Edith Cowan University

CSG2341
Intelligent Systems

Assignment 1B

Saucers Part 2

Martin Ponce
Student 10371381

Tutor: Philip Hingston

October 1, 2015

Contents

1 Introduction

This report examines fuzzy logic using Sugeno style inference, and its practical use in a video game called Saucers. Saucers is a multiplayer game where each player indirectly controls their flying saucer using a fuzzy logic controller. The saucers meet on a battle space, a rectangular xy plane, with the purpose of destroying each other. The walls of this space cannot be travelled through, and will cause the saucer to ricochet off the wall when hit.

Up to twelve saucers spawn at the start of the game, and each saucer begins with equal amounts of energy. This energy is consumed as they fly around, fire their auto aiming cannon mounted on a rotating turret, or use their shield to protect themselves from energy blasts. When a saucer is hit by an energy blast, energy is depleted. The amount of energy depletion depends on the amount of energy committed to firing, and how far away it was fired. Energy blasts lose energy the further they travel.

Saucers cannot stop flying and will always consume energy. However, the speed of a saucer can be controlled. The slowest speed consumes the least amount of energy, while the fastest speed consumes the most. The saucer's heading can also be controlled, and can turn left or right in any direction. There is however, a small energy penalty for turning.

Each saucer is equipped with multiple sensors to provide inputs for fuzzy logic. There is a sensor for all current enemy saucers, providing information about their distance, their direction in relation to the player, their current heading in relation to the player's heading, their current speed, their current energy level, and their ID number. Similarly, there is a sensor for power ups that randomly appear in the battle space, providing an energy boost to the first saucer that touches it, and a sensor for all energy blasts. These sensors provide the same information as the enemy saucer sensor. There is also a sensor providing information about the player's own energy level.

When a saucer loses all of its energy, it disappears from the battle space and loses. Each game round has a limited time, and if multiple saucers are still alive at the end of the round, it results in a tie for that spot in the ladder. The goal of this report is to design a fuzzy logic controller so that it's saucer will be the last saucer remaining during the end of a game round.

2 Idea

The main tactic for this controller is fly defensively in order to conserve energy and survive until there are a two enemies left. Turns for the most part will focus on the energy blast sensor, in order to dodge as many of them as possible. When there are only two enemy saucers left in the arena, turning will focus on the enemy saucer sensor, to track them down and shoot at them from a close distance.

However, when any power ups spawn nearby, the goal is to move straight to the power up, ignoring any energy blasts. If any close energy blasts close to the player while attempting to retrieve a power up, the player will deploy the shield. The rate of fire will be kept to a minimum for most situations to conserve energy, unless a nearby power up spawns, or if there is only two remaining enemy saucers left. In these cases, the rate of fire will increase, to either deter enemy saucers from

retrieving the power up, or attempt to destroy them.

Speed will also be kept at the minimum for most situations, only increasing speed when necessary for dodging, or when a power up spawns nearby, to try and get to it first. Speed will also be increased when there are only two remaining enemy saucers left, attempting to destroy them before the timer runs out.

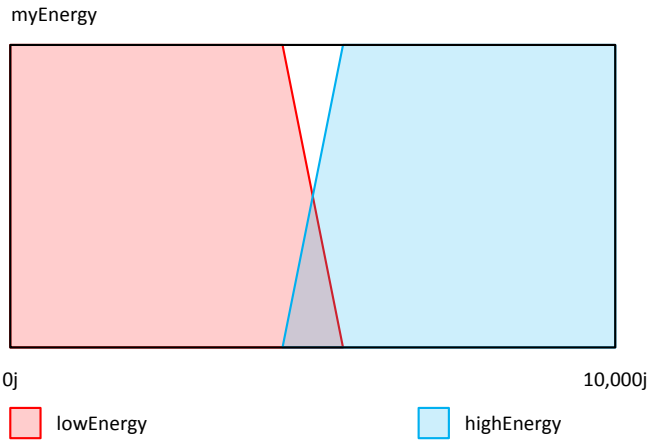
This controller attempts to implement these strategies with turning, speed, shield deployment and firepower, with the primary goal of flying defensively by dodging energy blasts and retrieving nearby power ups. When there are only two enemies left, the saucer will begin to fly offensively, increasing speed and firepower, and attempt to destroy the enemies before the timer runs out.

