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 of which have a ‘DumbTank’ class assigned to them and we need to implement a new ‘SmartTank’ class using the techniques we have learnt throughout the module. We are going to use a finite state machine that will allow our tank to perform multiple different actions when required to do so, allowing it to choose the best strategy available at the time. We hope to create several SmartTanks all equipped with unique characteristics. Each set of characteristics will be tested to check how effective they are against the 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 individually set targets, so that we can set goals at different points throughout 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 receive 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.

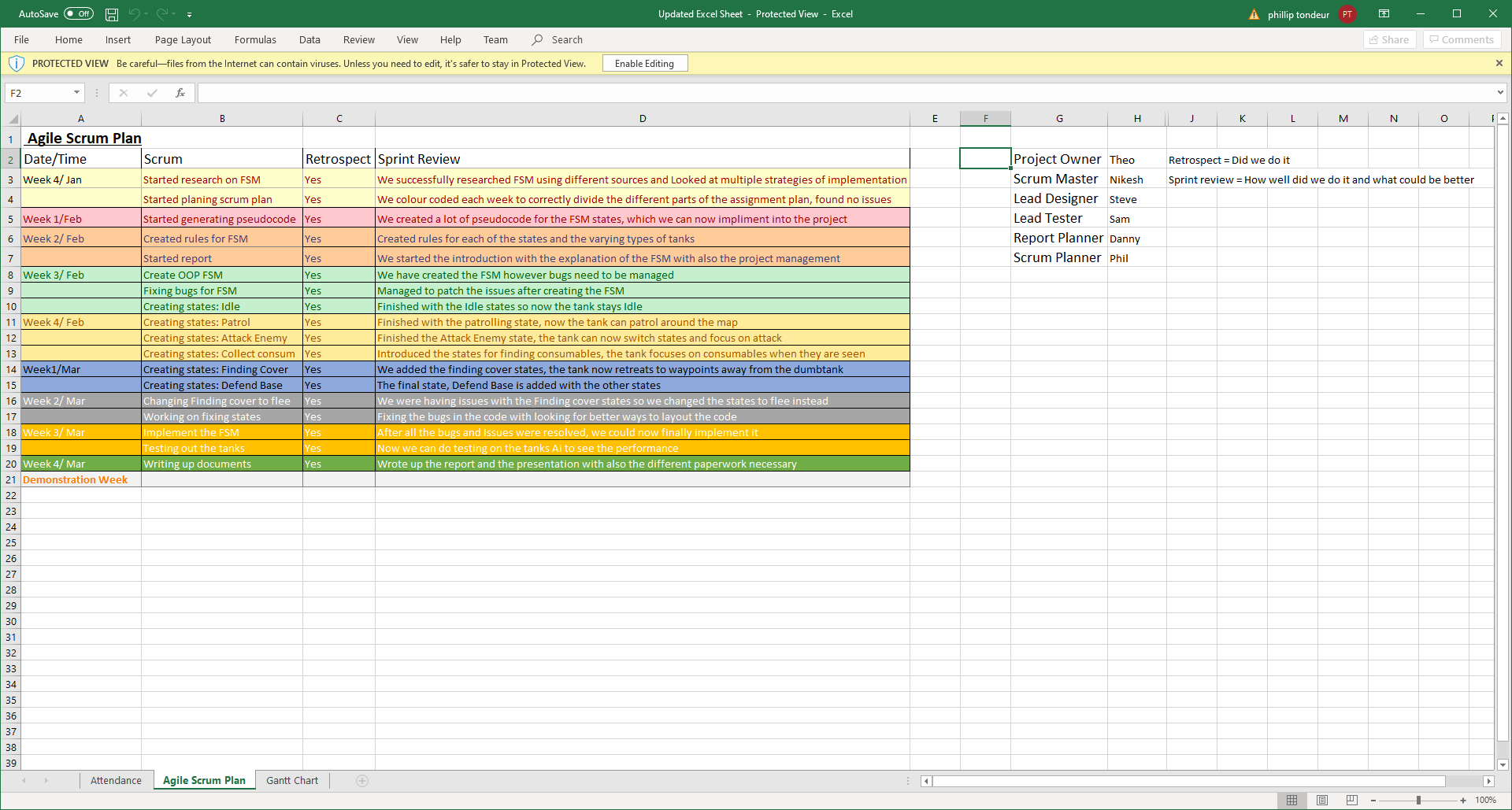


Fig [1] Scrum plan (PlanningSheet.xlsx/AgileScrumPlan)

