University of Hull
Department of Computer Science

Writing Games with Pygame

Vsn. 1.0 Rob Miles 2014

Week 3: Games and Classes

These notes describe the practical elements of the course. They are to be used in conjunction with the slide decks.

Practical Break 1: A Player Object in CrackerChase

In this break we are going to see how a game controls a Player object. This will be functionally very similar to the program that we created last time, but it will be structured in a way that will make it much easier to create large and complex games.

Getting started

We are going to use the programs that we wrote last week as the basis of the work this week. You will also need to find the `cheese.png` file and add that to the folder where you store your programs. If you didn't get as far as drawing the cheese, you can find the cheese image here:

`http://www.robmiles.com/s/cheese.png`

Later on you will need a cracker. You can find it here:

`http://www.robmiles.com/s/cracker.png`

These images are not subject to any form of copyright. Also, I've eaten them both and they were delicious. My next game will probably involve pies.

Before you go any further; perform the following:

1. Log into your system with the username and password that you know and love from the past.
2. Start the Idle environment.

CrackerChase Game

To show how the game might work we are going to create a game called "CrackerChase". The cheese will move over the screen chasing crackers that appear at random positions. The aim of the game is to get as many crackers as you can in ten seconds.

The Player Class

The starting point will be a cheese that the player can control with the keyboard. We are going to make this a separate class. There are lots of advantages to doing this, we will discover some of them later in the labs. The Player is quite a large object, but don't let this worry you. Please don't try and type all this in though, the best way to get hold of this is to download the file from here:

`http://www.robmiles.com/s/Player.txt`

Before you go any further; perform the following:

3. Using Idle create a new Python game file called CrackerChase.
4. Using Internet Explorer, browse to the file above and open it in Notepad. Select all the text in Player.txt and paste it into your Python game.

```
import pygame
import random

class Player:
    def __init__(self, position, limit, image):
        # set the initial position and record this as the reset position
        self.resetPosition = position
        self.rect = pygame.Rect(0,0,image.get_width(), image.get_height())
        self.setPosition(position)

        self.limit = [float(limit[0]), float(limit[1])]

        self.image = image

        self.movementSpeed = [200,200] # pixels per second
        self.reset()

    def draw(self, surface):
        surface.blit(self.image, self.position)

    def update(self, deltaTime):

        # Handle movement - if we are moving in that direction
        # change the current position by the speed value

        if self.movingUp:
            self.position[1] = self.position[1] - (self.movementSpeed[1]*deltaTime)
        if self.movingDown:
            self.position[1] = self.position[1] + (self.movementSpeed[1]*deltaTime)
        if self.movingLeft:
            self.position[0] = self.position[0] - (self.movementSpeed[0]*deltaTime)
        if self.movingRight:
            self.position[0] = self.position[0] + (self.movementSpeed[0]*deltaTime)

        # Clamp the position values so we can't move off the screen

        if self.position[0] < 0:
            self.position[0]=0
        if self.position[1] < 0:
            self.position[1]=0
        if self.position[0] + self.image.get_width() > self.limit[0]:
            self.position[0] = self.limit[0] - self.image.get_width()
        if self.position[1] + self.image.get_height() > self.limit[1]:
            self.position[1] = self.limit[1] - self.image.get_height()

        # Move to the new position

        self.setPosition(self.position)
    # Movement controllers - two for each direction

    def StartMoveUp(self):
        self.movingUp = True
    def StopMoveUp(self):
        self.movingUp = False

    def StartMoveDown(self):
        self.movingDown = True
    def StopMoveDown(self):
        self.movingDown = False

    def StartMoveLeft(self):
        self.movingLeft = True
    def StopMoveLeft(self):
        self.movingLeft = False

    def StartMoveRight(self):
        self.movingRight = True
    def StopMoveRight(self):
        self.movingRight = False

    def setPosition(self,position):
        self.position = [float(position[0]), float(position[1])]
        self.rect[0]= self.position[0]
        self.rect[1]= self.position[1]

    def reset(self):
        self.setPosition(self.resetPosition)
        self.movingUp = False
        self.movingDown = False
        self.movingLeft = False
        self.movingRight = False
```

