Minion Crusher

Tower Defense Game

University of Twente
Gonçalo Camelo Neves Pereira
Leon Roderick Wouter Klute
Nahuel Manterola
Remi Jonkman

Abstract **1**

Introduction **2**
Tower Defense Game 2

Game to build: Minion Crusher **3**
Interface 3
UML diagram 4

First Iteration **4**
Goals 4
Task division 5

Future Iterations **5**

Appendices **6**
Appendix A: Use Cases 6
Player Actions 6
Saving the Game 6
Editing Levels 7
Playing the Game 7

Abstract

As an assignment for the course Advanced Programming in University of Twente it was proposed to develop a team project in which C++ and SDL programming was involved not only to improve such skills but also to learn about project organization and planning.
This involved the use of an iterative process throughout the development of the product at hand and the used of diagrams, like UML and Use Case Diagrams.
To this end it is proposed by the team to develop a typical tower defense game, which is a type of game widely known in the game industry.

Introduction

Tower Defense Game

A tower defense game generally involves the player having to defend himself, portrayed as a base, by using certain entities like turrets, warriors and other type of damaging entities in order to avoid the main castle from being damaged by incoming enemies. An example of such game in given in Figure 1. The enemies walk the path which is set out at the start. Usually the enemies come in waves, where each wave increase the amount and or strength of the enemies. The player earns coins by killing enemies and or surviving rounds. This coins can be used to build new defense or upgrade the existing ones.

Figure 1 - Screenshot took from the game GemCraft (https://www.browsergames.de/gemcraft-chapter-1)

Game to build: Minion Crusher

In the first iteration the game is not intended to have an appearance as pleasing as GemCraft, instead sprites will be designed to a minimalistic look since it's only goal is to make the interface readable to focus on functionality of the game itself. This can be the focus of a future iteration.

Interface

The main part of the screen contains the map. In this map, a path is drawn to the players base. Enemy minions will travel the path towards the base. The player can buy towers and place them on the tiles next to the path, which are displayed to one side of the map. This can be done by clicking the tower and then clicking an empty slot on the map. These towers shoot at the minions to kill them before they reach the base. The towers can be upgraded by clicking them and then clicking the arrow that appears above them. Both building and upgrading towers will subtract some money from the player. Around the map, information is shown to the player, like health, money, current wave and the time until the next wave.