To help us keep track of the working solution we used Github, which allowed us to upload all the files into a single repository so we could keep them in one place whilst still all having access to it, Github also allowed us to create a projects page which would let us 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 similar to a scum board. It also helped us with seeing what the other people have worked on and what has not yet been started so that we could begin to work on those areas.

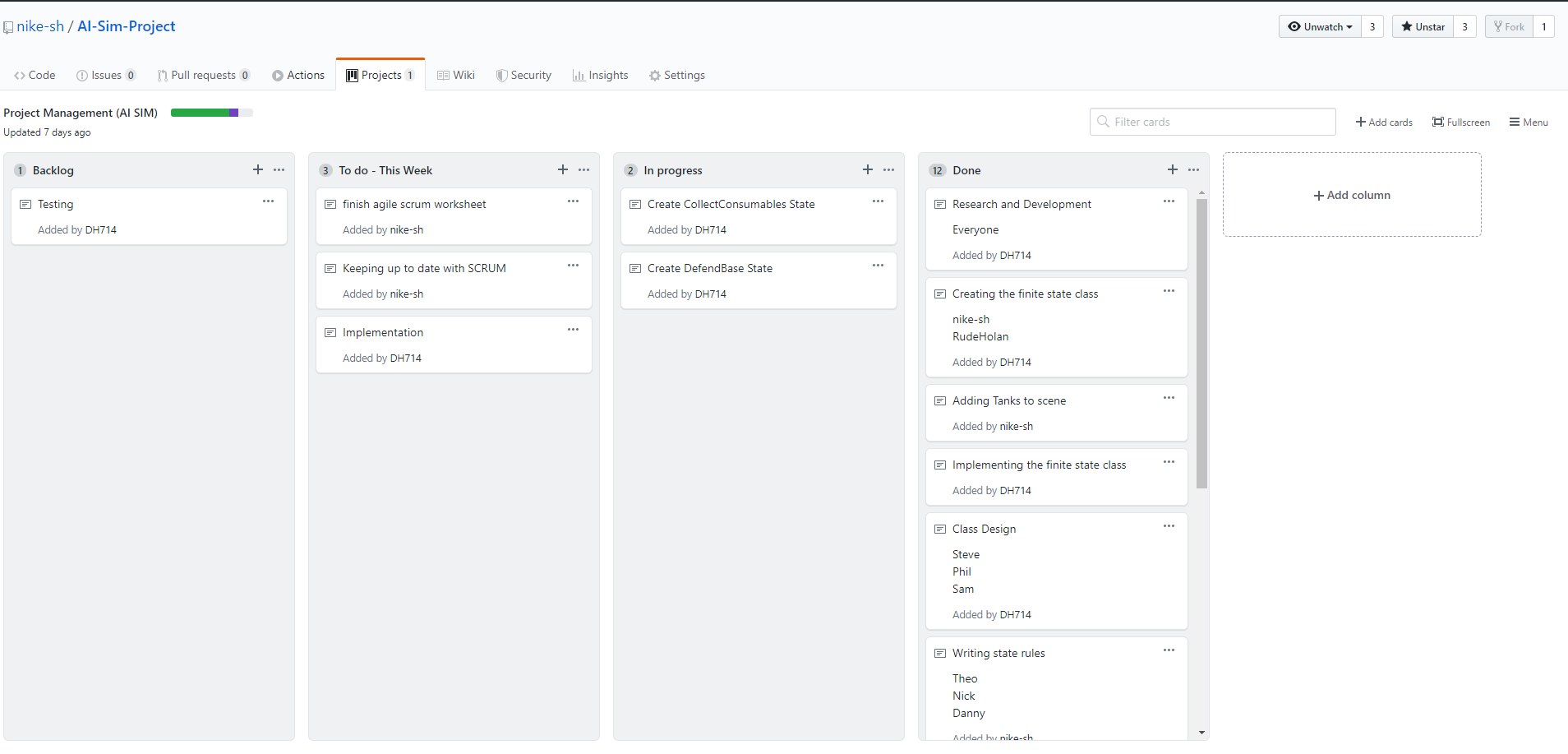


Fig [2] Sprint/Project board (<https://github.com/nike-sh/AI-Sim-Project>)

