# COMP222 – Assignment 2 (Robocode Tank)

## Behaviour model description (FSM)

My robot follows the AI behaviour model of a finite state machine. The finite state machine is an abstract model of computation meaning that it is a theoretical model of a system. Finite state machines have a set of states, starting state, inputs and transition functions. This allows the system to transition between states. The state machine can only be at one state at a time. State transitions don’t always have to transition to different states too, you can carry out actions in a certain state which might have a side effects, but it won’t necessarily change its state.

## Design Description

My robot has 2 main states: scanning and aiming. The scanning state is the first state where it will continuously spin until it has found a robot. If it finds an enemy, it transitions to the next state, aiming. Aiming doubles as the attacking state and will be in this state for as long as there are other robots in the match. As it is aiming, the robot will always move.

The robot uses random movements as an evasive manoeuvre to dodge enemy tank bullets and it’s fighting mechanism fires bullets at a predicted path of enemy tanks. My robot reacts to damage and collisions by switching directions of movements in battle, if it is heading towards bullets or into walls, instead of heading straight into danger, it can turn back the way it came. The movements made in the aiming state will randomly make large or small movements. Periodically, it can also change the direction the robot moves in too. Direction doesn’t have to change only when taking damage and the robot will change direction randomly.

As you can see below, I have created a finite state machine which explains how the robot transitions between states and how it reacts to the environment.

## C:\Users\Olee\Downloads\Process Map (1).png

I chose this method because it focusses on evasion and attack. Unpredictability is good as other robots won’t know where to shoot. I think my model will be successful because no matter how intelligent another bot’s AI is, it won’t be able to hit every shot on target if the target’s movements are 100% random. I don’t think that my robot will win the tournament with randomness alone. I do however think my robot’s attacking mechanism is strong enough to beat a lot of robots one on one. I don’t think the robot will be consistent enough to win every game because of the randomness, perhaps the robot could be unlucky. This is because the robot will sometimes be able to evade bullets and sometimes, it will not.

## Implementation description

The robots initial starting state initialised the robots, I did this in the ‘public void run()’ function. I then also set the guns and radars to not turn with the tank because it allowed me to handle those guns individually. As my tank would be moving and turning a lot, it gave me a lot of freedom to move without jeopardizing my aim or accuracy

I imported Robocode.util. This was very important as it allowed me to get the relative angles of the bot.

import robocode.\*;

import robocode.util.\*;

My robot must react to its environment, in order to survive. I used the standard built in methods with Robocode where you’re ‘hit’ by something. To react to the environment, I made the robot so that whenever it takes damage, it changes the direction on the robot. These functions are shown below:

public void onHitByBullets()

public void onHitRobot()

public void onHitWall()

The robots scan function is the main function. If there are robots scanned, it moves, aims and shoots in the main function the robot will always scan for other robots using the following code:

// Turns the radar to the right infinitely so it always has someone to shoot at

while (true) {

    turnRadarRightRadians(Double.POSITIVE\_INFINITY);}

When a robot is found, this function will run:

public void onScannedRobot(ScannedRobotEvent e){}

Firstly, the robot starts by getting the angle of the other robot relative to the robot’s current heading.

double angle = getHeadingRadians() + e.getBearingRadians();

// Conserving energy on turning

double energyConserved = Math.min( 3, getEnergy() / 10 );

// Amount enemy can move in the time it would take for bullet to hit

double movementPossibility = Math.asin(8.0 / ( 20 - ( 3 \* energyConserved)));

The gun then aims in a direction that they could be heading, there is an element of randomness to this and even when they’re stood still it doesn’t always hit.

// Amount to turn the gun

double angleToShootAt = Utils.normalRelativeAngle(

        angle - getGunHeadingRadians() + movementPossibility / 3 );

normalRelativeAngle, got the angles relative to where the robot is currently looking. Again, there is a prediction of where the robot is with this function. And it considers the movement possibility of where the robot is going and shoots in that general direction.

My robot would also turn perpendicular to the target so when the robot moves forward, it could perhaps randomly move forward in attempt to ram or even get closer to enemy tanks. This code is shown below:

// Turn perpendicular to target

setTurnLeftRadians( -90 - e.getBearingRadians() \* direction);

Afterwards, the gun would then take aim of the robots:

// Aims the gun towards the angle

setTurnGunRightRadians(angleToShootAt);

Then my robot would fire and fire to conserve energy.

// Fire, using less energy if we have low energy

double conserve\_energy = getEnergy()/10;

setFire(conserve\_energy);

To make sure that we’re still locking onto the target tank, we use the following code:

// Gets the angle relative to the direction that you're currently facing

double rel = Utils.normalRelativeAngle(angle-getRadarHeadingRadians());

// Lock on the radar

setTurnRadarRightRadians(rel);

Finally, the random movements are done as follows:

// Randomly move

setAhead(100\*Math.random()\*direction);

// Randomly turn direction

if(Math.random()>.9){

direction = -direction;

}

The direction variable will always be 1 or -1. This line: 100\*Math.random()\*direction allows me to head in the direction that direction is. If direction = -1 then it moves backwards if it’s positive, it moves forward. When the robot takes damage, it will also flip direction in the same way:

direction = -direction;