Tankgeon! project documentation

Teemu Taivainen, Janne Kantola, Otso Häkkänen, Jaakko Pulkkinen

Overview and structure

Our final product is a Wii play Classic: Tanks! -knockoff with some dungeon crawler elements. In this game you can control a tank in real time combat while avoiding enemies. The turret spins independently of the chassis and the chassis can go forwards, backwards and spin. The movement is not locked in the x-y directions, so the chassis can move diagonally. Movement is controlled by custom collision detection with walls and spikes. We had to implement custom collision detection since the implementation provided by SFML only works with axis-aligned hitboxes. The tanks can shoot, and projectiles ricochet from walls, go over holes and two projectiles hitting each other cancel out.

The levels are implemented with text files of a predetermined format and are automatically loaded in order. Adding levels can be done by copying the template and naming it as the next level (If last level is level7, name the new one level8 so that it is loaded last). Levels have walls, spikes and holes as well as shields and enemies. We did not implement walls between the rooms, as the player automatically moves between them when all enemies are hit. The game is won if the next level cannot be loaded and displays an end screen with your score.

Shields are items that are loaded with levels. They protect you from one projectile and disappear after being hit. We ended up not implementing an inventory UI, since we only have one item. We deviated from our original plan by only implementing shields as items, since the basic gameplay felt good enough without them. We also did not add the planned level up system to stay consistent with our inspiration, Wii Play: Tanks!

There are three types of enemies with different movement patterns and attacks. The enemy tracks the location of the player and acts accordingly while trying to hit them. The enemies do not fire, if they do not have a line of sight to the player tank. Moving is achieved with a random generator for directions.

The game has textures for visible objects and sound for main menu and levels. We also have a start menu with a button to start and menu music. The levels have their own music / sound effects and hits on tanks generate animated explosions. End screens for win and lose are both implemented,

with lose appearing when you are hit without a shield. The game resets and sends you back to the start screen after an end screen.

Software structure

Our program consists of classes that can roughly be divided into three modules: “Game model”, “Game elements” and “Hud”. Below (Graph 1) is a graph describing the basic class relationships of our program. We also used SFML in our program for graphics, sounds, music, capturing user input and basic collision detection. The gameplay loop is implemented in the main function, which updates the window for each frame and then draws it.

The “Game model” module consists of the “Game” and “Level” classes. The “Game” class is responsible for enforcing the game rules and it also keeps track of the current game state, high scores, win conditions, and lose conditions. The “Game” class is also responsible for implementing and drawing the menu and end screens, based on the current game state. For each game there exists one Level object which represents the current level that the player is on. The “Level” keeps track of the tanks, shields and obstacles that are located on it, and it also handles item pick-ups. If the level is completed, then the “Game” class orders the “Level” class to load the next level from a .txt file that contains the level data.