When you get time, you should take a good hard look at this code. There is some quite important stuff going on here, which is well worth knowing about. The **`Player`** value will be part of our game. Things to remember:

- The **`__init__`** method in a class runs when a new instance of the class is created. For the game class this is where we set up all the Pygame stuff. For the **`Player`** this is where all the settings for the player (position on screen, limits of the screen and image) are copied into the class instance.
- The **`self`** part means "use something which is part of this class". If it is a variable it means use (or create) a variable which is held in the class. If it is a method it means use the method with that name.
- We are using lists to hold coordinate value pairs. The **`position`** variable in the **`Player`** is a list that contains two values, x and y. We can get hold of the x coordinate by using **`position[0]`** and the y coordinate by using **`position[1]`**.

- The **Player** also contains a Pygame rectangle data member that is used to determine the region of the screen occupied by the player. This variable, called **rect**, is created when the Player is created and updated by the **setPosition** method. Games programs can use this variable to detect when the player has collided with another game object.
- The **Player** class contains an **update** method which is called at regular intervals by the game to update the state of a player. The **update** method is given a parameter, called **deltaTime**, that tells the player the number of seconds since the last time **update** was called. This allows a player object to adjust its position based on the speed it is presently moving.
- The Player class contains a **reset** method that is called at the start of a game. This puts the player back at the reset position and turns off all movement.

The CrackerChaseGame class

The game now has an object that can be used to represent a player on the screen. The next thing to do is create a class to hold the game itself. This will hold Player values and other objects that will represent the game in progress.

```
class CrackerChaseGame:

    def __init__(self):
        self.isRunning = False
        self.windowSize = (800, 600)
        self.backgroundColour = (255, 255, 255)
        self.textColour = (255,0,0)
        self.fps = 60
        pygame.init()
        self.surface = pygame.display.set_mode(self.windowSize)
        pygame.display.set_caption("Cracker Chase")

        cheeseImage = pygame.image.load("cheese.png")
        self.gameCheese = Player((0,0),self.windowSize,cheeseImage)

    def run(self):
        self.gameCheese.draw(self.surface)
        pygame.display.flip()
```

This is the starting code for our game. It creates a **Player** value (called **gameCheese**) and then draws it. Things to remember:

- This time the **__init__** method runs when a new **CrackerCheeseGame** instance is created. This is where the game is set up. The **Pygame** framework is initialised.
- The **__init__** method also creates an instance of the **Player** class, in a variable called **gameCheese**. The Python system knows that this is part of the **CrackerCheeseGame** class because it is referred to as **self.gameCheese**

Before you go any further; perform the following:

5. Add the above class into your program underneath the Player class.
6. If you run the program nothing will happen. This is because at the moment we have just described a couple of classes, there is no code that actually runs.

Running the Game

Now that we have our two classes we can use them to get the game working.

```
g = CrackerChaseGame()
g.run()
```

The above two statements create an instance of the game and then call the run method in that game instance. The instance is referred to by a variable called `g`. Things to remember:

- The `__init__` method inside `CrackerChaseGame` runs when the first statement is executed because constructors are always called when a new instance of a class is created. This method is where the game is set up and Pygame begins to run. If we ran the first statement on its own we would see an empty Pygame window appear on the screen.
- Neither of the two statements creates a `Player` instance. The player is created when the game is constructed.

Before you go any further; perform the following:

7. Add the above two statements into your program at the bottom.
8. Run the program and it should draw some cheese in the top right hand corner of the screen.

We now have two separate objects, one is the game and the other is the single game object. The object has a draw behaviour which we are now using to ask it to draw itself on the screen.

