CSCI 1133, Fall 2018 Due: 5:00 pm, Friday, Dec 7

Programming Examination 3

PLEASE NOTE: This is a graded assessment of individual programming understanding and ability, and is ***not*** a collaborative assignment; you must design, implement and test the solution(s) completely on your own without outside assistance from anyone. You may not consult or discuss the solution with anyone. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class. Note that *providing* a solution or assisting another student in any way on this examination is also considered academic misconduct. Failure to heed these directives will result in a failing grade for the course and an incident report filed with the Office for Student Conduct and Academic Integrity for further sanction.

A. (100 points) **Turtle Saves the World**

This examination consists of designing, modifying and adding functionality to a single well-structured Python program that must be committed and pushed to your remote GitHub examination repository *prior* to the deadline. Note that your program *must* be named `turtlegame.py` (all lower-case) and must modify the existing file `turtlegame.py` that is currently provided in the Modules section of the course Canvas page.

Late submissions will not be accepted and will result in a zero score for the exam.

TA help for this examination will not be provided. If you have clarification questions, they must be addressed to the graduate TAs for the class.

The total point value will be awarded for solutions that are *complete*, *correct*, and *well structured*. A *"well structured"* program entails good design that employs meaningful functional decomposition, appropriate comments and general readability (descriptive names for variables and procedures, appropriate use of blank space, etc.) If you are not sure what this means, review the "well-structured" program requirements provided in Lab2.

Note that your work will be graded using, and must function correctly with, the current version of Python 3 on CSE Labs UNIX machines. If you complete this programming exam using a different system, it is *your* responsibility to ensure it works on CSELabs machines prior to submitting it.

The rubric includes but is not limited to the following specific (accumulative) point deductions:

• Missing academic integrity pledge	-100 points
• Syntax errors	-50 to -100 points
• Misnamed source file or incorrect repository	-25 points
• Use of global variables	-25 points
• Missing main program function	-25 points

Examination Repository

Examination files must be submitted to GitHub using your remote exam repository. Exam repositories have already been created for each registered student and are named using the string `exam-` followed by your X500 userID (e.g., `exam-smit1234`). You must first clone your exam repository in your local home directory before submitting the materials for this exam. If you are having difficulty, consult the second Lab from earlier in the semester or the GitHub tutorial on the class webpage. If your exam repository is missing or something is amiss, please contact the graduate TA. Do not create any additional folders or directories in your exam repository. DO NOT SUBMIT YOUR EXAM FILES TO YOUR LAB/EXERCISE REPOSITORY!

Background

Programming Exam 1 and 2 asked you to create stand alone programs, essentially from scratch. While this is a valuable skill for learning programming, in many software development contexts we are expected to apply what we have learned to existing code in order to improve or augment existing functionality.

In **Turtle Saves the World**, you will be working within a pre-existing piece of Python code (currently saved to the course Canvas page as `turtlegame.py`) to finish the implementation of an interactive game that utilizes animation and motion.

In the game **Turtle Saves the World,** ghosts from the PacMan universe have escaped their arcade game and invaded the world! Only our fearless Turtle, armed with a jet pack and and ghost-destroying laser beams, can save us. A complete version of **Turtle Saves the World** allows players to interactively "fly" Turtle around the screen using input from the keyboard. During the course of the game, Turtle can shoot laser beams at a series of randomly flying ghosts in an effort to eliminate them all and save the world. If Turtle runs into a ghost, she loses a life — Turtle has only three spare lives, so the game will end once she has lost all of her lives or eliminated all of the ghosts.

Note: to avoid confusion with other Turtle Graphics objects, throughout this exam description we will refer to the flying turtle in this game as Tiny.

Writing an interactive program is hard work and a big job! We have developed this programming exam so that you can implement functionality one piece at a time. For each piece of functionality you are able to fully implement, you will earn points. At each stage, ensure the current functionality is working before trying to add another.

