

General strategy: I first asked Gemini Pro make a very readable version of Froggy, and then modify the code to be more like Crossy Roads. I played each version of the game, and provided it feedback to add in features and fix bugs that I was noticing, and improve graphics to make it look more like Crossy Roads.

what is the best gaming engine for python and what is a gaming engine



To dive into game development with Python, it's helpful to first understand what's happening "under the hood" of your favorite titles.

---

## What is a Game Engine?

Think of a game engine as a software construction kit. Instead of writing code from scratch to handle how light reflects off a surface, how gravity pulls a character down, or how a sound echoes in a hallway, the engine provides those tools for you.

A game engine typically handles these core systems:

- Rendering: Drawing 2D or 3D graphics.
  - Physics: Calculating collisions, gravity, and friction.
  - Input: Detecting when you press a key or move a mouse.
  - Audio: Playing music and sound effects.
  - Scripting: This is where you come in—writing the "brain" of the game (the logic).
- 

## The Best Python Gaming Engines

While Python isn't usually used for "triple-A" blockbusters (like *Call of Duty*) because it's slower than C++, it is fantastic for indie games, prototypes, and learning. Here are the top contenders:

### 1. Pygame (The Classic)

Pygame is the most famous library for Python game development. It's strictly 2D and very "bare-bones," which is actually a blessing if you want to learn the fundamentals of how games work.

- Best for: Beginners and 2D retro-style games.
- Pros: Massive community, easy to learn, very lightweight.

### 2. Ursina (The Modern 3D Pick)

If you want to make something that looks like *Minecraft* or a modern 3D platformer, Ursina is a powerhouse. It's built on top of a more complex engine called Panda3D but makes everything much simpler.

- Best for: Rapid 3D prototyping.
- Pros: You can get a 3D world running in just a few lines of code.

### 3. Godot (The Professional Path)

Godot is a full-featured, professional-grade engine. While its native language is GDScript (which looks and acts almost exactly like Python), it has excellent community support for "Pythonic" development.

- Best for: Serious developers wanting to publish to Steam or Consoles.
- Pros: Dedicated visual editor, handles complex 3D and 2D equally well.

## Comparison at a Glance

Feature	Pygame	Ursina	Godot
Dimension	2D	3D (and 2D)	Both
Ease of Use	High	High	Medium
Performance	Low	Medium	High
Visual Editor	No	No	Yes

Would you like me to help you write a simple "Hello World" script for Pygame to see how it feels?

You Asked 

make a readable version of the classic frogger game in pygame. have separate files if necessary to make the code easy to understand. have in-line comments explaining the more complicated parts of the code



Gemini 3 Pro

This is a great project to understand how object-oriented programming works in games. Because Frogger relies heavily on distinct "objects" (the frog, the cars, the logs), separating the code into files makes it much easier to manage.

We will create three files:

1. settings.py : Stores all the colors, screen sizes, and game constants.

2. `sprites.py` : Contains the classes for the Frog, Cars, and Logs.

3. `main.py` : The game loop that runs everything.

### File 1: `settings.py`

This file acts as a control panel. You can tweak speeds and colors here without breaking the game logic.

```
python
```

```
# settings.py

# Screen dimensions
WIDTH = 600
HEIGHT = 600
FPS = 60

# Grid setup (The game moves in a grid-like pattern)
TILE_SIZE = 40
GRID_WIDTH = WIDTH // TILE_SIZE
GRID_HEIGHT = HEIGHT // TILE_SIZE

# Colors (R, G, B)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (34, 139, 34)    # Frog color
GRAY = (128, 128, 128)   # Road color
BLUE = (65, 105, 225)    # River color
RED = (200, 50, 50)      # Car color
BROWN = (139, 69, 19)    # Log color

# Game Areas (Y-coordinates for where road and river start)
# We calculate these based on tile size so they fit the grid
RIVER_TOP = 2 * TILE_SIZE
RIVER_BOTTOM = 7 * TILE_SIZE
ROAD_TOP = 8 * TILE_SIZE
ROAD_BOTTOM = 13 * TILE_SIZE
```

### File 2: `sprites.py`

Here we define the objects. We use `pygame.sprite.Sprite`, which is a built-in Pygame class that makes handling movement and collisions much easier.