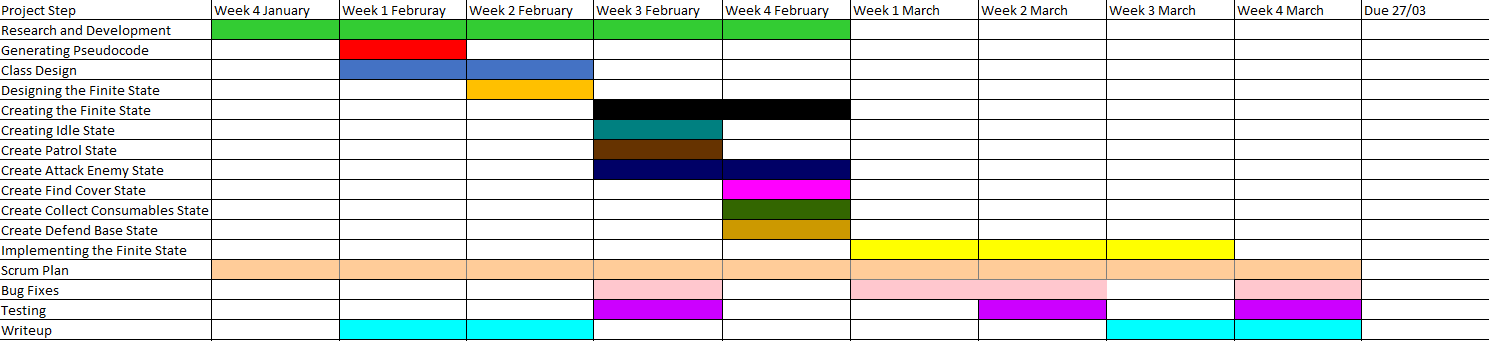
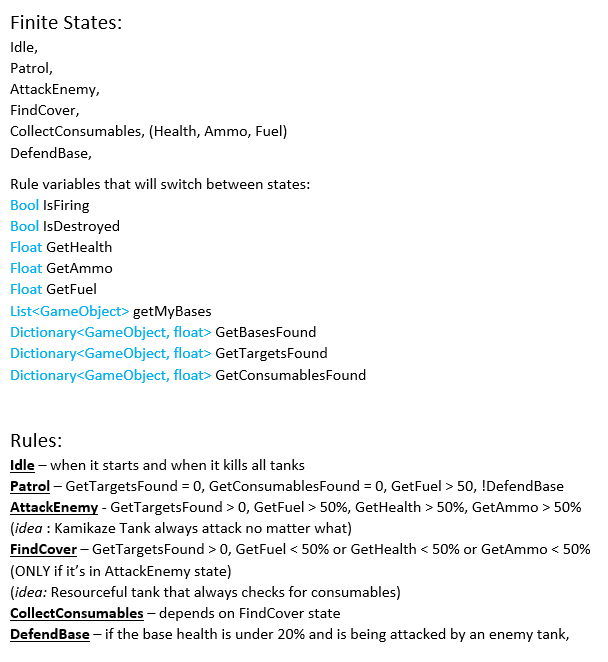
As well as using Github we created a Gantt chart as well, which allowed us to set goals early in the projects lifecycle to give us a rough estimate of time spent to follow and aim for. Meaning it gave us the ability to split up the project into more smaller set goals with time-based targets that each of the members had to complete and that made sure we were always on track and that we were able to complete the project in time for the submission date whilst giving each individual part of the project enough time to work on their tasks.

Fig [3] Gantt Chart ( PlanningSheet.xlsx/GanttChart)

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. Examples of these which can include but is not limited to the health percentage of the tank or whether a target has been found in the line of sight. Using these different rules, we can make the tank change its state to another state which would boost the efficiency levels of the Tanks performance.