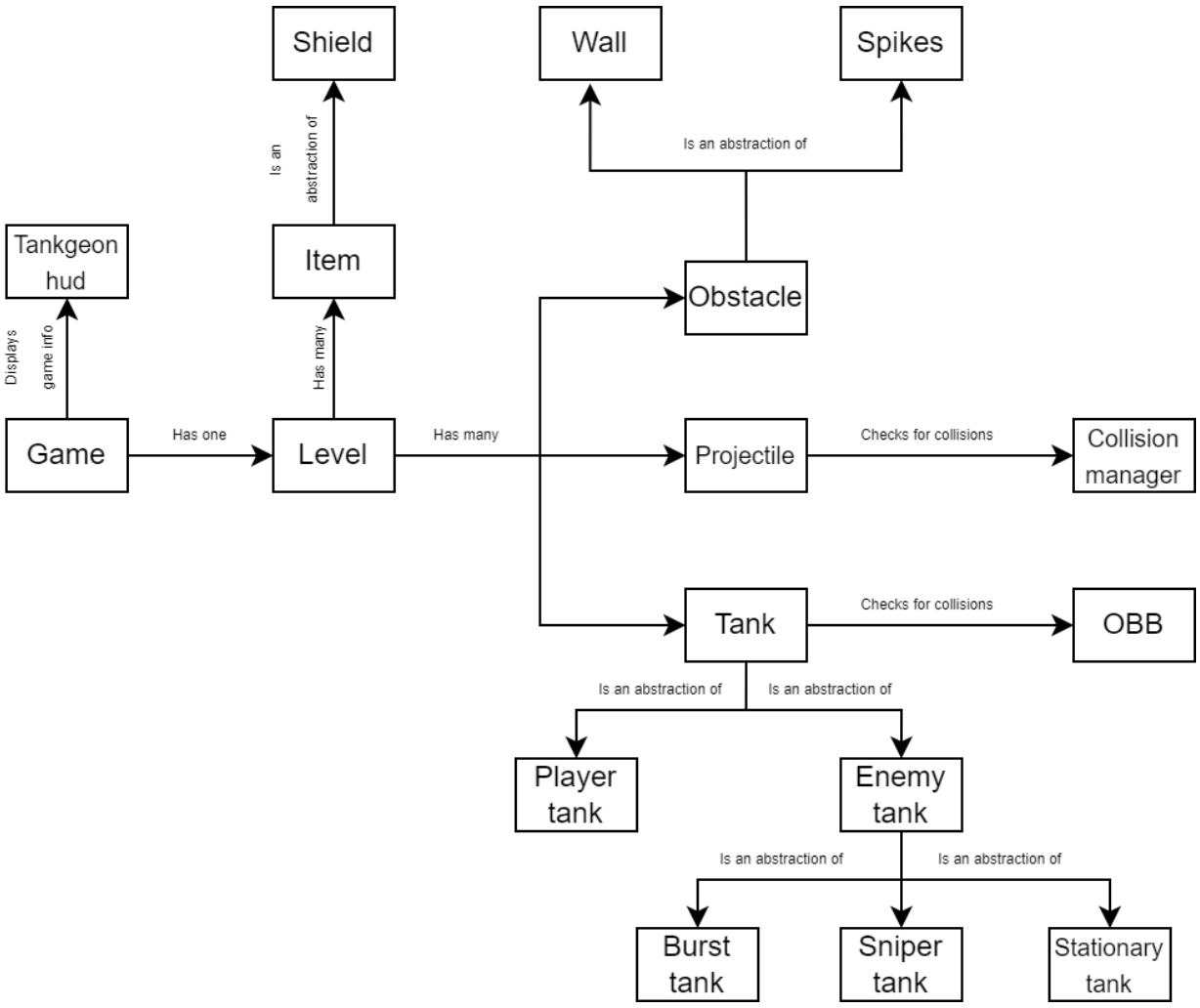
The ”Game model” is the biggest module in our program. One of its most abstract classes is the “Tank” class that that each represents one tank that is in a level. The “Tanks” class has 2 derived classes: The “Player tank” and “Enemy tank”. Furthermore, the “Enemy tank” class has 3 derived classes: “Burst tank”, Sniper tank”, and “Stationary tank”. All basic tank functionalities such as current position, moving, shooting, status, and drawing are implemented in the “Tank class”. The “Player Tank” class adds player-specific functionalities such as moving the tank according to the keyboard inputs, handling the shield status and upkeeping the score. The player tank also checks whether it can update its position based on the user commands or if it collides with an obstacle using the “OBB” class. The “OBB” class determines whether a collision will occur even when the player tank is rotated. This class was created to improve collision detection since the native bounding box created by SFML is always axis aligned. Vice versa, the “Enemy tank” class adds enemy specific functionalities such finding the player tank and getting the shooting angle to the player. The enemy-specific behavioral characteristics such as movement and shooting styles are implemented in the “Burst tank”, “Sniper tank”, and “Stationary tank” classes.

The “Game model” module also includes the “Obstacle”, “Item”, and “Projectile” classes. The “Obstacle” class is an abstract class that describes an obstacle that is located within a level. The class implements virtual functions for drawing the obstacle and determining whether a projectile can pass through them. Derived classes “Wall” and “Spike” then implement these.

The “Item” class is an abstract class for an item that is located within a level and the class implements a virtual function for drawing it. It only has one derived class, the “Shield” class that implements the drawing function. The “Shield” class represents a shield that the player can pick up and equip.

The “Projectile” class is a class, that’s instance describes a projectile that is shot either by the player tank or an enemy tank. The projectile class handles functionalities such as updating it’s current position, keeping track of its life time and ricocheting of walls. The collisions hand ricochets of the projectiles are handled in the “Level” class with the help of the “Collision manager” class that checks whether the bounding box of a projectile intersects another bounding box and what to do based on the object it collides with.

The final module is also the smallest module in our program. The “Hud” module consists only of the class “Tankgeon hud”. The class implements a basic hud that is shown at the bottom of the screen. The hud shows the current level number and player score.



Graph 1: Visual representation of the class relationships.

Building and using the software

These instructions assume Linux environment, but CMake makes it quite easy to build on other environments too. However, exact terminal commands differ. The following steps give detailed instructions on how to build and run the Tangeon game.