WARNING: WE STRONGLY RECOMMEND VERSIONING YOUR PROGRAM. This means we recommend saving your work as you go along, in stages. Git is designed to do this! After you get one piece of functionality working, ensure you save everything you have done up to that stage, and push it to your exam repository. When adding functionality to a program, it is very easy to make one small change and all of a sudden have several previously implemented additions break. If you do not have a record of what your program was when it previously worked, it can often be very hard to fix. Because your points will be attained cumulatively based on functionality completed, it is very important that you submit a working version of the program (even if it does not have every single piece of functionality implemented). Save your work as you go.

Implementing Turtle Saves the World

Head to the Modules page of our Canvas course page and download `turtlegame.py, pinkghost.gif,` and `blueghost.gif.` Make sure to save all of these files in the same directory (we will refer to the place you have saved all of these files as your "Working Directory")! **Note:** pinkghost.gif will be used only in an extra credit task, you do not initially need this file.

Once you have saved all of the files in your Working Directory, go ahead and compile `turtlegame.py` by typing `python3 turtlegame.py` into the terminal window.

You will see a bare bones, partial implementation of **Turtle Saves the World.** When the program is compiled, a window should pop up with the title "Turtle Saves the World," and you should see blue ghosts flying around on the screen and a purple turtle (we will call this purple turtle Tiny) placed on the screen.

Try using the left arrow — Tiny turns to the left! That's neat! If you press the up arrow, you active Tiny's jet pack boosters and she zooms into the direction she is pointing. Note that Tiny is only boosted from her starting position, so if you try to turn Tiny while she is drifting she will not change course.

Try turning Tiny to the left and pressing the up arrow again — now she picks up speed (she has received another boost!) and zooms in the direction you have pointed her.

Repeatedly pressing the up arrow makes Tiny go faster and faster. To stop Tiny, press the down arrow. Next, try to press the right arrow. Nothing happens!

This is the first piece of functionality that you will need to implement as Task 3. Before we get there, though, let's try to understand what's happening in this code by taking a deeper look at `turtlegame.py.`

Open `turtlegame.py` in Atom or another text editor.

At the top of the file, you can see the modules we need to import to create this game. We have the usual suspects, the `Turtle` module (note we use `from turtle import *` instead of `import turtle` — this allows us to use the names from `Turtle` without typing `turtle.` in front of each method name), `random`, and `math`, but we also have something new with the `datetime` module (we will use this to set a timer for our game), and the `tkinter` module and its associated `messagebox`. We will give you the code you need to use with these new modules, but you should still give a brief read to the Python 3 documentation describing how these work. You will often find that understanding new modules and methods is an important part of working with pre-existing code! Since you are working with something that was developed by another programmer, it is often essential to spend some time at the front end acquiring knowledge that is new to you.

Reading a little further down the file, we see that `turtlegame.py` utilizes classes to handle the objects that are flying around on the screen. Ah ha! We know something about that.

Take a look at the line `class Ghost(RawTurtle):`

`RawTurtle` is what is called a Parent class to `Turtle` — we will discuss this bit of technicality in lecture. For now, what you should know is that to use the types of graphics we want for this project, when we create a class for an object that will fly on the screen, we want to include RawTurtle as an 'argument' to the class name.

Inside of the `Ghost` class, we see the expected class constructor (we will be drawing on `canvasobj`), a `getRadius` function, and a `move` function, all of which will be explained in detail later. For now, let's focus on one important point: all of the attributes inside of `Ghost` are public! We would like to make these private.

TASK 1: Make all attributes inside of the Ghost class private, and write appropriate accessor and mutator functions (10 points)

Task 1 is to make any attributes that are created inside of the Ghost class constructor private, and to create and implement methods to access private attributes DX and DY, and to change DX and DY to new values. A successful implementation of Task 1 will set 3 attributes to private and create 2 accessor and 2 mutator methods that function correctly. HINT: you may need to look through other parts of the code to see where previously public variables were used!

