Lab – Teaching Accessible Pygame: Instructor Guide

Learning Goals

This guide supports CS instructors in teaching accessibility principles through game development. After using this module, instructors will be able to:

1. Lead students through building a working Pygame car-dodging game over 3–4 class sessions
2. Facilitate a structured accessibility audit using WCAG AA criteria
3. Guide students to implement four concrete accessibility features in their game
4. Assess student work using the provided rubric and lead reflection discussions

No prior accessibility background is required. WCAG references are provided in context throughout the guide.

Accessibility Context for Instructors

The **Web Content Accessibility Guidelines (WCAG) 2.1 AA** are the internationally recognized standard for digital accessibility, developed by the W3C Web Accessibility Initiative. While designed for web content, the four core principles — Perceivable, Operable, Understandable, Robust (POUR) — apply directly to games.

This module focuses on four WCAG-grounded features that are both teachable in introductory CS and genuinely impactful:

Feature	WCAG Criterion	Who it helps
High-contrast colors	1.4.3 (text, 4.5:1), 1.4.11 (objects, 3:1)	Low vision, color blindness
Adjustable speed (slow mode)	2.2.1 Timing Adjustable	Motor disabilities, cognitive disabilities, newcomers
Readable font (Atkinson Hyperlegible)	1.4.12 Text Spacing	Dyslexia, low vision
User-controlled volume	1.4.2 Audio Control	Sensory sensitivities (e.g., autism), hearing differences

> **Instructor Note: Universal Design Framing**

> Every accessibility feature in this lab also benefits non-disabled users: slow mode helps beginners, volume control helps anyone in a noisy environment, high contrast helps in bright lighting. Frame accessibility as good design, not accommodation.

Prerequisites & Setup

Student Prerequisites

- Basic Python: variables, functions, loops, conditionals
- Familiarity with classes is helpful but not required (Lesson 2 introduces OOP gently)
- No prior Pygame or game development experience needed

Environment Setup

Students install Pygame with:

```
pip install pygame
```

Required Assets

- `Player.png` and `Enemy.png` — provided in the starter kit, or students create colored rectangles
- `crash.wav` — a short sound file for collision feedback
- `Atkinson-Hyperlegible-Regular-102.ttf` — downloadable free from braille institute.org/freefont
- A road background image (optional; a solid dark rectangle works fine)

> **Tip: Distribute a starter kit**
>
> Consider providing a zip file with the base (inaccessible) game already running and all assets included. This focuses Lessons 3–4 on the audit and fixes rather than debugging asset paths.

Module Structure

Session	Topic	Key Concepts
Lesson 1	Pygame Foundations	Game loop, display surface, FPS, RGB colors, coordinate system

Lesson 2	Building the Game	OOP with sprites, collision detection, custom events, scoring
Lesson 3	Accessibility Audit	WCAG POUR principles, contrast ratios, sensory and cognitive needs
Lesson 4	Implementing Fixes	High contrast, speed modes, readable fonts, volume control

---

Part I – Lesson 1: Pygame Foundations & the Game Loop

Suggested Timeline (50 min)

0–10 min	**Warm-up:** Ask students to name a game they enjoy. "What would happen if you couldn't see the colors clearly? What if the game was too fast for you to react?" Record responses — they will surface naturally as fixes in Lessons 3–4.
10–20 min	**Lecture:** Introduce Pygame — importing, initializing, the display surface, RGB color model.
20–35 min	**Live code:** Build the game loop together. Explain FPS, demonstrate the coordinate system (origin top-left, y increases downward).
35–45 min	**Hands-on:** Students create a window, set a background color, display a rectangle that moves.
45–50 min	**Wrap-up:** Preview the OOP approach for Lesson 2.

Task 1 – Initializing Pygame

Have students create `game.py` and type these lines to start the engine:

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

pygame.init()
```

Explain: `pygame.init()` starts all subsystems (display, sound, etc.) and must come before any other Pygame call. The `pygame.locals` import provides constants like `QUIT`, `K_LEFT`, `K_SPACE`.

Task 2 – The Game Window and FPS

```
normal_mode = 60    # Standard speed: 60 frames per second
slow_mode   = 30    # Half speed: 30 frames per second
FPS = normal_mode
FramePerSec = pygame.time.Clock()

SCREEN_WIDTH  = 400
SCREEN_HEIGHT = 600
SPEED = 5
SCORE = 0

DISPLAYSURF = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("Accessible Car Game")
```

Task 3 – The Game Loop

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

    # Game logic here

    pygame.display.update()
    FramePerSec.tick(FPS)
```

The three phases of every game loop: **Handle events → Update state → Render**. `FramePerSec.tick(FPS)` limits execution to FPS iterations per second, giving consistent speed across machines.

> **Discussion Prompt**
>
> "What would happen if we removed the `tick(FPS)` line? Why might different players need different speeds?" This seeds the slow mode conversation for Lesson 4.

