How to Program a Game in Python with Pygame

This is an introduction to Pygame for people who already know Python. This article will teach you the steps to building a simple game that has the player dodging bouncing balls. Download Pygame. Find it for your platform from...

Part 1 of 8:

Installing Pygame

1. **Download Pygame.** Find it for your platform from http://www.pygame.org/download.shtml .
2. **Run the installer.**
3. **Verify that the installation worked.** Open a Python terminal. Type "import pygame." If you don't see any errors then Pygame was successfully installed.

```
import pygame
```

Part 2 of 8:

Setting Up A Basic Window

1. **Open a new file.**
2. **Import Pygame.** Pygame is a library that provides access to graphics functions. If you want more information on how these functions work, you can look them up on the Pygame website. https://www.pygame.org/docs/

```
import pygame from pygame.locals import *
```

3. **Set the window resolution.** You'll be making a global variable for the screen resolution so that can be referenced in several parts of the game. It's also easy to find at the top of the file so it can be changed later. For advanced projects, putting this information in a separate file would be a better idea.

```
resolution = (400,300)
```

4. **Define some colors.** Colors in pygame are (RBGA which range in values between 0 and 255. The alpha value (A) is optional but the other colors (red, blue, and green are mandatory).

```
white = (255,255,255) black = (0,0,0) red = (255,0,0)
```

5. **Initialize the screen.** Use the resolution variable that was defined earlier.

```
screen = pygame.display.set_mode(resolution)
```

6. **Make a game loop.** Repeat certain actions in every frame of our game. Make a loop that will always repeat to cycle through all these actions.

```
while True:
```

7. **Color the screen.**

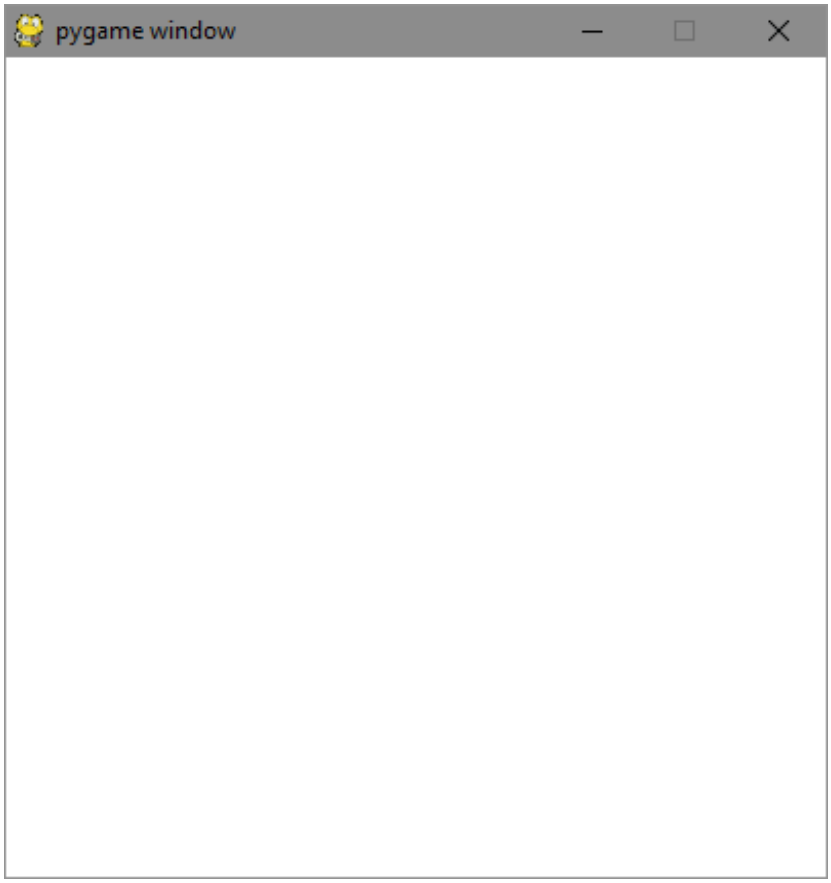
   ```
   screen.fill(white)
   ```

8. **Display the screen.** If you run the program, the screen will turn white and then the program will crash. This is because the operating system is sending events to the game and the game isn't doing anything with them. Once the game receives too many unhandled events, it will crash.

   ```
   while True: ... pygame.display.flip()
   ```

9. **Handle events.** Get a list of all events that have occurred in each frame. You're only going to care about one event, the quit event. This occurs when the user closes the game window. This will also prevent our program from crashing due to too many events.

