Edith Cowan University
CSG2341
Intelligent Systems
Assignment 1A

Martin Ponce
Student 10371381

Tutor: Philip Hingston

August 30, 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 two player 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.

The saucers begin with equal amounts of energy at the start of the game, and this energy is consumed as they fly around or fire their auto-aiming cannon mounted on a rotating turret. The rounds fired from the cannon are ballistic. They travel in a straight line from the cannon and are not guided. When a saucer is hit by a round, energy is depleted. Amount of energy depletion depends on how far away the round was fired. In other words, rounds fired will lose energy the further they have to travel, and will eventually fade away. They are much more lethal in close combat.

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. Currently, there is no energy penalty for turning. Each saucer is also equipped with a sensor, which determines how far away the opponent is, the opponent direction, and how much energy the opponent has, which are used as inputs for fuzzy logic.

When a saucer loses all of its energy, it disappears from the battle space and loses. The remaining saucer with energy left over is the winner. The goal of this report is to design a fuzzy logic controller so that it's saucer will have the most amount of remaining energy at the end of the battle.

2 Idea

The tactics of this controller are based on two facts:

- Saucers will always consume energy, no matter what they are doing.
- Cannon rounds are much more effective at close range.

Since saucers will always consume energy during all regimes of flight, this report believes that it is more efficient to be in a position to fire the weapon and attempt to degrade the enemy's energy at a faster rate, rather than fly defensively without firing at all. It is also more effective to be within close range of the enemy to cause more damage with the cannon, but on the other hand, the enemy's cannon will also be just as dangerous.

Therefore, the tactics of the controller will be as discussed below.

2.1 Flight regime

If the enemy is far while winning, commit to engagement and fly aggressively towards the enemy. Otherwise if the enemy is close, and winning or the score is even, fly at a slow speed to stay within close range. This will also allow the enemy to pass so that the saucer can stay behind the enemy and avoid overshooting him. During winning or even scores, the saucer will always turn towards the enemy attempting to close the distance to a much more lethal firing range.

If losing, break contact and fly away from the enemy at the appropriate speed. If close, fly as fast as possible away from the enemy. If near, fly moderately, and if far, then fly at the minimum speed to conserve energy.

2.2 Weapon employment

Since cannon shots lose energy the further they travel, only fire at close or near ranges to maximise efficiency of energy. When close and winning, fire at maximum power. When close and the score is even, fire at medium power. When near and winning or the score is even, fire the cannon at maximum power to ensure that the cannon round will reach the enemy. During other circumstances, hold fire to conserve energy.

3 Fuzzy variables

Three linguistic variables have been selected for input, based on the sensor readings of the saucer: *distance to target*, *energy difference* and *heading angle*.

Conversely, three linguistic variables have been selected for output: *turn*, *speed* and *firepower*.

3.1 Input variables

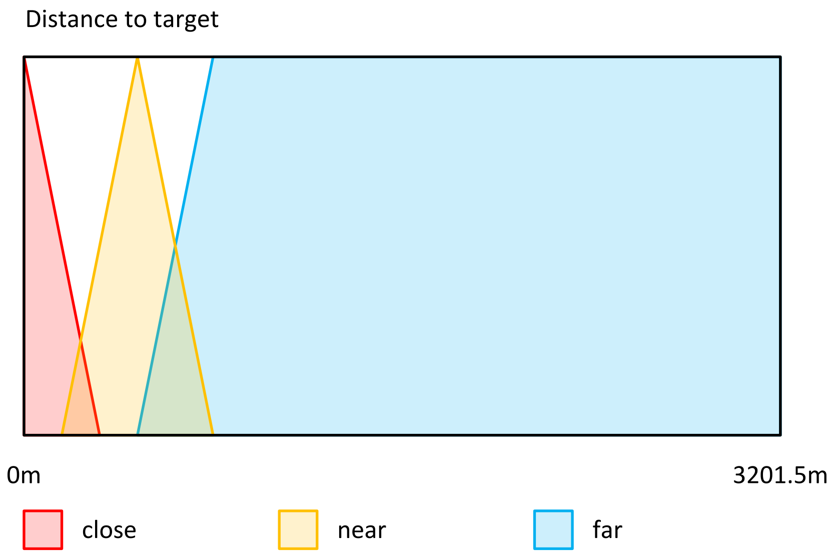
3.1.1 *Distance to target*

Distance to target is the distance from the saucer to the opponent, and is measured in meters. The universe of disclosure for *distance to target* is between 0 meters and the diagonal length of the battle space. The formula has been supplied in the existing code as:

$\sqrt{\text{width} \cdot \text{width} + \text{height} \cdot \text{height}}$

This linguistic variable is used to determine how much energy will be committed to firing the cannon. As mentioned previously, the cannon will only be fired at close or near distances. Therefore, three fuzzy sets are associated with *distance to target*:

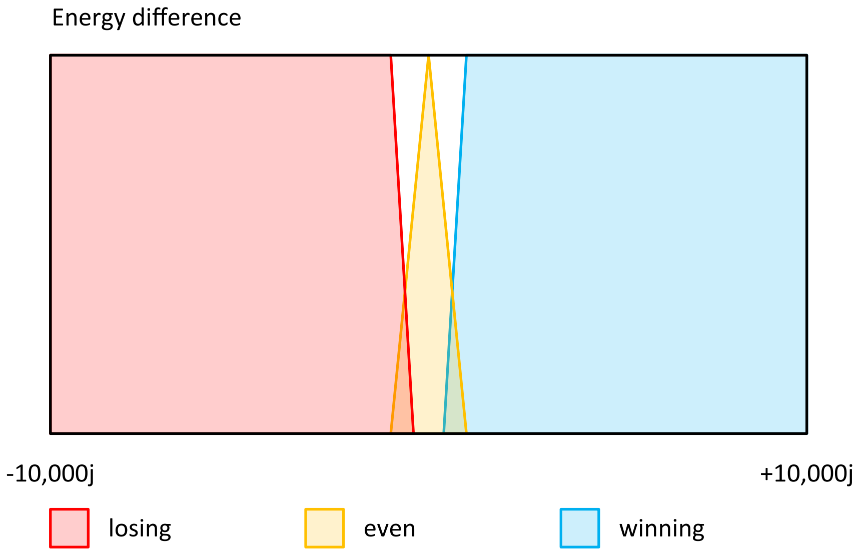
Figure 1: *Distance to target* fuzzy sets



3.1.2 *Energy difference*

Energy difference is the difference between the saucer's energy and the opponent's energy. The universe of disclosure for *energy difference* is between -10,000j to +10,000j, where 10,000j is the amount of energy that the saucers begin with. This linguistic variable determines who is winning, who is losing, or if the score is even. It is used as input to decide how much energy is committed in firing the weapon, as well as whether or not to turn into or away from the enemy. The following fuzzy sets are created for *energy difference*:

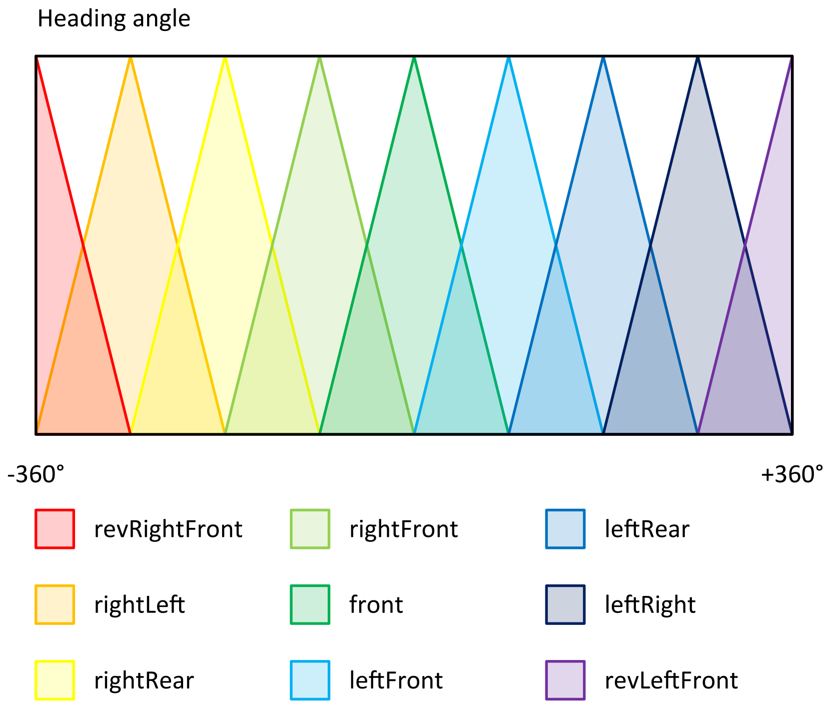
Figure 2: *Energy difference* fuzzy sets