Next, take a look at the `FlyingTurtle` class, defined below class `FlyingTurtle(RawTurtle):.` This class handles the tiny purple turtle that is zooming around on the screen. You can see similar attributes to the `Ghost` class, but a few different methods. Because we want to be able to fly Tiny around, we need a few extra methods that will give a `turboBoost` to her jet pack and also `stop` her flight path. But wait — all of the attributes in this class are public, too!

TASK 2: Make all attributes inside of the FlyingTurtle class private, and write appropriate accessor and mutator functions (10 points)

Task 2 is to make any attributes that are created inside of the FlyingTurtle class constructor private, and to create and implement methods to access private attributes DX and DY, and to change DX and DY to new values. A successful implementation of Task 3 will set 3 attributes to private and create 2 accessor and 2 mutator methods that function correctly. HINT: you may need to look through other parts of the code to see where previously public variables were used!

Make sure that after Tasks 1 and 2 are completed that your code still compiles, and fix any errors as needed.

OK, now we are ready to tackle the `main()` function. First, we do some initialization for the canvas, which is what we will be "drawing" on in our game. This code should not be altered. You will likely find it interesting to read about what some of these methods do in the Python documentation if you haven't seen them before.

Take a look at the line `screen.register_shape("blueghost.gif")`. This is what allows us to use customizable shapes in Turtle. If we wanted to make our game Turtle vs. the Flying Jellyfish, we would only need to substitute out a .gif image type of a jellyfish for our ghosts. This is a cool and fun thing to use with Turtle, so if you have extra time you might enjoy playing around with it! As long as we have registered the shape, we can use it elsewhere in our programs.

You can also see where we create Tiny, with the line `flyingturtle = FlyingTurtle(canvas,0,0,(screenMaxX-screenMinX)/2+screenMinX,(screenMaxY-screenMinY)/2 + screenMinY,3)`

This line calls the constructor of the `FlyingTurtle` class and sets the attributes to the variables we have passed in. Great!

Next, we need to create some `Ghosts`. Since we have more than one `Ghost`, and eventually we would like to zap our Ghosts with lasers and eliminate them, we will store the ghosts in a list: `ghosts[]`

The next few lines of code create several `Ghost` objects and randomly assign them a starting location on the screen. You should read through this code carefully and work out what is happening - it will be useful to you later in this exam.

Ok, we have now reached the `play()` function. This is where the meat of the action will occur! Right now, `play()` is pretty limited. As discussed in class, "motion" in an animation is secretly just moving an object small amounts, over and over, in a fast timeframe. We start the function with `start = datetime.datetime.now()` — this is simply starting a "clock" within the function that will let us know how long it takes to execute `play()`. We will stop the clock at the end of the function, and use that to set a timer for how frequently we should call the `play()` function (we want to start it again as soon as it has finished its computation). You don't need to worry about the timer operations - we have set all of this for you. Just leave the code as is and everything should work.

Now, you can see that we call the `.move()` methods to move our objects, and then set off our timer. When the timer goes off, the `play()` function gets called again[1]. The play function is called effectively as soon as we finish computation.[2]

Moving down through the code, we see that to start the timer going, we have an initial call to the timer that is outside of `play()`. This will only get called once, when we start up the game.

[1] Careful! Make sure that inside of the `screen.ontimer` function you use `play` and not `play()`

[2] Note that we have defined a function inside of our `main()` function. This is sometimes useful when we do not intend to use a function outside of one specific context - it allows us to use variables that are local to `main()` without passing them in. We don't always recommend this, but in this context it is OK since we handle all of the big jobs with classes, private attributes, and methods.

Below this, we define the functions that move our FlyingTurtle object forward, left, and stop. We also tie these functions to their relevant keyboard inputs.

Now that we've read through all of the existing code and fixed the class privacy issues, it's time to add some functionality!

TASK 3: Right turning (5 points)

Task 3 is to modify `turtlegame.py` to enable Tiny to turn right. A successful implementation of Task 3 will allow the user to turn Tiny to the right by pressing the right arrow key. A successful implementation of this task should turn Tiny by the same relative amount as turning Left.