   ```
   while True: ... for event in pygame.event.get(): if event.type == QUIT:
    pygame.quit()
   ```

10. 

    Try it out! Here's what the code should look like now:

    ```
    import pygame from pygame.locals import * resolution = (400,300) white
    = (255,255,255) black = (0,0,0) red = (255,0,0) screen = pygame.display
    .set_mode(resolution) while True: screen.fill(white) pygame.display.
    flip() for event in pygame.event.get(): if event.type == QUIT: pygame.
    quit()
    ```

Part 3 of 8:

Making a Game Object

1. **Make a new class and constructor.** Set all the properties of the object. You're also providing default values for all the properties.

   ```
   class Ball: def __init__(self, xPos = resolution[0] / 2, yPos = resolution[1] / 2, xVel = 1, yVel = 1, rad = 15): self.x = xPos self.y = yPos self.dx = xVel self.dy = yVel self.radius = rad self.type = "ball"
   ```

2. **Define how to draw the object.** Use the properties that were defined in the constructor to draw the ball as a circle as well as to pass a surface into the function to draw the object on. The surface will be the screen object that was created using the resolution earlier.

   ```
    def draw(self, surface): pygame.draw.circle(surface, black, (self.x, self.y), self.radius)
   ```

3. **Make an instance of the class as well as to tell the game loop to draw the ball in every loop.**

   ```
   ball = Ball() while True: ... ball.draw(screen)
   ```

4. **Make the object move.** Create a function that will update the position of the object. Call this function in every game loop.

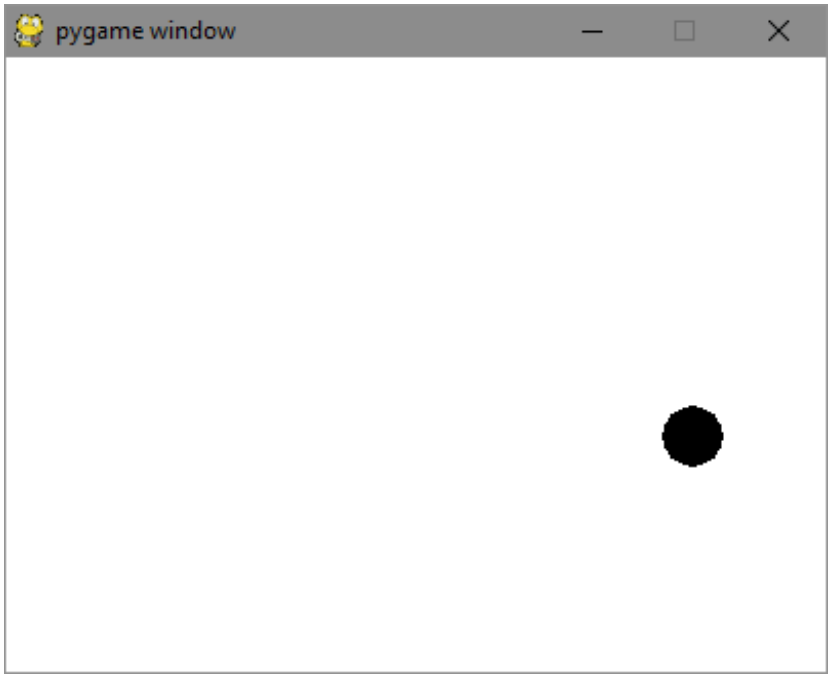
   ```
   class Ball: ... def update(self): self.x += self.dx self.y += self.dy
   ```

5. **Limit the frame rate.** The ball will move really fast because the game loop is running hundreds of times a second. Use Pygame's clock to limit the frame rate to 60 fps.

   ```
   clock = pygame.time.Clock() while True: ... clock.tick(60)
   ```

6. **Keep the ball on the screen.** Add checks in the update function to reverse the ball's direction if it hits one of the screen edges.

   ```
   class Ball: ... def update(self): ... if (self.x = 0 or self.x >= resolution[0]): self.dx *= -1 if (self.y = 0 or self.y >= resolution[1 ]): self.dy *= -1
   ```

7. 