3 Input linguistic variables

3.1 myEnergy

The linguistic variable *myEnergy* is the player's energy level and determines whether or not the player has *lowEnergy* or *highEnergy* remaining. The universe of disclosure for *myEnergy* is between 0 joules and 10,000 joules, the amount of energy that all saucers begin with.

Figure 1: *myEnergy* fuzzy sets



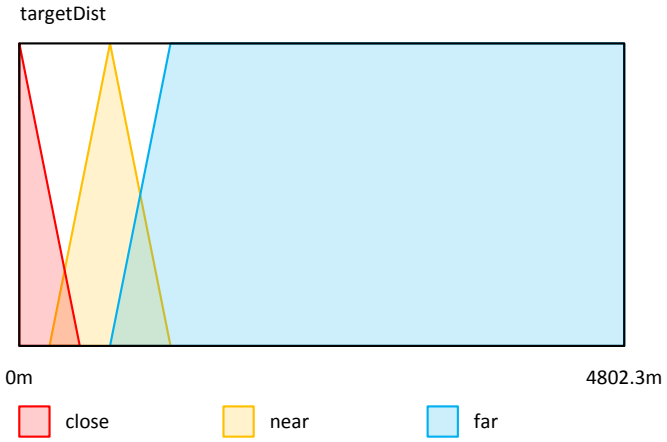
3.2 Target variables

The sensor returns a list of all the enemy saucers currently in the battle space. This controller only considers the closest saucer as the target, and ignores all other saucers in the list.

3.2.1 targetDist

The linguistic variable *targetDist* is the distance from the player to the target. The universe of disclosure for *targetDist* is between 0m and 4802.3m, and the fuzzy sets are defined as *close*, *near*, and *far*.

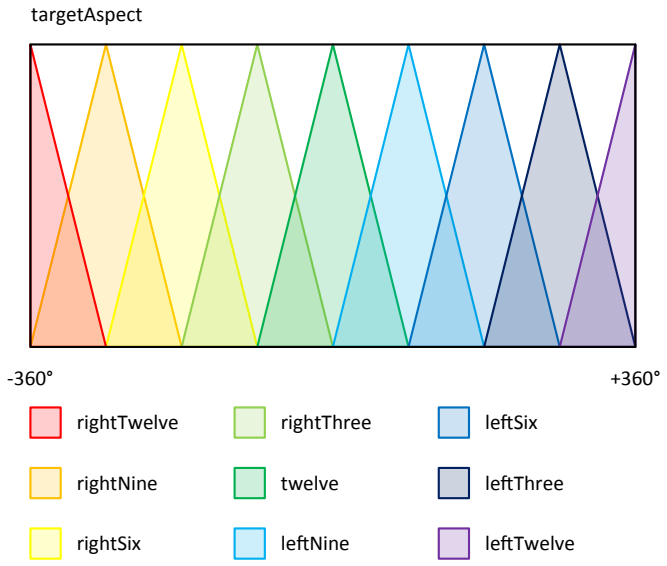
Figure 2: *targetDist* fuzzy sets



3.2.2 targetAspect

The linguistic variable *targetAspect* is the direction of the target in relation to the player. The universe of disclosure for *targetAspect* is between -360° and +360°. Positive values rotate to the left, and negative values rotate to the right. The fuzzy sets selected relate to clock positions, similar to what fighter pilots might call out in combat. There are three twelve o'clock positions due to the two revolutions between -360° and +360°.