Here are some of the notes we had initially made:

Fig [4] Finite State and Rules.doc

|  |  |  |
| --- | --- | --- |
| Current State | Transition | Next State |
| Idle | Game Start | Patrol |
| Patrol | Go To Random Point | Patrol |
| Patrol | Consumable Found | Collect Consumable |
| Collect Consumable | Consumable Collected | Patrol |
| Collect Consumable | Target Found | Attack |
| Attack | No Target or No Base | Patrol |
| Patrol | Base Found && Ammo > 4 | Attack |
| Patrol | Target Found && Ammo > 4 | Attack |
| Patrol | Our bases destroyed and health < 40 and ammo < 4 | Defend base |
| Attack | Bases Destroyed and Our bases > 0 and ammo > 4 and Health > 40 | Defend Base |
| Defend Base | Target Found | Attack |
| Attack | Bases Destroyed and Our bases > 0 and ammo < 4 or Health < 40 | Patrol |
| Attack | Bases Destroyed and Our bases destroyed and ammo > 4 and Health > 40 | Patrol |
| Attack | Ammo < 4 | Patrol |

Fig [5] – State Transition Table for updated State UML

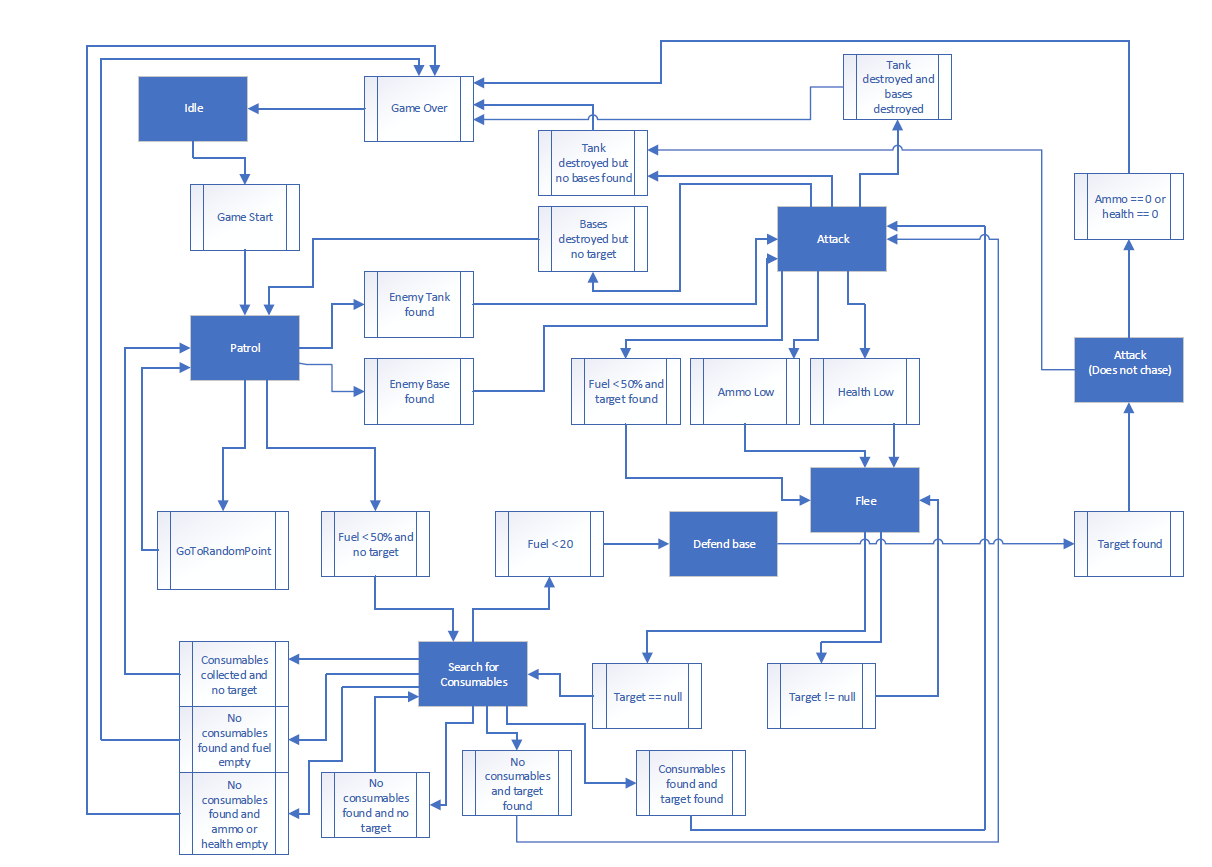
Fig [6] – Original UML diagram for the state machine

