COMP115: Assignment 2

April 1, 2015

Due Dates

Prac Class Week 9	Checkpoint submission
Week 12	Final Submission

Summary

In this assignment you will create a flying game set in space. The assignment is broken into parts, separate tasks which can be done in isolation and which combine to make a full solution. Each part has a pass level (worth half the marks allocated for that part) and a distinction level (worth the other half of the marks) for that part. For each part we indicate which module *is most useful* for completing it. Part 1 is actually the hardest part of this assignment but it requires only pixels and variables and conditionals and thus you can start it immediately. Part 2 is probably the simplest. You are encouraged to attempt Part 2 first if you are struggling with Part 1.

The total marks available in this assignment is 100. Part 1 is worth 40 marks, Part 2 is worth 20, Part 3 is worth 20, and Part 4 is worth 20 marks. In each part, half the marks are allocated to pass-level functionality and the other half to distinction level functionality. For example, pass-level functionality for Part 1 is worth 20 marks overall.

Starting Point

Start from a 600 by 600 black window with a space craft which can only fly in circles.

```
float x=300;
float y=300;
float direction=0;
float increment=1;
float speed = 5;

void setup(){
  size(600,600);
}

void draw(){
  background(0);
  x = x + speed*cos(direction);
  y = y + speed*sin(direction);
  direction = direction + increment* 0.03;
  ellipse(x,y,20,20);
}
```

Note that this program uses **float** variables rather than **int** variables. This is because the circular motion of the space craft can't be calculated with integer arithmetic. We recommend you use **float** variables for all numbers in this assignment.

Part 1: Flying Saucer - 40 marks

Most useful module is conditionals

You are the pilot of a stricken flying saucer. Your space craft can only fly in cirlces but you can at least swap between clockwise and anti-clockwise.

Pass Level

In this part you must code the space craft flying motion. The formula for calculating the space craft's position in the next frame (a,b) from its position in this frame (x,y) is $a = x + S * cos(D), b = y + S * sin(D)$ where S is speed and D is direction. This has been provided for you in the starting code. What you have to do for this task is to allow the craft to change direction and to deal with the edges of the screen.

Anti-clockwise The space craft is flying in an anti clockwise circle. When flying anti clockwise the craft's direction is decreasing by 0.03 every time the draw loop is called.

Clockwise The space craft is flying in an anti clockwise circle. When flying anti clockwise the craft's direction is increasing by 0.03 every time the draw loop is called.

You can swap from left to right and vice-versa by hitting any key.

If the craft goes off one edge of the screen it should re-appear on the opposite edge and continue its current motion.

Distinction Part

Randomly choose a location for a wormhole and have it drawn at that location. If the craft collides with the wormhole, it should disappear.

The wormhole is a circle which grows and shrinks in size between diameter 0 and diameter 80. It will grow and shrink on a constant loop.

Part 2: Starry Background - 20 marks

Most useful modules are loops and arrays and strings

Your job is to make the background look more like space by adding stars. We then use those stars to create a "warp" effect.

Pass Part

Have 100 stars drawn in the background. They should be 60 pixels apart from each other. You must use loops to get full marks for this part. Figure 1 gives the expected output for this part.

Distinction Part

Have the stars drawn in 100 randomly chosen locations. They must be "stable", i.e. the stars don't move around the screen while the space craft moves.

When the craft goes into the wormhole, it must be "warped" to a new part of space. Warping to a new part of space means the space craft does not appear to move but a new wormhole and a new set of random stars are generated. Figure 2 gives the expected output for this part.

You must use loops and arrays to get full marks for this part.

Part 3: Keep Score - 20 marks

Most useful module is arrays and strings

In this part you will keep track of the player's score.

Pass Part

Every time the player is "warped" through the wormhole, they get one point. Display this score on the bottom left of the screen.