Figure 3: *targetAspect* fuzzy sets

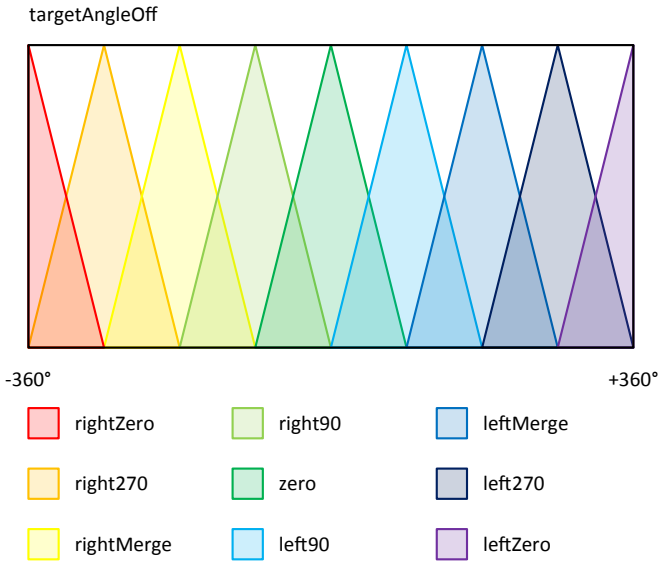


3.2.3 targetAngleOff

The linguistic variable *targetAngleOff* relates to the target's current heading, in relation to the player's current heading. For example, if the target is heading towards the player perpendicularly from the right, the target's angle-off would be +90°. Similarly, if the target has the exact same heading as the player, the target's angle-off would be 0°. Again, positive values rotate to the left, and negative values rotate to the right. The universe of disclosure is between -360° and +360°. The fuzzy

sets selected mimic clock positions, similar to *targetAspect*, however are named with degree values. A merge is when the player and target have opposite angle-off's, ie. 180°. In this situation, if the player and target were in front of each other, they would facing each other, and would be about to directly pass each other, in a "merge".

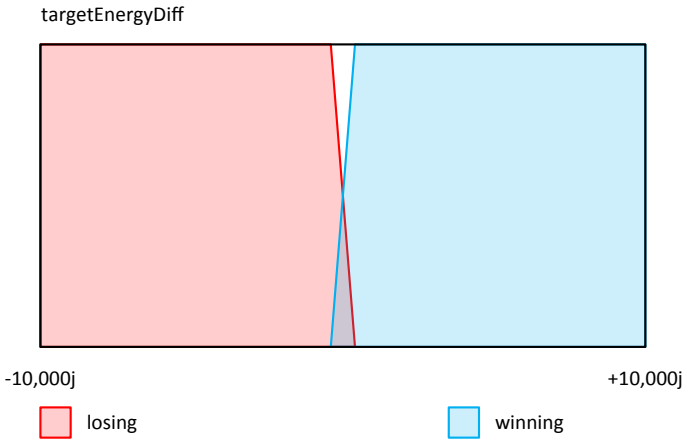
Figure 4: *targetAngleOff* fuzzy sets



3.2.4 targetEnergyDiff

The linguistic variable *targetEnergyDiff* relates to the difference between the player's energy and the current target's energy. The universe of disclosure for *targetEnergyDiff* is between -10,000j and +10,000j. The fuzzy sets selected for this linguistic variable are *losing* and *winning*.

Figure 5: *targetEnergyDiff* fuzzy sets



3.3 Blast variables

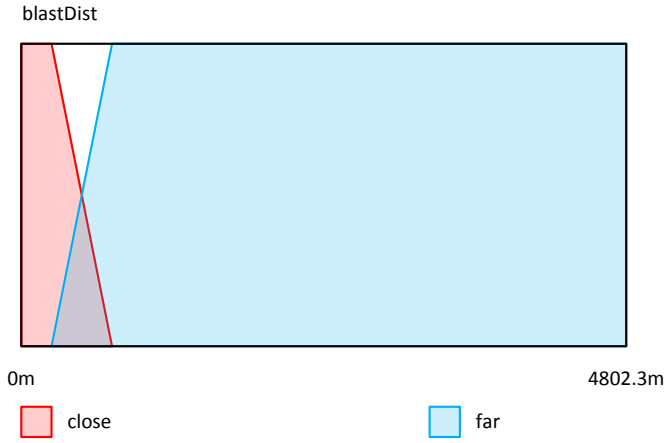
The sensor returns a list of all energy blasts currently in the battle space. This controller only considers the closest energy blast to dodge, and ignores all other

blasts in the list.

3.3.1 blastDist

The linguistic variable *blastDist* is the distance from the player to the energy blast. The universe of disclosure for *blastDist* is between 0m and 4802.3m, and the fuzzy sets are defined as *close* and *far*.