python

```
# sprites.py
import pygame
from settings import *

class Frog(pygame.sprite.Sprite):
    def __init__(self, x, y):
        # Initialize the parent Sprite class
        super().__init__()

        # Create the visual representation of the frog (a green square)
        self.image = pygame.Surface((TILE_SIZE - 4, TILE_SIZE - 4))
        self.image.fill(GREEN)

        # The 'rect' is used for positioning and collision detection
        self.rect = self.image.get_rect()
        self.start_pos = (x, y)
        self.rect.topleft = self.start_pos

        self.on_log = False

    def move(self, dx, dy):
        """Handles grid-based movement"""
        # Calculate new position
        new_x = self.rect.x + (dx * TILE_SIZE)
        new_y = self.rect.y + (dy * TILE_SIZE)

        # Boundary checks: Keep frog inside the screen
        if 0 <= new_x < WIDTH and 0 <= new_y < HEIGHT:
            self.rect.x = new_x
            self.rect.y = new_y

    def reset_position(self):
        """Sends frog back to start"""
        self.rect.topleft = self.start_pos

    def update(self):
        # We don't need constant updates here because movement is event-base
        # but we can use this to clamp the frog to the screen if it rides a log off-screen.
        if self.rect.left < 0:
            self.rect.left = 0
        if self.rect.right > WIDTH:
            self.rect.right = WIDTH
```

```

class Obstacle(pygame.sprite.Sprite):
    """Used for both Cars and Logs"""
    def __init__(self, x, y, width, color, speed):
        super().__init__()
        self.image = pygame.Surface((width, TILE_SIZE - 4))
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)
        self.speed = speed

    def update(self):
        """Move the obstacle and wrap around the screen"""
        self.rect.x += self.speed

        # If moving right and goes off screen, move to left side
        if self.speed > 0 and self.rect.left > WIDTH:
            self.rect.right = 0

        # If moving left and goes off screen, move to right side
        if self.speed < 0 and self.rect.right < 0:
            self.rect.left = WIDTH

```

### File 3: main.py

This runs the game loop. It handles the "Log Logic" (moving the frog if he is on a log) and "Water Logic" (killing the frog if he isn't).

python

```

# main.py
import pygame
import sys
from settings import *
from sprites import Frog, Obstacle

# 1. Initialize Pygame
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Python Frogger")
clock = pygame.time.Clock()

# 2. Group Setup
# Groups help us update and draw many objects at once
all_sprites = pygame.sprite.Group()
cars = pygame.sprite.Group()
logs = pygame.sprite.Group()

```



```

# 3. Create Objects

# Create the Player player = Frog(WIDTH // 2 - TILE_SIZE //
2, HEIGHT - TILE_SIZE) all_sprites.add(player)

# Create Cars (Rows 8 to 12)
# We loop through rows and create cars with alternating
speeds for row in range(8, 13):      y_pos = row * TILE_SIZE
    # Alternating direction based on even/odd row
speed = 3 if row % 2 == 0 else -3

    # Create 2 cars per lane      car1 = Obstacle(0, y_pos,
TILE_SIZE * 2, RED, speed)      car2 = Obstacle(WIDTH // 2,
y_pos, TILE_SIZE * 2, RED, speed)
        cars.add(car1, car2)
all_sprites.add(car1, car2)

# Create Logs (Rows 2 to 6)
for row in range(2, 7):
    y_pos = row * TILE_SIZE
speed = 2 if row % 2 == 0 else -2

    # Create 2 logs per lane, make them wider than cars      log1 =
Obstacle(0, y_pos, TILE_SIZE * 3, BROWN, speed)      log2 =
Obstacle(WIDTH // 2 + 100, y_pos, TILE_SIZE * 3, BROWN, speed)
        logs.add(log1, log2)
all_sprites.add(log1, log2)

# --- GAME LOOP ---
running = True
while running:
    # A. Event Handling (Input)
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False

    # Move frog only on Key Press (not
hold)          if event.type == pygame.KEYDOWN:
if event.key == pygame.K_UP:
        player.move(0, -1)
elif event.key == pygame.K_DOWN:
        player.move(0, 1)
elif event.key == pygame.K_LEFT:
        player.move(-1, 0)

```



```

        elif event.key == pygame.K_RIGHT:
player.move(1, 0)

    # B. Update everything
all_sprites.update()

    # C. Collision Logic

        # 1. Car Collision (Instant Death)
        # spritecollide checks if 'player' hits anything in 'cars'
group      # The 'False' argument means "do not delete the car on
hit"       if pygame.sprite.spritecollide(player, cars, False):
            print("Hit by car!")
player.reset_position()

        # 2. Water/Log Logic
        # First, check if player is in the river area
if RIVER_TOP <= player.rect.y < RIVER_BOTTOM:
    # Check if touching a log
    # hitting_log will be a list of logs the player is touching
hitting_log = pygame.sprite.spritecollide(player, logs, False)
    if
hitting_log:
        # Move the frog at the same speed as the
log           # We grab the first log in the list
[0]           player.rect.x += hitting_log[0].speed
else:
        # In river but NOT on a log = Drown
print("Drowned!")
player.reset_position()

        # D. Win Condition (Reached Top)
if player.rect.top < TILE_SIZE:
    print("You Win!")
player.reset_position()

        # E. Drawing
screen.fill(BLACK) # Clear screen

        # Draw simple background strips for Road and River
        # (Rect arguments: x, y, width, height)      pygame.draw.rect(screen,
BLUE, (0, RIVER_TOP, WIDTH, RIVER_BOTTOM - RIVER
_TOP))
        pygame.draw.rect(screen, GRAY, (0, ROAD_TOP, WIDTH, ROAD_BOTTOM - ROAD_TO
P))

