Lab – Building an Accessible Car Game with Python

Learning Goals

In this lab, you will build a car-dodging game in Python, then improve it so that *everyone* can play — including people with disabilities. By the end, you will be able to:

1. Set up Python and Pygame and run a basic program
2. Build a game with a moving player, moving enemies, and collision detection
3. Identify accessibility problems using a checklist
4. Add four features that make your game playable for more people

No coding experience is required to start. Every step is explained from scratch.

All code goes in one file: `game.py`

Before You Begin

What You Need

- A computer with Python installed (ask your teacher if unsure)
- A code editor — VS Code, IDLE, or whatever your class uses
- Two image files: `Player.png` and `Enemy.png` (your teacher will provide these)
- A sound file: `crash.wav` (provided)
- A font file: `Atkinson-Hyperlegible-Regular-102.ttf` (provided, or download free at brailleinstitute.org/freefont)

Vocabulary

Word	What it means
Python	A programming language — it's the text you'll type to tell the computer what to do
Pygame	A free add-on for Python that makes it easy to build games with graphics and sound
FPS	Frames per second — how many times the screen redraws each second. Higher FPS = smoother and faster
Sprite	A game character or object (your car, the enemy car) shown as an image

Rect	Short for "rectangle" — an invisible box around your sprite used to track position and detect collisions
WCAG	Web Content Accessibility Guidelines — rules created by experts for making digital things usable by everyone
Contrast ratio	A number that says how easy two colors are to tell apart. Higher = easier to see
Accessibility	Designing things so people with different abilities can use them

---

Part I – Setting Up

Task 1 – Install Pygame

Open your terminal (on Mac: search "Terminal"; on Windows: search "Command Prompt") and type exactly:

```
pip install pygame
```

You should see text appear and eventually a line saying it was installed successfully. You only need to do this once.

> **What is pip?**
>
> `pip` is Python's package manager — it downloads and installs free add-ons (called packages) for Python. Pygame is a package.

Task 2 – Create your file and import tools

Create a new file called `game.py`. Add these lines at the very top:

```
import pygame, sys
from pygame.locals import *
import random, time

pygame.init()
```

> **What does this do?**
>
> `import pygame` loads the game-making tools. `pygame.init()` turns them on. Think of it like plugging in a game console (import) and pressing the power button (init). You need both steps every time.

Task 3 – Set up the game window

Add these lines below your imports:

```
# Speed settings
normal_mode = 60    # Normal: 60 frames per second
slow_mode   = 30    # Slow:   30 frames per second
FPS = normal_mode
FramePerSec = pygame.time.Clock()

# Colors (3 numbers: Red, Green, Blue — each from 0 to 255)
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED   = (255, 0, 0)

# Game settings
SCREEN_WIDTH  = 400
SCREEN_HEIGHT = 600
SPEED = 5
SCORE = 0

# Create the window
DISPLAYSURF = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
DISPLAYSURF.fill(WHITE)
pygame.display.set_caption("My Game")
```

> **Colors are just 3 numbers**
>
> Every color on screen mixes Red, Green, and Blue light. Each goes from 0 (none) to 255 (full). So `(255, 0, 0)` is pure red, `(0, 0, 0)` is black (no light at all), and `(255, 255, 255)` is white (all the light mixed together).

Part II – Building Your Characters

Task 4 – Create the Enemy class

In games, we use **classes** to create characters. Think of a class like a **cookie cutter**: it's a template, and every character made from it has the same shape and behavior.

Add this code below your setup:

```
class Enemy(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
```

```
        self.image = pygame.image.load("Enemy.png")
        self.rect  = self.image.get_rect()
        # Start at a random spot at the top of the screen
        self.rect.center = (random.randint(40, SCREEN_WIDTH-40), 0)

    def move(self):
        global SCORE
        # Move down the screen
        self.rect.move_ip(0, SPEED)
        # If the enemy goes off the bottom, bring it back to the top
        if self.rect.top > 600:
            SCORE += 1
            self.rect.top = 0
            self.rect.center = (random.randint(40, SCREEN_WIDTH - 40), 0)
```

> **What's happening here?**
>
> `__init__` runs once when a new enemy is created — it loads the image and picks a random starting spot. `move()` is called every frame to slide the enemy downward. When it exits the bottom, the score goes up and the enemy resets.

