Artificial Intelligence for Simulation

# Tank to choose.

The tank you must choose is the Russia tank. This can be found under our folder titled:

RGLM -> IT'S DANGEROUS TO GO ALONE, TAKE THIS -> Russia.

Please do not take NO TOUCHY. This is not the correct tank.

# Introduction

The objectives of this coursework were to be able to create and implement a simple AI software system and to understand how the AI works. To do this, the task given was to create an artificially intelligent tank to fight other artificially intelligent tanks in a ranked tournament. We were told to organise ourselves in groups of 5 or 6, which for us, was easy to do due to our previous collaboration work from the first year.

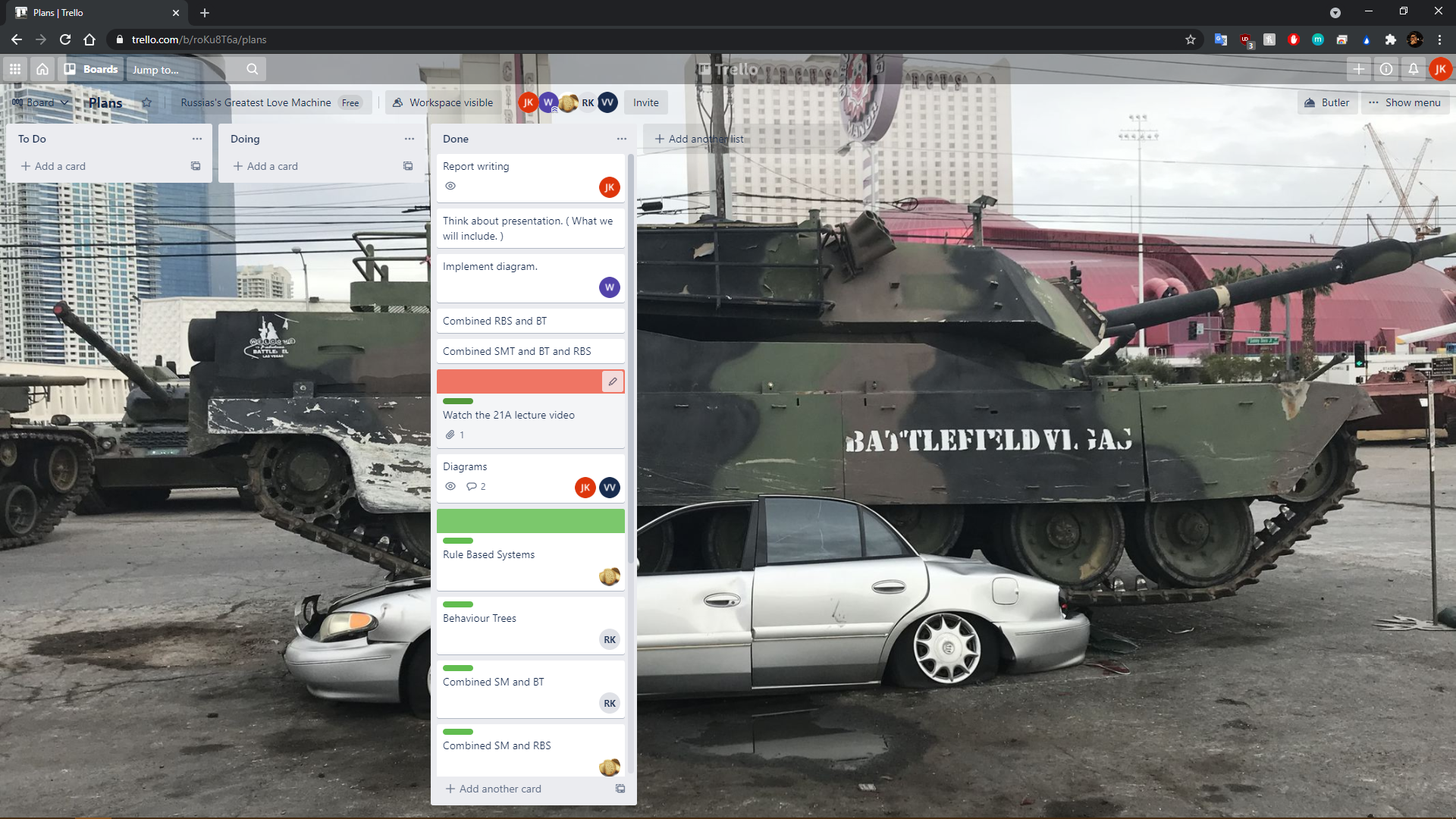
We decided very early on that we would like to be as efficient as we could be. We created a Discord group to be able to converse with one another and to be able to host meetings. We then created our repository on GitHub to be able to easily collaborate with each other and to be able to push changes to the projects with many problems. Romans set up the GitHub repository will all permission added for each user.

A screenshot of a computer

Description automatically generated with medium confidence

Completed GitHub Repository.

Next, we went with Romans idea of using Trello. We set up a Trello page where we could add tasks that needed to be done, assign people to those tasks to keep track of who has done what, check any changes that needed to be done as well as check the progression of the project. All these tools allowed the group to be as efficient and effective as possible whilst keeping disruptions and problems down considerably. This was all discussed in our first group meeting where we also discussed the questions featured in lecture 21b. A bonus of the Trello page was to allow us to keep track of new code that was being added to the project so if any problems were encountered, we would know when to roll back to.

Completed Trello Page.