At this point, we can fly our turtle around, but not much else. We'd like to equip Tiny with some abilities to fight back against the Ghosts!

TASK 4: Add Laser Beams (35 points)

A successful implementation of Task 4 will allow Tiny to shoot laser beams when the user presses the space bar. They won't hit anything just yet, but we will be able to see the laser beams shoot across the screen in the same trajectory as Tiny.

To complete Task 4, you will first need to implement a class called `LaserBeam`. This class should be defined just before the `Ghost` class. The class definition and the constructor of this class should be:

```
class LaserBeam(RawTurtle):
    def __init__(self,canvas,x,y,direction,dx,dy):
        super().__init__(canvas)
        self.penup()
        self.goto(x,y)
        self.setheading(direction)
        self.color("Green")
        self.lifespan = 200
        self.__dx = ?????? #this is Task 4A
        self.__dy = ?????? #this is Task 4A
        self.shape("laser") #this shape has already been registered.
```

This class creates a single laser beam object. We can call the constructor using Tiny's `direction` to figure out where the laser beam needs to shoot (HINT: investigate the `heading()` method in Turtle Graphics that is used in `turboBoost()` and the associated `setheading()` method). We have also created a laser "lifespan" equal to 200 (this will limit how long we allow your laser to exist in the game), and set a color and created a shape for our laser beam.

Task 4A: Create appropriate mathematical equations for self.dx and self.dy (5 points)

Remember, `self.__dx` and `self.__dy` will tell us how the laser should move. Imagine that the laser has a starting position of x,y — the same as Tiny's position. When Tiny shoots a laser, it should travel faster than Tiny does, but in the same direction.

HINT: Look at how we move Tiny in `turboBoost()`. You can assume that the laser beam should move twice as fast as Tiny. We will call the constructor using Tiny's current values of dx and dy.

Careful: `self.__dx = 2*dx` will *not* return the correct values! `self.__dx = (something)*2 + dx` is the form this line of code should take.

HINT, again: looking at `turboBoost()` will help you figure out what the `something` above should be! Draw a picture.

Task 4B: Add accessors to this class (5 points)

For task 4B, create accessors to get the lifespan, DX, and DY values. You should also have an "accessor" for the radius that returns a constant value, 4.

Task 4C: Create a method to move the laser (5 points)

For task 4C, create a method inside the LaserBeam class that moves the laser. This function should be very, very similar to the `move()` functions for `Ghost` and `FlyingTurtle.` However, the laser move function should also decrease the laser's lifespan by 1.

Task 4D: Shoot lasers! (20 points)

To shoot lasers, we will need to modify some things within the `main()` and `play()` functions. Here are a few tips for implementing the functionality of shooting lasers:

- Like Ghosts, you should keep track of the lasers that are "in play" using a list
- Inside the play function, you should keep track of "Dead Lasers" - lasers with a lifespan that is 0
- All lasers that are not dead should move when the play function is called (Hint: put this after the movement of Tiny)
- After moving, if a laser is dead, it should be added to the list of dead lasers, removed from the list of active lasers, told to `.goto(-screenMinX*2, -screenMinY*2)`, and it should be hidden (remember, it is a turtle object so you can use `ht()`).
- You should create a function called `fireLaser()` outside the play function that does the following:
    - creates a `LaserBeam` object with the x,y coordinates, heading, and dx, dy values of Tiny
    - Adds that `LaserBeam` object to the list of active lasers
- You should tie the function `fireLaser()` to the action of pressing the space bar with `screen.onkeypress(firelaser,"")`

If you have correctly implemented tasks 1-4, your program should now be successfully shooting lasers when the user presses the space bar, in the same trajectory as Tiny, but at a faster rate. This is tough work! But if you've made it this far, you've already acquired more than half the points on the exam.

