University of Hull
Department of Computer Science

Writing Games with Pygame

Vsn. 1.0 Rob Miles 2014

Week 4: Using the Game Framework

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

ChasedByCrackers

We are going to make a game called **`ChasedByCrackers`**. The Cheese will be chased by an increasing number of crackers until it is caught. This is pretty much the reverse of the **`CrackerChase`** game that we worked with last time.

Getting started

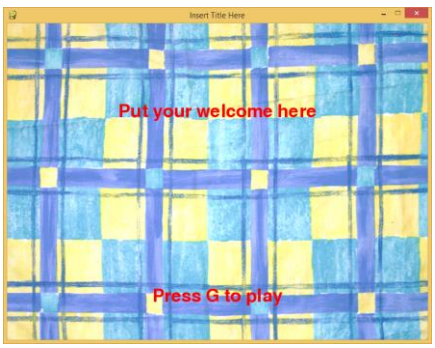
You will need to find the blank game framework that contains a game and some classes that represent items in the game. You will be copying and modifying elements in the framework to make the game that you want.

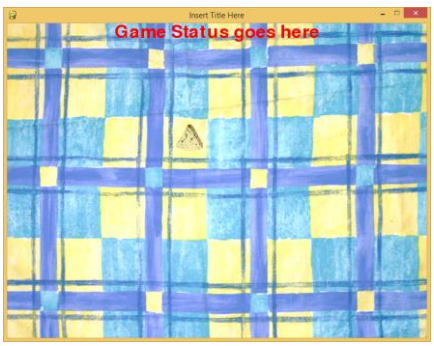
You can download the framework from:

`http://www.robmiles.com/s/PygameFramework.zip`

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.
3. Copy the game framework zip file from the web address above and unzip it into a folder on your machine.
4. Open the file **`Blankgame.py`** and run the program. You should see the first screen above. If you press the G key the game should change to the second screen.
5. Use the arrow keys to move the cheese around the screen. You should notice that it has a "physics" behaviour in that it will accelerate when you press an arrow key and then slow down when you release the key.
6. Press the ESC key to exit the game and close down Pygame.





These are the two game screens displayed by the framework application. Once you have entered the game there is no way back to the welcome screen. We will have to fix that when we make a game.

Creating a new game from the BlankGame template

Each time you make a new game you will start from the template folder. You can make a new folder simply by unzipping the archive into a different folder. The first thing we are going to do is change the name of the game class so that it reflects the game we are making.

Before you go any further; perform the following:

1. In the IDLE environment, find the line that declares the class:

   ```
   class BlankGame:
   ```

2. Change this so that the name reflects our new game:

   ```
   class ChasedByCrackers:
   ```

3. Save the file and run the program. Does it work?
4. The program won't work because you also have to change the code that actually runs the game. If you remember, we make games run by creating an instance of the game class and then calling the run method on that class.
5. Go right to the bottom of the file and find the line that creates the class instance:

   ```
   g = BlankGame()
   ```

6. Change this so that the class that is created has the right name:

   ```
   g = ChasedByCrackers()
   ```

7. Run the game, it should now work.
8. You should really change the name of the game file to reflect what it holds. Exit from IDLE, rename the file to **`ChasedByCrackers`** and re-open it.

Creating a Cracker class

At the moment the game just uses the **`PhysicsMover`** and **`Background`** object. It needs a third object, a class that will be used for each of the crackers. We are going to create this by making a copy of the **`Mover`** class and then customising it.

Before you go any further; perform the following:

1. Open the folder that contains your game and make a copy of the **`Mover`** python source file. Call the new file **`Cracker`**.
2. You can do the copy by right clicking the file and selecting copy from the menu that appears. Then you right click the new file, select rename from the menu and change the name to **`Cracker`**. You don't need to put the .py language extension.
3. Open the **`Cracker`** file using IDLE and change the name of the class from **`Mover`** to **`Cracker`**.

Using the Cracker class in the Game

The next thing we need to do is make the **`ChasedByCrackers`** class able to use the **`Cracker`** class. Python lets you spread programs over several source files, but if a program needs to use the contents of a source file it needs to be explicitly imported.

Before you go any further; perform the following:

1. You should have two Python files open in IDLE at the moment, **Cracker** and **ChasedByCrackers**. Go to the **ChasedByCrackers** file and look at the top. You should find a number of from statements, these are how we are importing the **PhysicsMover** and **Background** classes into the game.
2. Add the statement to the program to import the Cracker class into the game:

```
from Cracker import *
```

We can now use the **Cracker** class in our game. If we had several classes in the source file **Cracker.py** the statement above would import all of the classes, that is what the * means.