Practical Break 2: Controlling the Player Object

In this break we are going to see how a game controls a `Player` object and runs the Draw-Update behaviours.

```
class CrackerChaseGame:

    def __init__(self):
        self.isRunning = False
        self.windowSize = (800, 600)
        self.backgroundColour = (255, 255, 255)
        self.textColour = (255,0,0)
        self.fps = 60
        pygame.init()
        self.surface = pygame.display.set_mode(self.windowSize)
        pygame.display.set_caption("Cracker Chase")

        cheeseImage = pygame.image.load("cheese.png")
        self.gameCheese = Player((0,0),self.windowSize,cheeseImage)

    def drawGame(self):
        self.surface.fill(self.backgroundColour)
        self.gameCheese.draw(self.surface)
        pygame.display.flip()

    def updateGame(self,deltaTime):
        for e in pygame.event.get():
            if e.type == pygame.KEYDOWN:
                if e.key == pygame.K_ESCAPE:
                    self.isRunning = False
                    return
                if e.key == pygame.K_w:
                    self.gameCheese.StartMoveUp()
                if e.key == pygame.K_x:
                    self.gameCheese.StartMoveDown()
                if e.key == pygame.K_a:
                    self.gameCheese.StartMoveLeft()
                if e.key == pygame.K_d:
                    self.gameCheese.StartMoveRight()
            if e.type == pygame.KEYUP:
                if e.key == pygame.K_w:
                    self.gameCheese.StopMoveUp()
                if e.key == pygame.K_x:
                    self.gameCheese.StopMoveDown()
                if e.key == pygame.K_a:
                    self.gameCheese.StopMoveLeft()
                if e.key == pygame.K_d:
                    self.gameCheese.StopMoveRight()

        self.gameCheese.update(deltaTime)

    def run(self):
        clock = pygame.time.Clock()
        while True:
            # time to complete the last frame (in secs)
            deltaTime = clock.tick(self.fps)/1000.0
            self.updateGame(deltaTime)
            self.drawGame()
```

Some of this code will be familiar, but it has been arranged a little differently. The run method now implements a "game loop" which repeatedly calls **`updateGame`** and **`drawGame`** methods. The **`updateGame`** method updates the cheese and the **`drawGame`** method draws it.

Before you go any further; perform the following:

9. Replace the **`CrackerChaseGame`** class in your program with the code above. Be careful not to overwrite the bottom two statements that create the game and start it running.
10. Run the program and you should be able to steer the cheese around the screen with the keyboard.

Practical Break 3: Adding a Target

In this break we are going to add a target that randomly places itself on the screen. This is the cracker that we are going to chase.

Adding the Target Class

A target is like a very stupid player. It just draws itself. The only clever thing it can do is position itself randomly on the screen.

```
class Target:

    def __init__(self, limit, image):
        self.limit = [float(limit[0]), float(limit[1])]
        self.rect = pygame.Rect(0,0,image.get_width(), image.get_height())
        self.image = image
        self.randomPlace()

    def randomPlace(self):
        x = random.randint(0,self.limit[0]-self.image.get_width())
        y = random.randint(0,self.limit[1]-self.image.get_height())
        self.setPosition((x,y))

    def setPosition(self,position):
        self.position = [float(position[0]), float(position[1])]
        self.rect[0]= self.position[0]
        self.rect[1]= self.position[1]

    def draw(self, surface):
        surface.blit(self.image, self.position)
```

Before you go any further; perform the following:

11. Add the **`Target`** class above to your game. Put it above the **`CrackerChaseGame`** class.

Adding a Cracker to the game

Now we need to add a line to create a cracker for our game to use.

```
crackerImage = pygame.image.load("cracker.png")
self.gameCracker = Target(self.windowSize,crackerImage)
```

Before you go any further; perform the following:

12. Add the above lines to the **`__init__`** method in **`CrackerChaseGame`**. You want put them below the two lines in that create the **`gameCheese`** object. They look very similar.

Drawing the Cracker

Now that we have the cracker in the game, we have to draw it.

```
 self.gameCracker.draw(self.surface)
```

Before you go any further; perform the following:

13. Add the above statement to the **`drawGame`** method in **`CrackerChaseGame`**. You want put them below the statement that draws the **`GameCheese`** object.

If you run the game you should see a piece of cheese on the screen, and a cracker. You can move the cheese towards the cracker. Next time we will find out how to detect collisions between them.

Rob Miles June. 2014