CSG2341 Intelligent Systems Assignment

Title: Saucers – Part B

Related objectives from the unit outline:

- design and use computer packages incorporating the techniques associated with computational intelligence;
- investigate the application of these intelligent systems techniques to a relevant problem;
- apply relevant technique(s) to a task normally considered to require computational intelligence.

Specific learning outcomes:

After completing this assignment, you should be able to apply intelligent systems techniques to design and implement a solution to a “realistic” problem.

Marks allocation:

This assignment is worth 40% of the total mark for this unit. The assignment is in two parts, with two submission dates. Part A is worth 15%, and Part is worth 25%. Both parts can be done in groups of 1, 2 or 3 people.

Due dates:

Part A: Midnight Sunday 30th August

Part B: Midnight Sunday 18th October

Referencing:

All sources of references must be cited (in text citation) and listed (end reference list). For details about referencing and the required format, please refer to the ECU Referencing Guide, which can be found from the following URL: http://www.ecu.edu.au/LDS/pdf/refguide.pdf

Plagiarism:

Please ensure that you have read and understood the information on plagiarism provided on Blackboard under Unit Overview->Essential Information.

Submission requirements:

Assignment materials are to be handed in by the due date, in electronic form by BlackBoard submission.

Part B:

Please read these instructions carefully and thoroughly! There is a lot of information here, all important. It will answer many of your questions if you read it carefully.

Changes:

Some changes have been made to the Saucers game. The main changes are:

- **Multiple saucers** Each contest now takes place between 12 saucers. This is the main change: you will be competing against controllers created by everyone else in the class.
- **Power ups** Extra energy is available from "power ups" that appear from time to time.
- **Enhanced sensors** Sensors provide information about all enemy saucers, as well as power ups and photon blasts.
- **Shields** Saucers can use shields to protect against photon blasts. Shields reduce the impact of a blast, but the saucer uses more fuel while the shields are up, and you cannot fire a photon blast while your shields are up
- Some parameters such as the size of the battlefield speeds of saucers and photon blasts, maximum power of a blast, etc. have changed. Many constants that you might want to use have been moved to `Constants.java`. You can easily access these values in your controller by adding `"implements Constants"` to the class header, and importing `battle.Constants`.
- There is a time limit for each battle.

Some of these changes have required extra sensors for the saucers: saucers can now "see" all other saucers and power ups, and they can also see photon blasts. They also make the problem a lot more complicated, so you will need to make some decisions about how much information to take into account when you design your controller. It may be better to have a well-designed, well-tuned controller that only uses the most important information, and ignores the rest.

Though you can use the idea for your controller from Part A as a starting point, the `SaucerController` interface has changed so you will have to make changes before it will compile. Two example controllers have been provided (neither one uses any fuzzy or other Intelligent Systems methods).

The Game:

Each game starts with 12 saucers on the battlefield. As before, saucers move around the battlefield, using up energy. They can adjust their speed, steer, and shoot self-aiming photon blasts at other saucers. When a saucer runs out of energy, it dies and is removed from the battlefield. The last saucer standing is the winner of the game, and receives a rank of 1. The saucer that lasted next longest receives a rank of 2, and so on. And there are power ups, which appear and disappear from time to time. Saucers can run into power ups to receive an energy boost.

Sometimes, the time limit will be reached before anyone has won the game. In that case, all the saucers still alive receive the same rank (equal to the number of saucers alive).

You can run the game by running the `Main` class. When the battle completes, it prints out a list like this:

```
simple          2
simple          5
simple          10
simple          1
simple          3
simple          4
sitting duck          9
sitting duck          8
sitting duck          12
sitting duck          6
sitting duck          7
sitting duck          11
```

This list reports on the final rakings of the controllers. In this case, the winner was `simple`, with a rank of 1. Actually in this example there are 6 `simples` and 6 `sitting ducks.` The `simples` placed 1,2,3,4,5 and 10. These are examples that you can use to give you ideas and to see how a controller could be written.

In the following screenshot you can see 12 saucers and one power up (which is a red-green pulsing disk). Two of the saucers have their shields on (the ones with the circles around them).

