SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30002

Planet Wars

Task 5, Week 3

Matthew Coulter
S102573957

26/03/2020

Tutor: Tien Pham

Table of Contents

Getting the game to start

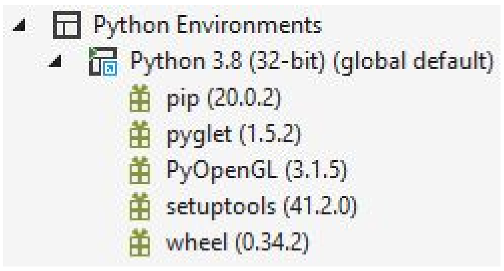
Packages

From the provided code, importing it into visual studio and attempting to compile will not work straight away. This is because there are particular packages required by python for the game to be able to open and run the game window. These packages are called pyglet and PyOpenGL. To install these, you first need to install pip which is a python package manager. To do this:

1. open a command window
2. change directory to where you wish to install pip
3. enter the command: 'curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py'
4. then enter the command: 'python get-pip.py'

Now that pip is installed, you will be able to use your command window to install the required packages. In this case, PyOpenGl and pyglet can be installed by entering the commands:

1. pip install pyglet
2. pip install PyOpenGl

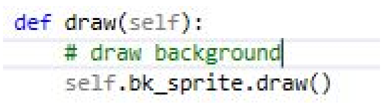


```
Python Environments
    Python 3.8 (32-bit) (global default)
        pip (20.0.2)
        pyglet (1.5.2)
        PyOpenGL (3.1.5)
        setuptools (41.2.0)
        wheel (0.34.2)
```

When you open your project solution in visual studio, you should now see the packages you just installed under Python Environments >> Python 3.8 (32-bit) (global default) [or whatever version you have]

Draw code modifications

The provided code actually commented out some of the drawing commands that would cause the background to be missing:



```python
def draw(self):
    # draw background
    self.bk_sprite.draw()
```

How planet wars works

Planet Wars is a strategy game. It functions as an accumulative one where the more ships you get, the more planets you will be able to take out which inturn will likely take out more planets. The game is over once one player has consumed ownership over all of the planets. At the beginning of the game, lots of planets are gaia. This means the first player to take ownership of the planet gets it. If an enemy fleet of ships comes into one of your planets, and they have more ships than you do, they will then own the planet and the difference between you and them will become the amount of ships that they own the planet with.

For example, if an enemy sends over a fleet of 250 ships into one of my planets that has 200 ships, they will own the planet with 50 ships.

Basic strategy for this AI

The move that is played is based off of 2 variables: src and dest.

src is where the attack is coming from,
dest is the destination for where the planet will attack.

The provided code:

```python
# Always send from the planet with the highest value
src = max(gameinfo.my_planets.values(), key=lambda p: p.num_ships)
# Always attacks the planet with the lowest value
dest = min(gameinfo.not_my_planets.values(), key=lambda p:
p.num_ships)
```

uses a very simple method which sends a fleet from the planet with the most ships into the opponent's planet with the least ships. Whilst this is very basic, it covers a large part of the strategy of the game, and in turn plays much better than the OneMove Ai that is already implemented. The OneMove simply grabs the first planet in the list of enemy planets and attacks from the first planet in the list of owned planets:

```python
src = list(gameinfo.my_planets.values())[0]
dest = list(gameinfo.not_my_planets.values())[0]
```

Here is a log of a game that displayed the choice of a handful of moves from the a given game of MyNewBot vs OneMove. In particular, pay close attention to the amount of ships on the left side that attacked the amount of ships on the left side.

```
Planet Planet:2, owner: 1, ships: 105 attacked Planet Planet:16, owner: 0, ships: 10
Planet Planet:2, owner: 1, ships: 88 attacked Planet Planet:17, owner: 0, ships: 10
Planet Planet:16, owner: 1, ships: 83 attacked Planet Planet:2, owner: 2, ships: 2
Planet Planet:17, owner: 1, ships: 77 attacked Planet Planet:22, owner: 0, ships: 18
Planet Planet:16, owner: 1, ships: 53 attacked Planet Planet:23, owner: 0, ships: 18
Planet Planet:22, owner: 1, ships: 61 attacked Planet Planet:12, owner: 0, ships: 19
Planet Planet:17, owner: 1, ships: 59 attacked Planet Planet:13, owner: 0, ships: 19
Planet Planet:23, owner: 1, ships: 71 attacked Planet Planet:10, owner: 0, ships: 27
Planet Planet:22, owner: 1, ships: 100 attacked Planet Planet:1, owner: 2, ships: 5
Planet Planet:12, owner: 1, ships: 94 attacked Planet Planet:11, owner: 0, ships: 27
Planet Planet:1, owner: 1, ships: 105 attacked Planet Planet:20, owner: 0, ships: 27
Planet Planet:10, owner: 1, ships: 134 attacked Planet Planet:21, owner: 0, ships: 27
Planet Planet:11, owner: 1, ships: 143 attacked Planet Planet:14, owner: 0, ships: 39
Planet Planet:20, owner: 1, ships: 179 attacked Planet Planet:1, owner: 2, ships: 5
Planet Planet:13, owner: 1, ships: 199 attacked Planet Planet:15, owner: 0, ships: 39
```

```
Planet Planet:21, owner: 1, ships: 261 attacked Planet Planet:10, owner: 2, ships: 31

Planet Planet:23, owner: 1, ships: 256 attacked Planet Planet:18, owner: 0, ships: 56

Planet Planet:14, owner: 1, ships: 256 attacked Planet Planet:19, owner: 0, ships: 56
```

The list goes on and on…

It is clear that the MyNewBot AI is sending over fleets way too big for what is necessary in order to capture the opposing planet. Another disadvantage of this gameplay style is that the AI are sending these fleets very far, even across the whole galaxy! This in turn creates a very sluggish gameplay. Whilst it withstands the OneMove AI, I don't believe it will hold up against anything more complex.

