DH2323 dgi22 VT22-1 Computer Graphics and Interaction
Andreas Wrife, David Åstrand, Léo Noharet
awrife@kth.se, dastrand@kth.se, lnoharet@kth.se

DH2323 LAB 3

1. Setup of the Lab Environment

For lab 3, we firstly exported the changes made in lab 2 to a unity package. Then, we created a new Unity project and imported the unity package with the source code for lab 3 from canvas. Finally, we imported the packages with the changes from lab 2, overwriting some the common scripts.

2.5 Collider Tasks

2.5.1 Radar Detection

To make the radar change colour we utilise the $OnTriggerEnter()$ and $OnTriggerExit()$ functions. Since we already had a render function for the radar alarm circle, we called it again with a different material path as an input. Since $OnTriggerExit()$ and $Start()$ both set the material to the green save colour, this colour was set as the default argument of the $DoRenderer()$ function.

To make it so that only the player tank triggered the alarm, we introduced an if statement using the *gameObject.tag* utility set to "Player". This way, other gameObjects such as enemy tanks do not trigger the alarm as can be seen on the right image of Figure 1.

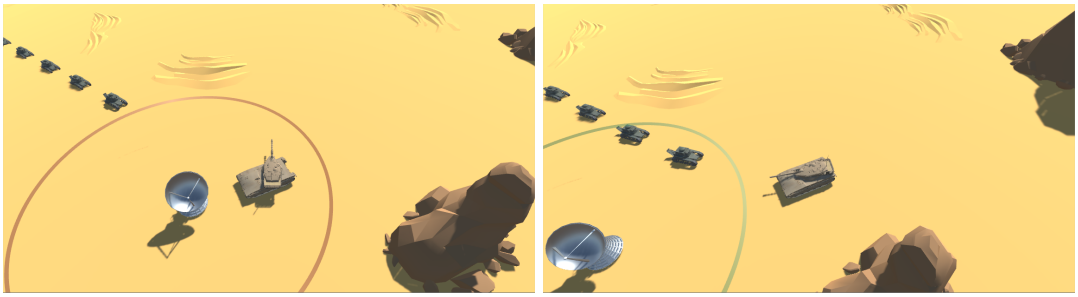

Figure 1: Images showing the alarm being on (left) when the player is inside the alarm and alarm begin off (right) when player is outside

2.5.2 Firing of the Secondary Gun

In the second task the secondary gun was supposed to fire a laser in the direction of the player mouse. Though similar to lab 2 implementing the turret aim, the difference being that if the laser hits a rock or boundary object it should be cut short, thus indicating that a collision with the target object had occurred. To make the laser shoot in the direction of the mouse, the second position of the *gunLine* was set to the *shootHit* point if it hit a Rock or Boundary object. Otherwise the second position of the *gunLine* was set to the maximal raycast range which is calculated by $shootRay.origin + shootRay.direction * range$.

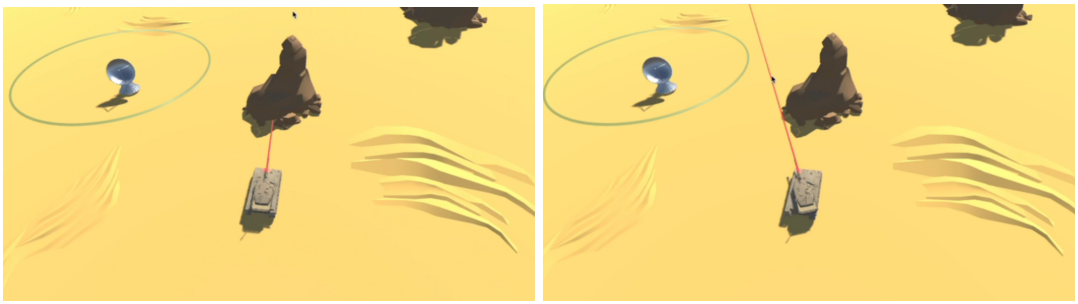

Figure 2: Images showing the shoot ray stopping when with a rock (left) and going to the maximal raycast range if no rocks or boundaries are hit (right)

After our first implementation attempt, the ray traversed the rocks and boundaries of the map. To fix this, we created a LayerMask called *rockMask* and passed it to the *Physics.Raycast*() function, similar to what was done in lab 2 with the floorMask. To make the LayerMask work, we had to create a new layer "Rock" which was applied to the Rocks and Boundary gameObjects. See Figures 2 and 3 for the results.

Figure 3: Image showing the shoot ray stopping when hitting a boundary of the map

2.5.3 Firing of the Main Gun

This task was easily implemented by adding a *transform.localScale* to the *shellinstance* objected created in the *Fire*() function. The object was then scaled by a factor of 0.2 which seemed to rescale the shells to a realistic proportion.

