Pygame - Dev Notes - Player Health - Intro

- Dr Nick Hayward

A brief intro on checking and recording a player's health with Pygame.

Contents

Intro

We may add a status bar, for a player's health, lives, &c. to the game window, and then dynamically update it relative to a defined health value. For example, this might be a simple percentage that we decrement for each collision.

Check player's health

Our current game only gives a player one chance to shoot and destroy any advancing mob objects before a single hit ends the game.

A single hit may make the game more challenging, but it is certainly not what most players would expect for this type of game. So, we may now consider monitoring, and updating the status of a player's health as they are hit by advancing mob objects, i.e. our falling and rotating asteroids.

One way to protect our player, and their ship, is to use a *Star Trek* like shield. This shield will offer full protection initially, and then weaken with each hit from a mob object until it eventually fails at value `0`. We'll set a default for this player shield in the `Player` class,

```
# set default health for our player - start at max 100% and then decrease...
self.stShield = 100
```

Then, we need to modify our logic for a mob collision to ensure that the way we handle such objects better reflects a decrease in the player's shield, and health.

For example, instead of allowing a mob object to continue after it has collided with the player, we now need to remove it from the game window. If we don't update this boolean to `True`, each mob object will simply continue to hit the player as it moves, pixel by pixel, through the player's ship. A single hit would quickly become compounded in the gameplay.

```
# add check for collision - enemy and player sprites (True = hit object is now
deleted from game window)
collisions = pygame.sprite.spritecollide(player, mob_sprites, True,
pygame.sprite.collide_circle)
```

Then, as our player may be hit by multiple mob objects, we also need to update our check from a simple conditional to a loop through the possible collisions,

```
# check collisions with player's ship - decrease shield for each hit
for collision in collisions:
    # decrease player's shield for each collision
    player.stShield -= 20
```

```
    # check overall shield value - quit game if no shield
    if player.stShield <= 0:
        running = False
```

So, our player's shield will now survive four direct collisions before the fifth will destroy the ship, and then end the game.

Replace mob objects

Whilst this now works as expected, we have an issue with losing mob objects if they collide with the player's ship. This follows the same underlying pattern as the player's laser beam firing on the asteroids, our mob objects.

So, we need to create a new object if it is removed after a collision. As this is a familiar pattern, we may now abstract the creation of the mob objects to avoid repetition of code, e.g.

```
# create a mob object
def createMob():
    mob = Mob()
    # add to game_sprites group to get object updated
    game_sprites.add(mob)
    # add to mob_sprites group - use for collision detection &c.
    mob_sprites.add(mob)
```

This simple abstracted function now allows us to easily recreate our mob objects by creating a mob object, adding it to the overall group of `game_sprites`, and then the specific group for the game's `mob_sprites`.

We can then call this abstracted function whenever a mob object collides with a projectile, or the player's ship. We may also call this function when we initially create our new mob objects as part of the loop to 10.

```
# create a new mob object
createMob()
```

Show player's health - status bar

We've already defined a default maximum for our player's shield, and we can now start to output its value to the game window.

Whilst we could simply output a numerical value, as we did for the player's score, it seems more interesting to show a graphical update for the status of a player's health.

So, we can define a new draw function to allow us to render a visual health bar for the player's shield,

```
# draw a status bar for the player's health - percentage of health
def drawStatusBar(surface, x, y, health_pct):
    # defaults for status bar dimension
    BAR_WIDTH = 100
    BAR_HEIGHT = 10
    # use health as percentage to calculate fill for status bar
    bar_fill = (health / 100) * BAR_WIDTH
    # rectangles - outline of status bar...
    bar_rect = pygame.Rect(x, y, BAR_WIDTH, BAR_HEIGHT)
    fill_rect = pygame.Rect(x, y, bar_fill, BAR_HEIGHT)
    # draw health status bar to the game window - 1 specifies pixels for border width
    pygame.draw.rect(surface, GREEN, fill_rect)
    pygame.draw.rect(surface, WHITE, bar_rect, 1)
```

This function accepts four parameters, which allow us to easily define a surface for rendering, its `x` and `y` location in the game window, and then update the status of the player's health. In this case, we're simply updating the health of the shield for the player's ship.

We can set a default width and height for the status bar, and then specify how much of this bar needs to be filled with colour relative to the player's current health status. This health status can be calculated as a percentage, which then allows us to easily modify the relative sizes for the status bar.

We may also specify our rectangles for the status bar, which includes an outer container for the status bar, and effectively an inner bar for the colour fill to represent the player's health.

Our current health status bar will now start by showing 100% fill, and then reduce by 20% for each collision between a mob object, one of our asteroids, and the player's ship. When the health hits 0, the game will then quit.

References

- pygame.draw
- pygame.sprite

Demo

- playerhealth1.py
- shooter1.0.py
- check player's health
  - set default health to 100%
  - decrement health per collision
    - quit game when health reaches 0
  - draw status bar to game window
    - green colour for good health
    - change to red colour below 40%