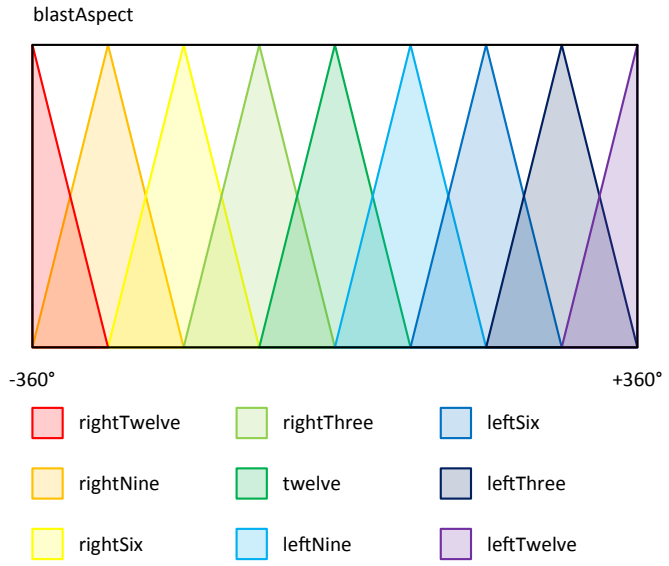
Figure 6: *blastDist* fuzzy sets



3.3.2 blastAspect

The linguistic variable *blastAspect* is similar to *targetAspect*, but relates to the direction from the player to the energy blast. Similar fuzzy sets, based on the clock analogy have been used.

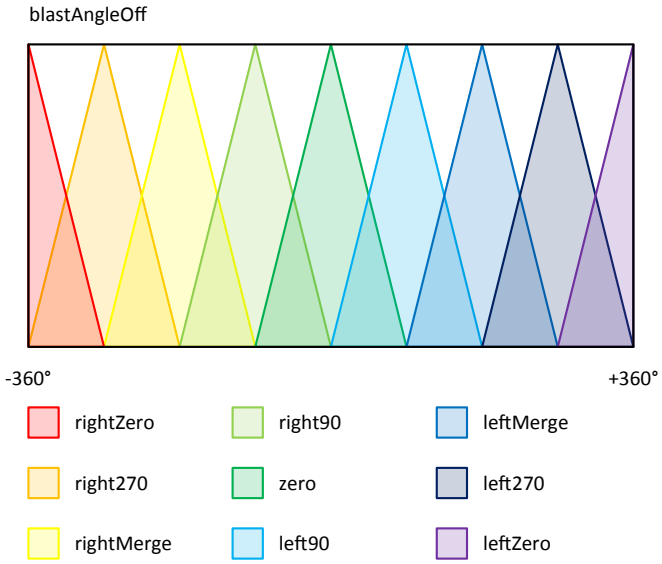
Figure 7: *blastAspect* fuzzy sets



3.3.3 blastAngleOff

The linguistic variable *blastAngleOff* is similar to *targetAngleOff*, but references the current heading of the energy blast in relation to the player's heading. Similar fuzzy sets have also been used.

Figure 8: *blastAngleOff* fuzzy sets



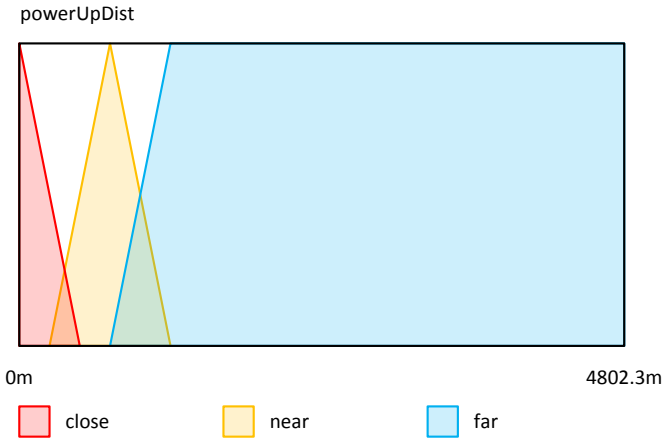
3.4 Powerup variables

The sensor returns a list of all powerups that currently exist in the battle space. To conserve energy, this controller only reacts to powerups only if they are nearby, and assumes that powerups that are far would most likely be consumed by an enemy by the time the player arrives.

3.4.1 powerUpDist

The linguistic variable *powerUpDist* is the distance between the target and the powerup. Similar to *targetDist*, the universe of disclosure is between 0m and 4802.3m. Similar fuzzy sets have also been used.