Task 5 – Create the Player class

Add this class below the Enemy class:

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("Player.png")
        self.rect  = self.image.get_rect()
        self.rect.center = (160, 520)    # Near the bottom of the screen

    def move(self):
        pressed_keys = pygame.key.get_pressed()
        # Move left (but don't go off screen)
        if self.rect.left > 0:
            if pressed_keys[K_LEFT]:
                self.rect.move_ip(-5, 0)
        # Move right (but don't go off screen)
        if self.rect.right < SCREEN_WIDTH:
            if pressed_keys[K_RIGHT]:
                self.rect.move_ip(5, 0)
```

> **What's a rect?**

> A rect (rectangle) is an invisible box around your character. The game uses it to know where the character is and to check whether two characters bumped into each other. You can't see it, but it's always there.

Part III – Making It Run

Task 6 – Set up characters and groups

Add this below your class definitions:

```
P1 = Player()    # Create one player
E1 = Enemy()     # Create one enemy

# Groups let us manage many sprites at once
enemies     = pygame.sprite.Group()
enemies.add(E1)
all_sprites = pygame.sprite.Group()
all_sprites.add(P1, E1)

# Make the game get harder over time (every 1 second)
INC_SPEED = pygame.USEREVENT + 1
pygame.time.set_timer(INC_SPEED, 1000)
```

Task 7 – Write the game loop

The game loop is the heartbeat of your game. It runs over and over — 60 times per second — doing three things: **check what happened → update everything → draw it on screen**.

Add this at the very bottom of your file:

```
while True:

    # 1. CHECK WHAT HAPPENED (keyboard, quit button, etc.)
    for event in pygame.event.get():
        if event.type == INC_SPEED:
            SPEED += 0.5               # Game gets faster every second!
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    # 2. CLEAR THE SCREEN AND MOVE EVERYTHING
    DISPLAYSURF.fill(WHITE)
    for entity in all_sprites:
        entity.move()
        DISPLAYSURF.blit(entity.image, entity.rect)

    # 3. CHECK FOR CRASHES
    if pygame.sprite.spritecollideany(P1, enemies):
        DISPLAYSURF.fill(RED)
        pygame.display.update()
```

```
        for entity in all_sprites:
            entity.kill()
        time.sleep(2)
        pygame.quit()
        sys.exit()

    # 4. SHOW THE FRAME AND WAIT
    pygame.display.update()
    FramePerSec.tick(FPS)
```

> **Run it!**
>
> Save your file and run it with `python game.py`. You should see your car at the bottom and an enemy falling from the top. Use left and right arrow keys to dodge! The game gets faster every second.

Part IV – Finding Problems (Accessibility Audit)

Task 8 – Spot the accessibility problems

Your game works — but it has serious problems that make it hard or impossible for some people to play:

Problem	Who it affects
Colors are hard to tell apart (low contrast)	People with low vision or color blindness
Score text is tiny and blends into the background	People with visual impairments
Only one speed — too fast for some players	People with motor or cognitive disabilities; beginners
No volume control for the crash sound	People with sensory sensitivities; people who are hard of hearing
Default font can be hard to read	People with dyslexia or low vision

Task 9 – Check the contrast using a tool

Go to **webaim.org/resources/contrastchecker** in a browser.

1. In "Foreground Color," type the hex code for your score text color
2. In "Background Color," type the hex code for your background

3. Look at the contrast ratio. For text, we need at least **4.5:1**. For game objects, at least **3:1**
4. If it says "Fail," try different colors until you find a pair that passes

> **Why does contrast matter?**
>
> About 1 in 12 men has color blindness. People with low vision or cataracts see colors as more washed-out. If your player car and the background are similar colors, some people literally cannot see your character on screen.

Task 10 – Complete the audit checklist

Play your game and mark each item below:

- □ All game objects are clearly visible against the background (3:1 contrast or better)
- □ The score text is easy to read (4.5:1 contrast or better)
- □ The game has a way to slow down
- □ The font is clear and readable
- □ Players can control the volume
- □ The game works using only the keyboard (no mouse needed)
- □ It's obvious when the game ends and what the score was

How many did you check off? The ones left unchecked are what you'll fix in Part V.

Part V – Fixing Your Game

Task 11 – Fix 1: High-Contrast Colors and Visible Score

Add your fonts at the top of the file (before the game loop), then update the drawing section inside the loop:

```
# Load the accessible font (add this before the game loop)
special     = "Atkinson-Hyperlegible-Regular-102.ttf"
font        = pygame.font.Font(special, 60)
font_medium = pygame.font.Font(special, 40)
font_small  = pygame.font.Font(special, 20)

# --- Inside the game loop, after DISPLAYSURF.blit(background, ...) ---
# Draw a bright yellow box, then put the score on top of it
scores = font.render(str(SCORE), True, BLACK)
pygame.draw.rect(DISPLAYSURF, (255, 200, 0), (0, 0, 60, 65))
DISPLAYSURF.blit(scores, (15, 15))
```

> **Why the yellow rectangle?**
>
> The yellow box goes behind the score number. No matter what color the road background is underneath, the score always appears on bright yellow — which gives very high contrast and is always readable.

Also update your background and sprite images to use high-contrast colors. Dark backgrounds with bright or light-colored sprites work well. Test your new colors at webaim.org/resources/contrastchecker.

Task 12 – Fix 2: Add a Slow Mode

Add one variable before the game loop, then add a spacebar check inside the event loop:

```
# Add this before the while loop:
mode_message_time = 0

# Add this inside the "for event" section:
if event.type == KEYDOWN:
    if event.key == K_SPACE:
        if FPS == normal_mode:
            FPS = slow_mode
        else:
            FPS = normal_mode
        mode_message_time = time.time()   # record when we toggled
```

```
# Add this after drawing the score (shows message for 2 seconds):
if time.time() - mode_message_time < 2:
    mode_text = "SLOW MODE" if FPS == slow_mode else "NORMAL MODE"
    mode_msg  = font_medium.render(mode_text, True, BLACK)
    rect      = mode_msg.get_rect(topleft=(110, 20))
    pygame.draw.rect(DISPLAYSURF, (255, 200, 0), rect)
    DISPLAYSURF.blit(mode_msg, rect)
```

> **How does slowing FPS slow the game?**
>
> At 60 FPS, the loop runs 60 times per second and moves things 60 times. At 30 FPS, everything moves only 30 times — literally half speed. The `time.time()` trick records when you pressed spacebar so the game can show a message for exactly 2 seconds afterward.

> **Who does slow mode help?**
>
> Players with motor disabilities who need more reaction time; players who process visual information more slowly; beginners practicing; children and older adults. Notice: slow mode helps *everyone* at some point — it's not only an "accessibility" feature, it's good game design.

Task 13 – Fix 3: Use an Accessible Font

If you haven't already done this in Task 11, add your font setup before the game loop:

```
special     = "Atkinson-Hyperlegible-Regular-102.ttf"
font        = pygame.font.Font(special, 60)
font_medium = pygame.font.Font(special, 40)
font_small  = pygame.font.Font(special, 20)
game_over   = font.render("Game Over", True, BLACK)
```

Make sure the font file is in the same folder as `game.py`.

> **Why Atkinson Hyperlegible?**
>
> This font was designed by the Braille Institute specifically for people with low vision. It makes characters that normally look confusing — like capital I, lowercase l, and the number 1 — look clearly different from each other. It also has generous spacing between letters.

> **Fonts to avoid:**
>
> pixelated fonts, cursive fonts, fonts with very thin strokes.

Task 14 – Fix 4: Let Players Control the Volume

Add a volume variable at the top of your file, then update the Player's `move()` method and the collision code:

```
# Add with your other variables at the top:
volume_level = 0.5    # Start at 50%

# Update the Player's move() method to include these lines:
def move(self):
    global volume_level
    pressed_keys = pygame.key.get_pressed()