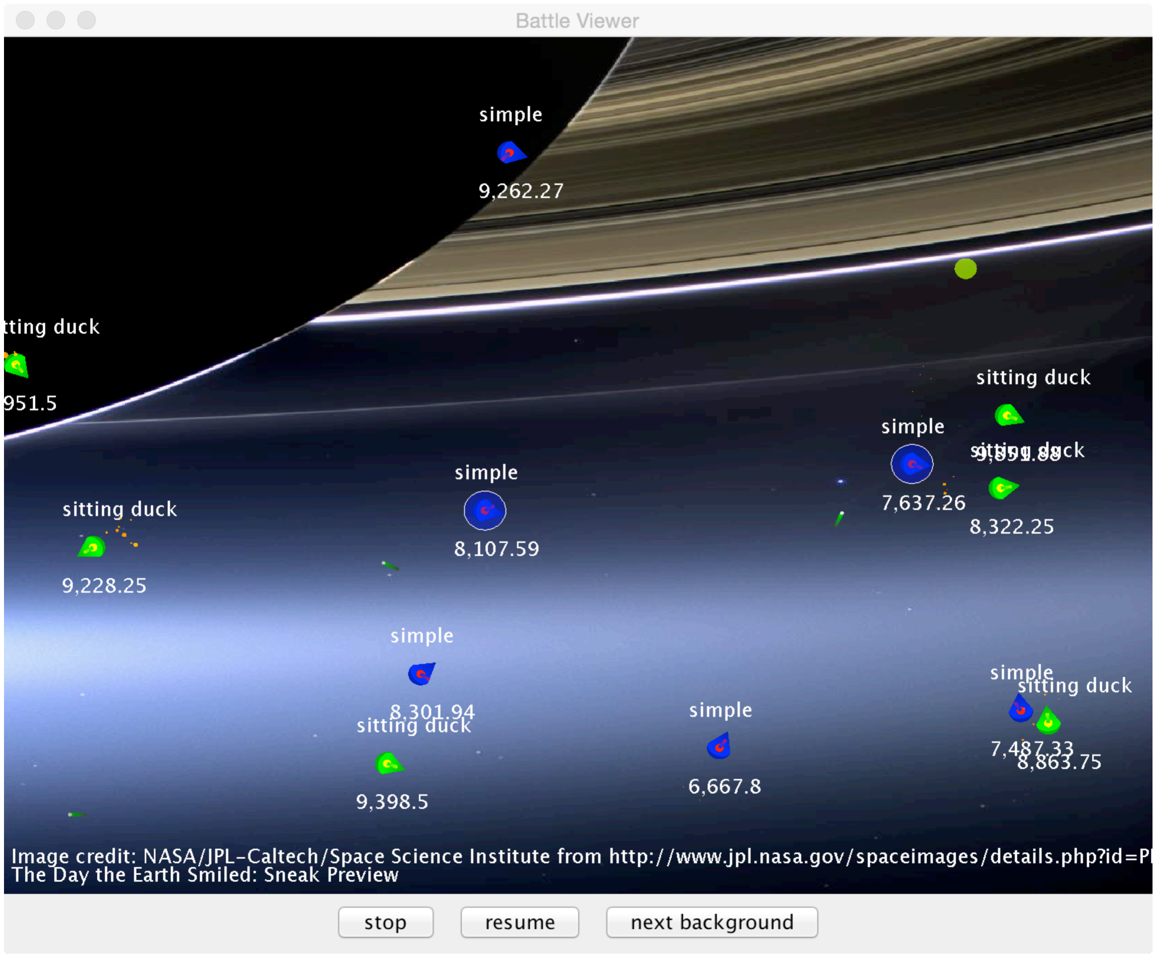

Image credit: NASA/JPL-Caltech/Space Science Institute from http://www.jpl.nasa.gov/spaceimages/details.php?id=Pl
The Day the Earth Smiled: Sneak Preview