Part II – Lesson 2: Building the Game — Sprites, Classes & Collision

Suggested Timeline (50 min)

0–10 min	**Review:** Quick recap of the game loop from Lesson 1.
10–25 min	**Lecture + live code:** Introduce OOP through the Player and Enemy classes. Frame classes as "blueprints."

25–40 min	**Hands-on:** Students implement sprite groups, collision detection, and the scoring system.
40–50 min	**Demo & discussion:** Play the completed (inaccessible) game. Ask students to identify barriers.

Task 4 – The Enemy Class

```
class Enemy(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("Enemy.png")
        self.rect  = self.image.get_rect()
        self.rect.center = (random.randint(40, SCREEN_WIDTH-40), 0)

    def move(self):
        global SCORE
        self.rect.move_ip(0, SPEED)
        if self.rect.top > 600:
            SCORE += 1
            self.rect.top = 0
            self.rect.center = (random.randint(40, SCREEN_WIDTH - 40), 0)
```

> **Teaching Note: OOP for Beginners**
>
> Frame it as a "blueprint": the Enemy class is the template; `E1 = Enemy()` creates one actual enemy from it. `self` means "this specific enemy." `__init__` runs once when the enemy is created. `move()` is an action the enemy can perform.

Task 5 – The Player 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)

    def move(self):
        pressed_keys = pygame.key.get_pressed()
        if self.rect.left > 0:
            if pressed_keys[K_LEFT]:
                self.rect.move_ip(-5, 0)
        if self.rect.right < SCREEN_WIDTH:
```

```
            if pressed_keys[K_RIGHT]:
                self.rect.move_ip(5, 0)
```

Task 6 – Sprite Groups, Collision Detection, and Increasing Difficulty

```
P1 = Player()
E1 = Enemy()

enemies     = pygame.sprite.Group()
enemies.add(E1)
all_sprites = pygame.sprite.Group()
all_sprites.add(P1, E1)

INC_SPEED = pygame.USEREVENT + 1
pygame.time.set_timer(INC_SPEED, 1000)   # fire every 1 second

# --- Inside the game loop ---
for event in pygame.event.get():
    if event.type == INC_SPEED:
        SPEED += 0.5

for entity in all_sprites:
    entity.move()
    DISPLAYSURF.blit(entity.image, entity.rect)

if pygame.sprite.spritecollideany(P1, enemies):
    # collision = game over
```

> **End-of-Lesson Discussion**
>
> Have students play the inaccessible game, then discuss: "Can you read the score easily? What if you were color blind — can you tell the player from the background? What if the game moves too fast to react? What if the crash sound is too loud?" Record their answers. These become the fix list for Lessons 3–4.

Part III – Lesson 3: Accessibility Audit — Applying WCAG to Games

Suggested Timeline (50 min)

0–15 min	**Lecture:** Introduction to WCAG AA. What is accessibility? Who benefits? The POUR framework.
15–30 min	**Activity:** Students audit the game using the WCAG checklist below.
30–40 min	**Group work:** Teams present findings and propose solutions.
40–50 min	**Planning:** Each student creates a prioritized fix list for Lesson 4.

Task 7 – Introduction to WCAG for Games

Present the four WCAG principles and their game equivalents:

WCAG Principle	Game Application
Perceivable	Can players see/hear all important game elements? Are colors distinguishable without relying on hue alone?
Operable	Can players control the game effectively? Is the speed manageable for users with different reaction times?
Understandable	Is text readable? Are game states (score, game over, mode changes) clearly communicated?
Robust	Does the game work with different input methods and configurations?

Task 8 – Contrast Audit Activity

Have students use WebAIM's Contrast Checker (webaim.org/resources/contrastchecker) to evaluate the original game colors. Example failures in the inaccessible version:

- Grey road (#808080) vs. white lane lines (#FFFFFF) → **1.86:1** (fails 3:1 minimum for objects)
- Grey road vs. yellow lane lines (#FFFF00) → **1.49:1** (fails)
- Grey road vs. blue car (#0000FF) → **1.07:1** (fails — nearly invisible)

Students then find accessible alternatives that pass 3:1 for game objects and 4.5:1 for text.

Task 9 – Full WCAG Audit Checklist

Category	Audit Question	WCAG Criterion
Contrast	Do all game objects meet at least 3:1 contrast against the background?	1.4.11
Contrast	Does all score/UI text meet at least 4.5:1 contrast?	1.4.3
Speed	Can players adjust game speed to suit their needs?	2.2.1
Font	Is the font readable with distinct characters (I vs. l vs. 1)?	1.4.12
Sound	Can users independently control audio volume?	1.4.2
Motion	Are there flashing elements faster than 3 per second?	2.3.1
Input	Is the entire game playable with keyboard only?	2.1.1
Feedback	Are game states (score, game over, mode change) clearly communicated?	3.3.1

Part IV – Lesson 4: Implementing Accessible Features

Suggested Timeline (50 min)

0–5 min	**Review:** Share audit findings. Prioritize the fix list.
5–40 min	**Hands-on coding:** Students implement fixes using the guided implementations below.
40–50 min	**Showcase:** Side-by-side before/after comparison. Reflection discussion.

Task 10 – Fix 1: High-Contrast Colors and Score Display

Before: Grey background, score text blending into the road, low-contrast car colors.

After: Dark background, bold score on a yellow rectangle, high-contrast sprites.

