**Setting up pygame:**

To begin using the pygame module, pygame first needs to be imported by typing:

import pygame

Next, pygame must be initialised by using:

pygame.init()

Which allows all the functionality of pygame to be used.

To create a **game screen** in pygame for the game to be graphically displayed you would need to write:

screen = pygame.display.set\_mode((800, 600))

Where the screen is set to the variable “screen”, and pygame’s display object is accessed to run the set\_mode method where the parameters are the height and then the width of the screen to be created in pixels.

The screen created is made up of 800 horizontal pixels and 600 vertical pixels. The top, left position of the screen is coordinate (0,0) so the x values get greater as you move right and the y values get greater as you move down.

If the program ends, then the game window will automatically close so to fix this, the game will run in an infinite loop until an exit condition is met (the player wants to quit the program).

To allow the player to quit the program, you would write:

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT():

running = False

pygame.quit()

Which keeps the game window open and checks every event in pygame until the player presses the quit button and then the loop will end and pygame will quit.

**Events:**

An **event** is any process which is executed inside the game window. Events can include inputs such as pressing arrow keys or moving your mouse. You can check if a certain event has happened by using:

for event in pygame.event.get():

if event.type == # The event you want to happen then

# Code that runs if the event happens

Here, a for loop is used to check every event which happened in the last frame and if the event type is the one you are looking for, then the corresponding code will trigger.

**Changing images and text:**

To change the **title and the icon** of the game window, you need to have an image file of the new icon in the same folder as the python file. To change the title, you would do:

pygame.display.set\_caption(“Space Invaders”)

The space invaders text can be changed to change the title and to change the icon, you would write:

icon = pygame.image.load('ufo.png')

pygame.display.set\_icon(icon)

Where ufo.png would be changed to whatever image would need to be used.

To change the background colour of the screen, you can use the screen.fill() method which takes in 3 parameters which are RGB values from 0 to 255.

To update the screen after changes are made you need to include the pygame.display.update() method which updates the screen.

The blit() function is used to draw an image onto the screen. For example:

screen.blit(playerImg, (playerX, playerY))

The first parameter of the blit function is the image to be painted onto the screen and the second parameter is the coordinates of the image to be placed at.

To add a background image, you set the image equal to a variable like so:

background = pygame.image.load('blue space.PNG')

Next, after the screen.fill command in the game loop, the background image will be displayed onto the screen like so:

screen.blit(background, (0,0))

**Movement in pygame:**

If you define a function to draw the player onto the screen, such as:

def player():

screen.blit(playerImg, (playerX, playerY))

And this is continuously run every frame inside a while loop, then if you update the playerX and playerY variables then the player image will move to a new coordinate accordingly.

Increasing an x value moves the image right whereas increasing a y value moves the image down.

**Keyboard inputs:**

In the game loop, inside the event detection you can check if an event that occurred during the last frame was an arrow key being pressed like so:

while running:

for event in pygame.event.get():

# if an arrowkey is pressed check if it is left or right

if event.type == pygame.KEYDOWN: # This means a key has been pressed

if event.key == pygame.K\_LEFT: #This means it was the left arrowkey

#code that runs when left key is pressed

if event.key == pygame.K\_RIGHT: #This means it was the right arrowkey

#code that runs when left key is pressed

pygame.KEYDOWN means a key has been pressed whereas pygame.KEYUP means that a key has been let go of. To update the position of the player you first define playerX\_change and playerY\_change variables which will later change the x and y coordinates and you set these values when arrow keys are pressed for example:

if event.type == pygame.KEYDOWN: # This means a key has been pressed

if event.key == pygame.K\_LEFT: #This means it was the left arrowkey

playerX\_change = -0.1

if event.key == pygame.K\_RIGHT: #This means it was the right arrowkey

playerX\_change = 0.1

if event.type == pygame.KEYUP:

if event.key == pygame.K\_LEFT or event.key == pygame.K\_RIGHT:

playerX\_change = 0

The playerX\_change being positive means that 0.1 will later be added to the x coordinate and it being negative means it will be subtracted.

To update the player position now you simply write:

playerX += playerX\_change

And this will automatically update the position of the player.

**Adding boundaries to the game:**

As there is no limit set to the possible x values of the player, it is able to exceed the boundaries of the screen and go off the screen which is not intended. To fix this, you can add an if statement after adding the playerX\_change which checks if the x coordinate has exceeded its limits and if so corrects it e.g.

if playerX <= 0:

playerX = 0

elif playerX >= 736:

playerX = 736

**Creating enemies:**

You can create enemies in pygame in the same way as the player storing their attributes as variables and displaying their images on the screen in the same way.