1. Make sure that you have libraries required by our graphics library SFML installed.

- On Ubuntu or other Debian-based OSes, that can be done with the commands below:

  sudo apt update

  sudo apt install \

  libxrandr-dev \

  libxcursor-dev \

  libudev-dev \

  libopenal-dev \

  libflac-dev \

  libvorbis-dev \

  libgl1-mesa-dev \

  libegl1-mesa-dev \

  libdrm-dev \

  libgbm-dev \

  libfreetype6-dev

- Also the CMake and Make are needed and can be installed with

  sudo apt install build-essential cmake

- CMake Tools extension for vscode is required to build project with preconfigured task.

2. Copy our git repository to your machine

- Use git clone or download zip folder

3. Compile project with cmake and makefile

- Use preconfigured vscode task build that is defined in .vscode/tasks.json. Open vscode command palette and select Run task, Build.
- OR do the steps below to build from command line.
- Navigate to project root directory
- Create build directory and navigate to it

  mkdir build

  cd build

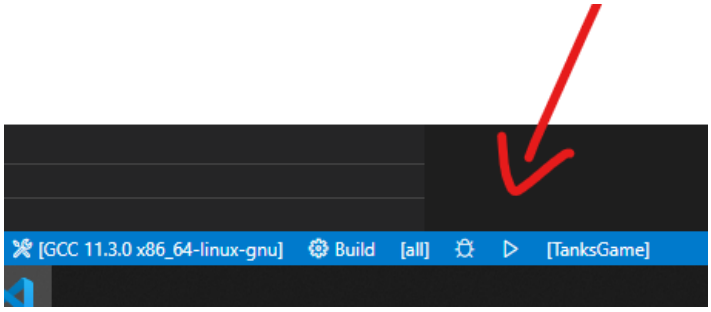
- Run cmake to generate Makefile (.. tells that CMakeLists is located in parent directory)

  cmake ..

- Compile and build the project with make

  make

4. Run the game.



- Press launch button provided by CMake Tools extension located at the bottom of the screen
- OR run the following on command line while in the build directory (dot at the beginning)

```
./TanksGame
```

How to play

Use A and D keys to rotate and W and S keys to move your tank. Aim and shoot using mouse and left click. All tanks, including yours, explode in one shot so be careful. Try to ricochet your shots with walls so enemies have harder time shoot you. You can shoot over holes. When all the enemy tanks shot you advance to the next level. Some levels contain Shields which give you one extra hit point. The game has different tanks: basic Blue tanks that shoot in random directions and doesn't move, Yellow tanks that shoot you in bursts and Red tanks that shoot fast moving bullets that don't ricochet. Try to survive all the levels if you can! Esc key exits the game and P pauses the game.

Testing

Most of the testing was done manually by just playtesting the game. This gave us the opportunity to fine-tune projectile speed and movement speed. Most of the bugs we found were discovered just by playing, and we felt that the issues discovered this way were the most important to fix. The bugs were communicated via GIT to the whole team and fixed in subsequent commits. Things like enemy tank movement and projectile ricochets could be easily tested by just playing and seeing if they worked properly.

In addition to substantial manual testing, unit test for level class was written. The test ensures that different objects are placed correctly within the level. The test creates a mock level and calls its loadFromFile method that populates the level using a level file meant for testing. After calling the method the function asserts that different objects, such as enemy and player, were placed to correct positions calculated using the mock window size and hard-coded positions of objects withing text file.

The test is run simply by including the test file in main and calling the test function before executing the code that actually starts the game. This test is useful to run if loadFromFile method is changed as it ensures that it works properly.

Division of work and work log

Otso	Player tank controls, enemy AI, Tank classes main methods, start and end screen, main menu, OBB collision detection
Janne	Spike class, Tank collision detection, Item and Shield classes, HUD, high score system, Tank scaling according to screen size
Teemu	Build system (CMakeLists), projectile class, level loader, enemy AI, different Tank classes, Game class
Jaakko	All visuals (for example tank textures), explosion animation, audio, level design, asset creation

week	Otso	Janne	Teemu	Jaakko
1	Player tank controls, added inherited class enemy tank 4 h	Abstract class obstacle and derived classes spike and wall implementation 8h	CMake + Projectile + PlayerTank 10 h	First prototype creation 7 h
2	Code refactoring for tank class and enemy tank, basic AI and first iteration of Level class 10 h	Spikes full implementation, Tank collision with obstacles 12 h	Projectiles ricochet from walls and have a limited lifetime 8 h	Graphics (Tank and Wall visuals) and lots of SFML learning 9 h
3	Improved collision detection for rotated shapes 12 h	Shield item implementation and HUD 9 h	Level loader and projectiles that cancel each other out, Game class 10 h	Graphics for the shield item and fully animated explosions 8 h
4	Endscreen, pausescreen, main menu and win conditions 6 h	Highscores and improved HUD 9 h	Two new enemy AI types 7 h	Sound design and missing textures added to game. 7 h