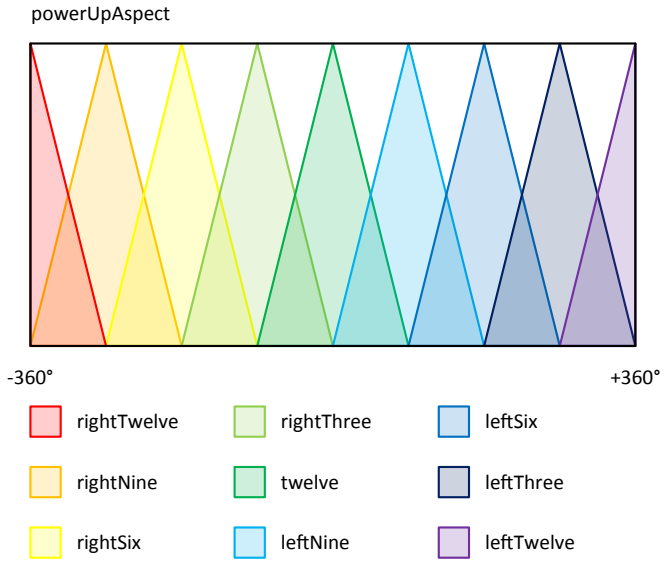
Figure 9: *powerUpDist* fuzzy sets



3.4.2 powerUpAspect

The linguistic variable *powerUpAspect* is the direction of the powerup in relation to the player. Again, the clock analogy has been used to determine the fuzzy sets. However, the angle-off of the powerup is not considered, since it is stationary and does not change its heading.

Figure 10: *powerUpAspect* fuzzy sets



4 Output linguistic variables

Due to the fact that the fuzzy logic API does not provide a method to prioritize rules, defensive and offensive versions of some output linguistic variables have been defined. Boolean variables are defined, which are then tested to determine which version of the rule is fired, during a given situation.

For example, the defensive turn rule only considers the blast linguistic variables as input, whereas the offensive turn rule considers both target and blast linguistic variables.

4.1 Defensive rules

4.1.1 defensiveTurn

The *defensiveTurn* linguistic variable defines which heading the saucer will take, in degrees, according to the rules that govern defensive turning. The linguistic variables used as input for *defensiveTurn* are: `layerVar = blastDist`, `rowVar = blastAngleOff`, and `colVar = blastAspect`, creating a 3D rule matrix. Note that *r* and *l* represent right and left in the table.

For example: IF (close) AND (rightSix) AND (zero) THEN (-90)
In other words, IF the energy blast is close, AND it is behind the player, AND it's heading towards the player, THEN turn right for 90°.

Table 1: *defensiveTurn blastDist close*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	0	0	-90	0	0	0	-90	0	0
r270	-90	0	-90	+180	+90	0	-90	+180	+90
rMerge	0	0	-90	0	0	0	-90	0	0
r90	-90	-180	-90	0	-90	-180	-90	0	+90
zero	0	0	-90	0	0	0	-90	0	0
l90	-90	0	-90	+180	+90	0	-90	+180	+90
lMerge	0	0	-90	0	0	0	-90	0	0
l270	-90	-180	-90	0	-90	-180	-90	0	+90
lZero	0	0	-90	0	0	0	-90	0	0

Table 2: *defensiveTurn blastDist far*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	-180	0	-90	0	0	0	+90	0	+180
r270	0	0	0	0	0	0	0	0	0
rMerge	0	0	0	0	0	0	0	0	0
r90	0	0	0	0	0	0	0	0	0
zero	-180	0	-90	0	0	0	+90	0	+180
l90	0	0	0	0	0	0	0	0	0
lMerge	0	0	0	0	0	0	0	0	0
l270	0	0	0	0	0	0	0	0	0
lZero	-180	0	-90	0	0	0	+90	0	+180

4.1.2 defensiveSpeed

The *defensiveSpeed* linguistic variable defines how fast the saucer will travel, according to the rules that govern defensive speed. The linguistic variables used for input for *defensiveSpeed* are: `layerVar = blastDist`, `rowVar = blastAngleOff`, and `colVar = blastAspect`, creating a 3D rule matrix.

For example: IF (close) AND (rightSix) AND (zero) THEN (125)

In other words, IF the energy blast is close, AND it is behind the player, AND it's heading towards the player, THEN speed is 125.

