Easy Pygame 2: Keyboard and Mouse

Event Handling

The nice thing about pygame is that you don't have to do any real event handling to respond to mouse and keyboard events. The pygame system captures any events that happen throughout the life of the game and stores them in a queue. It also takes certain actions in response to these events.

Most relevant for us, it keeps track of which keys and buttons are currently being held down and where the mouse is. This means that for most purposes, all you have to do to create a game with keyboard or mouse control is to build a game loop which handles the animations, repeatedly asks the pygame system where the mouse is and which keys and buttons are currently in the down position and responds appropriately.

As always, the `mohawk_pygame.py` module makes all this even easier for you! In this handout, we're assuming that you have imported this module using the following command:

```
import mohawk_pygame as mpg
```

A Simple Game Loop

A simple game loop can be built as an extension of the animation loop from the previous handout:

1. Initialize game variables (object positions, speeds, etc.)
2. Check keyboard and mouse and respond appropriately (e.g. if down key is currently pressed, change the y location of the player object)
3. Update position variables for objects that are moving automatically.
4. Clear and redraw the screen and all game objects.
5. Pause for 1/60 of a second or so (e.g. `mpg.sleep(1/60)`).
6. Go back to step 2.

An example of a game loop like this in action can be found in the file *keyboard_and_mouse_control.py*. There are no automatic animations in this, but the up and down keys, mouse movements, and mouse button presses all cause changes to the state of the game objects.

Mouse Position

Get the current mouse location like this:

```
    mx, my = mpg.get_mouse_pos()
```

Mouse Buttons

Check if a mouse button is currently depressed like this:

```
mpg.is_button_down(button_num)
```

The `button_num` must be `1`, `2`, or `3`. It returns `True` or `False` to indicate if the button is down.

Keys

You can find out if a key is currently depressed like this:

```
mpg.is_key_down(key)
```

If the specified `key` is down, this function returns `True`. Otherwise, it returns `False`.

The `key` can be a string containing a character (e.g. 'a') or if you want to know about the arrows, or the left or right SHIFT or ALT keys, you can use `mpg.LEFT`, `mpg.RIGHT`, `mpg.UP`, `mpg.DOWN`, `mpg.LSHIFT`, `mpg.RSHIFT`, `mpt.LALT`, or `mpg.RALT`..

A Minor Limitation

The above is all you really need to get going with making games using mohawk_pygame.py.

But one issue with the game loop pattern shown above is its timing. You are pausing for 1/60 of a second for a frame rate of 60 frames per second. But the actual frame rate will be lower and will depend on the speed of the computer on which the game is running. The fix for this is to keep track of the actual time elapsed at the top of each loop and then do some calculations to decide how many frames you need to process each time through the loop. But that's more complicated!

More Limitations

If your game loop is operating quickly this handout might be all you need to generate a cool game. But there is always a possibility that you will miss an event (for example, the user presses and releases a key so fast that it happens between calls to `mpg.is_key_down`). Another limitation is that every key and button press generates two events – one when the button is pressed down and another when it is released. The above methods don't let you process those events separately. And there are other events (such as scroll wheel events) that cannot be handled in this way.

If any of these situations apply to you, you will have to retrieve the entire list of recent events from pygame and sort through them. For more information on how to do this, see the pygame documentation. In particular, you should read:

http://www.pygame.org/docs/ref/event.html
http://www.pygame.org/docs/ref/key.html
http://www.pygame.org/docs/ref/mouse.html

Accessing Raw Pygame Stuff

If you see some cool pygame stuff you want to use, you can access it from within mohawk_pygame using `mpg.pygame.stuff` instead of `pygame.stuff`.