Tanks Kata

About this Kata

This kata is based on a class assignment by Russell White at New Mexico Tech.
https://github.com/Arctem/nmt_python_labs/tree/master/labs/project

The class assignment is based off a project called Dirtbags Tanks by Neale Pickett.
http://dirtbags.net/tanks/

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`, `main.py`, `sample_tank.py`, `sensor.py`, `tank.py`, and `tankutil.py`.

The project requires Python 3. To run the program, type "`python main.py`" at a command prompt.

Below is an explanation of each file in the project. A * next to the name means that you will need to modify the file for the kata, otherwise you should not edit the file.

main.py*

`main.py` contains the `main()` function of the program. You will need to modify this file in order to 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.

game.py

`game.py` contains `class Game`. `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.

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. 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__` to specify the tank's sensors as well as any other attributes that may differ from the default tank (left to your discretion).

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. Once again, you should not edit this file.

sensor.py

`sensor.py` contains `class Sensor`, which is used to define the sensors for tanks. `Sensor` contains nothing but a constructor 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.

As with `game.py`, `tank.py`, and `tank_util.py` you should not edit this file.

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 own tanks should be in separate files. `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 was 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.

Task 1: Coward Tank

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!

Task 2: Charger Tank

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 good it is at shooting things (by shooting them)! If the gun isn't ready, it should try to stay away from others so as not to reveal that its gun is not completely reliable.

Task 3: Turret Tank

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.

Task 4: Elephant Tank

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 `damage(self)`, which is called whenever a tank is shot or runs into another tank.

Task 5: Mouse Tank

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.