    # Volume up (up arrow)
    if pressed_keys[K_UP] and volume_level < 1.0:
        volume_level = min(volume_level + 0.1, 1.0)
    # Volume down (down arrow)
    if pressed_keys[K_DOWN] and volume_level > 0.0:
        volume_level = max(volume_level - 0.1, 0.0)

    # Keep the left/right movement code below this...

# Update the collision section to play sound at the user's volume:
if pygame.sprite.spritecollideany(P1, enemies):
    crash = pygame.mixer.Sound('crash.wav')
    crash.set_volume(volume_level)
    crash.play()
    time.sleep(1)
    # ... rest of game over code ...
```

> **What do min() and max() do?**
>
> `min(volume_level + 0.1, 1.0)` picks whichever value is smaller — so if volume somehow goes above 1.0, it gets capped back. `max()` does the opposite for the bottom. This keeps volume safely between 0.0 and 1.0 no matter how many times the player presses the button.

Part VI – Check Your Work

Task 15 – Final checklist

Play your finished game and make sure everything works:

- □ The game launches and runs without errors
- □ The player moves left and right with arrow keys
- □ The score increases when you dodge an enemy
- □ The game ends and shows "Game Over" when you crash
- □ All game objects and text are easy to see (high contrast)
- □ Pressing Spacebar switches between Normal and Slow mode
- □ The mode name appears on screen for 2 seconds after switching
- □ Up/Down arrows raise or lower the crash sound volume
- □ The font is clear and readable (Atkinson Hyperlegible)

> **You're done! Here's what you built:**
>
> - A working car-dodging game in Python
> - High-contrast visuals that are easy to see for people with low vision or color blindness
> - A slow mode that helps players who need more reaction time
> - An accessible font designed for low-vision readers
> - User-controlled volume for players with sensory sensitivities or hearing differences

---

Stretch Goal – Advanced Options

If you've finished everything above and want to go further, try one of these:

Stretch Task A – Color-Blind Mode with Shapes

Some players can't distinguish colors even with high contrast. Add a mode that puts a unique shape on each type of object (e.g., a circle on the player, a triangle on enemies) so players can tell them apart by shape, not just color.

```
# Example: draw a circle on top of the player sprite
pygame.draw.circle(DISPLAYSURF, WHITE, P1.rect.center, 10)
```

Research: Look up `pygame.draw.circle`, `pygame.draw.polygon`, and `pygame.draw.rect` in the Pygame docs (pygame.org/docs) to find the drawing functions you need.

Stretch Task B – Screen Reader Announcements

People who are blind use screen readers to hear what's on the screen. The `pyttsx3` library lets your game speak text out loud.

```
pip install pyttsx3
```

```
import pyttsx3
engine = pyttsx3.init()

# Announce the score when it changes:
engine.say(f"Score: {SCORE}")
engine.runAndWait()

# Announce game over:
engine.say("Game over")
engine.runAndWait()
```

Challenge: only announce the score when it changes (not every frame), and test that the announcement timing doesn't interrupt gameplay.

Stretch Task C – Remappable Controls

Not everyone can use the standard arrow keys. Add a setup screen at the beginning that lets players press any key to assign it as their "move left" or "move right" button.

```
# Example structure:
print("Press the key you want to use for MOVE LEFT:")
# Wait for a keypress, store it
left_key = wait_for_keypress()

print("Press the key you want to use for MOVE RIGHT:")
right_key = wait_for_keypress()
```

Research: Look up `pygame.event.wait()` and `event.key` to figure out how to capture a keypress and store it as a variable.

> **The big takeaway**
>
> Accessibility isn't about making a separate "special" version of your game. It's about making one game that works for as many people as possible. Every feature you added — slow mode, volume control, readable font, high contrast — makes the game better for all players, not just disabled ones. That's what good design looks like.

Resources

- **WebAIM Contrast Checker:** webaim.org/resources/contrastchecker
- **Atkinson Hyperlegible Font (free):** brailleinstitute.org/freefont
- **Pygame Documentation:** pygame.org/docs
- **WCAG Quick Reference:** w3.org/WAI/WCAG21/quickref

Created by Ava Sklar · Bucknell University · 2026