# Team Name and Roles

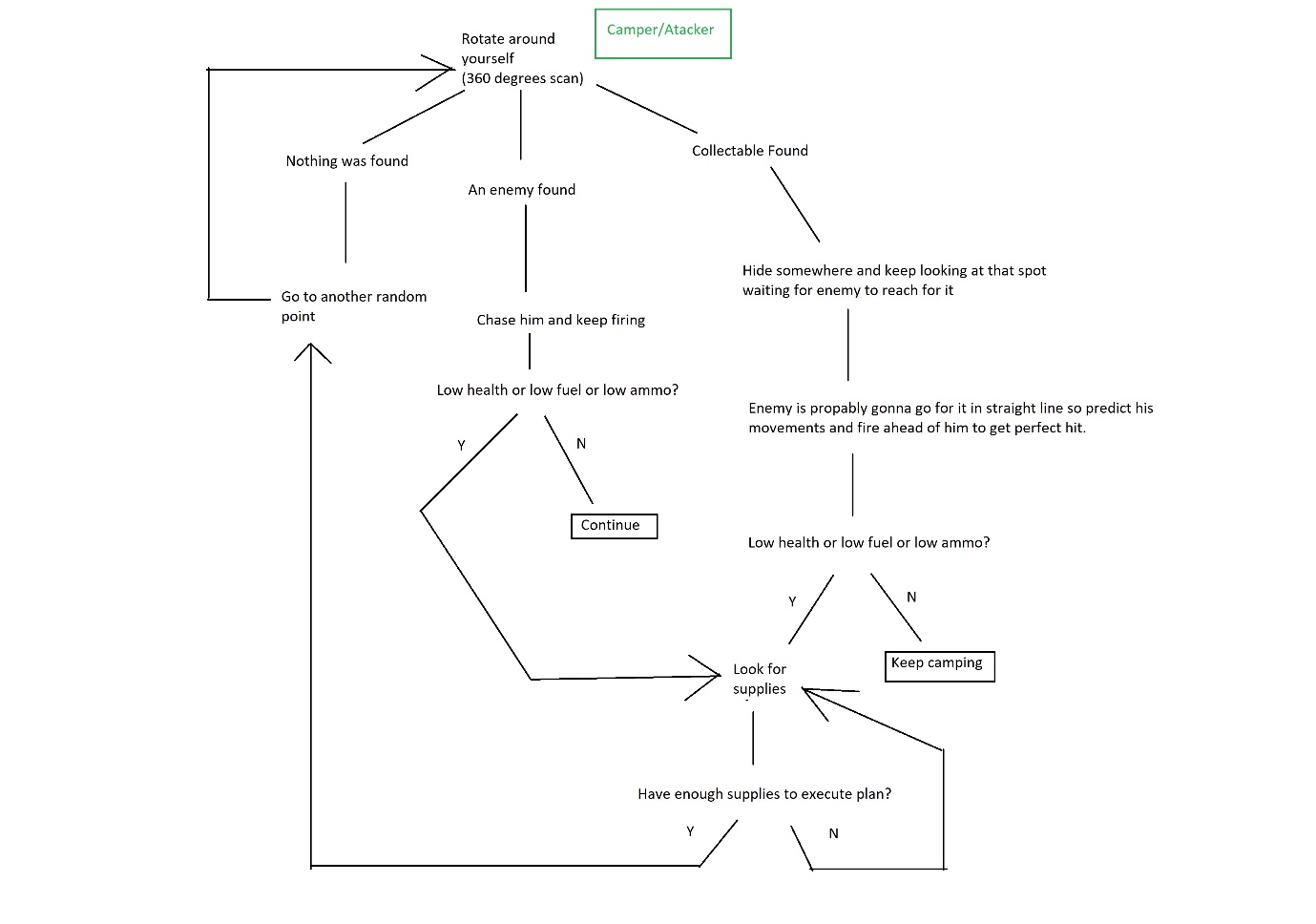
Once we formed the group, our first week was dedicated to creating the tools mentioned above. We had to decide on a group name and individual roles. Whilst on our first meeting call on Discord, we were throwing out names to see what everyone liked until Jai mentioned Russia’s Greatest Love Machine, named after the great Grigori Rasputin. Everyone enjoyed the idea of the tank being Russian, and so the name of the team was set. Next, we decided on the roles for the group. As the two strongest C# coders in the group, Oskar and Romans were given the roles of lead programmers with the rest of the group to assist them where necessary as programming assistants. Artur became lead tester and bug fixer. Jai was made the project manager to look over all the roles and was given the duty of creating the documentation. Vinushan began focusing on strategy for the tank as well as some of the documentation such as taking down minutes for meetings and after the project had been completed finishing the main report. We set up the first couple of to-dos on the Trello page and assigned everyone to watch all the lecture. We also decided then that a group meeting should be held roughly biweekly with more meetings closer to the deadline. We also gave ourselves an early deadline, approximately 2 weeks before final submission, so that if any problems were to arise, we would have time to solve them.

# Methodology

In addition, we decided to incorporate the extreme programming methodology with a few changes, notably the frequency of meetings. As mentioned before, we decided on holding biweekly meetings. These meetings would be held more frequently coming up to our set deadline. It was also suggested that Romans and Oskar would engage in pair programming. Before we came to a decision about the methodology, the group also explored options such as the Kanban programming standard and SCRUM method, but ultimately concluded that extreme programming is the best way forward. This was since some of the code was needed first whilst other could be done later as well as enable collaborations for more effective group work. Future meetings were organised in large part by Jai as the group leader, and this included setting the agenda for the group to give a progress update on each of their respective roles. These meetings were a good time to update Github, go through the Trello and add any more comments to certain items.