Try it out! Here's what the code should look like now:

```
import pygame from pygame.locals import * resolution = (400,300) white
= (255,255,255) black = (0,0,0) red = (255,0,0) screen = pygame.display
.set_mode(resolution) class Ball: def __init__(self, xPos = resolution[
0] / 2, yPos = resolution[1] / 2, xVel = 1, yVel = 1, rad = 15): self.x
 = xPos self.y = yPos self.dx = xVel self.dy = yVel self.radius = rad
self.type = "ball" def draw(self, surface): pygame.draw.circle(surface,
 black, (self.x, self.y), self.radius) def update(self): self.x += self
.dx self.y += self.dy if (self.x = 0 or self.x >= resolution[0]): self.
dx *= -1 if (self.y = 0 or self.y >= resolution[1]): self.dy *= -1 ball
 = Ball() clock = pygame.time.Clock() while True: screen.fill(white)
ball.draw(screen) ball.update() pygame.display.flip() clock.tick(60)
for event in pygame.event.get(): if event.type == QUIT: pygame.quit()
```

Part 4 of 8:

Organizing the Game

1. **Use classes to organize everything.** The game is going to get more complicated. Use object-oriented techniques to organize your code.
2. **Make the game loop into a class.** Since our game now has data including your game objects and functions, it makes sense to turn your game loop into a class.

   ```
   class game():
   ```

3. **Add a constructor.** Here you will instantiate some game objects, create our screen and clock and initialize Pygame. Pygame needs to be initialized to use certain features like text or sound.

   ```
   class game(): def __init__(self): pygame.init() self.screen = pygame.
   display.set_mode(resolution) self.clock = pygame.time.Clock()
   ```

4. **Handle events in a function.**

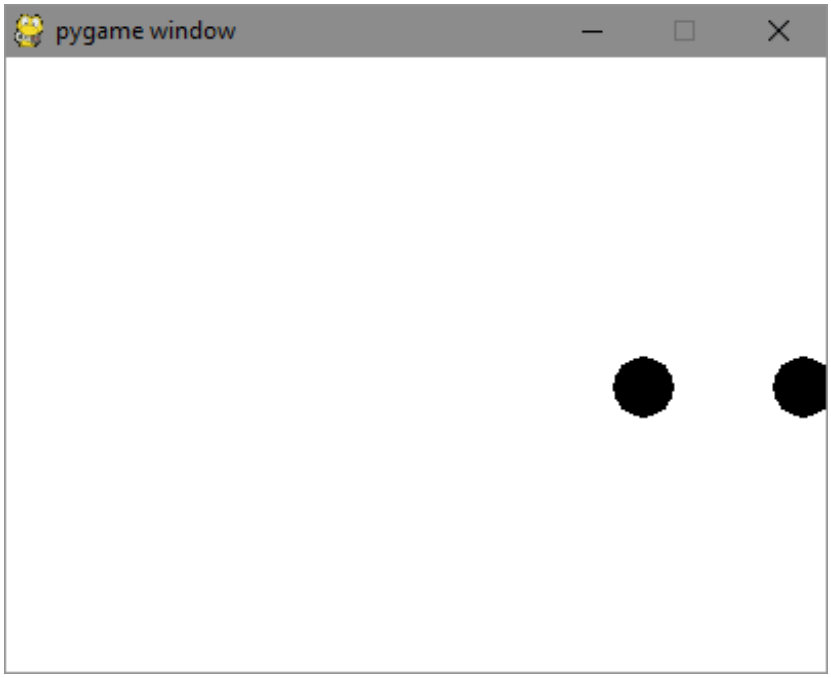
   ```
   class game(): ... def handleEvents(self): for event in pygame.event.get
   (): if event.type == QUIT: pygame.quit()
   ```

5. **Make the game loop a function.** Call the event handling function every loop.

   ```
   class game(): ... def run(self): while True: self.handleEvents() self.
   screen.fill(white) self.clock.tick(60) pygame.display.flip()
   ```

6. **Handle multiple game objects.** Right now this code has to call draw and update on our object each frame. This would get messy if you had a lot of objects. Let's add our object to an array and then update and draw all objects in the array every loop. Now you can easily add another object and give it a different starting position.

   ```
   class game(): def __init__(self): ... self.gameObjects = [] self.
   gameObjects.append(Ball()) self.gameObjects.append(Ball(100)) ... def
   run(self): while True: self.handleEvents() for gameObj in self.
   gameObjects: gameObj.update() self.screen.fill(white) for gameObj in
   self.gameObjects: gameObj.draw(self.screen) self.clock.tick(60) pygame.
   display.flip()
   ```

7. 