Fig [6] Original UML for Finite state machine (UMLDesigns/FSM-UML)

This was the original design for the UML Diagram or the foundation of what are UML looks like now. As you can see there are 7 states, including the Idle state, that show a basic understanding of which variables must be triggered for the AI to switch states.

The Idle state will be used for when the game is initially started and once the game has concluded.

The Patrol state will be used when it is searching for enemies or locating consumables and has not yet found/identified them. In the patrol state the tank will just continue to move around until a target has been found.

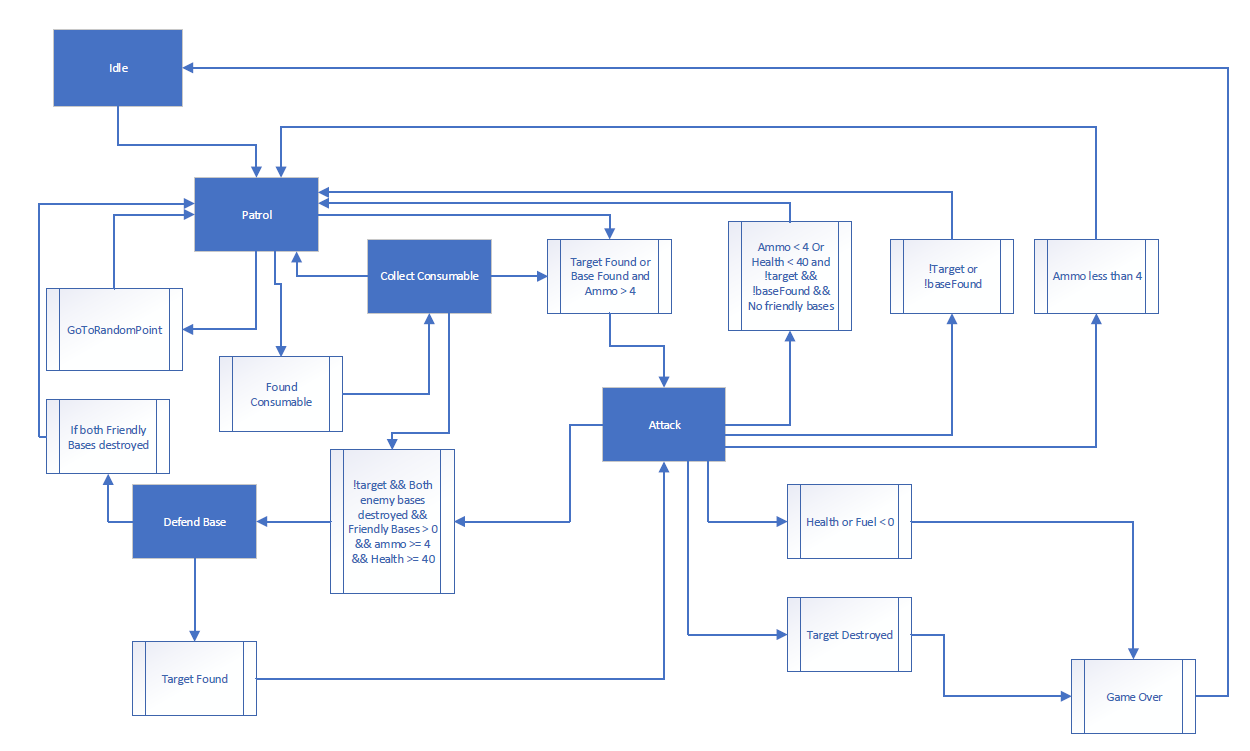
The AttackEnemy state is used when an enemy has been located but the tanks health, fuel and ammo are not low enough to begin to run. In the AttackEnemy state the tank will begin to move towards the enemy tank and once the tank is close enough, it will begin to fire until one of the rules are broken and then it will switch to a different state.