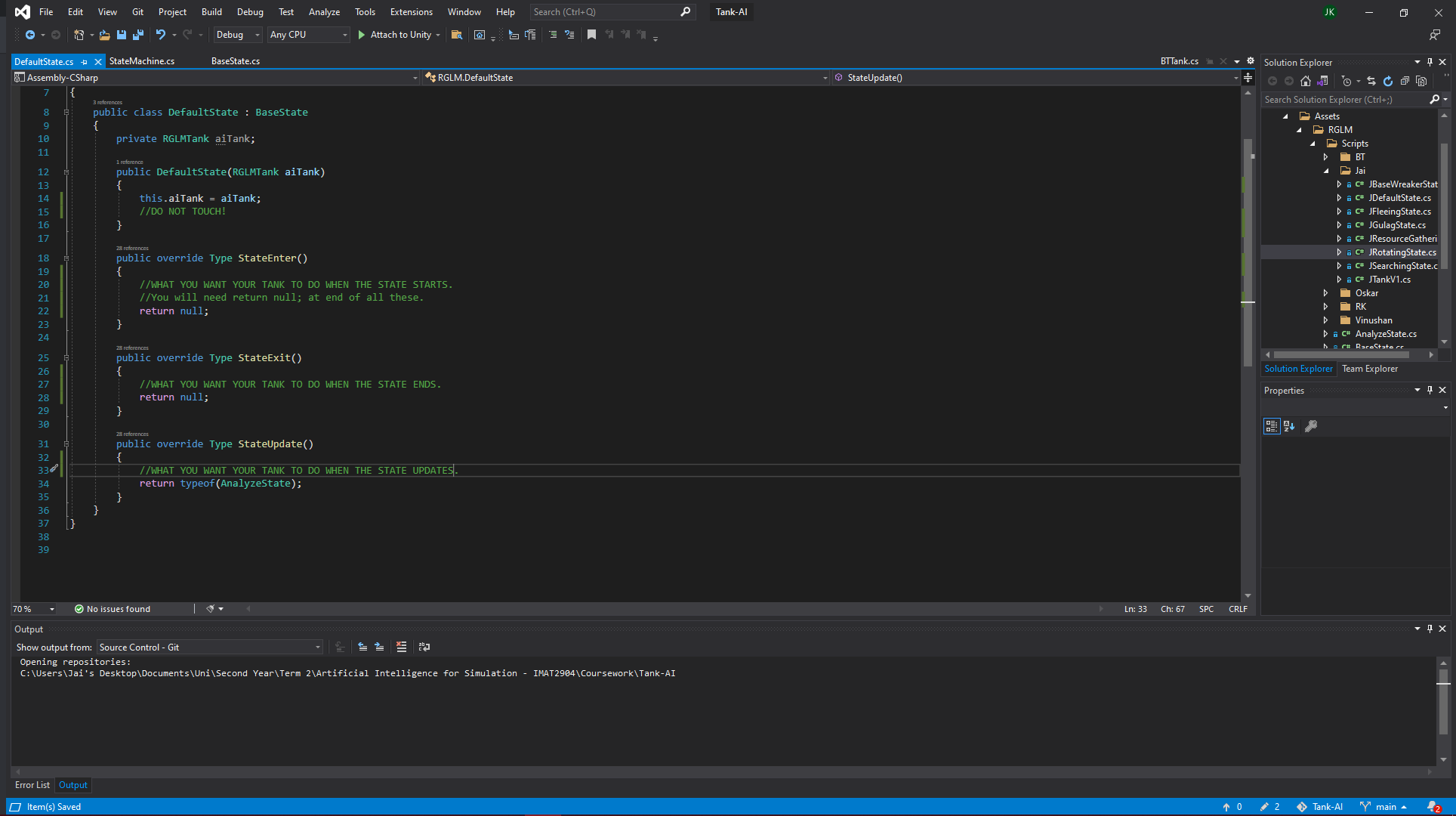
# State Machine

Once the lecture was released and everyone had had their lab session on finite state machines, we had a meeting to start to implement this in our AI. We first planned multiple strategies for our tank with this state machine in mind. This allowed us to further grasp the knowledge of what a finite state machine can and cannot do and help us understand what kinds of strategies we can make with just a simple finite state machine.

Initial base idea of tank strategy before splitting off to make our own strategies.

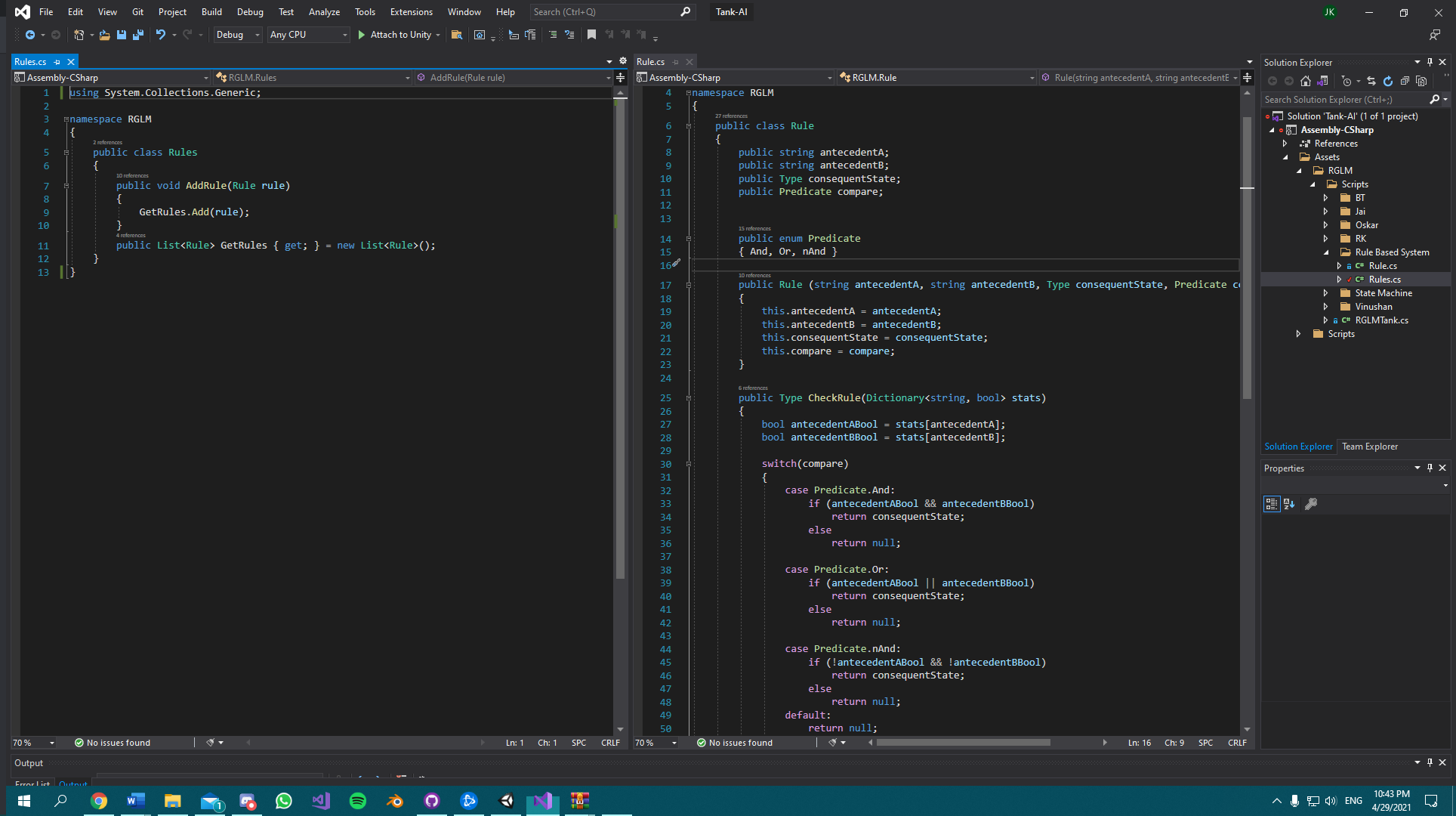
After some discussion, the group decided on a each making 2 strategies each with a vote in the Discord on whose tank ideas we would implement. We quickly narrowed that down to 4 separate diagrams and strategies ultimately deciding on which to use after all 4 have been coded and tested fully. This allowed us to see the number of states we would need and how to implement them as well as think about how to implement the rest of the diagram as code.