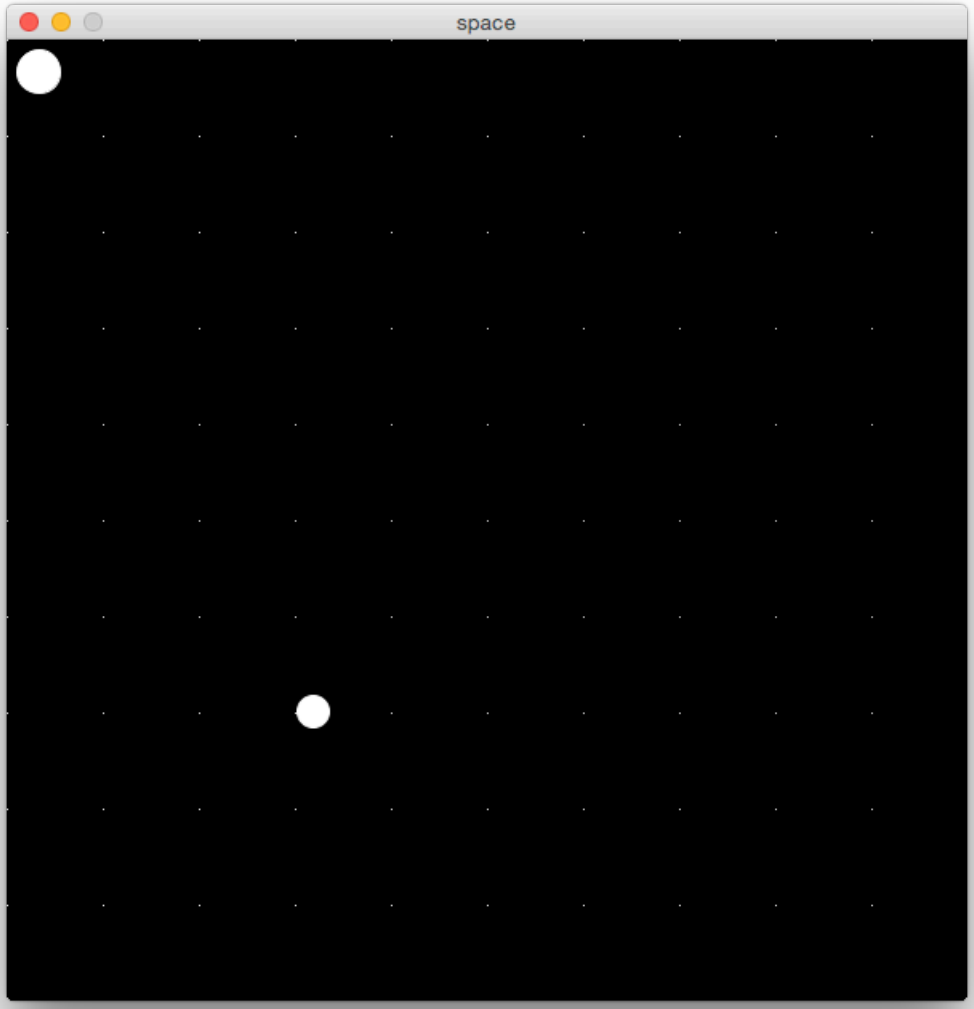

Figure 1: Expected pass-level starry background (with wormhole and craft)

Distinction Part

Create a parallax effect with the stars. We are looking "down"[1] on the ship with stars far behind it. If we move our location but keep looking at the ship, the stars will seem to move behind it. This effect is easy to achieve by moving the position of each star in the opposite direction to our supposed

