Artificial Intelligence for Simulation

Tank War Report

Introduction

Our group consists of six individuals, these are: Daniel, Nikesh, Theo, Phillip, Sam and Steve. We have been given a task based around a program provided where there are two tanks, both with a ‘DumbTank’ class assigned and we need to implement a new ‘SmartTank’ class using techniques we have learnt throughout the module. We are going to use a finite state machine to allow our tank to perform multiple different actions when required, allowing it to choose the best strategy available at the time. We hope to create several SmartTanks all with unique characteristics with each set of characteristics being tested to check how effective they are against other tanks.

Project Management

To help us manage our project we have decided to use the ‘Agile’ software development methodology this means that we will be doing each part of the project in short sprints with each sprint having its own individual targets so that we can set goals at different points through the assignment depending on what needs to be done and which parts we need to focus on. As well as this we plan to incorporate the scrum methodology which will allow us to work together to break down the end goal into smaller targets for each week organising what needs to be done. Since we had meetings at the start of each session this would allow us to keep track of how much progress we have made in each area as well as get feedback from other members of the group since each area would be assigned to specific people. For these Scrum meetings we would keep track of attendance as this allowed us to monitor who was coming to the meetings and keep up with what the others are doing.

Insert Scrum

To help keep track of who is doing what we used Github which allowed us to upload all the files so we could keep them in a single place whilst still all having access to them, Github also allowed us to create a projects page which would allow us to create each individual task and assign the relevant person to it whilst also tracking if it has been started, in progress or if it is completed. Github helped us with seeing what the other people have worked on and also what has not yet been started so that we could begin to work on those areas.

As well as using Github we created a Gantt chart as this allowed us to set goals early on for both what we wanted to finish and when we wanted it to be finished by allowing us to begin to split up the project into set goals with time-based targets for each of these which will allow us to complete the project in time for submission whilst giving each individual part of the project enough time.

Insert Gantt chart

Finite State Machine

Within our Finite State Machine, we plan on having multiple states which will change depending on when certain variables are met within the program which can include but is not limited to the health percentage of the tank or whether a target has been found.

Here are some of the notes we have made:

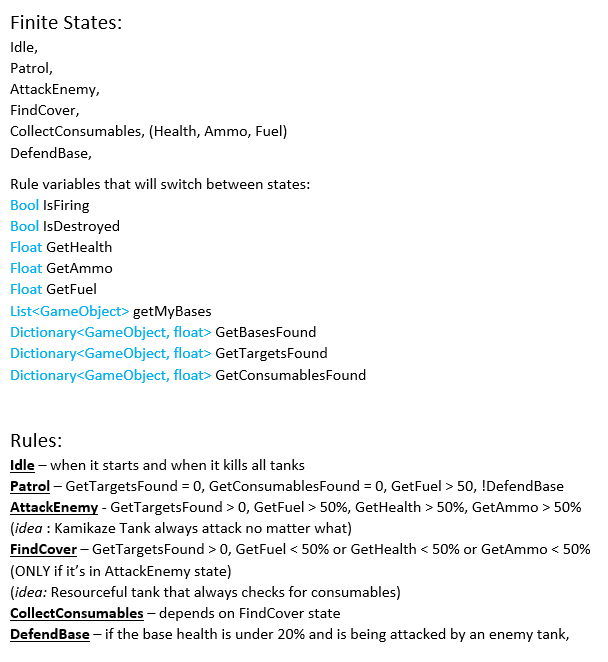


Fig [1]

We later changed the FindCover state to Flee as this made more logical sense due to the reduced amount of cover available on the map, fleeing was a more suitable option to try and get away from the DumbTank by creating more distance rather than trying to hide behind a tree or the rock.

Alternate AI States

We plan on adding alternative AI states with different rule sets in the finite state machine as this will allow us to test multiple different strategies out whilst seeing which one is the most effective, some of these ideas are:

Kamikaze Tank: Will continue to attack even if its resource levels are low. The AttackEnemy state rules (as seen in Fig [1]) would be changed so that GetFuel, GetHealth and GetAmmo are irrelevant and it just continues to attack no matter how low its resources are.