As one of the lead programmers, Oskar started on coding the basic elements of the state machine. The first was the virtual base class BaseState. This class has 3 functions that are all abstract and would be overridden in classes that inherited from this class and would serve for the base of all the code we individually made for our own implementations of the tanks. The base state works with the StateMachine class where all the logic resides. This allows for seamless switching between states and creating and setting new states. With this base, Oskar allowed the group to be able to code their respective state machines by giving examples for what is needed and where it is needed.

Example of default state by Oskar.

# Rule Based System

When the lecture for rule-based systems came out, we all saw how we could code the rules as well as the way rules are used to change behaviour. Oskar took initiative on this by using Salim’s code and tweaking it to suit our tanks. This allowed us to integrate the rule base systems with the finite state machine relatively easily and allowed for us to be able to show off more advanced techniques.

These classes allow base functionality for the Rule Based System.

# Behavioural Tree

Romans took lead when lectures came out for the behavioural trees. He managed to be able to code all the basic functionality of the AI technique. This allowed for yet more variations on what we had already developed up to this point. We managed to merge the state machine and the behavioural tree techniques together.

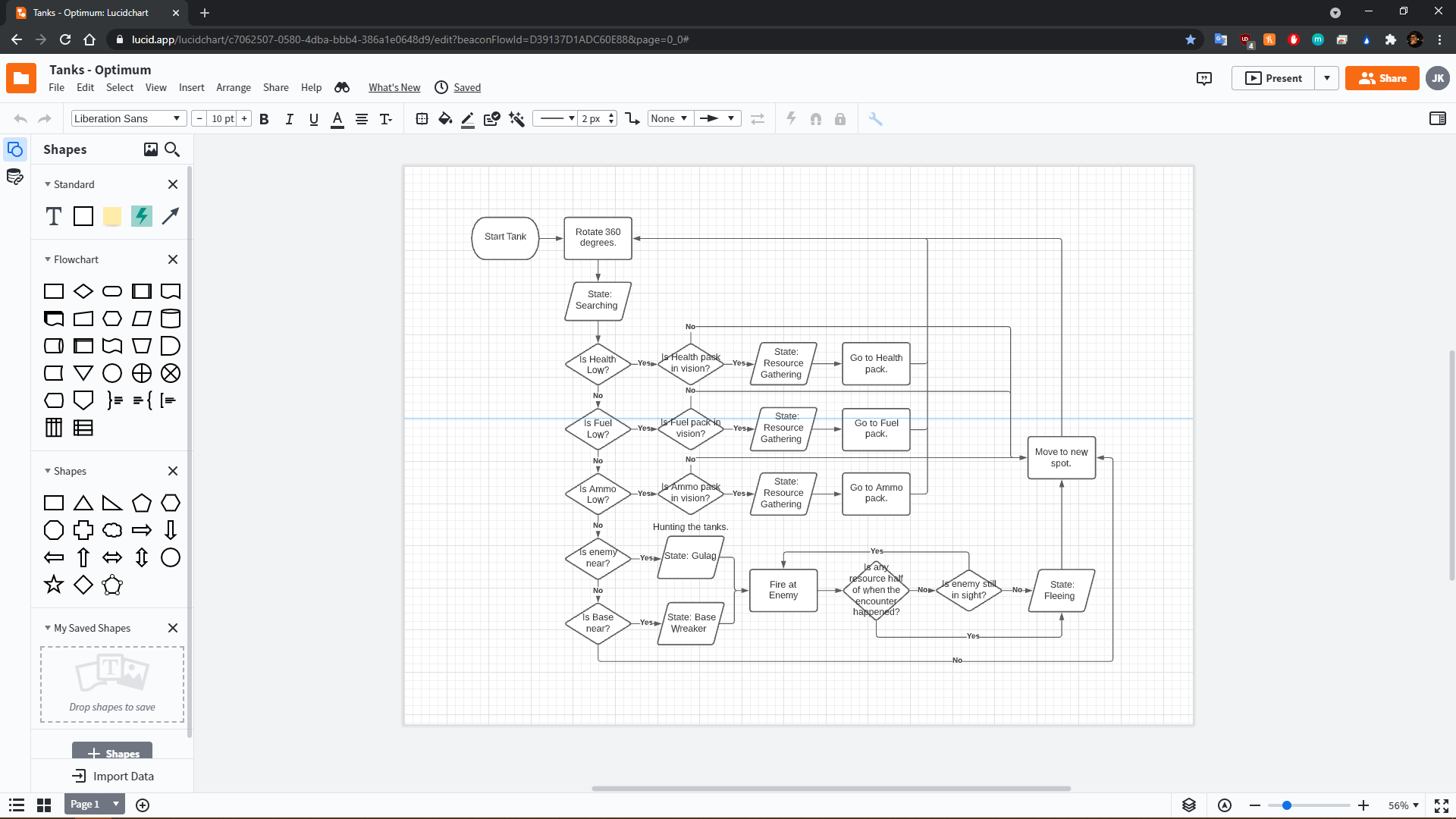
