Project: Tanks

CSE/IT 107

NMT Department of Computer Science and Engineering

> "TANK TANK TANK"
>
> — Intro to *Tank! Tank! Tank!*, Bandai Namco

---

1 Tanks

Dirtbags Tanks

This project is inspired by Dirtbags Tanks by Neale Pickett. More information is available at `http://woozle.org/tanks/intro.html`.

1.1 Project Overview

You will be provided with several files comprising the Tank project. This is the skeleton of a game involving several tanks. It consists of the files `game.py`, `sample_tank.py`, `sensor.py`, `tank.py`, and `tankutil.py`. For a description of what each of these files do, stay here. For a description of your assignment, see Section 1.2.

1.1.1 game.py

`game.py` contains two important things: `class Game` and the `main()` function of the program. `Game` contains the code that controls the interaction between tanks (collision detection and firing) as well as the drawing of objects to the tkinter window. You should not have to worry about how it does these things.

The `main()` function, however, you will need to edit, slightly. This is where you will add your own tanks to the `Game` instance that is already being created. To do this, simply imitate how the `SampleTanks` are being added. Note that you will also need to add an import at the start of the file in order to be able to access your tanks from other files.

1.1.2 tank.py

`tank.py` contains class Tank, which includes all the methods and attributes needed for general functionality of tanks. All of your custom tanks should inherit from `Tank`. *You should not edit this* *file whatsoever.* Each tank has a large collection of attributes that control its behavior. Most of these you should not touch, but for some of the required tanks you may need to change the values of specific attributes in your custom constructor.

For your tanks, you will need to redefine `ai(self, delta)`, which is the method called each simulation step to determine what the tank should do. Inside of this method, the only other methods you should call are those in the section of `tank.py` labeled "`METHODS TO BE USED BY AI`". They allow you to set the desired speed of the tank treads, the desired angle of the turret, and whether the tank should fire. They also allow you to check the current status of the treads, turret, and the sensors (described in Section1.1.4). Not using these methods will result in your tank not behaving properly, as properties such as tread acceleration, max speed, and turret speed may be ignored.

You should also redefine $_init_{tospecifythetank'ssensorsaswellasanyotherattributesthatmaydifferfromthedefaulttank(lefttoyourdiscretion)}$.

1.1.3 tankutil.py

`tankutil.py` contains a few miscellaneous functions used by the other files. It is relatively inconsequential and you won't need to use it, though it may be useful to look at if you want to set custom colors for your tanks.

1.1.4 sensor.py

`sensor.py` contains class Sensor, which is used to define the sensors for tanks. `Sensor` contains nothing but a connstructor and a few attributes. The actual logic for sensors resides inside `game.py`.

A sensor is defined from four attributes:

Direction A value in degrees that determines what direction the sensor is facing. This direction will indicate the middle of the sensor. 0 means directly in front of the tank, while 180 means directly behind the tank.

Width A value in degrees that determines how wide the sensor is. The sensor will detect tanks in half this number of degrees in each direction from the direction chosen above.

Size How far the sensor reaches. This distance is in pixels. For reference, the default range of a tank's gun is 50.

Tracking Whether or not the sensor will move as the gun's turret moves. This value should be either `True` or `False`. If this value is `False`, then the sensor's direction will be relative to the tank's current facing. If this value is `True`, then the sensor's direction will be relative to the turret. This allows you to have sensors that, for example, check if there is something to the left or right of the turret that could be shot if the turret was moved slightly.

The sensors for your custom tanks should be created inside of the tank's constructor. To check whether the sensor has detected a tank, use `self.read_sensor(num)`, where `num` is the index of the sensor inside of `self.sensors`.

Each sensor will be shown graphically, and if it detects a tank then it will be filled in with the color of its parent tank to indicate that it is active.

1.1.5 sample_tank.py

`sample_tank.py` is an example of how you might go about making your own custom tanks. You are free to edit this file, but your final submissions should be in separate files (as described in Section 1.2). `class SampleTank` defines two methods: `__init__` and `ai(self, delta)`. `__init__` is be used to define two sensors: one large one that looks for anything in front of the tank and one smaller one that follows the turret to check for anything that can be shot. It also determines whether the turret will turn clockwise or counter-clockwise.

`ai` is used to control the tank's behavior. The first thing this tank does is get the current angle of the turret, then set the desired angle to be higher or lower, depending on what chosen in the constructor. This results in the turret continuously turning in one direction at a constant rate.

Next, the turret reads from sensor 1 and checks if the turret is able to fire. Sensor 1 is the smaller sensor that is following the turret. If both values are true, then the turret fires.

Finally, sensor 0 is checked. This is the large one in front of the tank. If anything is detected, the tank reverses. If not, it continues forward, with the left tread at a slightly higher speed than the right one. This results in the tank slowly turning to the right.

1.2 Your Mission

You will be submitting 6 tanks that inherit from the `Tank` class in `tank.py`. Each of your tanks should override, at the very least, the `__init__` and `ai(self, delta)` methods of the parent class.

You will be graded primarily on code quality, so remember to follow PEP-8 and to document everything! Each tank file should give an overview of how its logic works and why you made it work that way.

Exercise 1.1 (README.txt).
You should have a README file briefly describing the logic behind each of your tanks. If you changed anything in the given files besides the main() of `game.py`, mention it here and explain why. Mention some changes you found that made tanks more or less successful.

Exercise 1.2 (coward.py).
This tank is nervous and doesn't like social interaction. It should do its best to avoid all interaction with other tanks and evade them as much as possible. Remember, this means more than simply going backwards if something is in front of it – think about what your tank should do if there's one tank in front of it and another behind it!

Exercise 1.3 (charger.py).
This tank is very enthusiastic to show you how well its turret works! It should chase down any tank it sees and give them a demonstration of how well its gun works![a] If the gun isn't ready, it should try to stay away from others so as not to reveal that the gun is not completely reliable.

[a] By shooting them.

Exercise 1.4 (turret.py).
This tank got its treads stuck in the mud. It should never move at all and should move its turret, shooting anyone who comes near. It should also attempt to track tanks as they approach if there isn't anyone currently in range to be shot at.

Exercise 1.5 (elephant.py).
The Elephant is a new experimental heavily armored tank. It is far slower and larger than the other tanks while also being harder to kill.

To accomplish this, you will need to change some of the default tank attributes, namely shape, radius, tread_accel, and tread_max. You will also need to redefine `kill(self)`, which is called whenever a tank is shot or runs into another tank.

Exercise 1.6 (mouse.py).
The Mouse is a new experimental hyper fast tank. It is far more agile and petite than the other tanks, attempting to succeed by dodging around other tanks.

Similarly to with the Elephant, you will need to change some of the default tank attributes.

Exercise 1.7 (custom.py).
Finally, design your own custom tank that works however you wish. Try to give it an AI that lets it combine the strategies of the other tanks in a way the makes it better than all of them!

1.3 Bug Bounty

The provided code was written fairly quickly and probably has some bugs. As such, if you report a bug that are not previously known, you will be given 5 points of extra credit. If you can provide a *fix* to a bug, you will be given 10 points. Report all bugs/fixes to Russell if you want to receive credit.

2 Submitting

You should submit your code as a tarball. It should contain all files used in the exercises for this lab. The submitted file should be named

`cse107_firstname_lastname_tanks.tar.gz`

Upload your tarball to Canvas.

List of Files to Submit