Table 3: *defensiveSpeed blastDist close*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	50	125	125	125	50	125	125	125	50
r270	50	125	50	125	50	125	50	125	50
rMerge	50	125	50	125	125	125	50	125	50
r90	50	125	50	125	125	125	50	125	50
zero	50	125	125	125	50	125	125	125	50
l90	50	125	50	125	125	125	50	125	50
lMerge	50	125	50	125	125	125	50	125	50
l270	50	125	50	125	50	125	50	125	50
lZero	50	125	125	125	50	125	125	125	50

Table 4: *defensiveSpeed blastDist far*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	50	50	75	50	50	50	75	50	50
r270	50	50	50	75	50	50	50	75	50
rMerge	50	50	50	50	75	50	50	50	50
r90	50	75	50	50	75	75	50	50	50
zero	50	50	75	50	50	50	75	50	50
l90	50	50	50	75	75	50	50	75	50
lMerge	50	50	50	50	75	50	50	50	50
l270	50	75	50	50	50	75	50	50	50
lZero	50	50	75	50	50	50	75	50	50

Singleton rules have also been added to *defensiveSpeed*, using *powerUpDist* as input:

- IF (powerUpDist IS far) THEN (defensiveSpeed IS 50)
- IF (powerUpDist IS near) THEN (defensiveSpeed IS 125)
- IF (powerUpDist IS close) THEN (defensiveSpeed IS 125)

4.2 Offensive rules

4.2.1 offensiveTurn

The *offensiveTurn* linguistic variable defines which heading the saucer will take, in degrees, according to the rules that govern defensive turning. Two sets of linguistic variables are combined, target variables and blast variables, creating two 3D rule matrices.

The target linguistic variables used as input for *offensiveTurn* are:
`layerVar = targetDist`, `rowVar = targetEnergyDiff`, and
`colVar = targetAspect`, creating the 3D rule matrix below.

Table 5: *offensiveTurn targetDist close*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
losing	+180	-90	0	+90	-180	-90	0	+90	-180
winning	0	+90	-180	-90	0	+90	+180	-90	0

Table 6: *offensiveTurn targetDist near*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
losing	+180	0	0	0	-180	0	0	0	-180
winning	0	+90	-180	-90	0	+90	+180	-90	0

The blast linguistic variables used as input for *offensiveTurn* are: `layerVar = blastDist`, `colVar = blastAngleOff`, and `rowVar = blastAspect`, creating another 3D rule matrix.

Table 7: *offensiveTurn blastDist close*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	0	0	-90	0	0	0	-90	0	0
r270	-90	0	-90	+180	+90	0	-90	+180	+90
rMerge	0	0	-90	0	0	0	-90	0	0
r90	-90	-180	-90	0	-90	-180	-90	0	+90
zero	0	0	-90	0	0	0	-90	0	0
l90	-90	0	-90	+180	+90	0	-90	+180	+90
lMerge	0	0	-90	0	0	0	-90	0	0
l270	-90	-180	-90	0	-90	-180	-90	0	+90
lZero	0	0	-90	0	0	0	-90	0	0

Table 8: *offensiveTurn blastDist far*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	-180	0	-90	0	0	0	+90	0	+180
r270	0	0	0	0	0	0	0	0	0
rMerge	0	0	0	0	0	0	0	0	0
r90	0	0	0	0	0	0	0	0	0
zero	-180	0	-90	0	0	0	+90	0	+180
l90	0	0	0	0	0	0	0	0	0
lMerge	0	0	0	0	0	0	0	0	0
l270	0	0	0	0	0	0	0	0	0
lZero	-180	0	-90	0	0	0	+90	0	+180

4.2.2 offensiveSpeed

The *offensiveSpeed* linguistic variable defines how fast the saucer will travel, according to the rules that govern offensive speed. Two sets of linguistic variables are used as input, target variables and blast variables.

