CS-UY 1114 — Final Project

Professors Katz, Mante, Epstein

1 Introduction

You have the option of submitting a final project, accounting for 10% of your final grade. If you choose not to submit a final project, you will be obligated to complete an additional section of questions on the final exam, in lieu of the project. You must choose one of these options (final project or additional section of final exam) by November 26th.

2 Projects

You must choose one of the following projects to implement. All projects must be completed in Python 3, using only built-in libraries.

You can see video demos of each program at the following address: `https://drive.google.com/drive/folders/1xZkg75hl_GwQdu_OmjwY7jXUFk4JHdGE?usp=sharing`

2.1 Tic Tac Toe

You will implement a version of the classic "tic-tac-toe" game (also known as "noughts and crosses"), in which the player competes against the computer to create a three-in-a-row pattern in vertical, horizontal, or diagonal directions.

The game is played in a 3-by-3 grid, displayed on the turtle canvas, which is initially empty. There are two players, the human and the computer, who take turns. The human always moves first. When it's the human's turn, he or she may place an "O" symbol in any empty position on the game grid by clicking on the grid position with the mouse; afterwards, it becomes the computer's move, and the computer will place a "X" symbol in any empty position on the grid. The players continue to alternate moves until the game ends. The game is over when any of the following conditions occurs:

- there are three "O" symbols in a line in any row or column, or diagonally from corner to corner, which means that the human has won
- there are three "X" symbols in a line in any row or column, or diagonally from corner to corner, which means that the computer has won
- there are no empty positions on the board, meaning that no further moves are possible, and the game ends in a stalemate

As part of this project, you must develop a reasonable (but simple) "artificial intelligence" to dictate the behavior of the computer player. Although you may provide a more advanced algorithm (which your instructor will help you develop, if you ask), at a minimum, you must implement the following logic for the computer player:

- If it's possible for the computer player to win in the current move (i.e. there are two "X"s and one empty position in a line somewhere on the board), the computer player must win by occupying that empty position.

- If it's possible for the human player to win on the next move (i.e. there are two "O"s and one empty position in a line somewhere on the board), the computer player must prevent the human's win by occupying that empty position.
- If neither of the above cases apply, the computer may randomly occupy any unoccupied position.

Getting started We provide you with starter code to help you begin this project. See the file in `starter.zip`. This is a starting point; you should replace the `pass` lines with code to perform the appropriate actions.

2.2 Pong

You will implement a version of the classic "pong" game, in which the player moves a virtual paddle to deflect a bouncing ball, while an opponent with artificial intelligence does the same. Both paddles and the ball are displayed on the turtle canvas.

The ball will move at a constant speed. Initially, the ball will be in the center of the window, moving in a random direction. The player may control the player's paddle by using the left and right arrow keys.

If the ball hits a paddle, it will deflect off at an angle depending only on what part of the paddle it hits. If the ball hits the exact center of the paddle, it will bounce off straight, perpendicular to the paddle (at a 0-degree angle). If the ball hits the extreme left end of the paddle, it will bounce off left, at a 70 degree angle relative to perpendicular. If the ball hits the extreme right end of the paddle, it will bounce off right, at a 70 degree angle relative to perpendicular. The angle of bounce in other positions is proportional between the extremes of 0 and 70 degrees.

If the ball hits a wall, its vertical velocity will not change, but its horizontal velocity will be mirrored, i.e. it will move at the same speed in the opposite direction.

If a paddle fails to hit the ball, that player's opponent will gain a point, and the game will continue with the ball moved to the center of the window. The total number of points for each player is displayed on the left side of the screen.

The opponent's paddle must make a reasonable attempt to deflect the ball. You are responsible for designing its artificial intelligence.

Getting started We provide you with starter code to help you begin this project. See the file in `starter.zip`. This is a starting point; you should replace the `pass` lines with code to perform the appropriate actions.

2.3 Space invaders

You will implement a version of the classic "space invaders" game, in which the player battles against an army of invading monsters. The game is displayed on the turtle canvas.

In this game, the user uses the arrow keys to control a space ship that can move along the horizontal axis. Pressing the space key will cause the ship to shoot a single bullet, which will move at a constant speed, starting from the current position of the ship. If the bullet hits an enemy, both the enemy and the bullet will be removed from the game. If the bullet leaves the visible area of the screen, it will be removed from the game. There may be multiple bullets in flight at the same time. If no enemies remain, the player has won, and the game ends.

