PyGame Primer

Due April 15th, 2016
CSE332 Programming Paradigms, Spring 2016

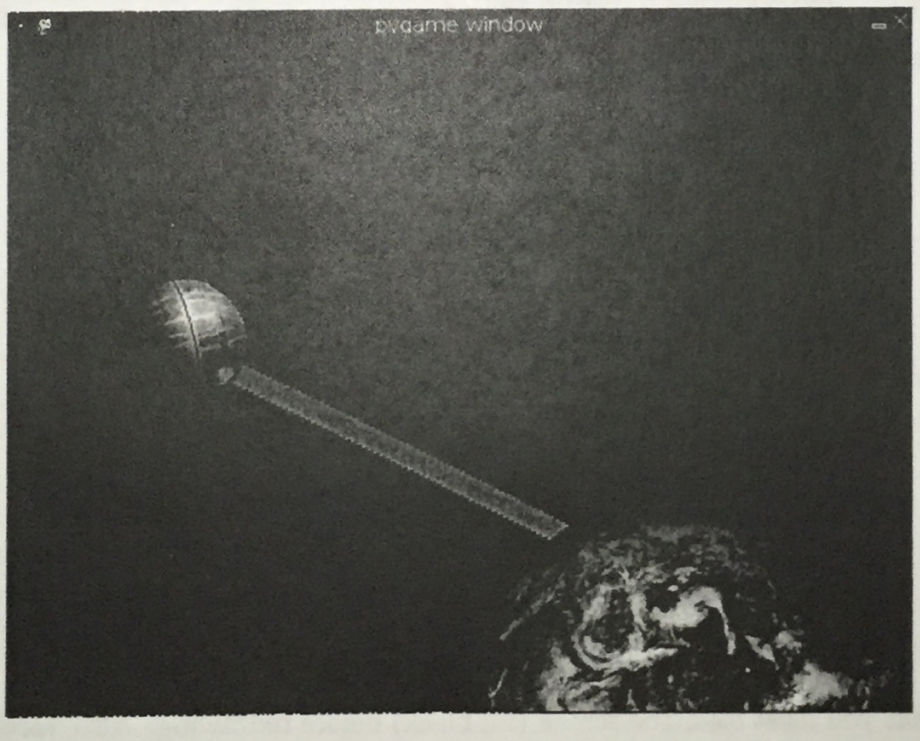
Overview Write a very simple game using PyGame. Your game may differ from the one discussed in class, but should include at least the following elements:

1. (5 points) A "player" avatar that is movable with the keyboard and rotates to face the mouse.
2. (5 points) An "enemy" avatar that either is static or moves on a fixed path.
3. (5 points) "Battle" functionality that fires a projectile, laser, etc.
4. (5 points) Enemy hitpoint reduction on projectile collision.
5. (5 points) Animated "disappear" (e.g., explosion) of enemy when hitpoints exhausted.
6. (5 points) Sounds for projectile launch and/or enemy elimination.

That's 30 points total! You may be creative with your design, or you may duplicate the "Deathstar Simulator" game from class. The media files for that simple game are in the AFS folder:

`/afs/nd.edu/user37/cmc/Public/cse332_sp16/pygame_primer/media/`

The "simulator":



Discussion To get started programming, first create a `GameSpace` class to hold your initialization logic and game loop.

```
class GameSpace:
        def main(self):
                # 1) basic initialization
                pygame.init()

                self.size = self.width, self.height = 640, 480
                self.black = 0, 0, 0

                self.screen = pygame.display.set_mode(self.size)

if __name__ == '__main__':
        gs = GameSpace()
        gs.main()
```

Next, try to set up just one game object, such as the player. This goes in the `GameSpace` `main()` function, after the basic initialization is all done. Remember that you also need a game clock to regulate the tick speed:

```
                # 2) set up game objects
                self.clock = pygame.time.Clock()

                self.player = Player(self)
```

Now you start your game loop, kind of like an event loop:

```
                # 3) start game loop
                while 1:
                        # 4) clock tick regulation (framerate)
                        self.clock.tick(60)

                        # 5) this is where you would handle user inputs...

                        # 6) send a tick to every game object!
                        self.player.tick()

                        # 7) and finally, display the game objects
                        self.screen.fill(self.black)

                        self.screen.blit(self.player.image, self.player.rect)

                        pygame.display.flip()
```

Notice that you must clear the screen (fill black) before redrawing. Also, recall from class that the `flip()` function is necessary due to the double-buffered animation system.

Now it's time to start the Player object. Write your game objects above the GameSpace class. Remember that they should inherit from Sprite!

```
class Player(pygame.sprite.Sprite):
        def __init__(self, gs=None):
                pygame.sprite.Sprite.__init__(self)

                self.gs = gs
                self.image = pygame.image.load("deathstar.png")
                self.rect = self.image.get_rect()

                # keep original image to limit resize errors
                self.orig_image = self.image

                # if I can fire laser beams, this flag will say
                # whether I should be firing them /right now/
                self.tofire = False
```

Notice that the self.gs will refer to the GameSpace main area. This is crucial as game objects (e.g., anything with a tick()) often need to access functions or "globals" from a common area. The GameSpace serves as this common area.

On each tick, your Player should update based on any events that have occurred. This can mean events in the classic sense, or it can mean "events" that you need to check for, such as mouse positions or sprite collisions. For example, the Player's tick() will look something like:

```
        def tick(self):
                # get the mouse x and y position on the screen
                mx, my = pygame.mouse.get_pos()

                # this conditional prevents movement while firing
                if self.tofire == True:
                        # code to emit a laser beam block
                else:
                        # code to calculate the angle between my current
                        # direction and the mouse position (see math.atan2)
                        # ... use this angle to rotate the image so that it
                        # faces the mouse
```

You're on your way now! Finish your homework by building one feature at a time, based on the feature list on the first page. Just do one at a time, so that you bank 5 points at a time. Don't try to do the assignment all in one "bite"...

Learning Objectives What you're really praticing here is the "clock tick" design paradigm, where the behavior of your program is controlled by a combination of events and clock ticks. Technically, this happens via a `tick()` function in each game object. The clock tick design is common not only in games, but also embedded systems and distributed programs. A major research area in computer science deals with how to ensure that the objects are properly synchronized on each tick, among other problems.

Turn in Package your source code and media files into a `zip` or `tar.gz` archive and submit it to `prog.paradigms.secN.sp16@gmail.com`, where `N` is your section number. Include a `README` file that briefly (3-4 sentences) describes your game and how to play it, if it is different from the one demonstrated in class.

Resources

`http://pygame.org/docs/`
`http://pygame.org/docs/tut/intro/intro.html`
`http://pygame.org/docs/tut/chimp/ChimpLineByLine.html`
`http://docs.python.org/2/library/math.html`
Trig. reminder for rotations:
`http://www.mathsisfun.com/algebra/trig-inverse-sin-cos-tan.html`