```

```
# Draw all sprites on top of background  
all_sprites.draw(screen)
```

```
pygame.display.flip() # Update the display
clock.tick(FPS)       # Keep consistent speed

pygame.quit()
sys.exit()
```

## How to Run This

1. Ensure you have Python installed.
2. Install Pygame: pip install pygame
3. Create a folder.
4. Save the code blocks above as three separate files inside that folder ( settings.py , sprites.py , main.py ).
5. Run main.py .

Would you like me to explain how to add a score counter or a "lives" system to this?

You Asked 

where do i run this in vs code"Install Pygame: pip install pygame"



Gemini 3 Pro

You run that command in the Integrated Terminal inside VS Code.

Here is exactly how to find it and use it:

### 1. Open the Terminal

You can open the terminal in three ways:

- Top Menu: Go to the top of the window, click Terminal, and select New Terminal.
- Keyboard Shortcut: Press Ctrl + ~ (tilde). This key is usually right below the Esc key.
- Right Click: Right-click your project folder in the file explorer sidebar and select "Open in Integrated Terminal".

### 2. Type the Command

Once the panel opens at the bottom of your screen, you will see a blinking cursor next to your folder path.

1. Click inside that black (or dark blue) panel.

2. Type: pip install pygame

3. Press Enter.

### 3. Watch for Success

You will see some text scrolling by. You know it worked when it stops scrolling and ends with a message like: Successfully installed pygame-2.x.x

Note: If you get an error saying "pip is not recognized," try typing py -m pip install pygame instead (common on some Windows setups).

Would you like me to explain how to run your main.py file once you've installed Pygame?

You Asked 

```
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge % pip install pygame
zsh: command not found: pip
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge % py -m pip install pygame
zsh: command not found: py
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge %
```

 Gemini 3 Pro

It looks like you are on a Mac (I can tell by nabilas-mbp and zsh ).

The command py is exclusively for Windows, and on Macs, pip is often named pip3 to distinguish it from older versions.

Try these commands in your terminal one by one.

#### Option 1: The Standard Mac Command

Type this and hit Enter:

```
bash
```

```
pip3 install pygame
```

#### Option 2: The "Safe" Way (Recommended)

If Option 1 doesn't work, this command forces your specific Python version to run the installer. It is usually the most reliable method:

```
bash
```

```
python3 -m pip install pygame
```

What if those still say "command not found"?

If `python3` is also not found, it means Python might not be installed on your Mac yet. You can check by typing:

```
bash
```

```
python3 --version
```

If that errors out, you need to download Python from [python.org](https://www.python.org).

Let me know if one of those commands worked!

You Asked 

```
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge % pip
in stall pygame zsh: command not found: pip school@nabilas-
mbp 1-hour-crossy-road-with-AI-challenge % py -m pip install
pygame zsh: command not found: py school@nabilas-mbp 1-
hour-crossy-road-with-AI-challenge % python3 -m pip install
pygame
Defaulting to user installation because normal site-packages is no
t writeable
Collecting pygame
  Downloading pygame-2.6.1.tar.gz (14.8 MB)
```

---

```
----- 14.8/14.8 MB 16.0 MB/s 0:00:00
```

```
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: pygame
  Building wheel for pygame (pyproject.toml) ... error
```

```
error: subprocess-exited-with-error
```

- × Building wheel for pygame (pyproject.toml) did not run successfully.

  | exit code: 1

  ↳ [14 lines of output]

```
src_c/_sdl2/sdl2.c:1237:10: fatal error: 'SDL.h' file not found
```

```
#include "SDL.h"
```

```
^~~~~~
```

```
1 error generated.
```

```
Skipping Cython compilation
```

---

For help with compilation see:

<https://www.pygame.org/wiki/MacCompile>

To contribute to pygame development see:

<https://www.pygame.org/contribute.html>

---

```
error: command '/usr/bin/clang' failed with exit code 1
```

[end of output]

note: This error originates from a subprocess, and is likely not a problem with pip.

