CS2013 Intermediate programming and problem solving II

Lab Assignment 1 – Part 1 of Project

Due: Friday February 16th, 5pm (week 5)
This assignment is worth 25% of your overall marks.

PLEASE READ THESE INSTRUCTIONS CAREFULLY.

Submission instructions.
Before submitting, ensure ALL of your python files are located in a folder. The folder should labelled in the following format:

StudentID_assignment_1

For example, if your student ID number is *123456*, then your folder should be named:

123456_assignment_1

The files that should be embedded in this folder are:

- main.py (containing main code, with comments)
- pacman.py (containing `Pacman` class, with comments)
- ghost.py (containing `Ghost` class, with comments)
- food.py (containing `Food` class, with comments)

Zip (compress) the folder before submitting.

<u>You should complete these actions by the due date specified above.</u> No late work will be accepted. The due date is final!

Plagiarism check
The Python code of the practical will be checked for plagiarism following the UCC guidelines:
https://www.ucc.ie/en/media/support/recordsandexaminations/documents/UCCPlagiarismPolicy-November2017V1.0-CLEAN.pdf
Plagiarism detection in a computer program is a simple process and experienced programmers can easily see if two programs are "modified" versions of the same one.

Project: Modified Pacman!

You will be creating a modified version of the Pacman game using python and circuitpython! The objective of the game is to create a Pacman whose mission it is to eat the food on the display screen. However, there is one ghost, let's call him pinky, who is determined to stop Pacman from eating the food! If Pacman eats the food before the ghost pinky catches Pacman, then the food should relocate to any position on the screen (though note the entire piece of food must be visible). However, if the ghost pinky catches Pacman before Pacman eats the food, both Pacman and the Ghost pinky should relocate. The overall game should be continuous until the player decides to close the window. Pacman's movements should be controlled via keyboard and gesture-based movements.
Note that the following characteristics must be implemented:

- Pacman should only be able to move either horizontally <u>OR</u> vertically. It should not be able to move diagonally.
- If Pacman meets one of the walls, then Pacman's location should move to the other side of the screen.
- The Ghost Pinky should appear to be following Pacman in some way.

This Assignment - Lab Assessment 1: Building the main pygame elements of the game

You will be creating classes (`Pacman` , `Food` and `Ghost`) and creating interactions between these classes in the main code.

Detailed guidelines

Note, you will be creating separate classes to control the gameplay, in the following files:

- In `pacman.py`, create the class `Pacman`, which is the blueprint for the `Pacman`
- In `food.py`, create the class Food, which is the blueprint for the `food`
- In `ghost.py`, create the class `Ghost`, which is the blueprint for the `Ghost`
- In `main.py`, create the main interactions of the game and create appropriate instances of the `Pacman`, Food and `Ghost` classes. This includes:
    - Creating a display window with a size of your choice.
    - Ensuring that Pacman, Food and Ghost objects are all drawn to the screen.
    - Pacman should be able to move up, down, left and right in response to keypressed. Pacman should only be able to move either horizontally or vertically at any given time. It should not be able to move diagonally.
      When pacman moves off one side of the screen, it should reappear on the other side.
    - Pacman's movements should be controlled by the keyboard.
    - Pinky should be able to "follow" pacman in some way.
    - The game should continue to play and the display window should remain open, until the user closes the window by clicking the "x" on the top corner of the screen.

Creating classes

Note that all classes below make use of pygame, so you should make sure it is imported in the file you are creating this class in.

1	**Creating the Pacman class**
A	The `Pacman` class contains the following variables, **<u>all of which should be private</u>**. You may decide whether the variables below are class or instance variables. However, you will need to defend your choice in your comments. (i) Variables for the coordinates of the Pacman (including x, y, radius or width and height) (ii) Variables for the Pacman colour and speed
B	**__init__(self, x, y, color, speed):**
	The constructor should be used to set up the `Pacman` class. This might include the coordinates, colour and speed of the Pacman (as defined above). You will need to defend your choice on parameters in comments.
C	**Create a string representation:** when the object is printed to the screen, it should contain reader-friendly information concerning the object's instance. This should include the colour and coordinates of the Pacman.
D	**draw(self, display):**
	This method draw the Pacman on the display window. To draw the Pacman, you will need to make use of pygame features (e.g. rectangles, circles, lines). **<u>Note you should not need to draw the Pacman in other methods.</u>**
E	**move_left(self):**
	This method should move the Pacman left. The speed the Pacman moves should be defined in an instance variable, containing a value of your choice. Note the following constraints: • The Pacman should reappear on the right hand side of the screen once it moves beyond the left hand side.