Meaningful Names

You might want to modify the names of the other game objects to better reflect what they do. At the moment that class that contains the cheese is called **PhysicsMover**. It might be better if it was called **PlayerCheese**. You can rename the class (and the file that it is in) just by changing the text in the class itself, renaming the file, changing the **from** statement and then using the new name in the game program. Then again, you might decide not to bother….

Adding a Cracker object to the game

We now have a **Cracker** class which we can use in the game, but at the moment the game doesn't contain any crackers. We are going to start by adding one cracker and making it chase the cheese. If the cracker catches the cheese, the game is over. We will need to add the cracker in a number of places in the game:

- When the game is initialised we need to create the cracker
- When the game is drawn we need to draw the cracker
- When the game is started we need to reset the cracker for the next game
- When the game is updated we need to update the cracker

You will perform this sequence for every new object that you want to add to the game. If we want to add "dangerous tomatoes" we would create a Tomato class and then add that in the same places.

Creating a Cracker when the game is initialised

Game objects are created when the game is started. At the moment our game contains two objects, the background and the cheese. We are going to add a third, a single **Cracker** that will chase the player. We will need to load the image from a file and then create a **Cracker** instance.

```
crackerImage = pygame.image.load("cracker.png")
self.cracker = Cracker((500,500),self.windowSize,(50,50),crackerImage,30)
```

Before you go any further; perform the following

1. Copy the **cracker.png** image into the folder where your game lives. You can find the cracker image here:

   http://www.robmiles.com/s/cracker.png
2. Find the **__init__** method in the **ChasedByCrackers** class.
3. Find the following statements in that method:

```
cheeseImage = pygame.image.load("cheese.png")
self.gameCheese = PhysicsMover((0,0),self.windowSize,(10,10),cheeseImage,50)
```

4. Add the two statements above just after these two.

If you run your program now it should work fine, but you won't see any crackers on the screen. You need to draw the cracker each time the screen is drawn.

Drawing the Cracker

We have seen that the game has a **drawGame** method that is called when the screen needs to be redrawn. At the moment this draws the background, the status message and the cheese. We are going to add another statement to draw the cracker.

```
self.cracker.draw(self.surface)
```

Before you go any further; perform the following

1. Find the **drawGame** method in the **ChasedByCrackers** class.
2. Find the following statements in that method:
   ```
   self.gameCheese.draw(self.surface)
   ```
3. Add the statement above just after this.
4. Run the game. You should see the cracker on the screen. We placed it at (500,500) so it is quite a way down the screen.

Note that the game program doesn't actually draw the cracker as such, it just asks the cracker to perform its draw action. The cracker knows where it is on the screen, and will draw itself in the right place, with the image that we gave it when it was created.

Resetting the Cracker at the Start of a Game

This is something I often forget to do. So we might as well do it now. When the game restarts we need to move the cracker back to its original location. The good news is that the **Mover** sprite that the **Cracker** is based on has a reset behaviour that was created specifically for this. In the case of the **Cracker** this just sets the sprite location to **(500,500)**, which was passed in when the sprite was created.

```
self.cracker.reset()
```

The statement above will ask the cracker to reset itself. We need to add this to the method that is called when the game starts running.

Before you go any further; perform the following

1. Find the **startGame** method in the **ChasedByCrackers** class.
2. Find the following statements in that method:
   ```
   self.gameCheese.reset()
   ```
3. Add the statement above just after this.
4. This will make no difference to the way that the game works at the moment, but it does mean that when we start the game the cracker will have been reset.

Note that if we added extra features to a cracker, for example we might give it three lives, we would reset the lives counter inside the reset method in the cracker.

Updating the Cracker

You must be used to the pattern by now. We are connecting behaviours in our game to behaviours in the objects that the game contains. The final behaviour that we need to address is the update behaviour of the cracker in our game. We want the cracker to chase the cheese and kill it. The first thing we need to do is to get the cracker to update when the game updates. We need to add another statement to the game:

```
self.cracker.update(deltaTime)
```

This statement calls the update method in the cracker and passes it a value that gives the number of seconds that have passed since the last time update was called. The cracker will then update itself.

Before you go any further; perform the following

1. Find the **`updateGame`** method in the **`ChasedByCrackers`** class.
2. Find the following statements in that method:
   ```
   self.gameCheese.update(deltaTime)
   ```
3. Add the statement above just after this statement.
4. Run the game. Does the cracker chase the cheese?

At the moment the cracker is not chasing the cheese because it has not been told how to do that. The cracker only knows how to move. At the moment the only way we can make a cracker move is by calling the Start and Stop move methods, in the same way as we control the player cheese.