You can also run a tournament by running the `Tournament` class. This will run lots of battles and print a summary like this at the end:

```
simple 2 8     9     1     4     7     6     4     4     1     4.6
simple 1 10    2     4     8     9     5     1     10    9     5.9
simple 5 6     3     8     10    1     1     2     3     10    4.9
simple 7 7     6     7     6     2     3     5     11    5     5.9
simple 3 4     7     3     2     4     4     6     7     2     4.2
simple 4 9     1     2     5     6     2     3     1     8     4.1
sitting duck 10 12 8     6     12    5     11    11    9     11    9.5
sitting duck 6 3   5     9     7     10    9     8     5     12    7.4
sitting duck 11 11 10    10    11    3     7     7     8     4     8.2
sitting duck 12 5  4     11    9     12    10    9     12    6     9.0
sitting duck 9 3   12    5     3     8     8     12    6     3     6.9
sitting duck 8 3   11    12    1     11    12    10    2     7     7.7
```

Each column shows the placings of the controllers in one battle. The last column is the average placing for each controller – so low numbers are better.

The Controller:

As in Part A, your task is to write a controller for a saucer in the game. The `SaucerController` interface has changed a bit from Part A. Examples of `SaucerControllers` are provided.

The `SaucerController` interface now looks like this:

```
public interface SaucerController
{
public void senseSaucers(ArrayList<SensorData> data) throws Exception;
public void sensePowerUps(ArrayList<SensorData> data) throws Exception;
public void senseBlasts(ArrayList<SensorData> data) throws Exception;
public void senseEnergy(double energy) throws Exception;
public SensorData getTarget() throws Exception;
public double getFirePower() throws Exception;
public double getSpeed() throws Exception;
public double getTurn() throws Exception; // in degrees
public boolean getShields() throws Exception;
public String getName();
public Color getBaseColor();
public Color getTurretColor();
}
```

When the battle runs, it will first call the methods `senseSaucers()`, `sensePowerUps()`, `senseBlasts()`, and `senseEnergy()` to give your controller information on the location and status of all the other saucers and all the power ups and photon blasts on the battle field, and your own energy level. You should save any of this information that you need and set your fuzzy variables when these methods are called. A convenient place to update your fuzzy rules would be at the end of the `senseEnergy()` method.

The simulation will then call your `getTarget()`, `getFirePower()`, `getTurn()`, `getShields()` and `getSpeed()` methods. It will then attempt to fire a blast at the specified target, turn the specified number of degrees, raise or lower shields, and set your speed to the specified value.

Please note that if your controller throws an error during a battle, it will be disqualified at that point and forced to commit suicide.

This cycle is repeated over and over until your saucer either dies or is declared the winner.

Now here is a description of each of these methods:

senseSaucers() This method gives you information about all the other saucers on the battlefield. This information is passed in via the variable `data`, which is an array of `SensorData` objects. If there are no other saucers, data will be have a `size()` of `0`. Each `SensorData` object in the array contains information about another saucer, in the following fields:

- `distance` – this is the distance between your saucer and this saucer;
- `direction` – this is the angle between the direction you are currently heading and a line of sight to the object, in degrees. So if the object is straight ahead of you, the direction would be 0 degrees.
- `heading` – this is direction that the object is heading, relative to the direction you are heading. So, if it was heading straight toward you and you were heading straight towards it, the heading would be 180 degrees.
- `speed` – this is the speed that the object is travelling at.
- `energy` – this is the energy level of the object.
- `ID` – this is a unique identifier for the saucer (you could use this to keep track of a particular opponent).

sensePowerUps() This method is the same as `senseSaucers(),` but gives information about any power ups on the battlefield.

senseBlasts() This method is the same as `senseSaucers(),` but gives information about any photon blasts on the battlefield (except ones that you fired yourself).

senseEnergy() This method tells you what your current energy level is.

getTarget() You use this method to specify which other object you wish to fire a photon blast at. Pass in the `SensorData` object for the saucer that you want to fire at. If you don't want to fire, pass in `null`.

getFirePower() You use this method to specify the strength of the blast. No matter what power you set, the blast will not be less than `0` or more than `SAUCER_MAX_POWER`.

getTurn() You use this method to specify the number of degrees that you want to turn.

getSpeed() You use this method to specify the speed that you want to move at. No matter what speed you ask for, the saucer will not go slower than `SAUCER_MIN_SPEED` or faster than `SAUCER_MAX_SPEED`.

getShields() Return true if you want shields up, false otherwise. Note that you will not be able to fire when the shields are up, and you will use extra fuel.

