Game Development 2 – Movement

Last week we started making our own “Birdie” game and the bird appeared on the screen. This week, we add movement to the game so that our main character can move around and avoid obstacles.

Task 1: Flapping Birdie’s Wings

If you have successfully completed task 4 from last week you do not have to program anything yourself here. Do take a look at the approach here though and see if this is what you did.

Birdie does flap his wings yet. As we have learnt, the main event loop must process all of the events. As it turns out, the code that you have already sends an event every time Birdie should move its wings. All that is left to do is to forward this information to the bird.

Take a look at the block of code starting at

```
def handle_events(events):
```

Currently, this block of code stops the program when the player clicks the cross because an event of type “pygame.QUIT” is sent.
Every time the bird should flap its wings an event of type “ANIMATE_BIRD_E” is sent. Use an if-statement to flap Birdie’s wings when this event is sent. To flap the wings, write:

```
bird.flap()
```

Task 2: Basic Movement

Remember the main event loop: in order to change the scene on the screen we first listen for events, then change the scene that is to be shown, and then finish by drawing it. This is repeated several times per second to create the illusion of movement.

Look at the main loop: By calling handle_events(pygame.event.get()) we have listened for events. The next step is to change the scene based on the information we have.

To find out which keys are currently pressed, use:

```
keys = pygame.key.get_pressed()
```

If the down-arrow key is pressed, Birdie should move down:

```
if keys[K_DOWN]:
    bird.move_down()
```

Add these lines of code in the place where you think they should go. Now extend this functionality by allowing Birdie to move in the other directions.

Task 3: Code Organisation

Our code base is now getting larger and more difficult to understand. One of the things software engineers do is to try and organise their code in ways that make it easier to understand and modify.
So-called python functions allow programmers to group bits of code and then reuse it in different areas. They are written as follows:

```
def some_function_name():
    # put some code here that you want to organise as a function
```

We have just added basic movement to Birdie and all the code that handles this has little to do with the other parts of the main loop, e.g. drawing onto the screen or handling other events. Instead, create a function containing all of this code to organise it:

```
def handle_movement():
    # Code to handle movement goes here
```

Now, inside the main loop you can "call" handle_movement() instead of using all of the lines of code that it contains:

```
handle_movement()
```

Can you think of other reasons that make the use of functions essential in programming?

Task 4: Obstacles

Currently, Birdie can freely move around the world. This does not make for a very exciting game which is why we will be adding obstacles next.

Add the following function to the code. What do you think it allows you to do?

```
def add_obstacle():
    top = random.randrange(20, 250)
    top_obstacle = Obstacle(True, top)
    bottom_obstacle = Obstacle(False, top + 150)

    top_obstacle.add(obstacles)
    bottom_obstacle.add(obstacles)
```

Once you know what this function does, call it in the right place of the code and run the code. What did we just do? Close the game and run it again – can you see a difference? Try and explain the difference by looking at the add_obstacle() code and the code starting with

```
class Obstacle(pygame.sprite.Sprite):
    ...
```

Advanced: What do you think "class Obstacle" represents and can you explain why it might be useful to use class?

Next time we will handle collision between Birdie and the obstacles.