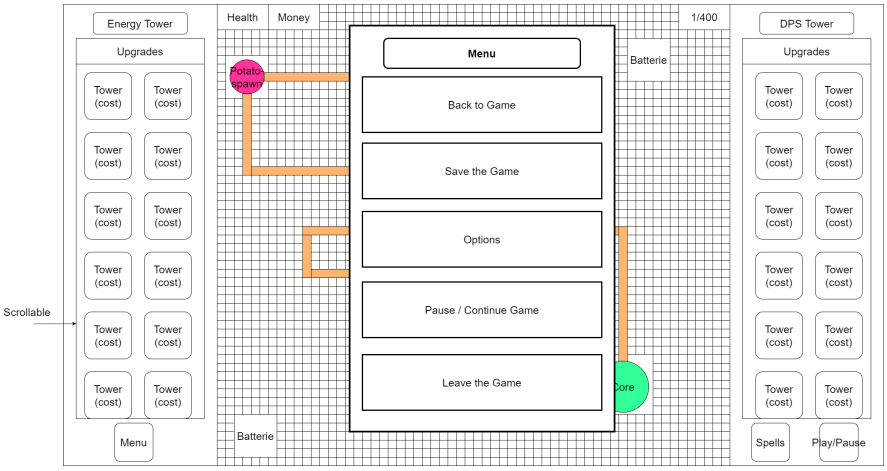


Figure 6: After opening the menu ingame