TASK 5: Hit ghosts with lasers (20 points)

Now that Tiny is able to shoot lasers, we would like our lasers to successfully "hit" the ghost objects.

To determine if a laser has "hit" a ghost, we need to establish if the objects have intersected. We will say that two objects have intersected if the distance between the two objects is less than or equal to the sum of their radii (recall that each object has a defined radius).

Task 5A: Create a function called `intersect(obj1, obj2)` to determine if two objects have intersected (5 pts)

- Hint: Recall the distance formula for determining the distance between two points in space (`distance = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)`)
- Hint: This function should return a boolean
- Hint: this is a function definition that exists outside the `main()` but is not part of any class

Task 5B: Modify the `play()` function to determine if a laser and a ghost have collided and remove the ghost from play (15 points)

- Hint: this should go just before the timer is updated at the end of `play()`

- Hint: create a list of hit ghosts to keep track of ghosts that have been hit
- If a laser hits a ghost, that laser is now dead
- If a laser hits a ghost, that ghost is also "dead" and should be removed from the game (HINT: use `ht()`)

TASK 6: Update Score when a Ghost is Hit (5 points)

Once you've gotten your game to successfully "hit" a ghost with a laser, you can start keeping track of your score.

Look for the line `frame.pack(side = tkinter.RIGHT,fill=tkinter.BOTH)` in your code (it should already be there).

Just underneath, add the following code:

```
scoreVal = tkinter.StringVar()
scoreVal.set("0")
scoreTitle = tkinter.Label(frame,text="Score")
scoreTitle.pack()
scoreFrame = tkinter.Frame(frame,height=2, bd=1, relief=tkinter.SUNKEN)
scoreFrame.pack()
score = tkinter.Label(scoreFrame,height=2,width=20,textvariable=scoreVal,
      \fg="Yellow",bg="black")

score.pack()
```

These lines embed a scoreboard into your graphics window using the `tkinter` module.

The score can be accessed using the accessor function `scoreVal.get()` and updated through the mutator `scoreVal.set(input)`. Note that scoreVal is a string, so if you wish to change the score your input must be converted into a string.

To successfully complete Task 6, add lines to your code to increase your score by 20 points each time a ghost is hit. HINT: you can create a temporary variable to store the integer conversion of `scoreVal.get()`, modify the temporary variable to add to the score each time you hit a ghost, and then update the scoreboard using `scoreVal.set(str(temp))`.

TASK 7: Check to see if all of the ghosts have been hit and end the game if so (5 points)

To complete Task 7, you should check to see if Turtle has successfully hit all of the ghosts. If yes, then the `play()` function should output a message to the user saying they have won, and `return`. Since the return means you will exit the `play()` function before calling the timer again, the game will not continue.

- HINT: this check should occur toward the beginning of the `play()` function, before you move the objects.
- HINT: you can use `tkinter.messagebox.showinfo("You Win!!", "You saved the world!")` to output a message to the user

TASK 8: Keep track of lives, end game if out of lives (10 points)

To implement task 8, we can add another element to our display window to show how many lives Tiny has remaining. Tiny starts the game with three extra lives, but loses a life if she collides with a ghost.

Look for the line `score.pack()` in your code (it should already be there).

Just underneath, add the following code:

```
livesTitle = tkinter.Label(frame, text="Extra Lives Remaining")
livesTitle.pack()
livesFrame = tkinter.Frame(frame,height=30,width=60,relief=tkinter.SUNKEN)
livesFrame.pack()
livesCanvas = ScrolledCanvas(livesFrame,150,40,150,40)
livesCanvas.pack()
livesTurtle = RawTurtle(livesCanvas)
livesTurtle.ht()
livesScreen = livesTurtle.getscreen()
life1 = FlyingTurtle(livesCanvas,0,0,-35,0)
life2 = FlyingTurtle(livesCanvas,0,0,0,0)
life3 = FlyingTurtle(livesCanvas,0,0,35,0)
lives = [life1, life2, life3]
```

When you compile your code, you should now see three turtles displayed to show how many extra lives Tiny has remaining.