The target linguistic variables used as input for *offensiveSpeed* are: `rowVar = targetDist`, and `colVar = targetEnergyDiff`

Table 9: *offensiveSpeed* target variables

	losing	winning
close	125	50
near	75	75
far	50	125

The blast linguistic variables used as input for *offensiveSpeed* are: `layerVar = blastDist`, `colVar = blastAngleOff`, and `rowVar = blastAspect`

Table 10: *offensiveSpeed blastDist close*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	50	50	125	50	50	50	125	50	50
r270	50	50	50	125	50	50	50	125	50
rMerge	50	50	50	50	125	50	50	50	50
r90	50	125	50	50	125	125	50	50	50
zero	50	50	125	50	50	50	125	50	50
l90	50	50	50	125	125	50	50	125	50
lMerge	50	50	50	50	125	50	50	50	50
l270	50	125	50	50	50	125	50	50	50
lZero	50	50	125	50	50	50	125	50	50

Table 11: *offensiveSpeed blastDist far*

	rTwelve	rNine	rSix	rThree	twelve	lNine	lSix	lThree	lTwelve
rZero	50	50	75	50	50	50	75	50	50
r270	50	50	50	75	50	50	50	75	50
rMerge	50	50	50	50	75	50	50	50	50
r90	50	75	50	50	75	75	50	50	50
zero	50	50	75	50	50	50	75	50	50
l90	50	50	50	75	75	50	50	75	50
lMerge	50	50	50	50	75	50	50	50	50
l270	50	75	50	50	50	75	50	50	50
lZero	50	50	75	50	50	50	75	50	50

4.3 Overrides

The following output linguistic variables do not have offensive or defensive versions, but are controlled by testing boolean variables which determine when to fire these rules. For example, *firepower*'s rate of fire is controlled by a boolean variable, which becomes `true` when there are only a certain number of enemies left in the battle space.

4.3.1 firepower

The *firepower* linguistic variable defines how much energy to commit to firing the cannon, according to the rules that govern firepower. *targetDist* is used as input. Additionally, the number of remaining enemy saucers and whether or not there is a nearby powerup determines the rate of fire, and the implementation of decision making will be shown as Java code.

The strategy is to be conservative with firepower unless there is a powerup nearby. If so, increase to maximum rate of fire to deter any enemy saucers that may be near the powerup from retrieving it, while the player moves directly to the powerup. If there are only two enemies left, increase rate of fire to medium, and if there is one enemy left, increase rate of fire to maximum. The strategy is to become more aggressive towards the end of the battle, since the result could be a tie if other saucers remain at the end of the battle.

The *firepower* rules relating to *targetDist* are:

- IF (targetDist IS far) THEN (firepower IS 0)
- IF (targetDist IS near) THEN (firepower IS 100)
- IF (targetDist IS close) THEN (firepower IS 100)

The following variables determine the rate of fire:

```
1  private static final double DEFENCE_ROF = 0.01;
2  private static final double OFFENCE_ROF = 0.05;
3  private static boolean isLastTwoTargets; // true when only two enemy saucers remain
4  private static boolean isLastTarget; // true when only one enemy saucer remains
5  private static boolean isPowerUpNear; // true when powerup within 1200.575m
```

Java code 4.1: *firepower* variables

Rate of fire is controlled by the `getFirePower()` method:

```
1  public double getFirePower() throws Exception {
2
3      // max rate of fire
4      if(isLastTarget || isPowerUpNear) {
5          return firePower.getValue();
6      // medium rate of fire
7      } else if(isLastTwoTargets) {
8          if(Math.random() < OFFENCE_ROF) {
9              return firePower.getValue();
10         } else {
11             return 0.0;
12         }
13     // min rate of fire
14     } else {
15         if(Math.random() < DEFENCE_ROF) {
16             return firePower.getValue();
17         } else {
18             return 0.0;
19         }
20     }
21 }
```

Java code 4.2: `getFirePower()`

4.3.2 getPowerUpTurn

The *getPowerUpTurn* linguistic variable defines which heading the saucer will take, in degrees, according to the rules that govern turning towards a nearby powerup. This linguistic variable is used to override the player's current turning regime, and ensure that it heads straight for a powerup without dodging any energy blasts. This rule is fired in conjunction with an increased rate of fire, as well as deploying the shield to protect itself as it attempts to retrieve a powerup.

powerUpDist and *powerUpAspect* are used as inputs, as well as a boolean variable which determines when this rule overrides all other turn rules.