# RGLM Tank

Oskar, with the help of Romans, was mainly responsible for tank design when he made the new texture in surface painter. It underwent some changes but the main idea that the group wanted was to see a Russian themed tank so that it would suit the group’s namesake, and then it was implemented in the RGLM tank. We also decided early on to have some fun with this project and thus decided on putting wide Putin as our mascot on our tank.

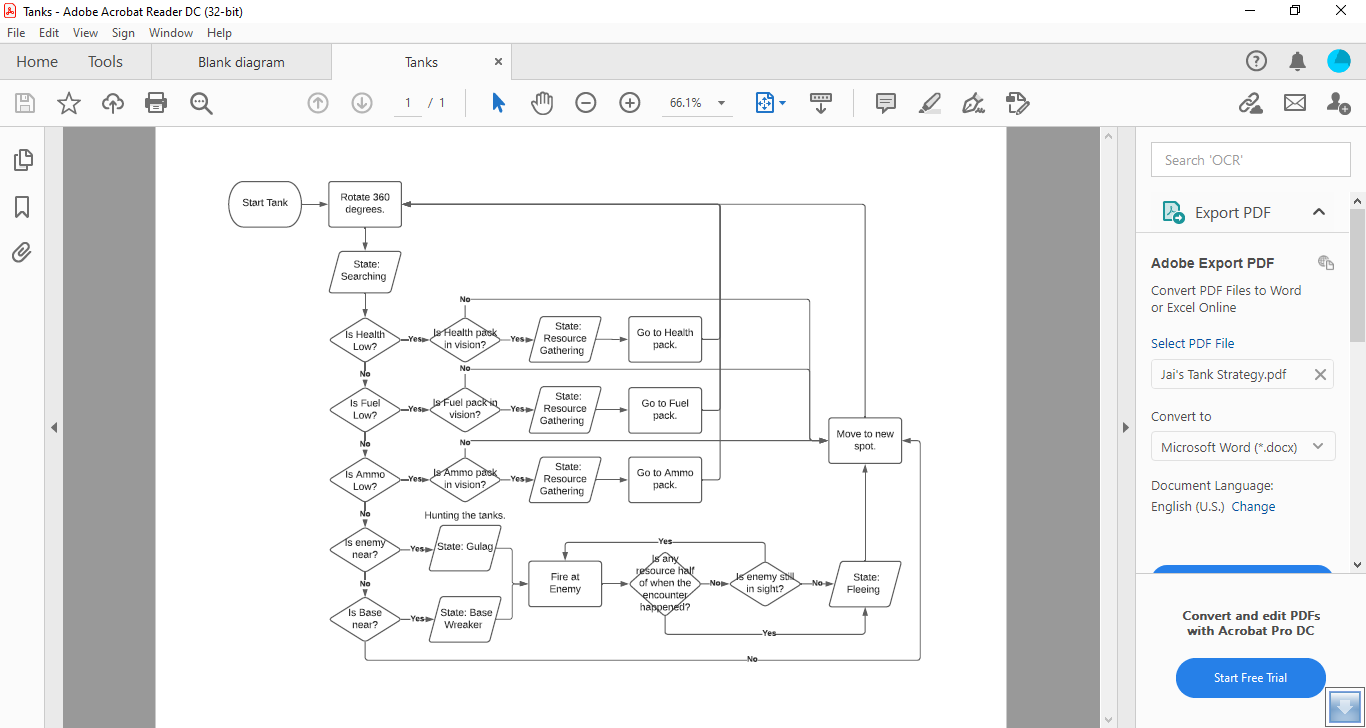
All group members helped with the coding of this main AI tank. Oskar was in large part responsible for setting up the class and functions that will eventually be inherited and used by the group for the rest of the project. Many of the code that could not be modified used protected functions. This meant that these functions could not be accessed by other classes most of the time, so Oskar re-wrote and made all the functions in the RGLM tank script which enabled the rest of the group members to use them in their states without any issues. He produced a pdf document of these functions to further assist the group. Oskar and Romans also created a function that rotated the tank by 360 degrees.

# Individual AI Tanks

Members of the group created their own separate AI tanks with a set of predetermined flowcharts showcasing the different strategies. We needed a set of AI tanks with different strategies to ensure that we can test to see how long each tank can survive in a survival setting. We used flowcharts created with the Lucidchart tool online to illustrate the AI strategies.