The enemies are initially arranged in rows. As you can see in the video, the enemies move in a distinctive, jerky pattern that your project must reproduce: they move all together at a regular intervals; initially, this time interval is one second, but as the game progresses, the time interval will get smaller and smaller, causing them to move increasingly fast. The enemies move horizontally (left or right), until the reach the end of the window, whereupon they will move one row closer to the player, and then continue moving horizontally in the opposite direction (if they were moving left in the previous row, they will move right in the subsequent

row), until they again reach the end of the window. Please consult the provided video for an example of enemy movement. If an enemy touches the player's ship, the player has lost, and the game ends.

You may choose to implement any of the following optional components: enemies that can shoot back; attractive and colorful graphics; scoring; a high-score table. Please discuss these options with your instructor.

Getting started We provide you with starter code to help you begin this project. See the file in `starter.zip`. This is a starting point; you should replace the `pass` lines with code to perform the appropriate actions.

2.4 *n*-bodies

You will implement a visualization of the classic "n-bodies" problem, a simulation of the effects of gravitation acting on various objects on a plane. Your program will graphically display the position all bodies in the system on the turtle canvas.

You will be given a file named `bodies.txt` containing the initial state of the simulation. The state of the simulation consists of: the gravitational constant G, as well as the name, position, velocity, and mass of an arbitrary number of objects. Your program will read this file in and simulate subsequent interactions between bodies, while displaying graphically a visualization of the simulation. Here is an example file:

```
.00005
Sun 500 500 0 0 100000000
Splat 255 255 2 0 1
Arg 745 745 -2 0 2
Magnavolt 270 740 -1.5 -3 12000
Spatula  253 450 0 -2 50000
Smoo 500 0 3 0 1000000
Volt 504 0 4 4 0.000001
```

The first line contains a value for G, the gravitational constant. In the real universe, G is always $6.67384 \times 10^{-11} m^3 kg^{-1} s^{-2}$, but in our simulation, it can be anything. Its magnitude is given as the first line in the input file.

The other lines contain information about each body. Each line gives:

- The body's name (a `str`).
- The body's initial horizontal position (a `float`).
- The body's initial vertical position (a `float`).
- The body's initial horizontal velocity (a `float`).
- The body's initial vertical velocity (a `float`).
- The body's mass (a `float`).

So, in the above file, the sun is the most massive object in the universe and it is not moving initially.

The heart of the simulation is the calculation of the forces acting on each body. Given a body located at position x_1, y_1 and a body located at x_2, y_2, the distance between them is

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The *horizontal distance* between the bodies is $|x_1 - x_2|$, and the *vertical distance* between them is $|y_1 - y_2|$.

The *gravitational force* F between two bodies is

$$F = G\frac{M_1 \times M_2}{d^2}$$

where

- G is the gravitational constant
- M_1 is the mass of the first body
- M_2 is the mass of the second body
- d is the distance between the two bodies

We will focus on a two-dimension simulation. Therefore we are interested in the horizontal force (F_x) and vertical force (F_y) on each body. These are given as:

$$F_x = G\frac{M_1 \times M_2 \times d_x}{d^3}$$

$$F_y = G\frac{M_1 \times M_2 \times d_y}{d^3}$$

where d_x and d_y are the horizontal and vertical distance between the bodies, respectively.

Recall that there is gravitational force between all objects in the universe. In particular, every object exerts gravitational force on every other object. The *net force* on a body is the sum of all forces on it. In the case of our simulation, this means the sum of the gravitational forces between a body and all other bodies in the simulation.

The *acceleration* of a body is given by $a = F/m$ where

- F is the net force acting on that body
- m is the mass of that body.

Similarly, the horizontal and vertical accelerations are given by $a_x = F_x/m$ and $a_y = F_y/m$.

At every frame of the animation, the velocity of each body must be updated, based on its current acceleration. In this program, the implicit units of the acceleration are in pixels per frame per frame. Thus, if the horizontal acceleration of a body is 3, then its horizontal velocity will be increased by 3 every frame. In other words: velocity is the integral of acceleration.

At every frame of the animation, the position of each body must be updated, based on its current velocity. In this program, the implicit units of the velocity are in pixels per frame. Thus, if the horizontal velocity of a body is 3, then its horizontal position will be increased by 3 pixels every frame. Likewise, if its vertical velocity is -1, its vertical position will be decreased by 1 pixel every frame. In other words: position is the integral of velocity.

