CLASSIC GAMES PROGRAMMING

WITH PYTHON & PYGAME

Nilo Ney Coutinho Menezes
nilo@pythonfromscratch.com

CodingDojo Belgium

November 2025

WELCOME TO CLASSIC GAMES PROGRAMMING!

Today we'll learn:

- How to create classic games with Python and PyGame
- Game programming fundamentals
- Code examples for iconic games

PREREQUISITES

- Basic Python knowledge
- Basic math
- Source code:

```
git clone https://github.com/lskbr/CodingDojoBelgium2025.git
```

WHAT IS GAME PROGRAMMING?

Game programming combines:

- **Logic** - Game rules and mechanics
- **Math** - Physics, collision detection, positioning
- **Art** - Graphics, animation, visual effects
- **Audio** - Sound effects and music
- **User Experience** - Controls, feedback, flow

ABOUT ME

- Nilo Ney Coutinho Menezes
- Amateur game programmer since 1984...
- Contact: questions@pythonfromscratch.com
- I wrote a Python book: Python from Scratch
- Web: https://pythonfromscratch.com

WHY START WITH CLASSIC GAMES?

- Simple mechanics, clear objectives
- Perfect for learning fundamentals
- Timeless concepts still used today

INTRODUCTION TO PYGAME

```python
import pygame

# Initialize PyGame
pygame.init()

# Create a window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("My Game")
```

```python
# Game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update game logic
    # Draw everything
    pygame.display.flip()

pygame.quit()
```

SETTING UP PYGAME

Installation:

```
pip install pygame-ce
```

You can also use uv or your preferred tool

Verify installation:

```python
import pygame
print(pygame.version.ver)  # Should show version number
```

THE GAME LOOP

Every game follows this pattern:

```python
def game_loop():
    while running:
        # 1. Handle events (input)
        handle_events()

        # 2. Update game state
        update_game()

        # 3. Render everything
        draw_everything()

        # 4. Control frame rate
        clock.tick(60)  # 60 FPS
```

Key concepts:

- **Events**: User input (keyboard, mouse)
- **Update**: Game logic, physics, AI
- **Render**: Draw sprites, backgrounds, UI
- **Frame Rate**: How many times per second the loop runs

COORDINATE SYSTEM

Screen coordinates:

- Origin (0,0) is top-left corner
- X increases to the right
- Y increases downward

```
# Screen: 800x600
# Top-left: (0, 0)
# Top-right: (800, 0)
# Bottom-left: (0, 600)
# Bottom-right: (800, 600)
# Center: (400, 300)
```

MOVING OBJECTS

```
# Move right
x += speed

# Move left
x -= speed

# Move down
y += speed

# Move up
y -= speed
```

SPRITES

Sprites are game objects (player, enemies, items):

```python
# Load an image
player_image = pygame.image.load("player.png")

# Create a sprite
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = player_image
        self.rect = self.image.get_rect()
        self.rect.x = 100
        self.rect.y = 100
```

SURFACES

Surfaces are like canvases for drawing:

```
# Create a surface
surface = pygame.Surface((100, 100))
surface.fill((255, 0, 0))  # Red color
```

COLLISION DETECTION

Rectangle collision:

```python
# Check if two rectangles overlap
if player.rect.colliderect(enemy.rect):
    print("Collision!")

# Check if point is inside rectangle
if player.rect.collidepoint(mouse_pos):
    print("Mouse over player!")
```

KEYBOARD INPUT

Getting keyboard state:

```python
keys = pygame.key.get_pressed()

if keys[pygame.K_LEFT]:
    player.x -= speed
if keys[pygame.K_RIGHT]:
    player.x += speed
if keys[pygame.K_SPACE]:
    player.jump()
```

KEY CONSTANTS

- Arrow keys: `pygame.K_LEFT`, `pygame.K_RIGHT`, `pygame.K_UP`, `pygame.K_DOWN`
- `pygame.K_SPACE`, `pygame.K_ENTER`, `pygame.K_ESCAPE`
- `pygame.K_a` through `pygame.K_z` for letters