[1] there is no direction in space, but you know what I mean

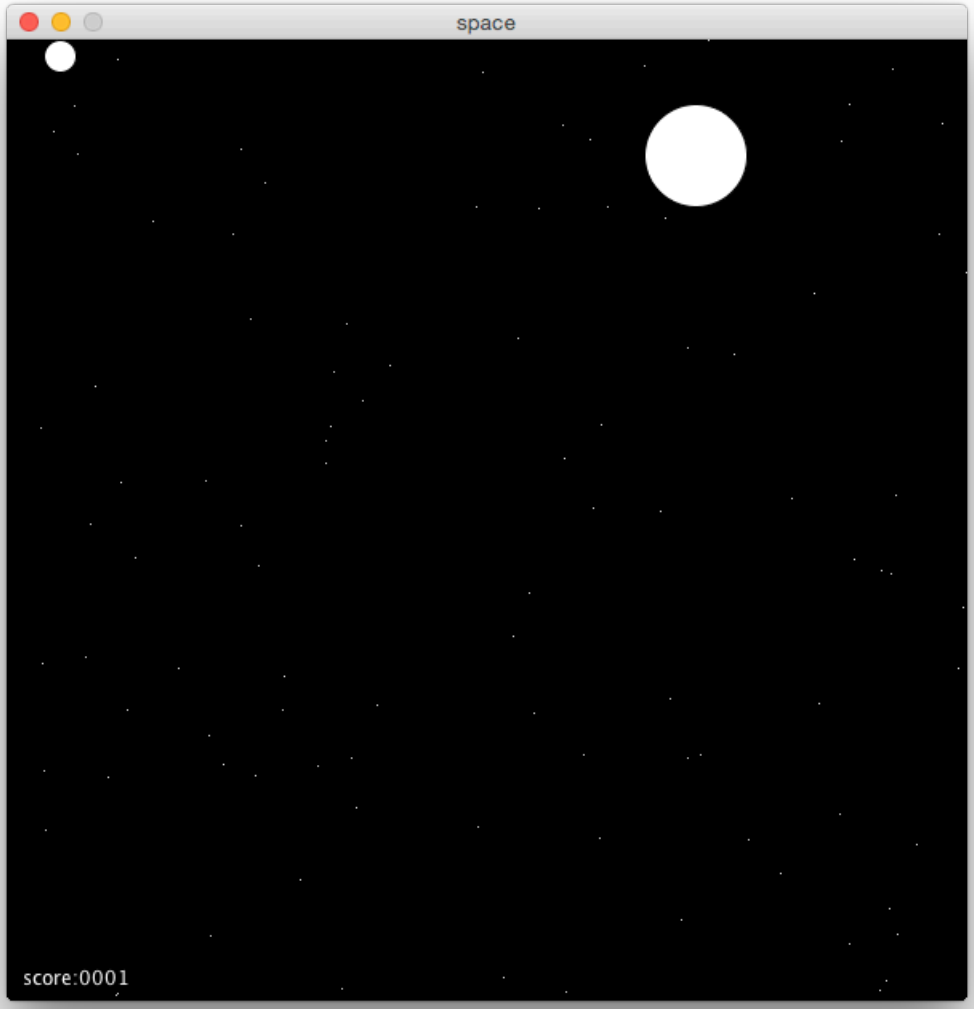

Figure 2: Expected distinction-level starry background (with wormhole and craft)

movement. Implement this by having 'w', 's', 'a', and 'd' act as movement keys for the viewer (i.e. they no longer change the ship's direction) with each having an effect according to the following table

key	*effect*
'w'	all the stars move down
's'	all the stars move up
'a'	all the stars move right
'd'	all the stars move left

You must implement this with loops for full marks.

Part 4: Functions - 20 marks

In this part you will *modularise* your code with functions.

Pass Part

Write a collision detection function with signature:

```
boolean warp();
```

which returns true whenever the craft has hit the wormhole. Use this function in the appropriate place in your draw loop. For full marks you must match the signature above and you must define that function *as the first function in your pde file (before setup and draw)*.

Distinction Part

Add two "black holes" to the game. They are always positioned at (100,40) and (400,500)[2]. They are smaller (40 pixels diameter) and they are black with a white edge. If the craft hits one, the game is over.

To get marks for this extra functionality, it must *all be coded in one function* with signature:

```
void blackHoleAt(float holeX, float holeY);
```

This function must be called two times in your draw function and it must perform both the drawing of the black hole and the collision detection against the craft. For full marks in this part the function must have exactly the signature above, must be called two times in your draw function and must be defined *directly under your warp function*.

Additional Functionality

We like to encourage creativity and for you to stretch your abilities on assignments. However, you can't add functionality which will hide things we are testing for. The following is a list of extra things you can add to your program which will make it better without hiding the functionality described above.

Make a better looking space craft You must ensure it has the same collision behavior as the basic flying saucer.

Make it look better in general The colours are uninspiring, but it is supposed to be deep space. You may fiddle them for something more appealing but it must still satisfy the requirements in part 2. The wormhole and the black holes are where you can be really creative. You may make them into anything you like as long as their "hit box" remains the same (and the wormhole pulsates).

High Score Have people give a 4 letter name when they start a game and then keep track of who has the highest score.

These tasks are not worth marks *but* the best submission overall will win a prize[3]. If, in the pursuit of the best possible game, you want to change something which would hide functionality worth marks, post your suggestion to the forum and we can all discuss how to implement it without making life hard for the markers.

Submission

Checkpoint Submission

In your practical class in week 9 you will have an opportunity to show your running program to your practical supervisor. If you have not yet got two parts of the assignment working, we will

[2]There must be some tear in the space-time continuum which allows the black holes to always be in the same place relative to us

[3]The prize has little value but is good for bragging rights

record a 10% penalty towards your final mark. As long as you have any two parts working and visible to your practical supervisor in week 9, you will avoid this penalty. "One part" means a pass or distinction part, not the two combined.

Final Submission

Before the due date and time, you must submit your Processing program online via the COMP115 iLearn site. You must submit your program as a single Processing source file called `ass2.pde`.

You can find the pde file inside the folder that stores your Processing sketch. Please rename it (if necessary) and submit just that file. Do not submit the whole sketch.

Marking

80% for the correctness of your code. We will check to see that your program correctly implements the specifications given above.

20% for the quality of your code. We will check to make sure that you have used reasonable names and types for your variables and that your code is presented in a manner that makes it understandable (e.g., good formatting, using layout to make the code more readable). Use the sample programs shown in lectures or the textbook as guidance.