Getting started We provide you with starter code to help you begin this project. See the file in `starter.zip`. This is a starting point; you should replace the `pass` lines with code to perform the appropriate actions.

If you've got a working prototype of your project, ask your instructor for additional versions of `bodies.txt` that you can use to test your program.

2.5 Automatic iris classification

In this project, you will write a program that can automatically classify an iris plant into one of three species, using machine learning techniques.

The three species of iris plants observed are *iris setosa*, *iris veriscolor*, and *iris virginica*. Flowers of *iris setosa* tend to be larger than flowers of *iris veriscolor* or *iris virginica*. If the length and width of the petals of a number of iris flowers are measured and the species of these flowers is known, we can use this information about known iris flowers of each species to attempt to classify an unknown flower based on the size and width of its petals. This technique is called *supervised machine learning*. Supervised learning requires a number of examples of iris flowers from the three species with known measurements of the length

and width of their petals. (In contrast, an unsupervised machine learning program might group like flowers together, but it wouldn't have any knowledge of which species the flowers actually belonged to.)

For this project, you will be creating a program that classifies an iris flower of unknown species. The petal size of the flower must be known because the supervised learning program will match this information to known flowers. In writing this program, some portion of the data for which the species, the petal length, and the petal width are known will act as the *training set*. When the program is presented with a new flower to classify, the program will find the most similar flower from the training set and classify the instance in the same way as the known similar instance. This method of making predictions is known as the *nearest neighbor algorithm*.

You will want to test your program to see if it makes accurate predictions for the species of an iris flower. Your program shouldn't make an assumption that the species of these particular iris flowers can be determined based only on their petal length and petal width. It may be that there is no relationship between the species of the flower and the petal length and petal width data that you have. You will test to see if your program is successful predicting the species based on the data that you have. The portion of the data that wasn't used in the training set will be used to test the feasibility of making this prediction based on your information.

To test your program's classification ability, we will use another set of data, the `test set`. This set contains data about flowers of unknown species that we want to classify. Actually, we *do* known their species, but your program should classify them based solely on their observed petal size; we then compare your program's conclusion to the plant's known species to determine the effectiveness of your program's classification strategy. If your program provides reasonable accuracy on this test set, you can be confident that it can predict the species of iris given the petal size. It is important that when you test your program using the test set, the program is given new examples of flowers that it has not seen before; in this way, we simulate using your program on data for which the species is really unkown.

We provide two data files: `iris_train.csv` (the training set) and `iris_test.csv` (the test set). These data files contain measurements for individual iris flowers and the corresponding type of each iris plant. [1]

Both files are in CSV format; open them in a text editor to see how their content is represented.

Each line in the file contains a single *observation* about a particular plant. In particular,

the first two comma-spearated columns in the dataset contain the petal length of the flower (in cm); and the petal width of flower (in cm). Each of these observable attributes (petal length and petal width) is called a *feature*. The last column in the dataset gives the type of iris, either *Iris-setosa*, *Iris-versicolor* or *Iris-virginica*. This column is called the *label*.

You will use the file `iris_train.csv` to build a predictive model. You will test your model on the file `iris_test.csv` and determine the accuracy of your model. You can only use modules from the standard distribution of Python in this project, and specifically the `numpy` and `pandas` modules are prohibited.

Starter code has been provided for you. We provide function headers and signatures for you; it's your task to complete the function bodies.

In the steps below, we walk you through the process of completing each function body in order. Make sure that you understand what each function should do before you write code for it. Before moving on to the next step, test your code thoroughly.

Do not modify the `main` function.

1. First, read the file `iris_train.csv` into a tuple containing two lists of type `float` and one list of type `string`. The features of the dataset should be of type `float` and the label should be of type `string`. Fill in the code for the function `create_table` which has been given.

