PyGame Car Crash Tutorial - Part 1

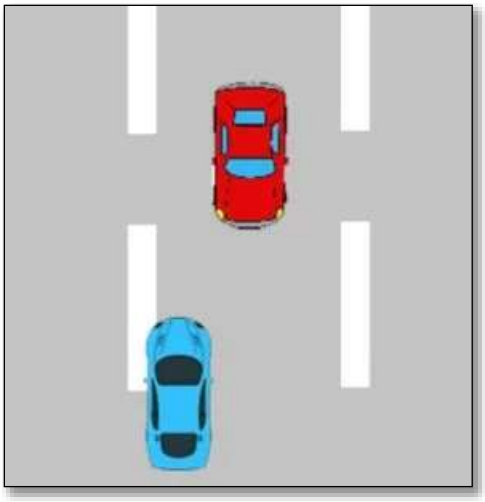
Contents

Time required: 90 minutes

Preview of the Game

Here's a sneak peak of the game that we are going to work on.

Car Crash Demo Video



Car Crash is simple arcade type game. The object is to move your blue car back and forth to avoid the oncoming red cars.

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 installed with Python. 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: **car_crash_1.py**.

Header Comments:

```
"""
    Name: car_crash_1.py
    Author:
    Date:
    Purpose: PyGame Hello World, an empty window
"""
```

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.

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

Setup Constants:

```
# Setup constants
WIDTH = 400
HEIGHT = 600
```

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

Initialize PyGame:

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

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

Create game surface:

```
21  # Create the game surface width (x) and height (y) as a tuple
22  surface = pygame.display.set_mode((WIDTH, HEIGHT))
```

This line creates a PyGame display surface with a size of 700 pixels in width and 500 pixels in height. The size is specified as a tuple **(700, 500)**.

Set Window Caption:

```
24  # Set window caption
25  pygame.display.set_caption("Car Crash")
```

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

Clock Object:

```
27  # Initialize 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.

```
30  """Infinite game loop"""
31  while True:
32      # Get the events from the event queue
33      # This is where the game waits for user input
34      for event in pygame.event.get():
35
36          # Closing the program causes the QUIT event to be fired
37          if event.type == pygame.QUIT:
38              # Quit Pygame
39              pygame.quit()
40              # Exit Python
41              exit()
42
43      # Copying the surface into video memory updates the game display
44      pygame.display.update()
```

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

Screen or Backbuffer

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 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:**

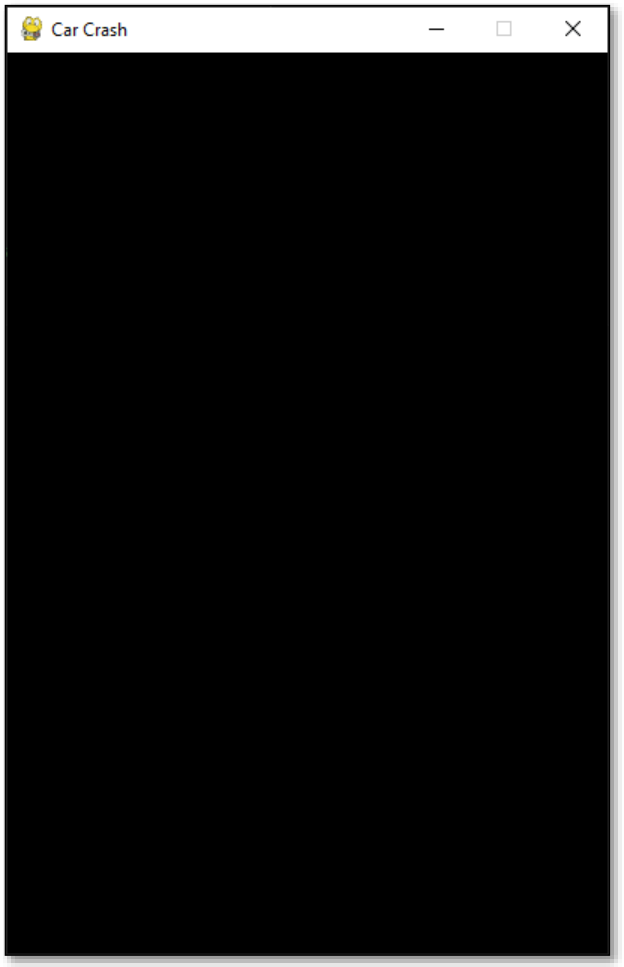
   a. Double buffering helps prevent flickering issues that can occur when directly rendering to the front buffer. The back buffer provides a stable environment for drawing, and the switch between buffers is done quickly to minimize any visible artifacts.

Control Frame Rate:

```
30  """Infinite game loop"""
31  while True:
32      # Get the events from the event queue
33      # This is where the game waits for user input
34      for event in pygame.event.get():
35
36          # Closing the program causes the QUIT event to be fired
37          if event.type == pygame.QUIT:
38              # Quit Pygame
39              pygame.quit()
40              # Exit Python
41              exit()
42
43      # Copying the surface into video memory updates the game display
44      pygame.display.update()
45
46      # Cap game speed at 60 frames per second
47      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 Car Crash at the top of the window in the title bar.

Creating a window and drawing an image is the first step to making graphical games.

Run the program to confirm that it works.

Assignment Submission

1. Attach a screenshot showing the operation of the program.
2. Zip up the program files folder and submit in Blackboard.