3.1.3 *Heading angle*

Heading angle is the direction of the opponent in relation to the saucer, in degrees. After printing `opponentDirection` during execution, it is assumed that the universe of disclosure for this linguistic variable is from -360° to +360°. The variable, in conjunction with *energy difference*, dictates how the saucer will turn, and has been configured with the following fuzzy sets:

Figure 3: *Heading angle* fuzzy sets



3.2 Output variables

3.2.1 *Turn*

The *turn* linguistic variable defines which heading the saucer will take, in degrees, according to the rules that govern turning. A zero value will turn towards the opponent, while 180° will turn away from the opponent and head into the opposite direction, turning left while doing so. A negative value will turn right, and a positive value will turn left. The linguistic variables used as input for the turning rules are *energy difference* and *heading angle.* See Table 1 for the *turn* rule table.

Table 1: *Turn* rule table

	Losing	Even	Winning
revRightFront	-180.0	0.0	0.0
rightLeft	-90.0	+90.0	+90.0
rightRear	0.0	-180.0	-180.0
rightFront	+180.0	-90.0	-90.0
front	-180.0	0.0	0.0
leftFront	-180.0	+90.0	+90.0
leftRear	0.0	+180.0	+180.0
leftRight	+90.0	-90.0	-90.0
revLeftFront	-180.0	0.0	0.0

When winning or if the score is even, turn towards the enemy and commit to the engagement. Otherwise, if losing, turn away from the enemy and keep him at a relatively safe distance to minimize damage from cannon fire.

For example: IF (winning) AND (front) THEN (0.0)
In other words, if winning and enemy is in front, then keep heading towards him.

In contrast: IF (losing) AND (front) THEN (-180.0)
This will perform the opposite, and the saucer will turn right, into the opposite direction if the enemy is in front.

3.2.2 *Speed*

The *speed* linguistic variable defines how much energy to use for flight and is governed by rules which use *distance to target* and *energy difference* as inputs. See Table 2 for the *speed* rule table.

Table 2: *Speed* rule table

	Losing	Even	Winning
Close	125.0	50.0	50.0
Near	100.0	50.0	125.0
Far	50.0	100.0	125.0

When winning, the strategy is as follows. If close, use the minimum speed to prevent overshooting the enemy, and also allow the enemy to overtake so that it can be followed. If near, use medium speed to catch up to get within a more lethal range for the cannon. If far, use maximum speed to close the distance to get within a more lethal firing range. When the score is even, the strategy is to be more conservative with energy and only use medium speed if far. Otherwise, use the minimum speed if close or near. When losing, the strategy is to fly fast away from the enemy if close. Otherwise use medium speed if close and minimum speed if far away.

3.2.3 ***Firepower***

The *firepower* linguistic variable determines how much energy to use when firing the cannon, and is controlled by rules which use *energy difference* and *distance to target* as inputs. See Table 3 for the *firepower* rule table.

Table 3: *Firepower* rule table

	Losing	Even	Winning
Close	50.0	100.0	100.0
Near	0.0	100.0	100.0
Far	0.0	0.0	0.0

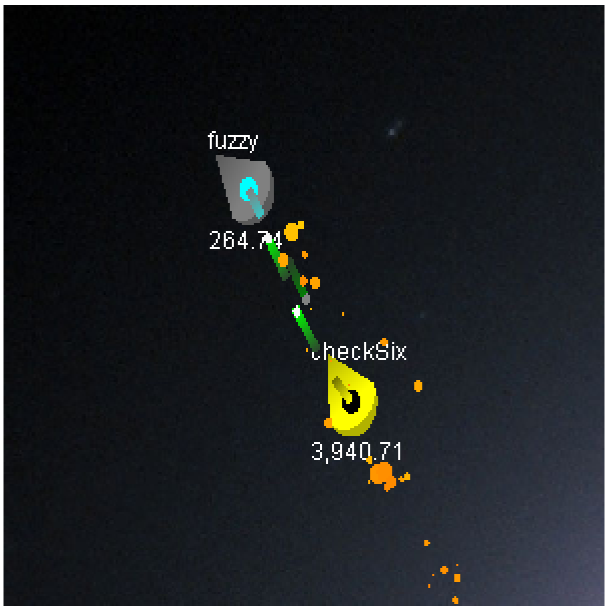
When winning or the score is even, and the enemy is close or near, fire the cannon at maximum power. Otherwise if the enemy is far, do not fire the cannon at all. This is to increase the chance that the cannon round will reach the target. Finally, if losing, do not fire the cannon at all at, unless at close range, and only fire at medium power.