2. Notice that the features of the dataset are both given in centimeters. Measure the range of each feature. This will require you to find the `min` and the `max` of each column in your dataset. Print your resulting

[1] This project uses the famous Iris Data Set, which was referenced in R.A. Fisher's 1936 paper detailing statistical methods to classify an iris plant You can read more about the dataset in the UCI Machine Learning Repository here: `https://archive.ics.uci.edu/ml/datasets/iris`. The dataset has been edited for use in this project.

min, max and range as shown. You will print your calculated values in place of the `xxx`. Fill in the code for the function `print_range_max_min` which has been given.

```
Feature 1 - min: xxx max: xxx range: xxx
Feature 2 - min: xxx max: xxx range: xxx
```

3. You will use the numerical values of each feature to make your predictions. If you use the values as they are, the features with higher values or a greater range might be given more importance in the model. To avoid this effect, you will *normalize* each of your features. This means that you will rescale all the values in a particular feature in terms of a mean of 0 and a standard deviation of 1. In order to normalize your features, find the mean and the standard deviation. Fill in the code for the given functions `find_mean` and `find_std_dev`. The formula for standard deviation is:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}}$$

where σ is the standard deviation, μ is the mean and N is the number of values in the feature. The syntax x_i indicates the i-th value of a particular feature. You should take each value in the feature, subtract it from the mean, μ, square the difference and accumulate the sum of these differences. Then divide by N, the number of values in the feature. Finally, take the square root to arrive at the standard deviation.

Print the mean and standard deviation for each feature as shown. Your calculated values should be printed in place of the `xxx`.

```
Feature 1 - mean: xxx std dev: xxx
Feature 2 - mean: xxx std dev: xxx
```

Determine the new normalized value, x_i', for each value, x_i, in a feature by subtracting each value from the mean, μ, and dividing by the standard deviation, σ, as shown below. Implement this functionality in the given function `normalize_data`.

$$x_i' = \frac{x_i - \mu}{\sigma}$$

Do this for all of the values in each feature. Check that you have correctly normalized the features by testing again for their mean and standard deviation. The output of your function `normalize_data` should now look at follows:

```
Feature 1 - mean: xxx std dev: xxx
Feature 1 after normalization - mean: xxx std dev: xxx
Feature 2 - mean: xxx std dev: xxx
Feature 2 after normalization - mean: xxx std dev: xxx
```

After normalization, each of your features should display a mean of 0 or very close to 0 and a standard deviation of 1 or very close to 1.

4. You will use an algorithm called k-Nearest Neighbors to predict the type for each iris observation in the test set. The test set is the file called `iris_test.csv`. In its simplest implementation, when k = 1, the k-Nearest Neighbors algorithm finds the observation in the training set that is 'closest' to the observation you are trying to make a prediction for and uses the known label of the closest observation

as the prediction value. In order to find the observation that is 'closest,' you can think about each of the training set observations as a point on a two-dimensional plane. For example, an observation will be plotted as *(x, y)*, where x is equal to the value of feature 1 (petal length) and y is equal to the value of feature 2 (petal width). In order to find the 'nearest neighbor,' you will calculate the Euclidean distance between the point in the test set that you are trying to label and all of the points in the training set. For this project, you will be implementing k-Nearest Neighbors with two features and with $k = 1$, but the algorithm can be implemented in other ways. [2]

Notice that the `main` function makes a second call to the `create_table` function, passing it the file `iris_test.csv`. The program reads in the `iris_test.csv` file and stores the data in a two-dimensional list called `test_data`. If you examine the `test_data` table, you'll notice that this file also contains labels for each observation. You will give a predicted label using the k-Nearest Neighbors algorithm as described and then compare your prediction to the actual label. You will be measuring your error for all of the observations in the test set.

The data in the test set also needs to be normalized. Notice that the `main` function makes a second call to `normalize_data`, passing it `test_data`.

Now that the test data is prepared, make a prediction for each observation in the test dataset. Implement this task in the function `make_predictions` which has been given. For each observation in the test set, you'll need to check all of the observations in the training set to see which is the 'nearest neighbor.' Fill in the code for the `find_dist` function which has been given. The `make_predictions` function should make a call to `find_dist`. Test your `find_dist` function independently to make sure it is working properly. Use the formula for Euclidean distance between two points (x_1, y_1) and (x_2, y_2):

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Your `make_predictions` function should accumulate a list of predicted iris types for each of the test set observations. The function will return this prediction list.

5. Determine the error of your predictions. Implement this task in the given function `find_error`. You will need to check the prediction list against the actual labels for the test set to determine how many errors were made. Return a percentage of how many observations in the test set were predicted incorrectly. Print the error percentage as shown:

   `The error percentage is:` *xxx*

   In practice (though not as part of this project), after you were confident that you could predict for your test set with good accuracy, you would use your model to make predictions for observations for which you did not know the label.

