COMP222: Principles of Computer Game Design and Implementation
Second CA Assignment: Option 1
SESSION 2022
Bradley P Grose
ID-201611354

---

1. Behavior Model:

I designed my model to follow a decision tree type structure with various rule-based systems in it. In this model I unintentionally made it unbalanced, as a large portion of my root decisions will result in one action looping back to the start of the decision tree, however once you go a few layers down it starts to branch out into different pathways.

I picked this model as it would help me achieve my strategy goals of having a reaction to various actions, but that not dictating all future actions made by the agent. A majority of ideas I had to implement followed an if then else clause, making this the ideal structure to use as that is the root all the decision tree structure. This allows fast and reactive decisions to different attribute tests resulting in different actions and decisions.

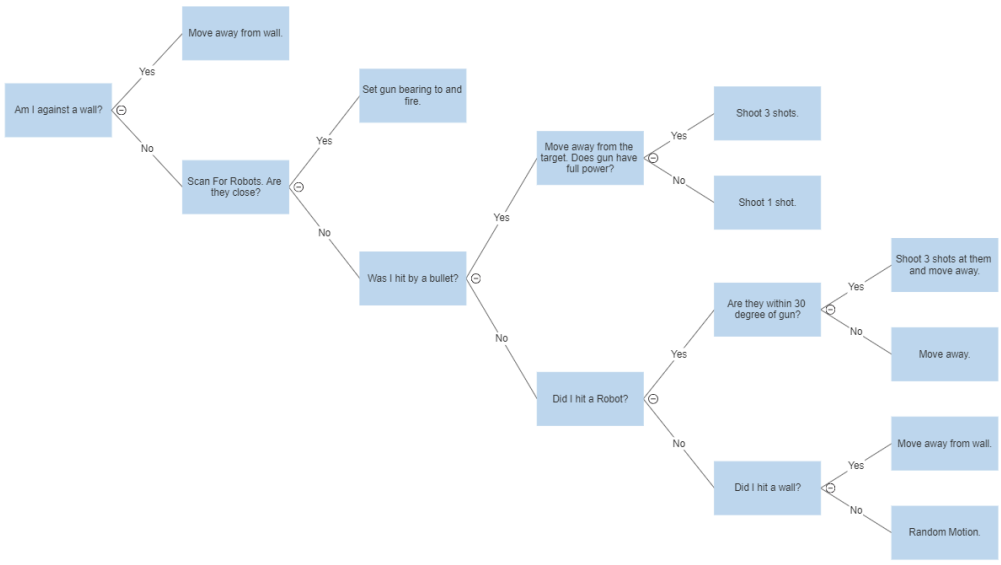
---

2. Design Description

Below is the heart of my decision tree for how my agent was designed and programmed to act in the battle simulation. The root of problems I ran into when testing was getting trapped against a wall, so my first check is getting off of walls or “trapped in a sense.” This will then result in moving away from the wall if that is the case. If not then the robot will scan for robots,

if the robot is close by then it will point the gun and shoot, if not it will keep going through its checks. I thought this would be taking a more conservative approach to fighting of not always wanting to engage in fights and rather wait until less teams exist and only flight if needed.

Following that, the robot will check if it was shot by a bullet, if it was it moves away from the agent that shot it, and then checks the power to send 3 or 1 shots. I selected this strategy to prevent multiple shots for hitting at once and following the more conservative avoid conflict approach. If it was not hit then it checks if it collided with a robot. If it did, the agent then checks how close the gun is to being pointed at the collision, if it is close then it will take three shots, if not it will move away. This strategy was chosen as if a robot was able to be used to have an easy shot on an enemy it will take it, however it allows it to also get away and then plan its next move. The final check is if the robot contacts a wall, which if it does it will move away from it, so it does not get stuck on a wall again. If all of these conditions are not met, then a random direction and distance is generated and the agent then moves that way, creating randomness to make it a harder to predict the motion of my robot.



3. Design Implementation

To implement my design, most of the code written implemented a hard coded in an ad-hoc manner following the if-then-else structure found in the design of the behavior tree structure. This is seen with the if then statements seen throughout the code as well as with the different robocode methods inherent in the library of advanced robot which I learned about through the API. Switching to an advanced robot inheriting from allowed me to implement non-blocking calls so multiple actions can be evaluated at the same time, such as motion and scanning.

In addition I also utilized the utils library for the relative angle functions allowing me to utilize that for when scanning for robots in the onScanned method to help aim at the scanned robot better using the bearings.

Most of the logic main ideas are based in call classes such as onScannedRobot, onHitBullet, onHitWall, or onHitRobot that allow it to evaluate classes if the event does happen, therefor not always running all the check code unless needed for a certain event. The only third-party library I utilized was the native java random function to create random motion through varying degrees to go in as well as randomizing the distance for the agent to move to get to a new spot, making it harder for other agents to hit the robot, hopefully.