```
DISPLAYSURF.blit(background, (0, 0))

# Yellow backing box guarantees contrast regardless of background image
scores = font.render(str(SCORE), True, BLACK)
pygame.draw.rect(DISPLAYSURF, (255, 200, 0), (0, 0, 60, 65))
DISPLAYSURF.blit(scores, (15, 15))
```

The yellow rectangle behind the score provides a guaranteed high-contrast surface that is independent of whatever background image is rendered beneath it.

Task 11 – Fix 2: Adjustable Speed (Slow Mode)

```
# Before the game loop:
mode_message_time = 0

# Inside the event loop:
if event.type == KEYDOWN:
    if event.key == K_SPACE:
        FPS = slow_mode if FPS == normal_mode else normal_mode
        mode_message_time = time.time()

# After drawing the score (show for 2 seconds after toggle):
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)
```

> **Discussion: Who Benefits from Slow Mode?**
>
> Players with motor disabilities who need more reaction time; players with cognitive disabilities who process visual information more slowly; new players learning the game; children and elderly players. Key insight: accessible features benefit everyone, not only disabled users.

Task 12 – Fix 3: Accessible Font (Atkinson Hyperlegible)

```
# Download from brailleinstitute.org/freefont, place .ttf in game folder
special     = "Atkinson-Hyperlegible-Regular-102.ttf"

font        = pygame.font.Font(special, 60)   # Large: "Game Over"
font_medium = pygame.font.Font(special, 40)   # Medium: mode display
font_small  = pygame.font.Font(special, 20)   # Small: labels
game_over   = font.render("Game Over", True, BLACK)
```

Atkinson Hyperlegible was designed by the Braille Institute for low-vision readers. It provides distinct letterforms (e.g., clear difference between capital I, lowercase l, and numeral 1), generous spacing, and open counters.

> **Teaching Note: Font Choice Criteria**
>
> Avoid pixelated/bitmap fonts, cursive or script fonts, and fonts with thin strokes. Recommended accessible fonts: Atkinson Hyperlegible, Verdana, Arial. WCAG 1.4.12

> addresses text spacing and readability.

Task 13 – Fix 4: Volume Control

```
# Top of file, with other variables:
volume_level = 0.5    # Default 50%

# In Player.move() — up/down arrows adjust volume:
def move(self):
    global volume_level
    pressed_keys = pygame.key.get_pressed()

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

    # Left/right movement unchanged...

# On collision:
crash = pygame.mixer.Sound('crash.wav')
crash.set_volume(volume_level)
crash.play()
```

People with sensory sensitivities (common in autism spectrum conditions) may find sudden loud sounds distressing. People who are hard of hearing may need higher volume. User-controlled audio is both a WCAG requirement and a UX best practice.

Part V – Assessment & Extension Activities

Assessment Rubric

Criterion	Excellent (A)	Proficient (B)	Developing (C)
Working game	Runs without errors; all features functional	Runs with minor issues	Significant bugs prevent play
Contrast fixes	All elements pass WCAG AA ratios; documented contrast checks	Most elements pass	Some improvements made
Speed mode	Toggle works; mode shown on screen; smooth transition	Toggle works	Attempted but buggy

Font & readability	Accessible font; appropriate sizing; high-contrast text	Font changed; sizing adequate	Default font unchanged
Volume control	User-controllable; clamped to [0.0, 1.0]; applied at play time	Volume control works	Sound plays but no control
Written reflection	Articulates who each fix helps and why; references WCAG	Basic reflection with examples	Minimal or generic reflection

Extension Activities

- **Screen reader support:** Add text-to-speech score announcements using the `pyttsx3` library
- **Color-blind mode:** Implement shape/pattern differentiation so game objects are distinguishable without relying on color alone
- **Remappable controls:** Let players assign their own key bindings at startup
- **Audit another game:** Have students apply the WCAG checklist to a classmate's project or a commercial game
- **Research project:** Compare WCAG to the Game Accessibility Guidelines (gameaccessibilityguidelines.com)

Resources & References

- **WCAG 2.1 AA Quick Reference:** w3.org/WAI/WCAG21/quickref
- **WebAIM Contrast Checker:** webaim.org/resources/contrastchecker
- **Atkinson Hyperlegible Font:** brailleinstitute.org/freefont
- **Game Accessibility Guidelines:** gameaccessibilityguidelines.com
- **Pygame Documentation:** pygame.org/docs
- **W3C WAI Introduction to WCAG:** w3.org/WAI/standards-guidelines/wcag

Created by Ava Sklar · Bucknell University · 2026