6. You will be using the `turtle` module to visualize the results of your analysis. See the sample plot for reference. In order to create your plot, fill in the code for the `plot_data` function. First, set the turtle window size to 500 x 500. Then draw the x and y axes in the window. Label your axes as shown in the sample plot.

   Plot each observation from your training set on the plane, using a circle shape and a different color for each type of iris. Use the value of the first feature for the x-coordinate and the value of the second feature for the y-coordinate. You can use a dot size of 10. Recall that the features have been normalized to have a mean of 0 and a standard deviation of 1. You will need to 'stretch' your features across the

[2] While it is beyond the scope of this project, the k-Nearest Neighbors algorithm can be used with more than two features. In this case, you can think about the values being plotted in an n-dimensional space with n being the number of features. Also, rather than consulting just one 'nearest neighbor,' the algorithm can find any number of 'nearest neighbors.' You can specify, k, as the number of nearest neighbors to find. In this case, the prediction would be for the majority class from the surrounding nearest neighbors.

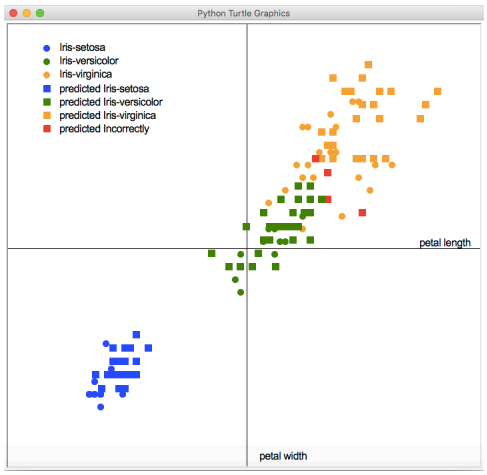

Figure 1: Sample Plot of Results

axes to make the best use of the 500 x 500 window. Ensure that none of your points are plotted off screen.

Also plot each correct prediction from your test set in the corresponding color. Use a square to indicate that the value is a prediction. Plot the incorrect predictions that were made for the test set in red, also using a square to indicate that it was a prediction.

Include a key in the upper left corner of the plot as shown in the sample plot. Fill in the code for the function `draw_key` to implement this task. Your function `plot_data` will make a call to the function `draw_key`.

2.6 Something else

Besides the above projects, you may pursue another project of your own choosing. You may choose a project relevant to your field of study: for example, if you study biology, you might want to look into a project involving DNA sequencing; of if you study art, you can develop a program for generating computer art. You may also propose a project relevant to your hobbies or interests. Your project must be of an appropriate level of difficulty; must demonstrate an understanding of the techniques taught in this course; and must solve an interesting problem or address a particular need. In all cases, your project must be approved by your instructor.

If you choose to develop a project of your own choosing, you should discuss your idea with your instructor in person as soon as possible. On the basis of our discussion, you must submit a written proposal to your instructor by email.

As a note to students, please bear in mind that implementing your own project typically requires *more* work than one of the default projects. In addition, it requires more communication with your instructor, to make sure that your development is continuing in the right direction and that your code meets the required standards. It is your responsibility to schedule meetings with your instructor in furtherance of these goals.

Your proposal represents a contract with your instructor, describing exactly what you intend to create. It must thoroughly cover all aspects of the behavior of your program. It should be at least as detailed as the descriptions given above for the other projects. Your description of your program should include:

- What is the purpose of the program? What problem is it solving?
- What does the program look like? You may provide drawings of the proposed user interface as illustrations.

- How does the user interact with the program? What input is expected from the user? What output is given to the user?
- How does your program behave? Describe in detail a typical use of your program, from beginning to end.
- What errors can arise while your program is running? For example, what will happen if the user provides unexpected or invalid input? Describe how your program will handle these cases.
- How will you structure the implementation of your program? As for the other possible projects, your program must reflect good design. This includes, but is not limited to, using multiple functions to break up larger problems and avoiding repetitive code.

Your proposal should be sufficiently detailed so that any competent programmer should be able to implement your proposed project solely on the basis of your written proposal.

Insofar as your finished project diverges from your project proposal, any differences must be cleared with your instructor well in advance of the due date.