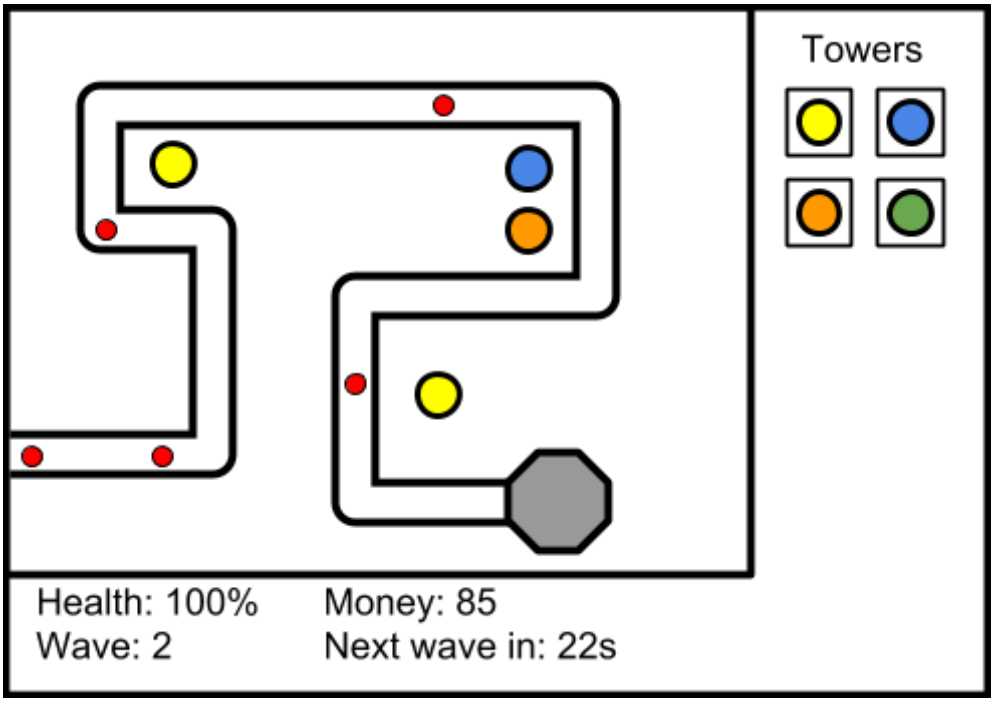

Figure 2 - General overview of the interface

UML diagram

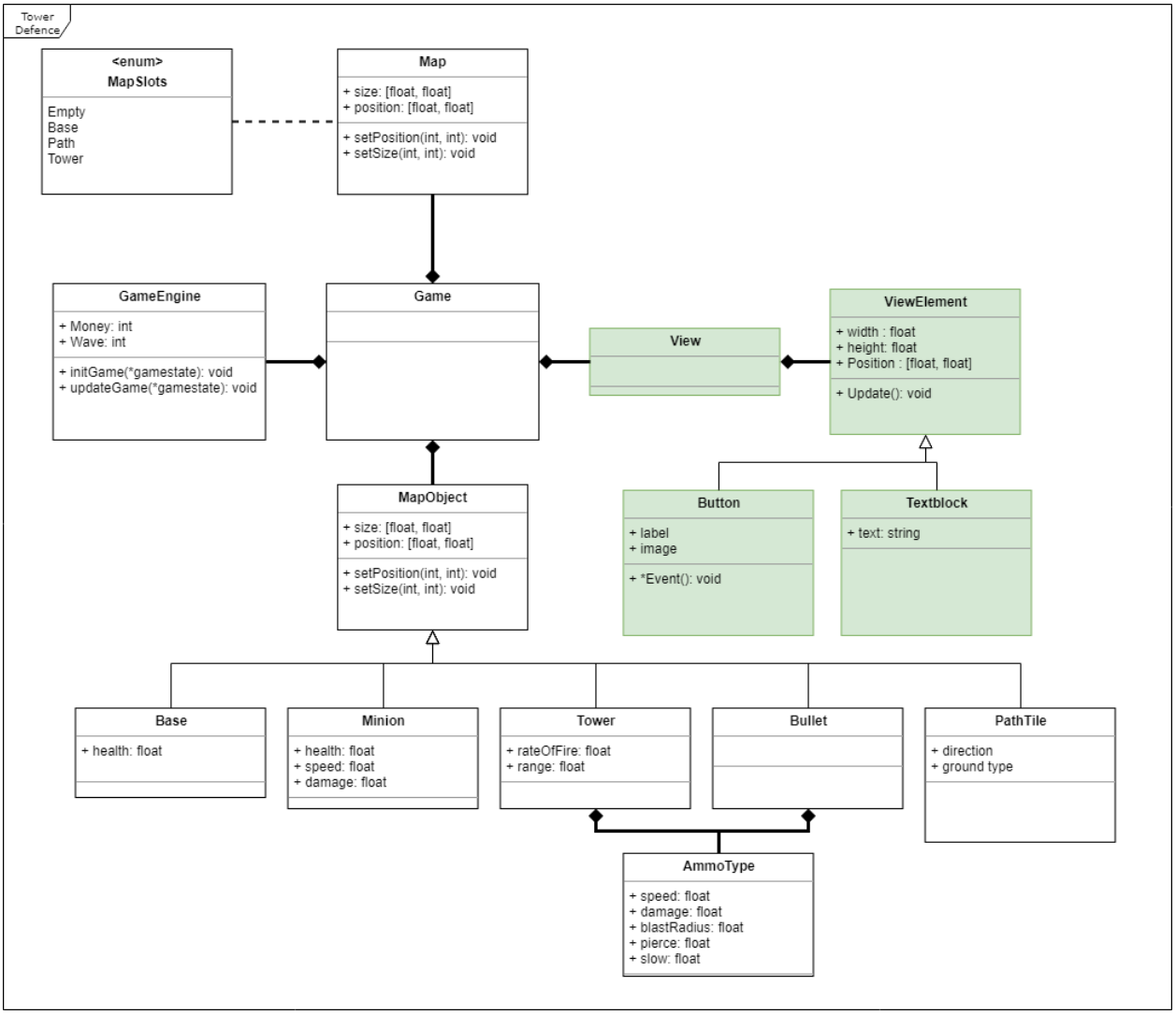

Figure 3 - UML diagram of the game

First Iteration

During the first iteration phase, focus will be on the basic functionality of the game. Standard game logic and a very simple GUI will be implemented. With this functionality it should be possible to play the game in a very simple manner.

Goals

- Loading a level
- Spawning minions;
- Show the level in the GUI
- Implement mouse interaction

- Placing towers
- Shooting projectiles
- Collision detection
- Generating money

Task division

In order to make team collaboration easy, tasks should be split inside the team, so by looking at the workload it can be seen the following almost independent work items:

- Map Generation
- Efficient collision detection
- GUI
- Game Engine
- Creation of map objects

Future Iterations

There are some features left out in this project iteration that could be implemented to improve the game, like the following:

- Level Editor - A specific interface to create a custom map
- Ńicer looking interface;
- Saving and loading your current game;
- Multiplayer;

Appendices

Appendix A: Use Cases

Player Actions

Description	This case describes how a user interacts with the GUI to place an object on the map.
Actors	Player, designer
Pre-conditions	The game should already be initialized.
Details	1. Objects can be selected in the GUI; 2. The user places the tower on one of the dedicated spots in the map; 3. The user exits object placement mode;
Post-conditions	The object is either a passive or active game object. If it is an active object, it will perform it's actions when the conditions are appropriate;
Comments	- In future versions, a tower might need more than one spot on the map.
Constraints	- In case of tower placement, the player should have enough money;
Exceptions	- Objects cannot be stacked (placed on the same spot);
Variants	- Tower placement - In level editing mode, one can place all types of objects to design a level;

Saving the Game