You can set enemies a continuous enemyX\_change value so that they are continuously moving and when they hit a wall, you can reverse their movement so they head in the opposite direction.

**Creating more game objects:**

To create other objects when an event is triggered, firstly you create all the attributes for that object. Then you create a function which creates that object e.g.

def fire\_bullet(x,y):

global bullet\_state

bullet\_state = "fire"

screen.blit(bulletImg, (x+16, y+10))

This function makes the bullet\_state attribute inside bullet global so that it can accessed by the function, then it sets the state of the bullet to fire as it has now been created and it creates an image of the bullet 16 pixels to the right of the x coordinate and 10 pixels down of the coordinate.

Then, you can make it when a button is pressed, a bullet is fired e.g.

if event.key == pygame.K\_SPACE:

fire\_bullet(playerX, bulletY)

Then, when the bullet has been fired, to update its position you can do:

if bullet\_state== "fire":

fire\_bullet(playerX, bulletY)

bulletY -= bulletY\_change

So that the bullet is constantly drawn and it moves over time.

However, this can only happen once so this must be reset when the bullet reaches the top of the screen like so:

if bulletY <= 0:

bulletY = 480

bullet\_state = "ready"

And as the fire\_bullet function takes the player’s x coordinate as a parameter, once the player moves the bullet will as well which is not correct so to fix this, you save the player’s x position when the bullet was fired and use that variable for all future uses of fire\_bullet().

**Collision detection:**

In order for most games to function properly, there needs to be a way to detect if one game object has collided with another game object. In pygame, as all the game objects we have made have a set of coordinates, these can be used to find the distance between two objects and then the distance can be used to decide if the objects are close enough to have collided. The mathematical equation:

Distance = √ (x2 – x1)2  + (x2 – x1)2

Can be used to find the distance between two sets of coordinates. For example:

def isCollision(enemyX, enemyY, bulletX, bulletY):

distance = math.sqrt(math.pow(enemyX - bulletX)+math.pow(enemyY - bulletY))

And then you can decide the amount of distance required for it to be too long for a collision to happen.

After you detect a collision has happened, you can then decide what you want to do with the colliding objects. For example with a bullet and an enemy:

collision = isCollision(enemyX,enemyY,bulletX,bulletY)

if collision:

bulletY = 480

bullet\_state = "ready"

score += 1

print(score)

enemyX = random.randint(0,736)

enemyY = random.randint(50,150)

**Creating multiple of the same game object:**

Game objects are not being defined in OOP so creating multiple of the same object is less straightforward. To do this procedurally, all 6 enemies will be stored inside of a list and the enemies will be created from that list using a for loop. For example:

enemyImg = []

enemyX = []

enemyY = []

enemyX\_change = []

enemyY\_change = []

num\_of\_enemies = 6

for i in range(num\_of\_enemies):

enemyImg.append(pygame.image.load('Space enemy.png'))

enemyX.append(random.randint(0,736))

enemyY.append(random.randint(50,150))

enemyX\_change.append(3)

enemyY\_change.append(40)

Here, each of the variables that were originally stored for a single enemy are now a list which can store the same variable for each different enemy in the game.

**Displaying text on screen:**

To display text on screen, you must first initialise a font object to decide which font is used e.g.

font = pygame.font.Font('freesansbold.ttf',32)

Where font is the name of the object its being assigned to, ‘freesansbold’ is just a free font that comes with pygame and 32 is the text size. Next, the text must be rendered using font’s render method. Rendering the text involves passing in: the string(s) which will be displayed on screen, True and the RGB values that the text will be.

score = font.render("Score: " + str(score\_value),True,(255,255,255))

Finally the text needs to be drawn onto screen which is done with:

screen.blit(score, (x, y))

However the rendering and drawing must be repeated each frame.

To add a different font than free sans bold then you can install them online and put them into your project folder.

**Sound effects and music:**

First, to play sounds you must have the audio files in the project folder and you must have imported mixer from pygame. Next you need to load the music and then play the music using mixer. For example:

mixer.music.load('background.wav')

mixer.music.play(-1)

The -1 will make the music play on loop.

If you want to do a sound effect instead, you will write something like:

bullet\_sound = mixer.Sound('laser.wav')

bullet\_sound.play()

Where mixer.Sound is used instead of mixer.music and there is no -1 and we do not want it to loop.

**Adding a game over screen:**

When you decide what your game over condition is, you can move all enemies off of the screen by giving them very large coordinates and then display game over text. The game over text is displayed in the same way as the score but should be large and in the centre of the screen instead.