The `simple` controller has example code to show how you might use the `ArrayLists` of `SensorData`.

To test your controller, locate this part of Main.java and Tournament.java:

```
private static SaucerController[] controllers;
static
{
     try {
          controllers = new SaucerController[]
          {
               new SimpleController(),
               new SimpleController (),
               new SimpleController (),
               new SimpleController (),
               new SimpleController (),
               new SimpleController(),
               new SittingDuckController (),
               new SittingDuckController (),
               new SittingDuckController (),
               new SittingDuckController (),
               new SittingDuckController (),
               new SittingDuckController ()
          };
     }
     catch(Exception e)
     {
          System.err.println("Unable to create
          controllers");
          e.printStackTrace(System.err);
          System.exit(0);
     }
}
```

and replace one of the `SimpleControllers` or `SittingDuckControllers` with your own controller. Please give your controller a unique name (not `MyController`, for example).

Performance measurement:

As in Part A, your task is to write the best controller you can, making use of fuzzy reasoning. There are now fewer marks allocated to the performance of your controller – just 10%, consisting of

- 5% completion bonus – So long as your controller genuinely uses fuzzy rules, and does not fail (throw exceptions), you will receive this 5%.
- 5% tournament ranking – All the successful final submissions will go into a tournament for ranking. If there are N submissions, each will get a rank between 1 (the best) and N (the worst), depending on performance in the tournament. Your mark out of 5 will then be calculated as 5x(N+1-rank)/N, rounded to the nearest ½ %. So the best submission gets 5% and the worst gets 0%.

The ranking will be calculated as follows: Each submitted controller will play some number of games (probably about 10 or 20) and will receive a rank for each game. Each game will be played against a different, randomly selected set of 11 other submitted controllers. The controller's score will then be calculated as the average rank achieved in these games. All of the controllers will then be sorted by score. The one with the highest score will receive a ranking of 1, the next highest a ranking of 2 etc.

You might like to help each other out by posting your controller's class file on the discussion board (not the java file), so you can have some competition to train and test against. Please note that in the past some unscrupulous students have decompiled other students' class files and submitted them. Also note that they were caught, reported for cheating, and given a mark of 0.

Report:

Write a 1500 word report documenting your work, with at least the following headings:

1. Introduction
   A short introduction. You don't need to provide details of the task – just refer to this document.
2. Idea
   Describe the idea behind your controller in simple English. Include a description of how your controller makes each important decision, including at a minimum how fast to go, what direction to go, whether to fire and what to fire at, whether to use shields or not.

3. Fuzzy Variables
   List the input and output (and any other) fuzzy variables and include membership diagrams for each fuzzy set.
4. Learnings
   How well did your controller work, and what did you learn from Part B?
5. Conclusion
   A simple conclusion should round out the report. Summarise what you did and what you learned.

1500 words is a guide. The report doesn't have to be exactly 1500 words, but if it's a lot less, say 700 or 800, or if important information is missing, expect to lose marks.

Please include details of your team members (names and student numbers).

Marking Key:

Part B is worth 25% of your mark for Intelligent Systems. This is made up of

- Appropriate use of Intelligent Systems techniques.
  This will be judged mainly on your project report.
  (15%)
- Performance of your controller.
  (10 %)

PLEASE READ THE NOTES ON THE NEXT PAGE CAREFULLY

Checklist of important things to remember for this assignment:

- You must work in groups of 1, 2 or 3.
- Only one submission for each group – be sure to include a list of group members in your report.
- If you are having trouble getting started, get help from your tutor in *plenty of time*.
- Do not change any of the code except your own controller and the sections of Main and Tournament where you add your controller.
- Your "controller" MUST use fuzzy reasoning in a non-trivial way.
- If your controller throws errors, you will not do well. Work out why the errors are being thrown and fix them.
- How well you do in the tournament depends on what other controllers are doing! Just doing well against the example controllers might not be enough.
- You must hand in a zip file containing
    - Your project report, and
    - Your source code.
    - **Please don't zip up your project folder with all the image and sound files.**