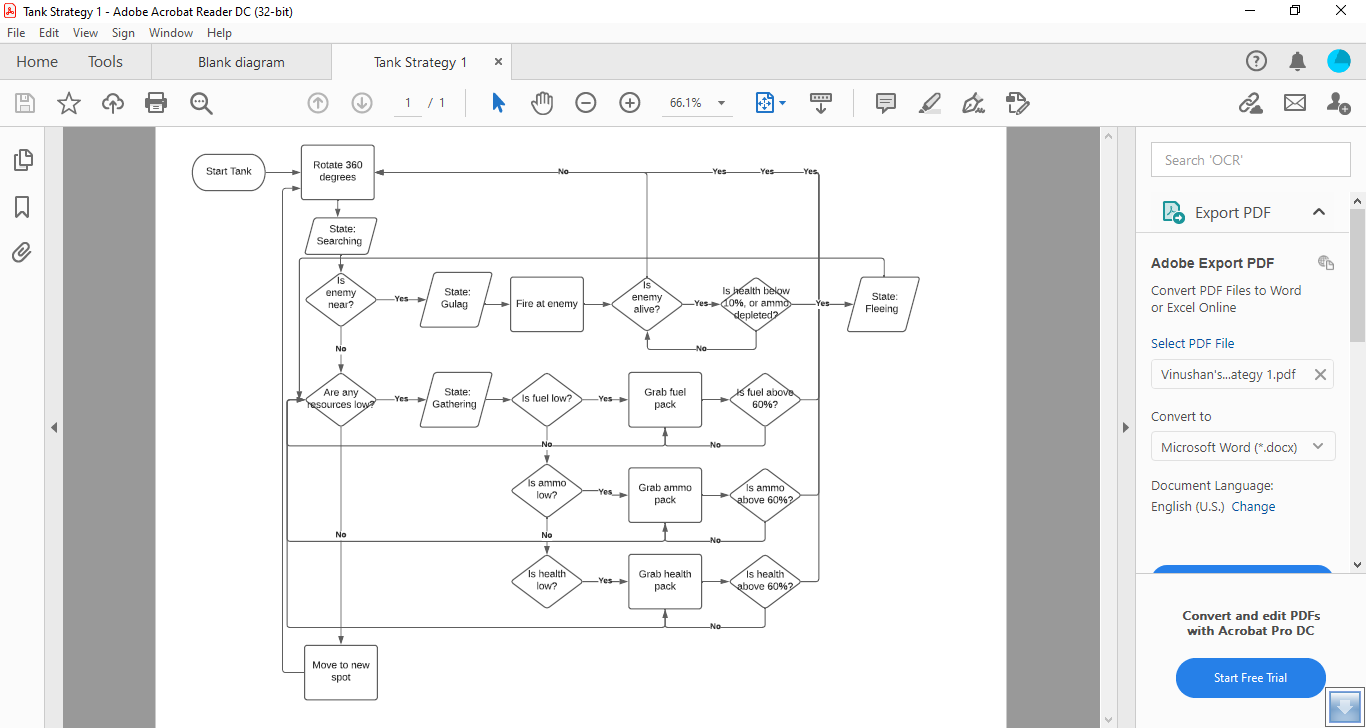


Jai created a strategy and began implementing his JTankV1 AI tank using a state machine. His tank will begin by looking for consumables in the ‘Searching’ state and change to a ‘Gathering’ state to collect any nearby consumable packs. If an enemy tank is near, it will enter the ‘Gulag’ state and start attacking. (Gulag was chosen here to score brownie points and the Russian aspect.) It will immediately switch to the ‘Fleeing’ state and move away to a new spot if resources are running low or the enemy is out of sight. This strategy ensures that the tank will only engage in combat when the enemy is in sight and will otherwise focus on survival to outlive the other enemy tanks. The primary goal of this AI is to maintain the resource levels and to flee as soon as one or more of the resource attributes are running low, which will increase the time at which this tank will stay alive in a survival setting. When testing this tank, the tank tended to win most of the time. We decided that this was the final tank, and this was our submission.



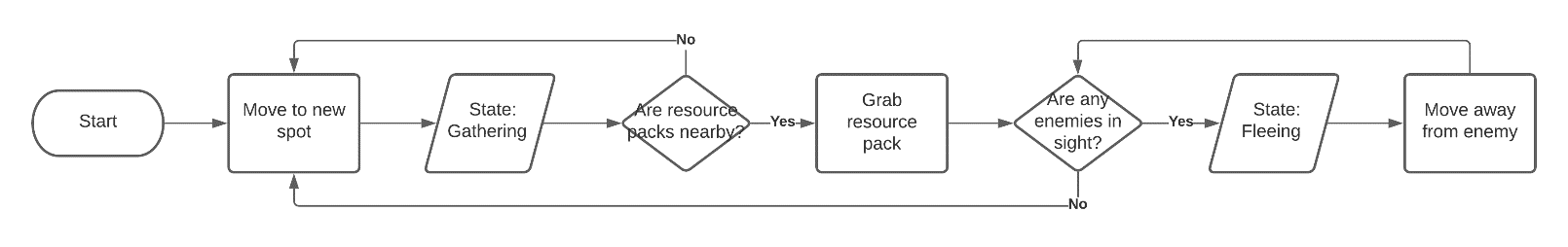
This is the flow chart for JTankV1.

Vinushan initially created 3 strategies for potential use by group members and decided to use strategy 1 to implement his VTankV1 AI tank using a state machine.



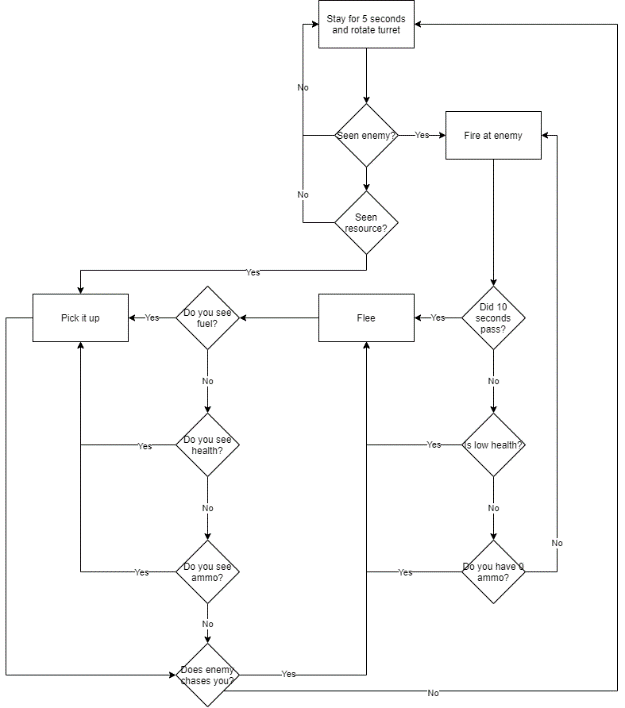
This is the flow chart for VTankV1.