FRAME RATE

Controlling speed:

```
clock = pygame.time.Clock()

# In game loop
dt = clock.tick(60) / 1000.0  # Delta time in seconds
player.x += speed * dt  # Frame-rate independent movement
```

TIMING

Timers:

```
# Create a timer event
pygame.time.set_timer(pygame.USEREVENT + 1, 1000)  # Every 1000ms

# Check for timer events
for event in pygame.event.get():
    if event.type == pygame.USEREVENT + 1:
        spawn_enemy()
```

PONG - GAME MECHANICS

Classic arcade game (1972)

- Two paddles, one ball
- Players hit ball back and forth
- Score when opponent misses
- Simple but engaging gameplay

PONG - CORE COMPONENTS

```
class Paddle:
    def __init__(self, x, y):
        self.rect = pygame.Rect(x, y, 15, 80)
        self.speed = 5

    def move_up(self):
        self.rect.y -= self.speed

    def move_down(self):
        self.rect.y += self.speed
```

```python
class Ball:
    def __init__(self, x, y):
        self.rect = pygame.Rect(x, y, 15, 15)
        self.speed_x = 3
        self.speed_y = 3

    def update(self):
        self.rect.x += self.speed_x
        self.rect.y += self.speed_y
```

PONG - PHYSICS AND COLLISION

```python
def update_ball(ball, left_paddle, right_paddle):
    # Move ball
    ball.rect.x += ball.speed_x
    ball.rect.y += ball.speed_y

    # Bounce off top/bottom walls
    if ball.rect.top <= 0 or ball.rect.bottom >= SCREEN_HEIGHT:
        ball.speed_y = -ball.speed_y
```

```python
# Bounce off paddles
if ball.rect.colliderect(left_paddle.rect):
    ball.speed_x = abs(ball.speed_x)  # Go right
if ball.rect.colliderect(right_paddle.rect):
    ball.speed_x = -abs(ball.speed_x)  # Go left

# Score points
if ball.rect.left <= 0:
    right_score += 1
if ball.rect.right >= SCREEN_WIDTH:
    left_score += 1
```

SNAKE - GAME MECHANICS

Classic mobile game

- Snake moves continuously
- Player controls direction
- Snake grows when eating food
- Game ends if snake hits wall or itself

Key concepts:

- Grid-based movement
- Snake body segments
- Food spawning
- Collision detection

SNAKE - GRID MOVEMENT

```python
class Snake:
    def __init__(self):
        self.body = [(10, 10)]  # List of (x, y) positions
        self.direction = (1, 0)  # Moving right
        self.grow = False
```

```
def move(self):
    head_x, head_y = self.body[0]
    new_head = (head_x + self.direction[0],
  head_y + self.direction[1])

    self.body.insert(0, new_head)

    if not self.grow:
        self.body.pop()  # Remove tail
    else:
        self.grow = False
    def change_direction(self, new_direction):
        # Prevent moving backwards into self
        if (new_direction[0] * -1,
            new_direction[1] * -1) != self.direction:
  self.direction = new_direction
```

SNAKE

GROWTH AND COLLISION

```python
def check_collisions(snake, food):
    head = snake.body[0]

    # Check wall collision
    if (head[0] < 0 or head[0] >= GRID_WIDTH or
        head[1] < 0 or head[1] >= GRID_HEIGHT):
        return "wall"

    # Check self collision
    if head in snake.body[1:]:
        return "self"
    # Check food collision
    if head == food.position:
        snake.grow = True
        food.spawn_new()
        return "food"

    return None

def draw_snake(screen, snake):
    for segment in snake.body:
        x = segment[0] * CELL_SIZE
        y = segment[1] * CELL_SIZE
        pygame.draw.rect(screen, GREEN,
          (x, y, CELL_SIZE, CELL_SIZE))
```

SPACE INVADERS - GAME MECHANICS

Classic arcade shooter (1978)

- Player controls ship at bottom
- Rows of aliens move down screen
- Player shoots aliens
- Aliens shoot back and move faster over time

Key components:

- Player ship with shooting
- Alien grid with movement patterns
- Bullets and collision detection
- Progressive difficulty

