Tankgeon! game project plan

Teemu Taivainen, Janne Kantola, Jaakko Pulkkinen, Otso Häkkänen

Scope of the work

Our plan is to create a dungeon crawler in the style of Wii play Classic: Tanks! The main idea is tank-on-tank combat while advancing through different rooms. Because we want the combat to be more dynamic, the game will be real time instead of turn based. For this idea, we need to implement a controllable player chassis with a turret that spins independently. For projectiles we'll build in rudimentary physics to achieve ricochets from objects. The rooms will have obstacles, enemy tanks and doors to adjacent rooms. The enemies will have to have basic AI to move around and shoot at the player tank. Going through the rooms and beating enemies rewards the player with items which will need an inventory system to manage them. The items could be different kinds of ammunition, mines, shields or repair kits.

For more ambitious features, we plan on implementing a level-up system with the ability to upgrade the player vehicle. Adding music and sound effects will also be a goal for us. Fine tuning the AI to be more interesting to play against will presumably also require quite a bit of work.

The game will be played through a GUI in a window, using the mouse to turn the turret and the keyboard to angle the chassis. We plan on having a custom UI to use the level-up and inventory systems. The whole game will be designed around a top-down perspective.

Structure of the program

We are planning to separate our program into the following modules: "UI", "Input" and "Model".

The "UI" module is responsible for rendering the game's graphics and displaying its user interface such as the menus, health bar and the inventory. The module consists of a single class called "Tanks GUI".

The “Input” module is responsible for handling the player inputs and it consists of a single class called “Tanks input”. The class reads both keyboard and mouse inputs from the player and takes care of event handling for them.

The “Model” module is the largest module in our project, and it handles the game logic of our program. It can be separated into 7 main classes: “Game”, “Player”, “Level”, “Enemy tank”, “Obstacle”, “Item” and “Ordnance”.

The “Game” class enforces the game rules such as objectives, game states, win conditions and lose conditions.

The “Player” class describes the players tank, and it handles its conditions such as current healthpoints, turret direction, inventory status and its current location on the play grid.

The “Enemy tank” class is an abstract class for the non-player enemies in the game. “Red tank”, “Blue tank” and “Black tank” are all derived classes from the “Enemy tank” class and they represent different enemy tanks that have different abilities compared to each other such as special projectiles or different AI.

The “Item” class is an abstract class for different elements that increase the health points or add armor to the player. It is implemented by the “Repari kit” and “Armor” classes. An instance of “Repair kit” is an item that is located within a level and it increases the healthpoints of the player. It keeps track of it’s location on the map and wether it has been used. An instance of the “Armor” is an item that is located within a level and it decreases the amount of lost health points when the player is hit.

The “Ordnance” class is an abstract class for different elements that negatively affect the health points of the player tank and the enemy tanks, and it is implemented by two classes: “Projectile” and “Mine”. An instance of the “Projectile” class represents one projectile that is shot either by the player or the enemy. Projectiles can riochet of walls only a limited number of times so the objective of the class is to keep track of it’s current location on map, how many times it has hit a wall and when to play a soundeffect. An instance of the “Mine” class represents a mine that is deployed either by the player or by an enemy. A mine can explode either when the player or the enemy gets too close to it or it has reached a certain lifetime, so the “Mine” class keeps track of how long it has been since it was laid and it’s location on the playgrid.

The “Obstacle” class is an abstract class for different obstacles that are located within a level. The two implementations of the “Obstacle” class are the “Wall” and “Spikes” classes. The common element of both classes is that the player nor the enemies can pass through them and they keep track of their location on the play

grid. The key difference between the two implementations is that projectiles can pass through “Spikes” objects while not through “Wall” objects.

An instance of the “Level” class describes one level in the game and its job is to keep track of different properties of a level such as which enemies, items and walls are included in it. One “Game” object contains several “Level” objects.

