PyGame Tractor Pong Tutorial - Part 1

Contents

Time required: 30 minutes

Preview of the Game

Atari. - the year: 1973 - the date: - November 29th

That game is called Pong . . . . Then there was Tractor Pong.

Tractor Pong Demo Video

Imagine the year is 1973. November 29th. All the talk is about a new arcade game which has been released by Atari. That game is called Pong.

It might not seem much of a game by today's standards. It was a massive hit in its day.

Don't be deceived, although a simple game, Pong covers a wide range of aspects of computer game programming. There is movement, control, collision detection, scoring, artificial intelligence. It's all in there!

Being able to program Pong is a doorway to being able to program a lot of other games.

Watch out! Once you start playing Tractor Pong you might find less time to program, as it is quite addictive!

How Video Games Work

Games are like films. A film is a collection of images played quickly in sequence (usually 24 frames per second). A 90-minute file would be about 129,600 images. Your brain stitches together the images into continuous movement. The only difference between movies and games is how each image is created.

Games work in a similar way. We create a lot of frames per second. We try to have at least 30 frames per second, 60 is typical. The difference is that each image is dynamically generated as you play.

What is PyGame?

The PyGame library is the most well-known python library for making games. It's not the most advanced or high-level library. It is simple and easy to learn (comparatively). PyGame serves as a great entry point into the world of graphics and game development.

PyGame is a framework that includes several modules with functions for drawing graphics, playing sounds, handling mouse input, and other game like things. PyGame provides functions for creating programs with a **graphical user interface,** or **GUI** (pronounced, "gooey").

Install and Update PyGame with pip

pip is a package manager for **Python**. It's a tool that allows you to install and manage additional libraries and dependencies that are not distributed as part of the standard library.

We are going to use the community edition of PyGame. It is updated more frequently than the original PyGame.

Install PyGame:

```
pip install pygame-ce
```

If PyGame is already installed, you may want to update PyGame:

```
pip install pygame-ce -U
```

You are now ready to embark on your successful and prosperous game development career in Python!

Tutorial 1: Hello World

This code is a Python script using the PyGame library to create a window. That is the PyGame equivalent of the traditional Hello World program.

Create a Python program named: **tractor_pong_1.py**.

Header Comments:

```
"""
    Name: tractor_pong_1.py
    Author:
    Date:
    Purpose: Draw playing surface, Hello World!
"""
```

The header provides information about the script, such as its name, author, date, and purpose.

Import pygame:

```
# pip install pygame-ce
# Import pygame library
import pygame
# Import exit for a clean program shutdown
from sys import exit
```

This line imports the PyGame library, which is used for developing games and multimedia applications in Python.

Importing exit from the sys library to ensure a clean shutdown of the program.

Setup Constants:

```
# Initialize RGB color constant
COUGAR_GOLD = (249, 190, 0)

# Screen width and height
WIDTH = 525
HEIGHT = 700
```

Colors can be defined using RGB (Red, Green, Blue). These are the official Cougar Gold RGB values.

Setup constants for the width and height of the program window.

Initialize pygame:

```
16    # Initialize the pygame game engine
17    pygame.init()
```

Initializes the PyGame game engine. This must be done before using any PyGame functions.

Create game surface:

```
23    # Initialize game surface with width (x) and height (y) as a tuple
24    surface = pygame.display.set_mode((WIDTH, HEIGHT))
```

This line creates a PyGame display surface with a size of 525 pixels in width and 700 pixels in height. The size is specified using constants as a tuple **(WIDTH, HEIGHT)**. The game surface created by this function will be used to display graphics and handle user input throughout the game.

Set Window Caption:

```
27    # Set window caption
28    pygame.display.set_caption("Tractor Pong")
```

This sets the caption or title for the PyGame window to "Tractor Pong". It is the text that typically appears at the top of the window.

Clock Object:

```
27    # Setup computer clock object to control the speed of the game
28    clock = pygame.time.Clock()
```

This line creates a PyGame clock object (**clock**). The clock is used to control the frame rate of the game, ensuring that the game runs at a consistent speed across different systems. It is often used with the **tick()** method to regulate the frame rate.

Game Loop:

```
30  while True:
31      # Listen for all window events
32      for event in pygame.event.get():
33
34          # Closing the program causes the QUIT event to be fired
35          if event.type == pygame.QUIT:
36              # Quit Pygame
37              pygame.quit()
38              # Exit Python
39              exit()
```

This creates an infinite loop, commonly known as the game loop. It continuously repeats the code inside, managing the game's ongoing processes.

A game loop keeps the game running smoothly by handling input, updating the game state, rendering graphics, and repeating these steps in a loop. It allows games to react to user actions in real-time and maintain a coherent and engaging experience. The loop continues until the game is exited or a specific condition is met.