PAC-MAN - GAME MECHANICS

Classic maze game (1980)

- Player navigates maze eating dots
- Avoid ghosts that chase the player
- Power pellets make ghosts vulnerable
- Clear all dots to advance level

Key concepts:

- Maze navigation
- Ghost AI behavior
- Path finding algorithms
- Game states

COMMON PYGAME PATTERNS

Sprite Groups:

```
# Create groups for different types of sprites
all_sprites = pygame.sprite.Group()
enemies = pygame.sprite.Group()
bullets = pygame.sprite.Group()
# Add sprites to groups
all_sprites.add(player)
enemies.add(enemy1, enemy2)
bullets.add(bullet)
# Update all sprites
all_sprites.update()
# Draw all sprites
all_sprites.draw(screen)
```

State Machine:

```
class GameState:
    def __init__(self):
        self.state = "menu"

    def handle_event(self, event):
        if self.state == "menu":
            return self.handle_menu_event(event)
        elif self.state == "playing":
            return self.handle_game_event(event)

    def update(self):
        if self.state == "menu":
            self.update_menu()
        elif self.state == "playing":
            self.update_game()
```

DEBUGGING GAME CODE

Common debugging techniques:

```python
# Print debug information - avoid
print(f"Player position: {player.rect.x}, {player.rect.y}")
print(f"Velocity: {player.velocity_x}, {player.velocity_y}")
# Draw debug rectangles
pygame.draw.rect(screen, (255, 0, 0), player.rect, 2)
# Frame rate debugging
fps = clock.get_fps()
font = pygame.font.Font(None, 36)
text = font.render(f"FPS: {fps:.1f}", True, (255, 255, 255))
screen.blit(text, (10, 10))
# Pause game for debugging
if keys[pygame.K_p]:
    paused = not paused
```

Useful debugging tools:

- Visual debugging with colored rectangles
- Frame rate monitoring
- Pause functionality

ADDING SOUND AND MUSIC

```python
# Load sounds
jump_sound = pygame.mixer.Sound("jump.wav")
coin_sound = pygame.mixer.Sound("coin.wav")
background_music = pygame.mixer.music.load("background.mp3")

# Play sounds
def play_jump():
    jump_sound.play()

def play_coin():
    coin_sound.play()

# Play background music
pygame.mixer.music.play(-1)  # Loop forever
```

Sound file formats:

- `.wav` - Uncompressed, good for short sounds
- `.ogg` - Compressed, good for music
- `.mp3` - Compressed, widely supported

GAME ASSETS AND RESOURCES

Where to find free assets:

- **OpenGameArt.org** - Free game graphics and sounds
- **Freesound.org** - Free sound effects
- **Kenney.nl** - High-quality game assets
- **Itch.io** - Indie game assets
- **Pixabay** - Free images and sounds

Creating your own:

- **Graphics**: GIMP, Inkscape, Aseprite
- **Sounds**: Audacity, Bfxr
- **Music**: LMMS, MuseScore

NEXT STEPS AND PROJECT IDEAS

Beginner projects:

- Tic-tac-toe
- Memory card game
- Simple puzzle game

Intermediate projects:

- Basic platformer
- Racing game

Advanced projects:

- 3D game

Learning resources:

- PyGame documentation
- Game development tutorials
- Open source game projects
- Game development communities

ADDITIONAL LEARNING RESOURCES

Books:

- "Making Games with Python & Pygame" by Al Sweigart
- "Game Programming Patterns" by Robert Nystrom
- "The Art of Game Design" by Jesse Schell

Practice platforms:

- GitHub: Contribute to open source games
- Itch.io: Share your games
- Game jams: Participate in competitions

Q&A SESSION

Common questions:

- How do I make my game run smoothly?
- What's the best way to organize my code?
- How do I add multiplayer functionality?
- What are some common performance issues?

Tips

- Start small and build up
- Learn from existing games
- Don't be afraid to experiment
- Join the game development community
- Practice regularly

THANK YOU!

Contact information:

- Email: questions@pythonfromscratch.com
- GitHub: https://github.com/lskbr
- Book: https://pythonfromscratch.com

Keep coding and keep gaming!