How Lambda Functions Work

This task introduced a new type of function in python which is called a Lambda function. It is important that we cover this first as it can be used for our smart AI. Lambda functions are a one-line , disposable function. They still take in a given argument and return a result, however they can only be used on a single line. Whilst this may seem very limiting, it can in turn create a very efficient way of calculating something as python libraries allow one line to do a lot!

A simple Lambda function can look like this:

```
add10 = lambda x: x + 10
```

`add10(5)` would return `15.`

A more complex Lambda function looks like this:

```
dest = max(gameinfo.not_my_planets.values(), key=lambda p: 1.0 / (1
+ p.num_ships))
```

This function is the other provided solution on how to find which planet should be attacked.

First of all, max will find the highest value in a given list. Without the lambda function here, the function would return the planet with the most amount of ship on it. However, because the lambda function is here, the `gameinfo.not_my_planets.values()` is sent into the lambda function as an argument, modified then the maximum is found from the modified values.

For example, if the num_ships for a given set of planets is {100,50,6,75,318}, this lambda function returns the maximum value once `1.0 / (1 + p.num_ships)` is calculated. Once this function is calculated on each planet, the new list is {0.0099,0.0196,0.1428,0.0131,0.0031}. From here, the maximum value is 0.1428, hence the planet with a ship fleet size of 6 would be returned.

How to implement a smarter AI

Given the current disadvantages of:

- Travel distance
- Efficiency of required size of fleets

Here we are able to create a better AI finding a ship nearby that just less ships than our planet does.

Here is the process:

1. Ignore any planets that has more ships that the fleet that will be sent out to attack it

```
# Ships that have less ships than our src fleet
# src.num_ships is multiplied by 0.75 as that is the size of the fleet
that is being sent out
lessShips = filter(lambda x: x.num_ships < round(src.num_ships*0.75),
gameinfo.not_my_planets.values())
```

2. From this list, calculate a value for each planet that determines how worthwhile the attack is. This value is determined by dividing the number of ships the enemy planet has by the travel distance.

```
# Chooses destination based off of the highest value that represents the
ratio between distance and value is calculated

dest = max(lessShips, key=lambda p: p.num_ships/p.distance_to(src))
```

This method works rather well. It gains capital over most of the planets before the enemy has a chance to, then each one of those planets is growing so MyNewBot AI is able to build up a lot of points quickly. There is only one more condition that needs to be put into place:

After MyNewBot has obtained most of the ships, there may be a point where the enemy planets have more ships than any of ours, hence lessShips will be None. We can fix through the pythonic way of adding a ‘default =’ statement into the max function. In the case the lessShips is empty, it will use the default instead. For the default, I have made it so I use the same lambda function but instead use the list for all enemy ships instead of the lessShips.

```
# Chooses destination based off of the highest value that represents the
ratio between distance and value is calculated

dest = max(lessShips, default=gameinfo.not_my_planets.values(),
      key=lambda p: p.num_ships/p.distance_to(src))
```

Below is a similar output where we can analyse the performance of this new and improved AI:

Planet 2 attacked Planet 5 from a distance of **3** with **79** ships

Planet 2 attacked Planet 20 from a distance of **3** with **32** ships

Planet 2 attacked Planet 13 from a distance of **7** with **23** ships

```
Planet 5 attacked Planet 15 from a distance of 11 with 40 ships
Planet 20 attacked Planet 19 from a distance of 11 with 60 ships
Planet 5 attacked Planet 18 from a distance of 10 with 62 ships
Planet 20 attacked Planet 8 from a distance of 4 with 81 ships
Planet 15 attacked Planet 23 from a distance of 2 with 79 ships
Planet 13 attacked Planet 14 from a distance of 8 with 62 ships
Planet 5 attacked Planet 4 from a distance of 19 with 72 ships
Planet 20 attacked Planet 7 from a distance of 8 with 125 ships
Planet 15 attacked Planet 11 from a distance of 3 with 137 ships
Planet 14 attacked Planet 22 from a distance of 2 with 110 ships
Planet 23 attacked Planet 6 from a distance of 15 with 108 ships
Planet 11 attacked Planet 9 from a distance of 15 with 136 ships
Planet 20 attacked Planet 10 from a distance of 13 with 166 ships
Planet 5 attacked Planet 21 from a distance of 20 with 194 ships
Planet 15 attacked Planet 12 from a distance of 8 with 244 ships
```

The code used to print this is below

```python
print("Planet %s attacked Planet %s from a distance of %s with %s ships" %
(src.id,dest.id,round(src.distance_to(dest)),round(src.num_ships*0.75)))
```

By looking at the distance attacked and the number of ships, it is clear that the AI will only travel far if it carries a very large fleet. Otherwise, it is attacking closeby with a moderate amount of ships.