3 Code quality

Your program's code must adhere to certain quality guidelines. Your grade may be penalized if your code does not meet standards of clarity, organization, and readability that are expected from all programmers, regardless of whether the code works.

Code quality includes, but is not limited to, the following:

- Programs should be structured modularly. That is, when approaching a larger problem, you should break it down into smaller problems. Write functions to solve particular needs, and then tie those functions together. If you find yourself writing your whole program in one giant function, you should reconsider.
- Your program should avoid unnecessarily repetitive code. If you find yourself writing similar code again and again, or copy-and-pasting code, it's a sign that you should use a loop, or break out the code into a generalized function.
- All of your functions must begin with a comment header containing a signature (what types it takes as parameters and what type it returns) and a description, in English, of the function's behavior. See the starter code for an example of what a correct comment header looks like.
- In addition to the comment header, your code should contain a sufficient number of comments (beginning with `#`) to make reading your code easy. Any confusing or complicated area of code should have a comment explaining what is happening, and even straightforward code can benefit from prose elucidation.

If you have any questions about these requirements, please contact your instructor.

4 Hints and rules

- Your instructor has provided a starting framework to help you get started. For the projects proposed above, find the corresponding starting framework in `starter.zip`. You are welcome to make use of these files to help you develop your project. If you are developing a different project, you may still benefit from reading or adapting the starter code.

  Please note that the code provided in the starting frameworks is designed to help you, but that does not absolve you of the necessity to understand the code contained therein. Before you start writing

new code, you should make sure that you understand the structure of the code provided for you, what it does, and exactly what remains to be done. If you have any doubts, contact your instructor. Many students wasted time writing wrong code because they failed to thoroughly read and understand the framework. Everything you need to know in order to complete the project has been covered in class; if you find yourself consulting outside resources, or looking for solutions on the internet, you're going in the wrong direction.

Also, please note that while these frameworks are intended to get you started and provide helpful suggestions, they should not constrain you. You can feel free to deviate from them or to develop your own implementation from scratch.

- This project represents a larger programming task than you've encountered before. To make it manageable, you should break it down into several, smaller tasks. The starter code that I've provided takes care of this for you: each function to be written represents a task that can be completed more-or-less independently.

  Furthermore, please try to test each smaller component, in order to verify the correctness of your solution, before moving on to the next task. There are are some strategies that will make testing easier:

  - Frequently pause to run what you've written so far. It's good to structure your code in such a way so that as you add functionality, you can test each new feature. Doing so will greatly increase the chances that the program as a whole will work as you hope.
  - Add temporary `print` calls into your code at critical points. For example, in n-bodies, to determine if the position of a body is being updated in the way that you think it should be, put a call to `print` inside the `update_positions` function, so that it prints out the current position of a given planet. That way, even if the rest of your code isn't fully working, you can run your code and determine if it's getting the right position results. In Tic-tac-toe, you could print out the state of the board in `do_user_move`, to help determine whether an unexpected result is caused by that function or another.

    Before you submit the final version of your code, you can remove these temporary aids.
  - If you're writing a program with a graphical display, such as any of the default projects, consider implementing the graphical code first (that would be the `draw_frame()` function for Pong, Space invaders, and n-bodies, and the or `draw_board()` function for Tic tac toe). That way, as you implement the remaining tasks, you'll be able to see their impact immediately, by how they affect the game's display. In this way you can avoid working in the dark.

- The Pong project, the Space Invaders project, and the n-bodies project all use animations. Before undertaking these projects, it's important that you understand the basic idea of how animations work in these projects.

  First, you should understand that animation is an illusion of motion. What's really happening is that the program is drawing a sequence of *frames*, or images, each of which represents the state of the game at a particular point in time. Between frames, the program calculates a new state, and then replaces the old frame with an updated one. Therefore, these programs consist of a *main loop*, which repeats until the game ends. The basic idea of the main loop is this:

  1. Calculate where stuff should be
  2. Clear the screen
  3. Draw the stuff on the screen
  4. Wait a small amount of time before the next frame
  5. Repeat

Looking at the code for the `main` function, you can see this strategy reflected in code: the screen is cleared with the `turtle.clear` function; your program updates the state of the variables in the `physics`, `ai`, `update_positions`, and/or `update_velocities` functions; the current positions of stuff is drawn out using various `turtle` functions inside your `draw_frame` function; and finally, we wait a very small amount of time before the next frame when we call `time.sleep`.

