Tank Assignment Report

# Patrol and Targeting

## 1.1 Implementation

For patrolling I use the following data to keep track:

* Two external CVector3 arrays called FrontPatrolPoints and BackPatrolPoints with a size of NumPoints
* An EPatrolType variable called m\_PatrolType which can be either ‘Front’ or ‘Back’
* An int m\_PatrolPointCounter set to 0
* A CVector3 called m\_TargetPoint

Depending on the state of m\_PatrolType tanks will set m\_TargetPoint to the m\_PatrolPointCounter index of the FrontPatrolPoints or BackPatrolPoints list. When its position is within 5 units of m\_TargetPoint, m\_PatrolPointCounter is increased by 1 unless it is equal to the last index of the array, in which case it is set to 0 and m\_PatrolType flips. Tanks move towards the m\_TargetPoint by increasing m\_Speed by m\_Acceleration, until m\_MaxSpeed is reached.

For aiming the turret and turning the tank I use the GetTurnAmount function, which calculates the dot product of the XAxis of the tank and the (target - tank position) to determine if the target is to the left or right of the tank (value > 0 if target is to the right). The function returns a positive or negative inverse cosine of the dot product between the ZAxis and the target. If the GetTurnAmount is greater than 0.1 the tank or turret with turn right by the respective m\_TurnSpeed or m\_TurretTurnSpeed, and if it less than -0.1 it will turn right instead. The gap between the turns leaves room for staying true to the current angle.

To determine if an enemy is within range to fire, I calculate the dot product between the turret ZAxis and enemy position, calculate the length of the turret ZAxis multiplied by the length of the enemy position, and return the inverse cosine of these values. If this angle is between 15 degrees in radians, then a recursive axis-aligned bounding box (from the position of the building) check is made using (enemy pos – turret ZAxis), adding the same vector each iteration to determine if the turret is facing the building. If all conditions are met the tank will fire a shell.

# 2.0 State Transition Diagram

## Diagram Description automatically generated2.1 Diagram

## 2.2 States and Messages

In this assignment messages are utilised to help perform transitions between the states of the tank, which can be sent by other tank entities or the system, to manage AI behaviour of the tanks. These messages may also pass important data between entities such as passing m\_ShellDamage variables between the firing tank and the tank that has been hit, to deduct health. States control the specific behaviour of each tank, making it easier to add new states without editing existing states and producing more reusable code.

# 3.0 Update functions

## 3.1 Single-Scene Update

To create a single scene update function, I would create functions to return an array of each entity type (tank, shell .etc) from the EntityManager. I could then iterate over these arrays using a loop and perform the specific behaviour required for the current state of those entities, this would require if-else statements based on the entities state. It would also require multiple getters/setters for many of the tanks members variables or to make them public which conflicts with OOP concepts.

## 3.2 Single-Scene vs Entity Update

Single-scene updating would be much slower than entity updates using the above method, this is due to iterating through the EntityManager hash table multiple times to find all entities of each type, then further iteration through each of the arrays for each entity type, whereas entity updating just iterates once through the hash table and call the update function. Single-scene may also produce many more function calls to get/set all the required member variables.

Not only this, entity updating is much more manageable, as the single-scene update function would become very long if more entity types were added, as well as requiring more iterations through the EntityManager. Of course, using public member variables can reduce function calls in single-scene updating, however this can create potential errors by accidental alteration of these member variables.