2.5.4 Destroy Enemy Tank

First, we created a new tag "EnemyTank", and proceeded to set the tag of the 'enemytank' prefab to this. The script then always destroys the shell object, and if the collided gameObject has the "EnemyTank" tag, the collided object is likewise destroyed and a particle effect object is subsequently instantiated and then destroyed when it has run its course. This part of the script, as well as the explosion particle system from lab 2, were imported for this purpose.

Afterwards, we renamed the Rock layer to *ShootRayObstacle* layer, and added this layer to the enemy tanks. Then, in *SecondaryGunShoot.cs* when the shootRay hit something of the *ShootRayObstacle* layer, we check if the *shootHit.collider* has the tag "enemytank". If it does then remove the enemy tank and explosion particle system. Added explosion script from lab2 to the secondary gun aswell, which instantiates the explosion and deletes its prefab after it had triggered. An error that we encountered at first was that since the secondary gun shoot only in a horizontal direction, the shootRay did not hit the enemy tanks because it was too high as the secondary gun is located higher than the height of the enemy tanks. To fix this, we rescaled the enemy tanks prefab instances through transform.localScale. The new scale of the prefab instances was set to (50, 50, 50).

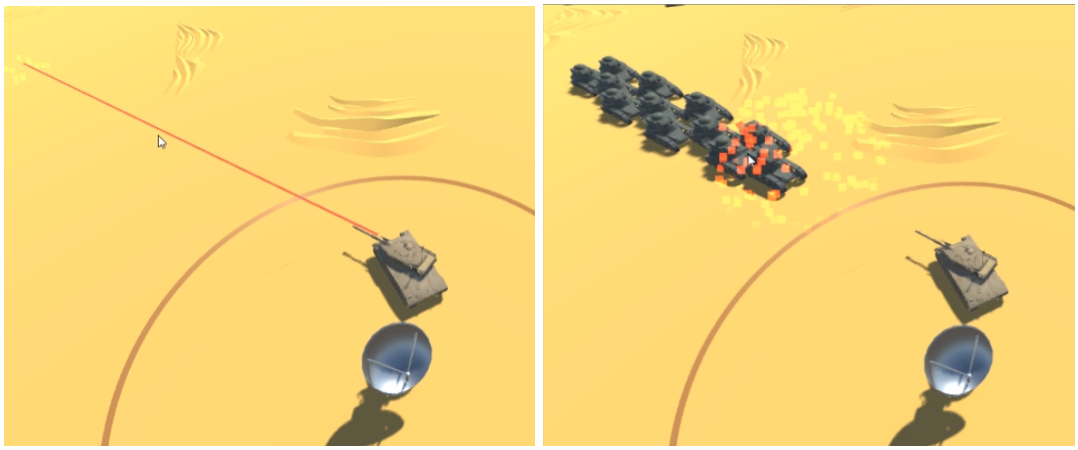

Figure 4: Image showing the laser and gun shooting and destroying an enemy tank showing the beginning of the explosion particles.

3 Level of Details Tasks

3.1 **LOD** In this task, we encountered a fair bit of problems.

Initially, we added a LOD component to the *enemytank* prefab and added the *enemytanklowmesh* prefab to the LOD component. However, as this is a prefab with its own collider and rigidbody, this entailed many weird glitches.

One of these problems were that since the objects pushed each other away, we ended up with two different objects at different places - each of which could be destroyed individually, but that were only visible one at a time.

The *enemytanklowmesh* had quite a large difference in size compared to the *enemytank*, see Figure 5, which gave some funny results ingame. The problem here was that the two prefabs had both size and scale which clashed with each other. Since these two objects were children to a parent object which had a scale this made the issue even worse. The problem was solved by setting the scale of the *enemytanklowmesh* to 0.01 and the parent's scale to (50, 50, 50) so that the collider of the object would be hit by the laser. This could also been done by changing the y-axis of the collider's hitbox to reach further up.