The starter framework isolates drawing code in the `draw_frame` function, while other functions (`ai`, `physics`, etc) are responsible for updating the variables that store the state of the game.

In order to ensure that your project's animation works as expected, if you are using the proposed starter framework, it's important that your program does drawing (i.e. calls functions in the `turtle` module that put images on the turtle canvas) *only* in the `draw_frame` function.

The most common cause of student confusion at early stages of the project happens when trying to do drawing at a phase of the program intended for calculations. Keep in mind that functions like `ai` and `physics` are only for doing calclulations. All drawing should happen in your `draw_frame` function or in your `draw_board` function.

- Due to a known bug in certain versions of Python on the Mac, the `turtle.write` function may be very slow. While this normally wouldn't be a problem, the slowness of the function makes creating smooth animation in Python on the Mac impossible. Therefore, if you do the Pong project, the n-bodies project, or another project that calls `turtle.write` in an animation loop, and if you encounter slowness in your animations, I encourage you to develop your project on a Windows machine rather than on a Mac.

- You may make use of the wide variety of drawing functions available within the `turtle` module. We've discussed some of these functions in lecture or lab (such as `turtle.forward`, `turtle.right`, `turtle.color`), and some others you may discover in the starter code, but there are many more. You are welcome to browse the official Python turtle documentation, where you may find other interesting ways to make your program visually interesting: `https://docs.python.org/3/library/turtle.html`

  Other functions that you will probably want to use:

  - `turtle.goto` – move the turtle to a specific x, y coordinate
  - `turtle.screen_width, turtle.screen_height` – return the width and height, respectively, of the turtle canvas window
  - `turtle.up, turtle.down` – stop an start, respectively, drawing when the turtle moves
  - `turtle.begin_fill, turtle.end_fill` – mark the beginning and end, respectively, of the drawing of a filled shape
  - `turtle.circle, turtle.dot` – draw, respectively, a circle and a dot at the turtle's current location
  - `turtle.write` – put text on the canvas at the turtle's current location
  - `turtle.width` – set the size of the turtle's width
  - `turtle.degrees, turtle.radians` – convert to degrees and radians, respectively, from radians and degrees

- Some online sources suggest that you use `turtle.Turtle` to create multiple simultaneous turtles. The multiple-turtles approach is mostly incompatible with the starter code that I present. Therefore, without prior written approval from your instructor, use of `turtle.Turtle` is prohibited.

- Use of any operating system-specific function calls (such as `os.system`) is prohibited. Do not make any assumptions about the operating system that your program will run on or what external programs are available.

- To accept keyboard input in a turtle window, use the `onkey` function to indicate which function should be called when a particular key is pressed. After you call `onkey` (possibly multiple times, for multiple keys) you should call `listen` only once.

  For example:

  ```
  def key_left():
      print("You pressed the left button")

  turtle.onkey(key_left, "Left")
  turtle.listen()
  while True:
      # main body of program here
  ```

  The starter code already calls `onkey` for you, so you need to only implement the body of the handler functions. If you are writing a project other than one of those proposed in this document, you should use a similar approach to that used in the starter code.

  If you want the user to enter a string, rather than a single character (for example, to enter their name for registration in a high scores table), you should not use the `input` function, since that function accepts input only in the Python console, not on the turtle canvas window. To accept text input in the turtle canvas window, use the function `turtle.textinput`.

  Code for `turtle.onkey` and `turtle.listen` have been added to the starter code for you.

- By default, the turtle moves fairly slowly, so you can see what it's doing. In the starter code, we call `turtle.tracer(0,0)` once at the beginning of your program, which has the effect of greatly speeding up the movement of the turtle. This is necessary in order to achieve smooth animations. However, if you find yourself confused by what the turtle is doing, it may be helpful to temporarily remove or comment out this line, so that you can follow what the turtle is actually doing.

  Note that because of the call to `turtle.tracer(0,0)`, we must use the `turtle.update()` function to ensure that turtle movements are completely reflected on the screen.

  Code for `turtle.tracer` and `tracer.update` have been added for you to the starter code.

- A common mistake when writing code for drawing is to assume that the turtle's heading is the same at the beginning of each frame. In reality, adjusting the turtle's heading (with `turtle.left` and `turtle.right`) will cause the turtle to maintain its new heading until you explicitly change it. To reset the turtle's heading to its default (pointing to the right), call `turtle.setheading(0)`.