   Try it out! Here's what the code should look like now:

   ```
   import pygame from pygame.locals import * resolution = (400,300) white
   = (255,255,255) black = (0,0,0) red = (255,0,0) screen = pygame.display
   .set_mode(resolution) class Ball: def __init__(self, xPos = resolution[
   0] / 2, yPos = resolution[1] / 2, xVel = 1, yVel = 1, rad = 15): self.x
    = xPos self.y = yPos self.dx = xVel self.dy = yVel self.radius = rad
   self.type = "ball" def draw(self, surface): pygame.draw.circle(surface,
    black, (self.x, self.y), self.radius) def update(self): self.x += self
   .dx self.y += self.dy if (self.x = 0 or self.x >= resolution[0]): self.
   dx *= -1 if (self.y = 0 or self.y >= resolution[1]): self.dy *= -1
   ```

```
class game(): def __init__(self): pygame.init() self.screen = pygame.
display.set_mode(resolution) self.clock = pygame.time.Clock() self.
gameObjects = [] self.gameObjects.append(Ball()) self.gameObjects.
append(Ball(100)) def handleEvents(self): for event in pygame.event.get
(): if event.type == QUIT: pygame.quit() def run(self): while True:
self.handleEvents() for gameObj in self.gameObjects: gameObj.update()
self.screen.fill(white) for gameObj in self.gameObjects: gameObj.draw(
self.screen) self.clock.tick(60) pygame.display.flip() game().run()
```

Part 5 of 8:

Adding a Player Object

1. **Make a player class and constructor.** You're going to make another circle that is controlled by the mouse. Initialize the values in the constructor. The radius is the only important value.

   ```
   class Player: def __init__(self, rad = 20): self.x = 0 self.y = 0 self.
   radius = rad
   ```

2. **Define how to draw the player object.** It's going to be the same way you drew the other game objects.

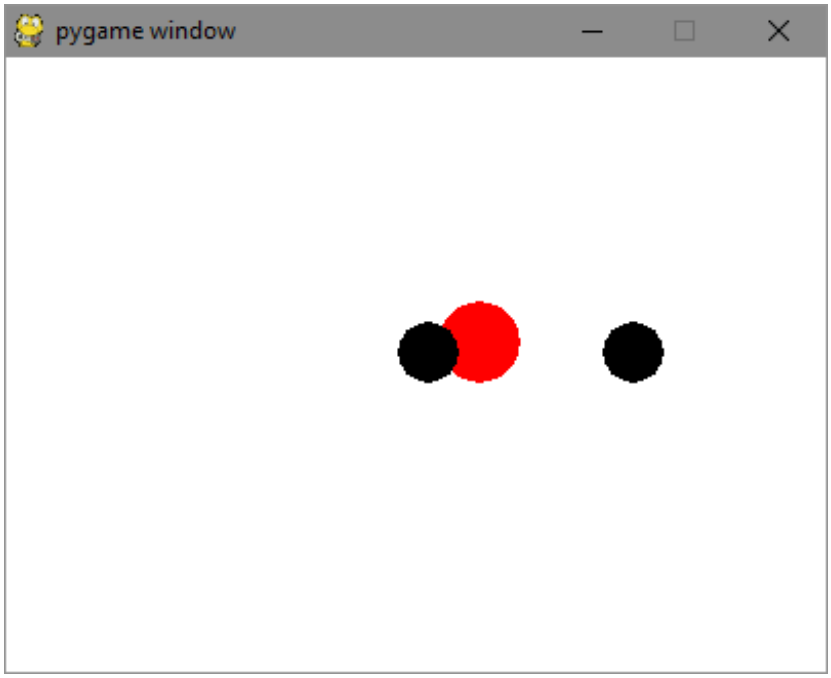
   ```
   class Player: ... def draw(self, surface): pygame.draw.circle(surface,
   red, (self.x, self.y), self.radius)
   ```

3. **Add mouse control for the player object.** In every frame, check the location of the mouse and set the players' objects' location to that point.

   ```
   class Player: ... def update(self): cord = pygame.mouse.get_pos() self.
   x = cord[0] self.y = cord[1]
   ```

4. **Add a player object to gameObjects.** Create a new player instance and add it to the list.

   ```
   class game(): def __init__(self): ... self.gameObjects.append(Player())
   ```

5. 