However, we want the cracker to chase the cheese. Which means the cracker needs to be told where the cheese is so that it can chase it. The best place to do this is when we create the cracker.

```
# Initialise the Cracker object
#    position - the initial position of the player
#    limit - a tuple giving the size of the screen
#    speed - a tuple giving the speed of movement in pixels/second
#    image - the image to be drawn
#    width - the x size of the image in pixels
def __init__(self, position, limit, speed, image, width):
```

This is the start of the **`__init__`** method for the **`Cracker`**. It is given the information to set up a **`Cracker`** instance. I've added a set of comments so that users of the **`Cracker`** class will know how it works. We need to add another parameter that gives the item to be chased:

```
# Initialise the Cracker object
#    position - the initial position of the player
#    limit - a tuple giving the size of the screen
#    speed - a tuple giving the speed of movement in pixels/second
#    image - the image to be drawn
#    width - the x size of the image in pixels
#    target - the object the cracker is chasing
def __init__(self, position, limit, speed, image, width, target):


    # Set the target for the chase
    self.target = target
```

The above statements show the new comments and the start of the method. There are also two statements that set the target.

Before you go any further; perform the following

1. Find the **`__init__`** method in the **`Cracker`** class.
2. Change the top of the method to the statements above. Don't forget to include the two new statements.
3. Save the **`Cracker`** class file.
4. Run the **`ChasedByCrackers`** game. Does it work?

The program won't work because we have broken the process of creating a **Cracker**. When we create a new **Cracker** we now have to tell it the target, as well as the other information. We are going to have to revisit the part of the game that creates the Cracker. This is the statement that we used to originally:

```
self.cracker = Cracker((500,500),self.windowSize,(50,50),crackerImage,30)
```

This is what we need to change it to. Note that it is exactly the same, except that it now has an extra parameter on the end which is the cheese we want it to chase.

```
self.cracker = Cracker((500,500),self.windowSize,(50,50),crackerImage,30, self.gameCheese)
```

Before you go any further; perform the following

1. Find the **__init__** method in the **ChasedByCrackers** class.
2. Change the code that creates the cracker and update it to the second of the two statements above.
3. Run the game. It should work, but the cracker still won't chase the cheese I'm afraid.

You will go through this process every time that you add something new to a game object that needs to be set up when the game starts.

Adding the Chase behaviour to a Cracker

The last thing we need to do is get the Cracker to chase the Cheese. The actual chase code is very simple in this case. If the cracker is below the cheese it moves up, if it is above it moves down. Repeat that for left and right and you have a very simple piece of "artificial intelligence".

```
cpos = self.getCentre()
tpos = self.target.getCentre()
if(cpos[0]<tpos[0]):
    # sprite is to the left of the target
    # must move right towards it
    self.StopMoveLeft()
    self.StartMoveRight()
else:
    # Go the other way
    self.StopMoveRight()
    self.StartMoveLeft()

if(cpos[1]>tpos[1]):
    # sprite is below the target
    # must move up towards it
    self.StopMoveDown()
    self.StartMoveUp()
else:
    # Go the other way
    self.StopMoveUp()
    self.StartMoveDown()
```

Note that all the sprites provide a method called **getCentre** which returns a tuple giving the X and Y coordinates of the sprite on the screen. The statements above use this to achieve the chase behaviour.

Before you go any further; perform the following

4. Find the **`update`** method in the **`ChasedByCrackers`** class.
5. Add the statements above to the method.
6. Run the game. It should work, and the cracker should chase the cheese.

This should show you how you can add a new type of object to a game and then change the behaviour of the object and how the game uses it to suit the particular gameplay that you want.

However it is not finished. Here are some things that you could consider.

Finishing the game

When the cracker collides with the cheese the game should end. The best way to do this would be for the cracker to set a flag in the cheese (perhaps called **`cheeseDead`**) to **`True`**. The game update method will check this flag in the cheese player and if the flag becomes set the game will end.

When the game is reset the flag will be set to false. To do this you will need to:

1. Create a **`cheeseDead`** flag in the **`PhysicsMover`** class (unless you have renamed the class to something more sensible).
2. Add a statement to the **`reset`** method in **`PhysicsMover`** to set the **`cheeseDead`** flag to **`False`** when the game is reset.
3. Add a test in the **`Cracker`** class to set the **`cheeseDead`** flag to **`True`** when the cracker gets close to the cheese. You could use the **`getDistanceFrom`** method for this or you could check to see if the rectangles from the two objects intersect.
4. Add a test in the game to check the **`cheeseDead`** flag and end the game when it is **`True`**.

Before you go any further; perform the following

1. Do all this.

Adding lots of crackers

At the moment the game is a bit (OK a lot) boring. You could make it more interesting by having lots of crackers. You could create a list of crackers and then update and draw each of these in turn.

Adding Scoring

We could also add scoring. One way to score would be to keep track of how long the player has survived.

Stealing Code

Don't forget that you can use the CrackerChase game and anything else as the basis of your new game. Feel free to grab as much as you can from any of these resources.

Rob Miles June. 2014