Praktikum: Echtzeit Computergrafik

Assignment 8 – *Enemy Ships*

10 Points

In this assignment, we will populate the sky with multiple enemy ships. The ships should spawn in regular intervals and move along randomly generated paths. You will have to define several enemy types in the config file, add and manage enemy instances at runtime and render each enemy at its respective position in the sky.

Resources

First, we need to add new resources for the enemy ships to the content pipeline.

- Find the required resources (.obj and .png files) for at least three enemy ship models of your choice in `external/art`, and modify the NMake script to generate the corresponding .t3d and .dds files (like you did for the ground and cockpit objects in Ass. 6).
- For each ship model, add a corresponding “Mesh” line to your `game.cfg`.

Enemy Types (1P)

In the config file, we will define several enemy types. Every time an enemy ship is spawned, its type should be randomly picked from all types defined in the config. Here is an example enemy type definition:

```
EnemyType AmyShip 100 10 50 Amy 0.5 0 180 0 0 0 0
```

This line defines an “EnemyType” called “AmyShip” with 100 hitpoints, a size of 10 units and a speed of 50 (size and hitpoints are not important for this assignment). It uses the mesh named “Amy” with the rest of the numbers specifying a transformation for the mesh as usual (i.e. scale, x-/y-/z-rotation and x-/y-/z-translation).

Unlike for Ground- and CockpitObjects this transformation does not specify an absolute positioning on the ground or in the cockpit. Instead we will use it to unify the pose of all enemy ship meshes in object space, before they are rotated and translated to their actual position in world space. This accounts for the problem that e.g. not all ship meshes are facing in the same direction in object space.

- Add at least five EnemyType definitions like the one above to your config file. Use all your meshes at least once, and vary properties like speed, hitpoints etc.
  - Hint: You can get the transformations right later. Start with zero rotation/translation and e.g. instantiate your ship meshes as GroundObjects to find a reasonable scaling.
- Similar to Ground- and CockpitObjects, create a type (struct or class) that mirrors the definition of EnemyTypes in the config.

- Extend your config parser to read in the enemy types defined in the config. Use a `std::map` to store all EnemyTypes so you can easily access them by their name (like for the meshes).

This is a good time to check (using e.g. the Visual Studio Debugger) that everything you did so far works as expected, i.e. that your `std::map` contains all the enemy types specified in the config.

Enemy Instances (1P)

In addition to enemy types, we need to manage instances of enemies internally, which have to be added/removed when enemies spawn, get destroyed or leave the map. Each enemy instance should be defined by (at least) the name of its enemy type, its current position $\boldsymbol{p} \in R^3$ and velocity $\boldsymbol{v} \in R^3$in world space and a number of remaining hitpoints.

- Create a type for enemy instances (class or struct) that contains all necessary information.
- Declare a global list (`std::list`) of enemy instances that we will use in the following to store all existing enemies (`#include <list>`).
    - Note: We need to be able to efficiently remove enemy instances from the game, so a list is a better choice than e.g. an array (`std::vector`).

Spawn & Movement (5P)

Now we will actually spawn enemies (i.e. add instances to the list) and move them. You can stick to the spawn model from the slides, or you can be creative and spawn enemies in your own way.

Your spawn model should at least fulfill the following requirements:

- *Enemies and their paths are randomly generated at regular or semi-random time intervals.*
- *All enemies pass the turret cockpit at close distances so that we will be able to shoot at them in future assignments.*
- *Enemies should at least move along straight, horizontal paths. If you want, you can also try to vary speed, direction or height of your enemy ships while they are in the air.*
- *Due to the random nature of our spawn model your ships are allowed to touch or fly through each other.*

- In your config file, define (parse in your config parser) all major properties that control your spawn behavior. For example:

```
Spawn 2.5 0.4 0.8
```

This example line tells our game to spawn an enemy ship every 2.5 seconds in a random height ∈ [0.4*`g_TerrainHeight`; 0.8*`g_TerrainHeight`]. **Add or remove more parameters as needed!**

- In `OnFrameMove()`: If a new enemy has to be spawned, add a new enemy instance to your list. Choose a random enemy type and initialize its position $\boldsymbol{p}$ and velocity $\boldsymbol{v}$ according to your spawn model.

    - Hint: In order to easily pick a random enemy type name, it might be a good idea to collect all available names in a `std::vector<std::string>` when parsing the config file (in addition to the `std::map` of enemy types).
- In `OnFrameMove()`: Remove all enemies from the list of enemy instances that have left the map according to your spawn model.
- In `OnFrameMove()`: Update the position of all enemy instances according to the rule $\boldsymbol{p} \coloneqq \boldsymbol{p} + h \cdot \boldsymbol{v}$ (where h = `fElapsedTime`).
    - Hint: During update, enemy ships might fly through the terrain. You can avoid this by either checking the terrain height a the new position and correting the path accordingly, and / or choosing your spawn positions / directions appropriate. However, this will not be graded in this assignment.
    -

Rendering (3P)

Now it's time to render the enemy ships.

- In `OnD3D11FrameRender():` Render all enemy instances at their respective positions $\boldsymbol{p}$ facing into the direction of their velocity $\boldsymbol{d} = \boldsymbol{v}/\|\boldsymbol{v}\|$. For the world transformation, firstly apply the transformations specified in the config file for the enemy type, and then use rotation & translation (= M_{Anim} on the slides) to position the object in world space.
- Make final adjustments to the transformations for the enemy types in your config file. Adjust rotation such that all ships are facing forward (for some ship models it's not completely clear where the front is, so there is more than one correct solution). The tool MeshLab might be of help here. All ship models already are roughly centered at the origin, so it's ok to use zero translation.
- Finally, make sure once again that your program produces no memory leaks during a normal run, and that it works in both the Debug and Release configurations.