Try it out! Here's what the code should look like now:

```
import pygame from pygame.locals import * resolution = (400,300) white
= (255,255,255) black = (0,0,0) red = (255,0,0) screen = pygame.display
.set_mode(resolution) class Ball: def __init__(self, xPos = resolution[
0] / 2, yPos = resolution[1] / 2, xVel = 1, yVel = 1, rad = 15): self.x
 = xPos self.y = yPos self.dx = xVel self.dy = yVel self.radius = rad
self.type = "ball" def draw(self, surface): pygame.draw.circle(surface,
 black, (self.x, self.y), self.radius) def update(self): self.x += self
.dx self.y += self.dy if (self.x = 0 or self.x >= resolution[0]): self.
dx *= -1 if (self.y = 0 or self.y >= resolution[1]): self.dy *= -1
class Player: def __init__(self, rad = 20): self.x = 0 self.y = 0 self.
radius = rad self.type = "player" def draw(self, surface): pygame.draw.
circle(surface, red, (self.x, self.y), self.radius) def update(self):
cord = pygame.mouse.get_pos() self.x = cord[0] self.y = cord[1] class
game(): def __init__(self): pygame.init() self.screen = pygame.display.
set_mode(resolution) self.clock = pygame.time.Clock() self.gameObjects
= [] self.gameObjects.append(Player()) self.gameObjects.append(Ball())
self.gameObjects.append(Ball(100)) def handleEvents(self): for event in
 pygame.event.get(): if event.type == QUIT: pygame.quit() def run(self
): while True: self.handleEvents() for gameObj in self.gameObjects:
gameObj.update() self.screen.fill(white) for gameObj in self.
gameObjects: gameObj.draw(self.screen) self.clock.tick(60) pygame.
display.flip() game().run()
```

Part 6 of 8:

Making Objects Interact with the Player

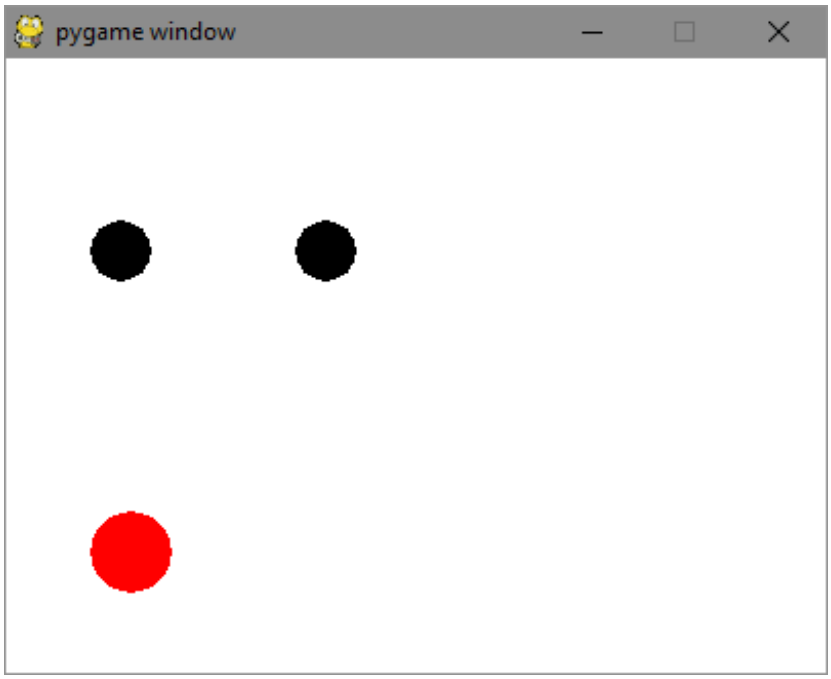
1. **Change the Update functions.** In order for objects to interact, they will need to have access to each other. Let's add another parameter to Update to pass in the gameObjects list. You'll have to add it to both the player object and the Ball objects. If you have a lot of game objects, inheritance could help you keep all

your method signatures the same.

```
class Ball: ... def update(self, gameObjects): ... class Player: ... def update(self, gameObjects):
```

2. **Check for collisions between the player and balls.** Go through all the game objects and check if the objects' type is ball. Then use the two objects' radii and the distance formula to check if they are colliding. Circles are really easy to check collisions on. This is the biggest reason that you didn't use some other shape for this game.

```
class Player: ... def update(self, gameObjects): ... for gameObj in gameObjects: if gameObj.type == "ball": if (gameObj.x - self.x)**2 + ( gameObj.y - self.y)**2 = (gameObj.radius + self.radius)**2:
```

3. 

End the game if the player gets "hit". Lets just quit the game for now.

```
if (gameObj.x - self.x)**2 + (gameObj.y - self.y)**2 = (gameObj.radius + self.radius)**2: pygame.quit()
```

4. **Try it out!** Here's What the code should look like now:

```
import pygame from pygame.locals import * resolution = (400, 300) white  = (255,255,255) black = (0,0,
```

You finished reading the article "**How to Program a Game in Python with Pygame**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and

© 2019 TipsMake.com