However, in practice, it became clear that there were many similarities to Jai’s AI tank, and it was decided that Vinushan will create a new strategy with a completely different perspective on how to achieve victory. This was implemented using both state machine and rule-based system. VTankV1 was redesigned to never stop moving, collecting resources where possible and running away from enemies at double speed if in proximity. At first, the tank will always enter the ‘Searching’ state and continuously look for consumables, switching to a ‘Gathering’ state to collect them when they are near. If an enemy is sighted, the tank will enter the ‘Fleeing’ state where it will move away to a new area at double the usual speed until the enemy is no longer near VTankV1 and the tank will return to the ‘Searching’ state. This strategy has the main aim of surviving longer than all its enemy tanks by always replenishing resources regardless of the current levels and without ever engaging in combat. When testing, this tank tended to run out of fuel the quickest and thus would not work for this submission.

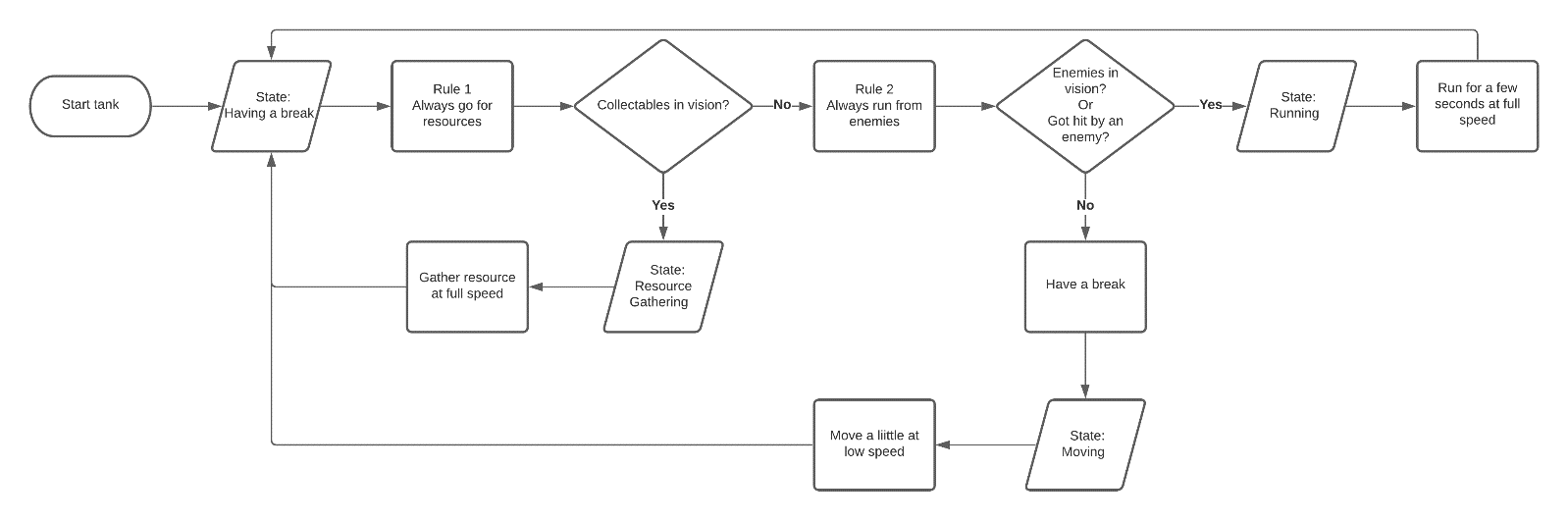


This is the flow chart for VTankV2.

Romans created both the BT tank and the state machine tank with a unique strategy. Due to many of the final chosen strategies were similar in many ways, his tank did a similar strategy to Jai’s tank with the Gulag state being handled by the behavioural trees. Whilst the tank won in situations, there were points when it would lose badly and thus was not chosen for the final tank.