4 Sample fuzzy rule

4.1 Turning rule

This section will examine the implementation of the fuzzy rules that govern the saucer's ability to turn, and will use the following situation, as shown in Figure 4 to demonstrate how the *turn* output rules are fired:

Figure 4: Sample game screen



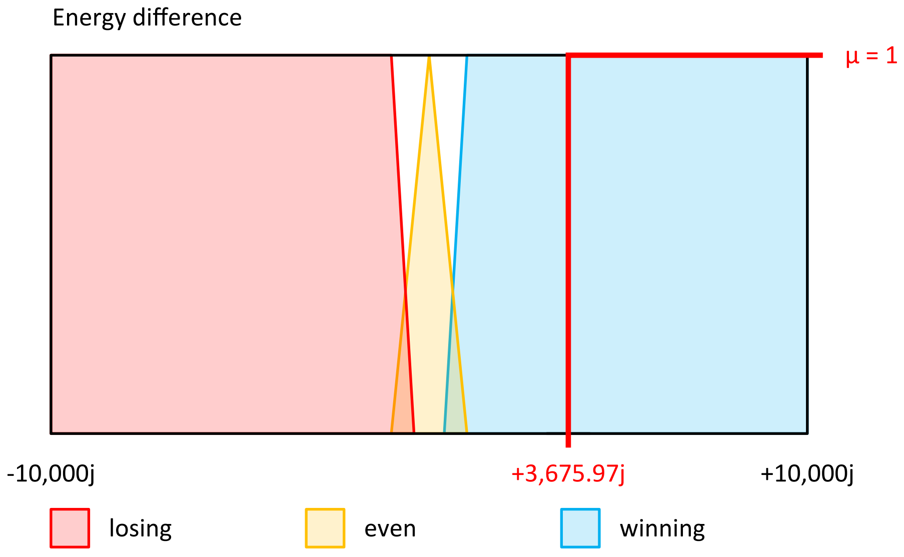
The yellow saucer in Figure 4 is the *player*, called *checkSix*, while the enemy is the grey saucer, called *fuzzy*. Note that *fuzzy* is controlled with the original, existing fuzzy controller supplied with the assignment. Two *turn* rules are fired during this sequence:

- Rule 1:
  - IF (*heading angle* IS front) AND (*energy difference* IS winning) THEN (*turn* IS 0)
- Rule 2:
  - IF (*heading angle* IS left front) AND (*energy difference* IS winning) THEN (*turn* IS 90)

4.1.1 *Energy difference* antecedent

Energy difference will be explored first, since the value is the same in both instances of Rule 1 and Rule 2 above. The *energy difference* antecedent, which in this case is *winning*, is demonstrated by the difference of energy between the two saucers. *checkSix* has 3940.71 joules of energy remaining, while *fuzzy* only has 264.74. The *energy difference* is +3675.97j. Therefore, *checkSix* is *winning*. Figure 5 below demonstrates the firing of this antecedent, and shows that the fuzzy set value is +3675.97, while μ, the membership value to the *winning* fuzzy set, is 1.

Figure 5: *Energy difference* antecedent



4.1.2 *Heading angle* antecedent

Heading angle causes Rule 1 and Rule 2 to fire, with *front* and *leftFront*. This is due to the configuration of the two fuzzy sets. As seen previously in Figure 3, the *leftFront* fuzzy set begins at 50% of the *front* fuzzy set range. In other words, any value higher than the median value of *front* will also belong to the *leftFront* set. Conversely, any value lower than the median value of *front* will also belong to the *rightFront* set. Figure 6 below demonstrates the firing of these two antecedents, their fuzzy set values and their μ values.