Pay attention to the last line of that code — we have created a list called `lives` that keeps track of how many lives Tiny has remaining. Once all of those lives have been removed from the list, the game will be over.

To successfully implement task 8, you need to determine if Tiny has lost all of her lives. This is somewhat complex, but as this is the last task of the game and worth only 10 points, if you've made it here you're doing great!

HINTS:

- All of this should occur within the `play()` function
- You can make a list to keep track of ghosts that have been hit by Tiny — remember that when ghosts are hit, they are hidden from our display, but that doesn't mean they can't be hit again. So, it is useful to keep track of which ghosts have already been collided with. Similarly, you can use a Boolean variable to denote if a ghost has already been hit.
- investigate the `.pop()` method for lists — you can use this to take a turtle out of the lives list, and then hide that turtle from the `livesCanvas`
- You can use the `tkinter.messagebox.showwarning( "Uh-Oh","You Lost a Life!")` to display a warning message to the user
- Once all lives have been lost, you should return from the `play()` function before calling the timer again. Just like in Task 7, this ends the game.
- Remember that if you haven't already hit a ghost, once a collision occurs that ghost should be hidden from the canvas.

TOTAL: 100 points

Extra tasks for extra credit will be posted by Monday, November 19.

Program Requirements

Because this program is extremely specific and builds on itself, it must be implemented in order. That is to say, you need to get one task fully working before you can move on to earn points for the next task. You can attain points if and only if a task has been completed, in order that they are given.

The sole exception to this is Task 6: you may continue on to Task 7 and 8 if your program does not successfully update the score (just leave the score as 0).

To help your TAs, please include a comment at the top of your file indicating the **highest** task you believe you have achieved. For example, if you completed tasks 1, 2, 3, 4, and 5a, please include a comment saying `#Completed through Task 5a` at the top of your file.

As always, you must avoid using global variables (you may use global constants, although you will likely find you do not need to use them).

To earn full points for a task, you must carry it out exactly as specified. For example, if your Tiny Turtle shoots lasers but does not do so in the appropriate direction or at the appropriate speed, you will not earn full credit for that task. Your programs will be graded primarily on their functionality; that is to say, whether you have accomplished the task as specified.

Your code must modify the existing `turtlegame.py` provided to you on Canvas, and implement tasks as specified in this document to earn credit.

Constraints:

- You may use any *built-in* Python object class methods (string, list, etc.)
- You may use *imported* functions and class methods from the modules already included in `turtlegame.py` only
- You may freely read and reference the Python 3 documentation, but should not consult any other sources outside of the course readings, graduate TAs, and instructor. While many of the ideas in this exam may be new to you, everything you need to succeed is provided to you in this document or in the Python 3 documentation.
- You should not in general alter the code provided to you in `turtlegame.py,` with the exceptions of tasks 1 and 2.

Academic Integrity Pledge

The first lines of your program *must* contain the following text (exactly as it appears), replacing the first line with your full name and X500 ID. If this statement does not appear in your program source, you will receive a score of zero for the exam.

```
# <replace this line with your name and x500 ID (e.g., John Smith smit1234>

# I understand this is a graded, individual examination that may not be
# discussed with anyone.  I also understand that obtaining solutions or
# partial solutions from outside sources, or discussing
# any aspect of the examination with anyone will result in failing the course.
# I further certify that this program represents my own work and that none of
# it was obtained from any source other than material presented as part of the
# course.
```

Suggestions and Hints:

- Start early - this project is larger in scope than your previous projects, but actually has less coding. It is most manageable in small chunks. Plan to accomplish one task at a time!
- Save versions of your program. This is critical for success in a large scale project.
- You may consult the instructor and graduate TAs during office hours to ask clarifying questions regarding the code provided to you.
- Ensure you thoroughly read the initial file `turtlegame.py` and understand how it works before attempting to implement any part of this exam.
- Have fun! :)