	Note that you should not draw the Pacman in this method.
F	**move_right(self):**
	This method should move the Pacman right. The speed the Pacman moves should be defined in an instance variable, containing a value of your choice. Note the following constraints: • The Pacman should reappear on the left hand side of the screen once it moves beyond the right hand side. Note that you should not draw the Pacman in this method.
G	**move_up(self):**
	This method should move the Pacman up. The speed the Pacman moves should be defined in an instance variable, containing a value of your choice. Note the following constraints: • The Pacman should reappear on the bottom of the screen once it moves beyond the top side. Note that you should not draw the Pacman in this method.
H	**move_down(self):**
	This method should move the Pacman down. The speed the Pacman moves should be defined in an instance variable, containing a value of your choice. Note the following constraints: • The Pacman should reappear on the top side of the screen once it moves beyond the bottom side. Note that you should not draw the Pacman in this method.
I	**Getters for the coordinates (x, y) and size of the Pacman**
	Create these getter methods.

2	**Creating a Ghost class**
A	The `Ghost` class contains the following variables, **<u>all of which should be private</u>**. You may decide whether the variables below are class or instance variables. However, you will need to defend your choice in your comments. (i) Variables for the coordinates of the Ghost (including x, y, radius or size in width/height) (ii) Variables for the Ghost speed on the x and y axis
B	**__init__(self, x, y, colour, radius, speed_x, speed_y):**
	The constructor should be used to set up the Ghost class. This might include the coordinates, colour, speed of the Ghost as shown above. You will need to defend your choice on parameters in comments.
C	**Create a string representation:** You should ensure that when the object is printed to the screen, it contains reader-friendly information concerning the object's instance, including the color and position of the Ghost.
D	**draw(self, display):**
	This method accepts the following parameters: display (pygame.display): the display window to draw objects on This method draw the Ghost on the display window. The display window is passed in with this method. To draw the Ghost, you will need to make use of pygame features (e.g. rectangle, circle or line). **<u>Note you should not need to draw the Ghost in other methods.</u>**
E	**move(self,direction,display_width=None, display_height=None)**

	This method accepts the following parameters: direction (String): a string that refers to which direction the Ghost should move – LEFT, RIGHT, UP or DOWN. display_width: the width of the screen display_height: the height of the screen This method should move the Ghost according to the direction passed in the variable direction. The speed the Ghost moves should be defined in instance variables, containing a value of your choice. Note the following constraints for the Ghost: • The Ghost's movement should be constrained within the bounds defined in the parameters `display_width_bound`, `display_height`. It should not be able to move fully off screen.
F	**relocate(self, display_width, display_height):**
	This method accepts the following parameters: display_width: the width of the screen display_height: the height of the screen This method should relocate the Ghost to a random location, that is within the bounds of the screen. In your solution, make sure the Ghost is fully visible (not only partially) when relocated.
G	**Getters for the x, y, and size the Ghost**
	Create getters to retrieve (i.e. return) the x, y, radius (or size) of the Ghost.