Figure 6: *Heading angle* antecedent

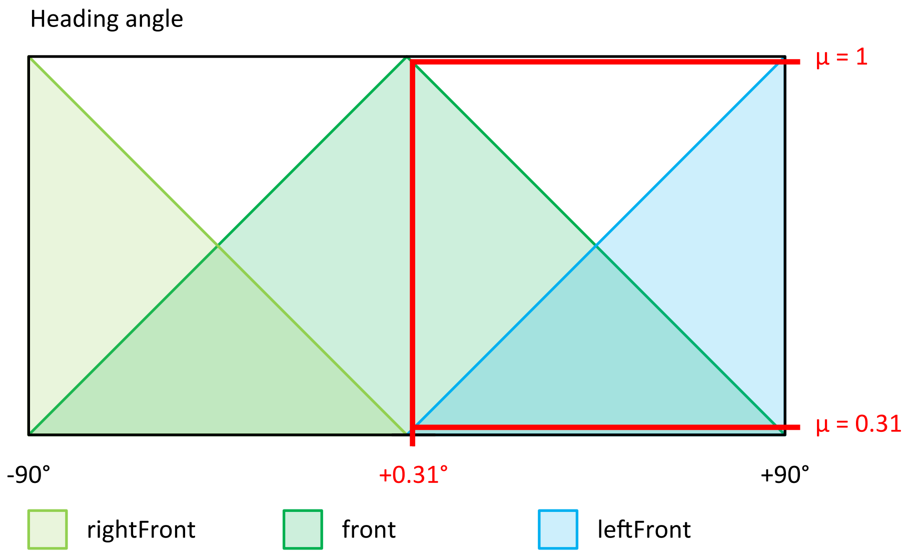


Figure 6 shows a simplified view of the *heading angle* sets, only displaying -90° to +90° during the firing of Rule 1 and Rule 2. The *heading angle* value of +0.31° belongs to the *front* fuzzy set at $\mu = 1$, and also belongs to the *leftFront* fuzzy set at $\mu = 0.31$, hence the firing of the two rules, thus providing two inputs. These inputs, along with the *energy difference* input will be aggregated to create a single, crisp output, as explained in the next section.

4.1.3 Rule aggregation

When multiple inputs are provided by the same linguistic variable antecedent, as seen in Section 4.1.2, Sugeno style inference calculates the weighted average of multiple μ values to determine a single μ value, using the following formula:

$$\text{WA} = \frac{\sum_{i=1}^{N} \mu(k_i) k_i}{\sum_{i=1}^{N} k_i} \tag{4.1}$$

Therefore, to calculate the weighted average for the two *heading angle* inputs for the example shown in Figure 6:

$$\text{WA} = \frac{1 \times 0.31 + 0.31 \times 0.31}{1 + 0.31} \tag{4.2}$$

$$= 0.31 \tag{4.3}$$

Subsequently, the weighted average of the two *heading angle* antecedents is 0.31. However, μ for the *energy difference* antecedent must also be considered. As discussed in Section 4.1.1, *winning* $\mu = 1$.

Since the example rule performs an AND operation between the two antecedents, fuzzy set operations dictate that an intersection operation must occur to determine the final, crisp output. Therefore, the following function is appropriate:

$$\text{output} = \min(\mu(k_1), \mu(k_2), ..., \mu(k_n)) \tag{4.4}$$

When applied to our example rules that have been fired:

$$\text{output} = \min(1, 0.31) \tag{4.5}$$

$$= 0.31 \tag{4.6}$$

Therefore, the final, crisp consequent output for the example scenario is a turn of 0.31°. In other words, from its current direction of 0.0°, the saucer will turn **left** for 0.31°, in order to follow the enemy, who is in front, and slightly to the left.

5 Learnings

Although I believe I have a firm grasp on the basic concepts of fuzzy logic, I had a considerable amount of trouble understanding the logic behind *turning* the saucer. I ran several tests, printing the opponent direction values during runtime to understand the relationship between where the player saucer is, and where the enemy is, and the values that were returned each frame. It was only until I imagined that the positive, left-hard turn values as "counter-clockwise", and negative, right-hand turn values as "clockwise" did I manage to tune the turn rules to a working state.