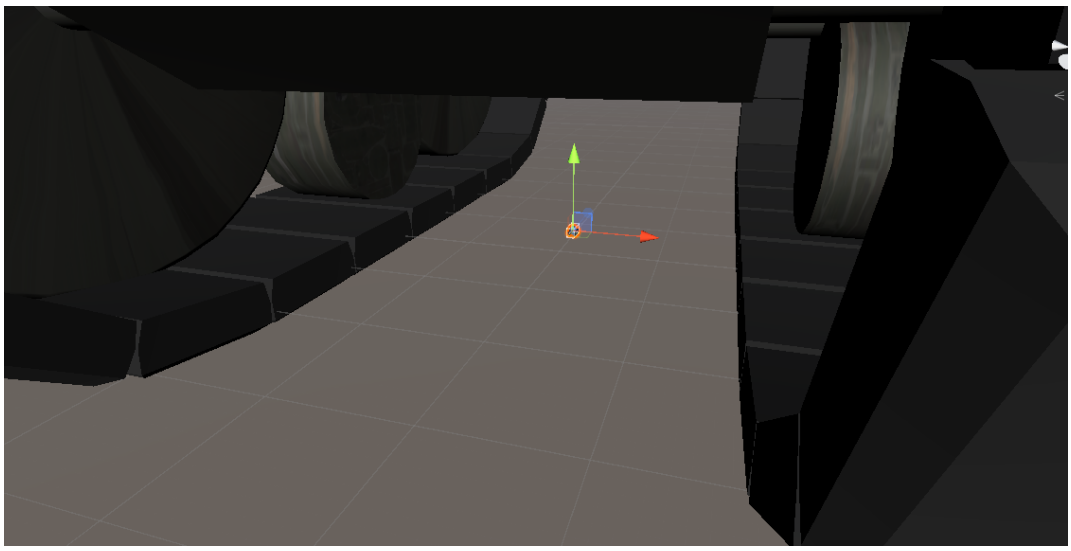

Figure 5: Image showing the two tanks in proportion to each other.

After following the Unity documentation on LOD, we realised that we needed to only add the meshes to the LOD instead of the whole prefab. Therefore, we created a new prefab called *TankRenderers* which contains only the mesh filters and mesh renderers of the *enemytanklowmesh* prefab and not any rigidbody or colliders. Then, we created a new prefab *LOD*2 that had the rigidbody and the collider of the *enemytank* prefab as well as a LOD component and the EnemyMovement script. AS a result, we had to set the LOD2 prefab as an input to the RadarAlarm.cs script. As its children, *LOD*2 had both the regular *enemytank* meshes (again, with the rigidbody and collider removed) and the *enemytanklowmesh* meshes. This way, the LOD component is able to find the meshes that it is supposed to switch between without changing any other functionality.

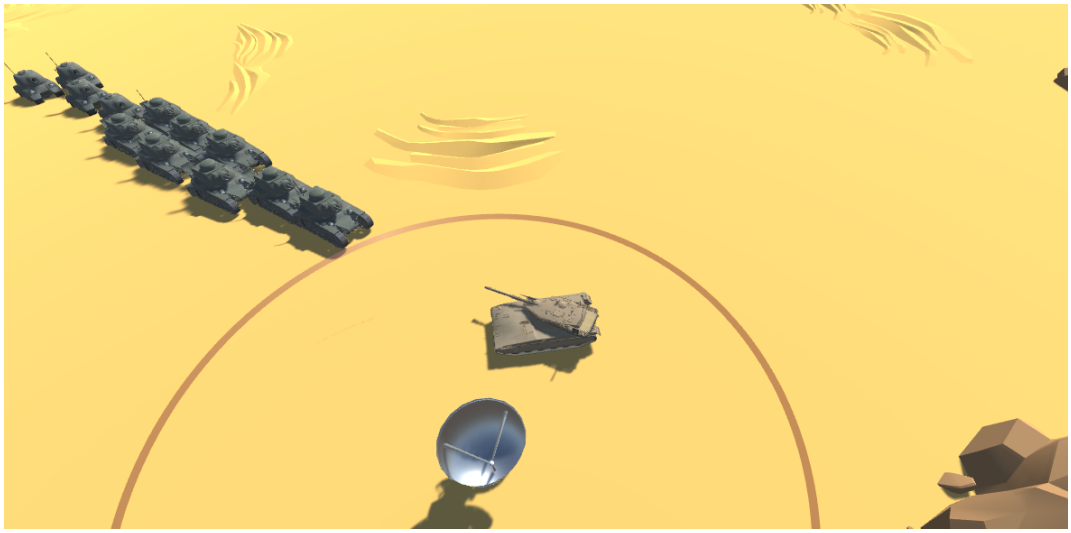

Figure 6: Image showing how the enemy tank changed mesh when moving the closer to the player tank.

Workload Distribution

Similarly to lab 2, this one was done individually on each group members' personal computer since it required Unity and we believed that everyone should get hands on experience operating Unity on their own system. Throughout the lab, we discussed the tasks and then tried different approaches individually, while sharing ideas and help. When everyone felt ready we compared and discussed our implementations, sharing screen and pointing out pros and cons with each application. The report was written together, where we discussed the content before writing it down.