Table 12: *getPowerUpTurn*

	close	near	far
rTwelve	0	0	0
rNine	+90	+90	0
rSix	-180	-180	0
rThree	-90	-90	0
twelve	0	0	0
lNine	+90	+90	0
lSix	+180	+180	0
lThree	-90	-90	0
lTwelve	0	0	0

The boolean variable determines when *getPowerUpTurn* overrides other turn rules:

```
1 private static boolean isPowerUpNear; // true when powerup within 1200.575m
```

Java code 4.3: *getPowerUpTurn* variable

getPowerUpTurn overrides other turn rules in the `getTurn()` method:

```
1  public double getTurn() throws Exception {
2
3      // override turn rules if powerup is nearby
4      if(isPowerUpNear) {
5          return getPowerUpTurn.getValue();
6      }
7
8      // use offensiveTurn if last two, or last target
9      if(isLastTarget || isLastTwoTargets) {
10         return offensiveTurn.getValue();
11     } else {
12         return defensiveTurn.getValue();
13     }
14 }
```

Java code 4.4: `getTurn()`

4.3.3 shield

The *shield* linguistic variable defines when to deploy the shield, according to the rules that govern shield deployment. The universe of disclosure is between 0 and

1. This linguistic variable uses powerup, blast and *myEnergy* variables as input, and uses the following sets: `layerVar = powerUpDist`, `rowVar = myEnergy`, and `colVar = blastDist`, creating a 3D rule matrix.

Table 13: *shield powerUpDist close*

	close	far
lowEnergy	1	0
highEnergy	1	0

Table 14: *shield powerUpDist near*

	close	far
lowEnergy	0	0
highEnergy	1	0

Table 15: *shield powerUpDist near*

	close	far
lowEnergy	0	0
highEnergy	0	0

Boolean variable `isPowerUpNear` is tested to determine when to deploy the shield, and the crisp output is converted to a boolean value in `getShields()` in order to function:

```
1  public boolean getShields() throws Exception {
2
3      // only deploy shield if powerup is nearby
4      if(isPowerUpNear) {
5          // convert output to boolean
6          return shield.getValue() > 0.5;
7      } else {
8          return false;
9      }
10 }
```

Java code 4.5: `getShields()`

5 Learnings

Due to the greater amount of input sensors provided, this controller is vastly different to the one submitted with the first assignment. It is now possible to define rules with higher complexity, and therefore 3D rule matrices are a necessity. The larger player count also requires a completely different strategy. In the first assignment, the main strategy was to be aggressive from the start. In comparison, this controller needs to be much more defensive and aim to survive until the end of the battle.

With that in mind, I began with blast sensor inputs. Rather than attempt to implement multiple sensors at once, I was able to focus on the controller's performance on dodging energy blasts and developed a decent ruleset just for turning. Experience from the first assignment assisted in defining controlled rules, resulting in less erratic movement. I reused the clock analogy for target/blast/powerup directions, and also considered the heading of moving objects to decide which direction to turn the player.

However, I came across an issue where the controller would switch between -180 and +180 when an energy blast was behind the player. Rules dictated that the player would turn either left or right, depending on if the energy blast position was a positive or negative number. Because the sensor value was switching between positive and negative values erratically, the controller also flicked between left and right turns, resulting in the player not turning at all, and continued to move straight and was always hit by energy blasts from behind. To remedy this, I decided to stick to a single direction of turn, which in this case was a right hand 90° turn. This resolved the issue and the player now dodges energy blasts very well from the 6 o'clock position, albiet only in right hand turns.

I encountered an issue where I wished to prioritize turn rules that tell the player to move directly to a powerup over turn rules that dodge energy blasts. I emailed the tutor, Philip Hingston, who advised me that this was not possible. The alternative was to set up a combination of fuzzy and if-then rules. I implemented this concept and realised this opened up other possibilities for rules, as seen with the defensive and offensive versions of my output variables. I found similarities in this concept with the Tartarus problem of Workshop 7, and assisted in designing the if-then rules.

The controller has potential to be improved. For example, firepower rules could utilize target heading, and only fire if the target is facing the player. I observe that the player can sometimes waste shots if the target is running away and does a good job at dodging energy blasts. The turn rules also have room for improvement. There are some situations where the player makes a wrong turn, or does not react quick enough to an incoming energy blast. I am pleased with the powerup strategy I have implemented, as it ignores energy blasts and suppresses enemies nearby while heading straight for the powerup. However, there are cases where the nearest enemy to the player is not the one closest to the powerup, and energy is wasted firing at the maximum rate. This could be improved by determining which enemy is closest to the powerup and firing at it, or by having the ability to fire at the powerup directly.

6 Conclusion