My initial concept was to aggressively attack the opponent as soon as the game started, which worked well. However, the existing opponent controllers did not test my defensive strategy at all, and were immediately overwhelmed by incoming fire. My saucer was always within *winning* or *even* fuzzy set limits, and never triggered *losing* rules. Eventually, I realised that the original `FuzzyController.java` class could be modified to become a more challenging opponent by changing the values, as listed below:

```java
public FuzzyController() throws Fuzzy Exception {
  // ...
  final double maxPower = Saucer.MAX_POWER;
  final double midPower = maxPower; // originally divided by 5.0
  final double lowPower = maxPower; // originally divided by 20.0
  // ...
}
```

Java code 5.1: FuzzyController.java modifications

After the changes were made, the *fuzzy* opponent fired its cannons at maximum power, which assisted in testing whether or not my offensive strategy to constantly stay within close range to the enemy worked. Initial tests proved otherwise, and my saucer was destroyed immediately. I needed to develop a substantial defensive strategy.

The *heading angle* fuzzy sets were modified from my original, arbitrarily chosen sets, to sets that are based on clock positions in relation to the player's position, similar to what a fighter pilot might say during combat, ie. "Bandit at my six o'clock", or "Bogey at my nine". The *turn* output spikes were also chosen based on the clock analogy, resulting in a more controlled behaviour, and enabled me to define a defensive strategy through *turn* rules.

However, the rules require refinement. Confusion can occur when the player attempts to travel beyond the battle space limits to maintain its heading and causes erratic turns. This may be resolved if sensors were in place to detect the battle space boundaries, and added as linguistic variables and rules. Confusion can also occur when too many *heading angle* antecedents fire rules which result in twitchy turning movements. This would be a major problem if turns cost energy. *Firepower* also has the potential for improvement as it may occasionally fire weak shots even though the enemy is further than the shot's range, resulting in wasted energy.

My offensive strategy is also far from perfect and has much room for improvement. Sitting right behind the enemy and giving chase during offence may suit well to space/aircraft with fixed, forward facing armament, but is a dangerous place to be when the enemy can rotate his weapon to face the rear.

6 Conclusion

This report examined the application of fuzzy logic using Sugeno style inference in a video game. The video game involved developing a fuzzy logic controller, which controls an armed flying saucer with the sole purpose of destroying the enemy saucer.

The linguistic input variables and their fuzzy sets were explored, as well as the output variables and the associated rules. These rules governed the behaviour of the saucer and attempted to implement the two overall tactics developed for this assignment:

- Commit to the battle and fly offensively when the score is even or if winning
- Disengage from the battle and fly defensively when losing

An example rule was examined, and the aggregation of a crisp output was explained, based on Sugeno style inference. Finally, learning experiences throughout the work of this assignment were discussed.

6.1 Results

The following tables on the final page display the results of tournaments between the *checkSix* controller versus the *simple* controller, *fuzzy* controller, as well as the modified fuzzy controller, as discussed in Section 5. The modified fuzzy controller is referred to as *fuzzyMaxPower* in the table. The average score over 10 battles per tournament is shown, with five consecutive tournaments executed per opponent.

The final table shows the results when *checkSix* goes head to head with itself. No changes were made to the *evilCheckSix* controller, other than its name and colour. It was expected that the scores would have been more equal. Surprisingly, that is not the case, although it is assumed that further tournament executions may balance out the score averages.

Table 4: *checkSix* vs. *simple*

Tournament	checkSix	simple
1	3568.44	0.0
2	3813.32	0.0
3	3518.27	0.0
4	3433.35	0.0
5	3181.94	0.0

Table 5: *checkSix* vs. *fuzzy*

Tournament	checkSix	fuzzy
1	2586.32	0.0
2	2709.84	0.0
3	2652.23	0.0
4	2245.64	0.0
5	2769.11	0.0

Table 6: *checkSix* vs. *fuzzyMaxPower*

Tournament	checkSix	fuzzyMaxPower
1	83.05	80.34
2	67.07	50.50
3	75.71	94.48
4	29.69	61.04
5	115.98	30.90

Table 7: Bonus round *checkSix* vs. *evilCheckSix*

Tournament	checkSix	evilCheckSix
1	109.72	69.16
2	140.43	54.01
3	194.95	39.28
4	898.02	2.83
5	406.34	7.17