Description	Whenever a player is playing the game, it should always be possible to save the current state, exit the game and resume later on.
Actors	Player
Pre-conditions	The player should have initialized and started a game;
Details	1. The player has been gaming for a certain time and has to stop; 2. He/she saves the game to a specific location; 3. He/she exits the game;
Post-conditions	The game state has been written to a save file and can be restored afterwards;

Comments	- The bullets that have been fired have a target; - A wave has a certain amount of minions that is spawned; - Time; - Level; - Tower placement;
Constraints	- Every state of every game object must be summarized and should be recoverable;
Exceptions	- No permission to save at location; - No disk space;

Editing Levels

Description	Whenever a player is bored of playing one level. It hires a designer to create a new level.
Includes	Object Placement
Actors	Designer
Pre-conditions	The designer loads the level editor;
Details	1. Load the level editor; 2. The designer selects the different block types (path, empty spot, tower spot) and places them on the map; 3. The designer saves the newly created map;
Post-conditions	A new level saved on disk;
Comments	The map has a fixed size;
Constraints	- The designer must place a base; - The designer must place a minion entry; - The designer must design a path between the base and entry;
Exceptions	- No permission to save at location; - No disk space;
Variants	None.

Playing the Game

Description	The general scenario in which the player has successfully started a game.
Includes	Tower Placement, spawning minions, towers firing bullets, wave identification, money generation, pause/play/speed game
Actors	Player
Pre-conditions	None.

Details	The player decides based on the amount of money, number of minions and wave counter what it must do to survive the wave.
Post-conditions	None.
Comments	None.
Constraints	- The game must run smoothly at a certain FPS; - The game must not eat memory (check for memory leaks with Valgrind)
Exceptions	None.
Variants	None (later on different skins can be added);