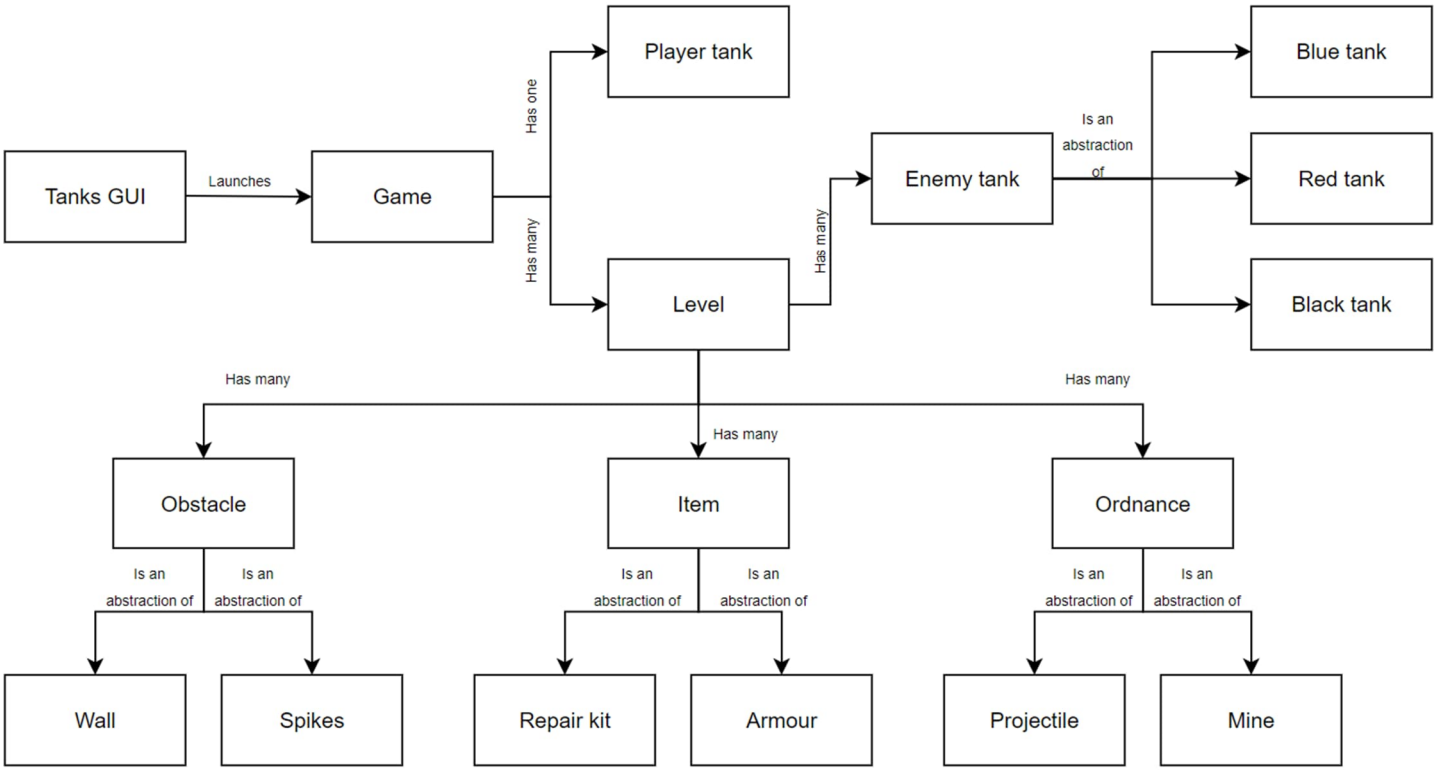

Figure 1: Visual representation of the planned class relationships

Use of external libraries

Raylib is open source 2D (and 3D) graphics rendering library that is made in C. It supports different forms of audio, image and video drawing for sprites, lighting, fonts and math for rotations, collisions and physics. In our game we use mostly basic shapes, like rectangles and circles, need collision detection and simple math and a bit of audio. Raylib is good choice for those.

Division of work

This division of work is preliminary and will probably change according to current needs.

Controls	Otso
Main (abstract) classes	Janne, Teemu
GUI	Teemu ja Otso
Levels and assets design	Jaakko
Concrete (derived) classes	Jaakko
Sound design	Jaakko, Janne
AI-design	Otso

Working practices

Every group member develops in their own git branch. Everyone pulls frequently from master to avoid merge conflicts. Once a feature is developed and tested, the working branch is merged with master. Master should always contain a working version of the software.

Schedule

Date	Milestone
31.10.	Planning done
8.11.	Asset creation and main class implementations done
13.11.	Prototype ready
27.11.	Working game
1.12	Testing and QA done
6.12	Bugs found in testing fixed
8.12.	Project ready