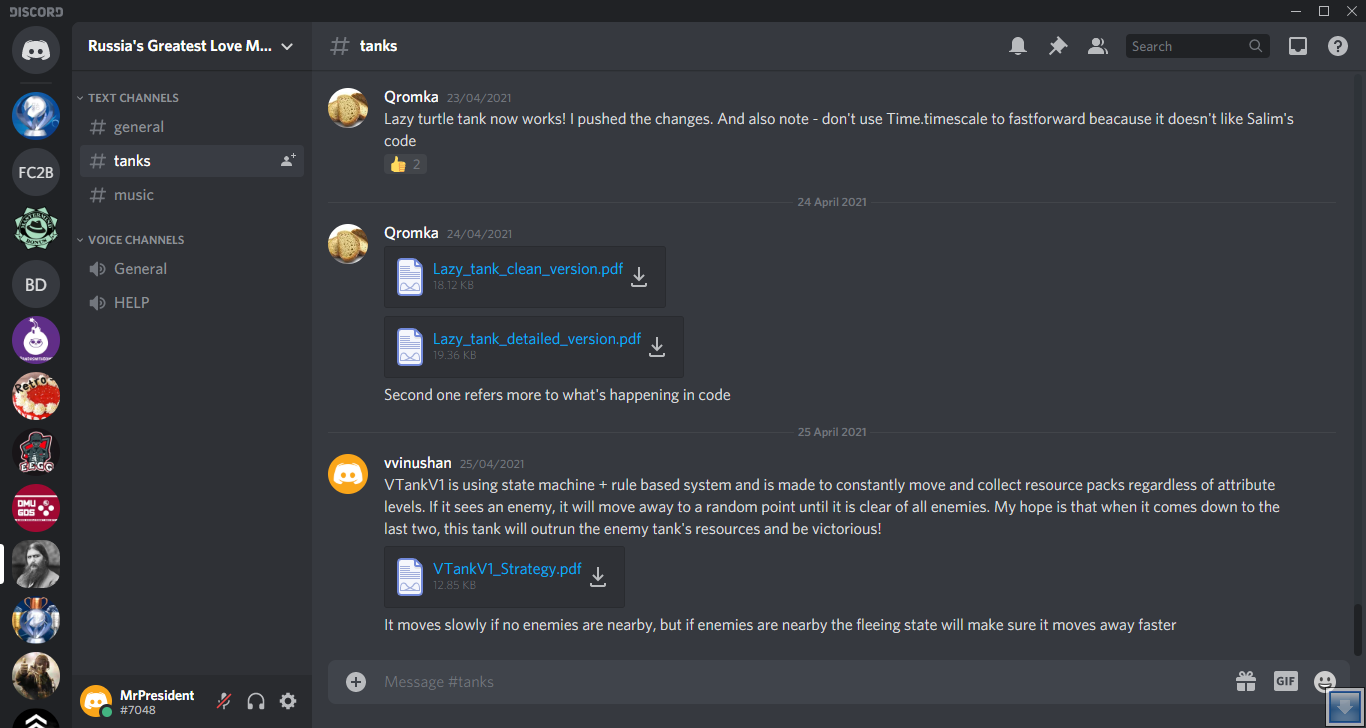


Oskar created a ‘lazy passive tank’ with a state machine and rule-based system. This tank is based on the approach of conserving resources and only moving if there is an enemy or consumables nearby. As it can be seen in the flow chart below, the tank takes a break almost always and it fulfils its namesake of being lazy where it will not engage in combat and focus on saving resources to be the last one standing. Although it had a couple of good runs, ultimately its downfall was the fact that it was too passive and thus would either run out of fuel or die from another tank.



Suggestions were made for Artur to use one of Vinushan’s strategies for an AI tank, but he ultimately assisted the group and tested all aspects of the project. This proved best for the group as he also assisted with many bug fixes and logic problems.

# Programming help

Oskar and Romans created a separate voice channel in the Discord server titled ‘Help’. This was used by all members of the group when assistance was needed with scripts and strategies and usually featured one on one support for everyone’s roles. Romans and Vinushan spent most of the time programming in their own time.

Oskar provided a demo to all group members on how to access functions in the RGLM class in the state machines, and the programming standard to be used throughout the project.

# Testing

All members of the group contributed to finding bugs in the code and testing out the tanks. Everyone worked on their own bug fixes, while Artur and Jai are responsible for most of testing the tanks and creating the test table. The test tables entries are for when the tank was failing. This allowed us to easily fix the small bugs that were present in the code.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tank1 Jai** | **What was expected** | **Was it working** | **What was wrong** |
| Fixed | Keep looking for resources | No | Lost track of base/stopped moving |
| Fixed | Gather resources when they are in sight | No | Lost resources position and started rotating for 1s then went to pick up resources |
| Fixed | Move to resource gathering state | No | Kept chasing enemy after running out of ammo |
| Fixed | Gather resources when they are in sight | No | Lost resources position and started fleeing |
| Fixed | Start rotating at the beginning | No | Tank would skip rotating state and immediately into searching state |
| Fixed | Attack enemy base after spotting it | No | Tank would notice enemy base and immediately lose track of it |
| Fixed | Look for resources after leaving gulag state | No | Tank would just stop in place |
| **Tank2 Oscar** | **What was expected** | **Was it working** | **What was wrong** |
| Fixed | Flee when getting hit | No | Tank would just stay in moving state |
| Fixed | Move at 0.1 of max speed when searching | No | Would move at max speed after coming out of fleeing state |
| Fixed | Flee after getting hit | No | Tank would stand still after getting hit |
| Fixed | Move to resource after spotting it | No | Tank would ignore collectable and run away |
| Fixed | Flee at max speed after spotting an enemy | No | Tank would stay at 0.1 of max speed |
| **Tank3 Roman** | **What was expected** | **Was it working** | **What was wrong** |
| Fixed | Go to fleeing/looking for resources | No | Crashes after rotating state |
| Fixed | Enter gulag state after spotting enemy | No | State was not initialized properly |
| Fixed | BT for enemy check should return true | No | BT would return false |
| **Tank4 Vinushan** | **What was expected** | **Was it working** | **What was wrong** |
| Not Fixed | Flee after getting hit | No | Tank would just stop after getting hit |
| Not Fixed | Flee for some time after spotting an enemy | No | Goes back to searching immediately after losing track of enemy |
| Fixed | Go back to slow speed after coming out of fleeing | No | Tank would stay at max speed all the time |
| Fixed | Keep searching after collecting the resource | No | Tank would go into fleeing after collecting the resource |
| Fixed | Keep searching after collecting the resource | No | Tank would stop after collecting the resource |

# Conclusion

Towards the final week, the deadline was extended by an extra week which allowed more time to finish and improve the project. The group decided to use the previous deadline as a ‘soft deadline’ to finish all scripts and testing, with the following week being used for completing documentation and bug fixing. This allowed the group to be well prepared in advance and begin preparations for giving a presentation on the final version of this project. All of us have worked well as a group and really enjoyed this project. We have all learnt a great deal about how AI techniques are used in games like Call of Duty Zombies and Ghost Recon. We feel confident that we would know how to implement a basic AI system in a game and which techniques to use in each scenario.