Resourceful Tank: Will always prioritise keeping consumables full. The AttackEnemy and FindCover state rules (as seen in Fig [1] ) would be changed so that the GetFuel, GetHealth and GetAmmo values would be set at a threshold of 120% whilst it is below this threshold it will Patrol until it finds consumables.

Defensive Tank: Will always be in cover, attacking when it is safe too, defending another tank (if multiple tanks on field). The FindCover state rule (seen in Fig [1]) would be the highest rule for this tank next being AttackEnemy (seen in Fig [1]) these rules would be switched depending on several dependences (in enemies’ line of sight, behind an object, if we have enough ammo and what our health is on. GetFuel, GetHealth and GetAmmo values would be at a threshold of 30% while DefendBase would be 70% so it would defend the base as it is created to defend/be defensive.

Tory Tank: Uses less fuel by staying idle when no enemy is found and

Testing

We have discussed how we are going to test our SmartTank script and we have all come into agreeance that what we will open the project on all of our computers and run our SmartTank script simultaneously 3 times each, this will give us a clear indication on how our smart tank performs as we can calculate our Win/Loss ratio and many other things such as the rate our tank attacks the enemy base etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **What we are testing** | **How we are testing** | **Result: Pass/Fail** | **How we can improve** |
| Idle State: |  |  |  |
| Patrol State: Patrolling random points | Running the program and testing each ruleset that sets the FSM to patrol | Pass | N/A |
| Patrol State: Transaction to attack Enemy [1] | Running program with Enemy tank in arena | Fail | Smart tank went into attack state but did not fire, modify state variables and retest |
| Patrol State: Transaction to attack Enemy [2] | Running program with Enemy tank in arena | Pass | N/A |
| Patrol State: Transaction to attack Base | Running program with Enemy tank outside arena so SmartTank can only go for Bases. | Fail | Smart tank did not attack base; revisited script and moved, so we changed the order of the variables within the attack function |
| Collecting consumables State: Fuel [1] | Allowing program to run, moved enemy tank out of arena so enemy can focus on consumables | Fail | The SmartTank does not go for the fuel and when it gets stuck in the collect consumable state |
| Collecting consumables State: Fuel [2] | Allowing program to run, moved enemy tank out of arena so enemy can focus on consumables | Pass | The Smart tank can collect the Fuel but the only issue is the A\* Pathfinding when the consumables spawn on the edge of the map. Both tanks get stuck and can’t collect it |
| AttackEnemy: No Enemies in sight switch to patrol | We get the enemy insight and then pause the game and then move the enemy away, hoping our SmartTank switches to patrol. | Pass | N/A |
| AttackEnemy: Goes to defend base if enemy bases destroyed and has enough health/ ammo [1] | We move the enemy tank outside of the arena so our tank has chance to destroy the enemy bases. | Fail | We changed some parameters and moved when change state is performed. |
| AttackEnemy: Goes to defend base if enemy bases destroyed and has enough health/ ammo [2] | We move the enemy tank outside of the arena so our tank has chance to destroy the enemy bases. | Pass | N/A |
| AttackEnemy: Goes to patrol base if enemy bases destroyed and hasn’t got enough health/ ammo | We let the game play out until our ammo and health is below what we specified, then we see what our tank will do. | Pass | N/A |
| AttackEnemy: ammo < 4 go to patrol | If ammo is below 4 then tank will enter patrol state and will not attack enemy. | Pass | N/A |
| Defend Base: Change state to patrol if both bases are destroyed or to attack if enemy is close to the tank | Let the game play out until the enemy destroy our base and then see if our tank goes into patrol if enemy is not near and attack if enemy is close | Pass | N/A |

We tested our project regularly by functionality testing each time we added a new state to ensure that it worked as required without having any errors, by doing this it allowed us to isolate any errors much sooner as it would be down to the most recent bit of code so we would go and fix the bugs before working on more code, this allowed us to make sure it continued to compile at each point rather than completing the code and discovering an error that has not been isolated.

We also used regression testing so that each time we began to work on the project we would run it again to make sure no new errors have been found before we began to update the code as this helped us isolate any future issues making it easier to fix them.