Surface Fill:

```
33  """Infinite game loop"""
34  while True:
35      # Listen for all program events
36      for event in pygame.event.get():
37
38          # Closing the program causes the QUIT event to be fired
39          if event.type == pygame.QUIT:
40              # Quit Pygame
41              pygame.quit()
42              # Exit Python
43              exit()
44
45      # Draw everything on the surface first
46      # Filling the surface with COUGAR_GOLD clears the previous frame
47      surface.fill(COUGAR_GOLD)
```

This code fills the entire surface of a PyGame window with the color "COUGAR_GOLD".

Update Display:

```
33  """Infinite game loop"""
34  while True:
35      # Listen for all program events
36      for event in pygame.event.get():
37
38          # Closing the program causes the QUIT event to be fired
39          if event.type == pygame.QUIT:
40              # Quit Pygame
41              pygame.quit()
42              # Exit Python
43              exit()
44
45      # Draw everything on the surface first
46      # Filling the surface with COUGAR_GOLD clears the previous frame
47      surface.fill(COUGAR_GOLD)
48
49      # Copying surface into video memory updates the game display
50      pygame.display.update()
51
```

Update the display to show the changes made in the current frame. This is essential for rendering the surface on the screen.

Backbuffer/surface

The term "backbuffer" refers to a secondary or off-screen buffer used during the rendering process. The backbuffer is closely related to the concept of double buffering, which is employed to prevent visual artifacts like flickering in animations.

Here's how it relates to pygame.display.update():

1. **Double Buffering:**
    a. In a double-buffering system, there are two buffers: the front buffer and the back buffer. The front buffer is the buffer currently being displayed on the screen, while the back buffer is where rendering operations occur without being immediately visible to the user.
2. **Rendering to the Back Buffer:**
    a. When you perform drawing operations using PyGame, you are often drawing to the back buffer. This includes drawing shapes, images, or any other visual elements that make up the current frame of your game or application.

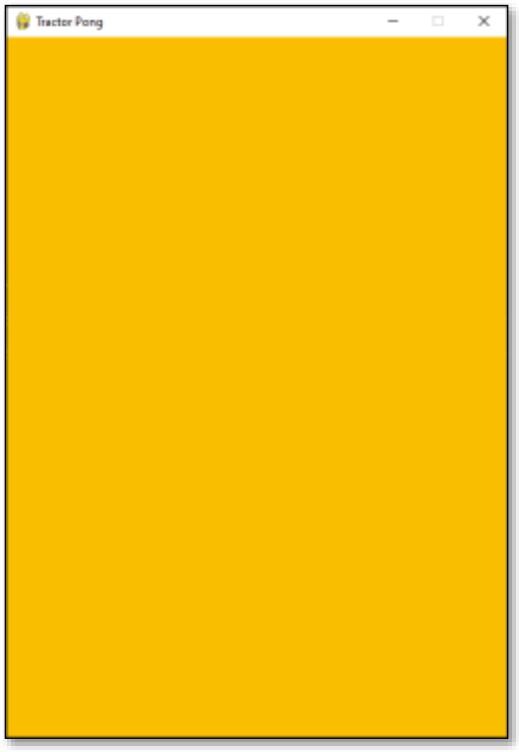
3. **Flipping Buffers:**
    a. After rendering the entire frame to the back buffer, the buffers are "flipped." This means the back buffer becomes the front buffer, and vice versa.
4. **pygame.display.update():**
    a. When you call **pygame.display.update()**, it tells PyGame to update the contents of the front buffer, making the changes from the back buffer visible on the screen. It's the point where the rendered frame becomes visible to the user.
5. **Preventing Flickering:**
    a. Double buffering helps prevent flickering issues that can occur when directly rendering to the front buffer. The back buffer provides a stable environment for drawing, and the switch between buffers is done quickly to minimize any visible artifacts.

Control Frame Rate:

```
33  """Infinite game loop"""
34  while True:
35      # Listen for all program events
36      for event in pygame.event.get():
37
38          # Closing the program causes the QUIT event to be fired
39          if event.type == pygame.QUIT:
40              # Quit Pygame
41              pygame.quit()
42              # Exit Python
43              exit()
44
45      # Draw everything on the surface first
46      # Filling the surface with COUGAR_GOLD clears the previous frame
47      surface.fill(COUGAR_GOLD)
48
49      # Copying surface into video memory updates the game display
50      pygame.display.update()
51
52      # Cap game speed at 60 frames per second
53      clock.tick(60)
```

Limits the frame rate to 60 frames per second (fps). It uses the clock object (**clock**) created earlier to control the speed of the game loop. This ensures that the game runs at a consistent frame rate, providing smooth animation.

Example run:



Yay! You've just made the world's most boring video game! It's just a blank window with Tractor Pong at the top of the window in the title bar.

Creating a window is the first step to making graphical games.

Run the program to confirm that it works.

Assignment Submission

1. Attach all tutorials and assignments.
2. Attach screenshots showing the successful operation of each tutorial program.
3. Submit in Blackboard.