3	**Creating a Food class**
A	The Food class contains the following variables, **<u>all of which should be private</u>**. You may decide whether the variables below are class or instance variables. However, you will need to defend your choice in your comments. (i) Variables for the coordinates of the Food (including x, y, radius or size). (ii) Variables for the Food colour
B	**__init__(self, width, height, colour):**
	The constructor should be used to set up the Food class. This might include the size, colour of the Food as shown above. However, your implementation may be different depending on what shape you have decided to draw (e.g. a line, rectangle or circle)
C	**Create a string representation:** when the object is printed to the screen, it should contain reader-friendly information concerning the object's instance. This should include the colour and position of the Food.
D	**draw(self, display):**
	This method draw the Food on the display window. The display window is passed in with this method. To draw the Food, you will need to make use of pygame features. **<u>Note you should not need to draw the Food in other methods.</u>**
F	**relocate(self, display_width, display_height):**
	This method accepts the following parameters: display_width: the width of the screen display_height: the height of the screen

	• This method should relocate the Ghost to a random location, that is within the bounds of the screen. In your solution, make sure the Ghost is fully visible (not only partially) when relocated.
G	**Getters for the coordinates and size (or radius) of Food**
	Create getters to retrieve (i.e. return) the x, y, radius (or size) of the Food

		Not achieved	**Somewhat achieved**	**Nearly there**	**Achieved**
Definition of game classes	15	Pygame is not used or referenced. The methods are defined, but most of them are not completed or working properly. **Comments** are not included or comments do not describe what each method in the class does.	Pygame is imported and used appropriately to draw shapes or images for the Food, Pacman and Ghost. Most of the methods are working, but there are some issues with the methods causing them not to work as described. However the student highlights those issues as comments and provides a possible reason why the code is not working. **Comments:** The student explains what the methods are doing in their own words, demonstrating clear understanding. The student also explains in the comments how each class is designed both draw and manipulate the objects as separate GUI components using their own words.	Pygame is imported and used appropriately to draw shapes or images for the Food, Pacman and Ghost. The methods inside the Food, Pacman and Ghost class are defined and working correctly. There may be 1-2 issues, but these are small and do not interfere with the code working. Where there are issues, the student clearly provides a comment as to what they are and how the student will address these in the next iteration. **Comments:** The student explains what the methods are doing in their own words, demonstrating clear understanding. The student also explains in the comments how each class is designed to draw and manipulate the objects as separate GUI components using their own words.	Pygame is imported and used appropriately to draw shapes or images for the Food, Pacman and Ghost. The methods inside the Food, Pacman and Ghost class are defined and working correctly. There are no issues with the code and everything works as described. **Comments:** The student explains what the methods are doing in their own words, demonstrating clear understanding. The student also explains in the comments how each class is designed both draw and manipulate the objects as separate GUI components using their own words.
Gameplay interactions	10	No, or incorrect use of objects in the main code. The Food, Pacman and Ghost classes are not interacting in the main.py code. None of the pygame elements are applied, or are mostly not working. There is no pygame window displayed. There is no game loop. No comments in the code.	The Food, Pacman and Ghost classes are imported in main.py. Instances of the class are created. Some of the pygame elements are applied, fundamentally the window. The objects (Pacman/Food/Ghost) are drawn to the screen. Objects are moving but not properly. Event driven programming is applied to control object movements and collisions, where necessary. **Comments:** the student explains, in their own words, what the pygame interactions are doing. Additionally, the student explains event driven programming, and how it was employed to control each of the objects created in their own words. In cases where interactions are not working, the student explains how event driven programming could have been employed in their own words.	The Food, Pacman and Ghost classes are imported in main.py. Instances of the class are created. Most of the pygame elements are applied, fundamentally the window. The objects (Pacman/Food/Ghost)are drawn to the screen. Event driven programming is applied to control object movements and collisions, where necessary. **Comments:** the student explains, in their own words, what the pygame interactions are doing. Additionally, the student explains event driven programming, and how it was employed to control each of the objects created in their own words. In cases where interactions are not working, the student explains how event driven programming could have been employed in their own words.	The Food, Pacman and Ghost classes are imported in main.py. Instances of the class are created. All pygame elements described are working. The objects (Pacman/Food/Ghost)are drawn to the screen. Objects are moving and working as described. Event driven programming is applied to control object movements and collisions, where necessary. **Comments:** the student explains, in their own words, what the pygame interactions are doing. Additionally, the student explains event driven programming, and how it was employed to control each of the objects created in their own words. In cases where interactions are not working, the student explains how event driven programming could have been employed in their own words.