- Recall that *local variables* are those variables defined inside a function, and *global variables* are those defined outside a function. Local variables are private to the function they are defined in, but global variables can be read from any function. However, assigning to a global variable from within a function requires an extra step. Be careful about a quirk of Python: by default, assignment refers only to local variables. For example, consider this code, which attempts to modify a global variable:

  ```
  x_position = 5

  def change_position():
      x_position = x_position + 1
  ```

  Although the programmer wants to change the global `x_position` variable from within the `change_position` function, it does not work as expected. What is actually happening is that the programmer is creating a *new* local variable also named `x_position` – the global variable remains unchanged. To ensure that assignment to global variables is possible, use the `global` statement within the given function. When you use the `global` statement, future assignments to variables of that name will update global, and not local, variables. The correct code should be:

```
x_position = 5

def change_position():
    global x_position
    x_position = x_position + 1
```

The starter code provides `global` statements where necessary so that you can modify global variables from within functions.

5 Academic honesty

You must write this assignment alone. Your submitted work must represent your own work; sharing code or discussing the project with other students is prohibited.

You may not consult code from any other source, including from other students. If you need help, contact your instructor or a TA.

6 Status report

In addition to the final project, you are required to submit a status report, reflecting your current progress in developing the project. This will give your instructor the opportunity to make sure that you are on the right track and making adequate progress.

Your status report should include all code you have written so far for the project. Your grade for the status report will be based on what I consider "adequate progress." This means that you will not be penalized for a non-working or incomplete program; however, you may be penalized if you have not yet started work, are progressing too slowly, or are writing code that does not reflect an understanding of the assignment or basic programming concepts.

The status report is worth 10% of the grade of the project.

7 Due dates

The final project is officially assigned to you on November 5th, 2018.

If you plan to work on a project other than those described in this document, you must submit your proposal to your instructor on or before November 12th, 2018.

You must submit a status report by November 26th. If you do not submit a status report, you forfeit the option of submitting a project for a grade, and will be obligated to take the additional section of the final exam. If you do submit a status report, you give up the option of taking the additional section of the final exam.

Your final project must be submitted before December 10th, 2018 at 11:55pm. Absolutely no work will be accepted after this time.

8 The `README` file

Your final project and status report must be accompanied by a so-called `README` file: a narrative, English-language commentary about your project. It should help your instructor understand your perspective and development process. It's your opportunity to provide feedback and discuss your project.

Your `README` file must be a *plain text* file (for example, created by IDLE) and not a Word document or other propretiary format.

It should include the following information:

- Your name and NetID.

- Your section.
- The state of your work. Please mention issues such as:
  - Did you manage to complete the assignment?
  - If you did not complete the assigment, what is missing? (If your assignment is incomplete or has known bugs, I prefer that students let me know, rather than let me discover these deficiencies on my own.)
  - How closely does your project meet the requirements specified in the project demos? Have you taken any artistic liberties?
  - Are there additional features, above and beyond the project requirements, that you've implemented?
  - Are there additional features, above and beyond the project requirements, that would have liked to implement?
  - Are you satisfied with the final result?
- Your retrospective on the development process. For example:
  - What parts of the assignment were particularly challenging?
  - What parts of the assignment were easier than you expected?
  - What parts of your solution are you particularly proud of?
  - Would you have approached the problem differently?
- Any additional thoughts or questions about the assignment, such as:
  - Were the instructions clear?
  - Were you given enough support in developing your solution?
  - Was the assignment interesting?
  - What insight has this project given you into programming, beyond the homework assignments during the semester?

9 Getting help

Your final project is challenging; it should expand your programming skill. As you contemplate the project, a moment of terror is to be expected. Fortunately, you have many avenues to get help:

- For questions regarding the requirements, design, and implementation of the project, you should first turn to your instructor. We can discuss project-related issues during class, by email, during office hours, or at other times by appointment.
- For specific questions about particular programming problems (such as, "What is causing this weird error?") you can also turn to a TA during a help session.

I strongly recommend having a discussion with your instructor *prior* to beginning to write code, in order to discuss overall strategy and to make sure that you understand the goals of the project.

10 Submission

When submitting your status report as well as the final version of your project, submit all code files, as well as the `README` file. In the absence of a `README` file, your project will not be graded.

Submit all files through Gradescope.