The FindCover state is used when the tanks health, fuel or ammo are low and in the FindCover state the tank will try to flee to behind cover getting out of sight of the enemy before then searching for consumables.

The CollectConsumables state is used when a consumable has been found in the Patrol state and it will then go pick up the consumable.

The DefendBase state is used when the SmartTanks base is under attack and the tank will return to its base in order to defend it from the DumbTank.

We later changed the FindCover state to Flee as this made more logical sense to our group. The reason behind this was due to the reduced amount of cover available on the map and this meant that 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 rock. We also discovered that it would make more sense for our tank to instead defend the base rather than just flee because if it flees that would leave the base open to attack.



This is the current UML Diagram and as you can see it has been worked upon significantly, with a much clearer view on where we would like the Ai to switch when the right conditions are met. We believed the states we have chosen are ideal, the reason for this is because the Tank can switch between the states without losing the functionality.

The Idle state is only present in the Game Over and when it switches to the Patrol state, that is it.

The Patrol state is being used here to if it finds a consumable it goes into the Consumable state; go to random points in the map or, when the variable that the AI finds the tank or base and holds more than 4 ammo, it attacks.

The Attack state contains many outputs, 3 of which is that if the ammo count is less than 4 it goes back to the patrol state, if the target or base is not within the AI’s vision then it will go back to patrolling and the same with if Ammo < 4 or Health < 40 with no enemy in sight. There are also two which output to a Game over either by killing the DumbTank and the other way is by having 0 Health or fuel.

The Consumable state as mentioned before is only triggered when a consumable is within the AI tanks view.

The Defend Base is used when both friendly bases are destroyed, and your tank health is less than 40 and ammo is less than 4.

Fig [7] Final updated UML for Finite state machine (UMLDesigns/updatedUML)

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 [4]) would be changed so that GetFuel, GetHealth and GetAmmo are irrelevant and it just continues to attack no matter how low its resources are. The idea behind this tank is it will always be attacking the enemy tank hoping to destroy it before it gets destroyed itself.

Resourceful Tank: Will always prioritise keeping consumables full. The AttackEnemy and FindCover state rules (as seen in Fig [4]) 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. This tank will aim to win by simply outlasting the other tank whilst also defending itself when attacked, this will be more effective after the later change where the tanks explode when they have no fuel left.

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 [4]) would be the highest rule for this tank next being AttackEnemy (seen in Fig [4]) 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%

Due to changes we later made to the finite state machine with no longer having the FindCover state and replacing it with the DefendBase state we will no longer have this initial idea.

Conservative Tank: This tank will always try to consume less fuel which would be done by the tank entering the Idle state when no enemy has found this would allow its fuel usage to go down so the dumb tank should run out of fuel sooner and then explode.

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.

When we tested the project, we would have each member of the group download the current repo and then play through the game three times which would allow us to find any bugs within the code, some of the bugs we found were when collectables spawns near a tree our tank would not go close enough to the collectable to collect it. In the first iteration of our code we found a bug where whilst the tanks were unable to drive into a rock they were capable of reversing through it instead and they would then get stuck. We also had a bug with entering the DefendBase state where the tank was not entering the state when the rule conditions were met, in order to fix this, we added a waypoint near our base so that the tank knew where it needed to go in order to get back to its base.

Conclusion

This Report highlights how we have coordinated this project, with the details of Scrum Plans and UML Diagrams to show our progress during the lifecycle of the project. In addition to this, there’s also valid testing with the inclusion of alternate solutions by having different AI tanks focus heavily on different things, an example of this would be conserving fuel, focus on attacking or focusing on defence.

If we had more time, we’d like to have spent it adding to the single smart tank by giving it secondary states, using the alternate smart tanks we made earlier. For example having the tank go into Kamikaze State when near enemy base and defensive state during the first 10 seconds of gameplay etc.