ERROR: Failed building wheel for pygame

Failed to build pygame

```
[notice] A new release of pip is available: 25.3 -> 26.0
```

```
[notice] To update, run: pip3 install --upgrade pip
```

```
error: failed-wheel-build-for-install
```

- × Failed to build installable wheels for some pyproject.toml based projects

  ↳ pygame

```
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge %
```

```
pytho n3 --version Python 3.14.2 school@nabilas-mbp 1-hour-
```

```
crossy-road-with-AI-challenge % pytho n 3.14.2 -m pip install
```

```
pygame
```

```
/System/Library/Frameworks/Python.framework/Versions/2.7/Resources/Python.app/Contents/MacOS/Python: can't open file '3.1
```



```
4.2': [Errno 2] No such file or directory school@nabilas-mbp 1-hour-  
crossy-road-with-AI-challenge % pip3 install pygame
```

```
Defaulting to user installation because normal site-packages is  
not writeable
```

```
Collecting pygame
```

```
Using cached pygame-2.6.1.tar.gz (14.8 MB)
```

```
Installing build dependencies ... done
```

```
Getting requirements to build wheel ... done
```

```
Preparing metadata (pyproject.toml) ... done
```

```
Building wheels for collected packages: pygame
```

```
Building wheel for pygame (pyproject.toml) ... error
```

```
error: subprocess-exited-with-error
```

```
    × Building wheel for pygame (pyproject.toml) did not run  
    success fully.
```

```
    | exit code: 1
```

```
    ↵ [14 lines of output]
```

```
      src_c/_sdl2/sdl2.c:1237:10: fatal error: 'SDL.h' file not found
```

```
#include "SDL.h"
```

```
      ^~~~~~
```

```
1 error generated.
```

```
Skipping Cython compilation
```

```
---
```

```
For help with compilation see:
```

```
https://www.pygame.org/wiki/MacCompile
```

```
To contribute to pygame development see:
```

```
https://www.pygame.org/contribute.html
```

```
---
```

```
      error: command '/usr/bin/clang' failed with exit code 1  
[end of output]
```

```
note: This error originates from a subprocess, and is likely not a problem with pip.
```

```
ERROR: Failed building wheel for pygame
```

```
Failed to build pygame
```

```
[notice] A new release of pip is available: 25.3 -> 26.0
```

```
[notice] To update, run: pip3 install --upgrade pip
```

```
error: failed-wheel-build-for-install
```

```
    × Failed to build installable wheels for some pyproject.toml based  
    projects  
    ↳ - > pygame  
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge %
```



This error looks scary, but it has a very simple cause: your version of Python (3.14.2) is too new.

Here is the breakdown:

1. The Problem: You are on the "bleeding edge" with Python 3.14. The creators of Pygame haven't built a "pre-made" version (a wheel) for 3.14 yet.
2. The Result: Your computer tried to build Pygame from scratch (raw source code), but it failed because you don't have the heavy-duty developer tools (like SDL.h ) installed on your Mac.

### The Fix: Use a Standard Python Version

The easiest, stress-free solution is to install Python 3.12. It is stable, and Pygame has a "pre-made" version for it that will install instantly.

#### Step 1: Install Python 3.12

1. Go to the [Python Downloads page](#).
2. Look for Python 3.12.x (any version starting with 3.12).
3. Download the macOS 64-bit universal2 installer.
4. Run the installer and follow the prompts.

#### Step 2: Install Pygame on the Correct Version

Once installed, you need to tell your terminal to use *that* specific version, not the 3.14 one.

Go back to your VS Code terminal and run this exact command:

```
bash
```

```
python3.12 -m pip install pygame
```

(Note: We use `python3.12` explicitly to bypass the 3.14 version).

#### Step 3: Update VS Code Interpreter

Now that you have the right Python, you need to tell VS Code to use it for your game.

1. Open any of your Python files (like main.py ).

2. Look at the bottom right corner of the VS Code window. You probably see 3.14.2 .

3. Click that version number.

4. A menu will pop up at the top. Select Python 3.12.x from the list.

Now, try running your game code!

You Asked 

```
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge %
/usr/bin/env /usr/local/bin/python3.14
/Users/school/.vscode/extensions/ms-python.debugpy-2
025.18.0-darwin-arm64/bundled/libs/debugpy/adapter/../debugpy/launcher 5
0739 -- /Users/school/Documents/GitHub/1-hour-crossy-road-with-
-AI-challenge
/main.py
Traceback (most recent call last):
  File "/Users/school/Documents/GitHub/1-hour-crossy-road-with-
AI-challenge/main.py", line 2, in <module>
    import pygame
ModuleNotFoundError: No module named 'pygame'
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge % cd /
Users/school
/Documents/GitHub/1-hour-crossy-road-with-AI-challenge ; /usr/bi
n/env           /usr/           local/bin/python3.14
/Users/school/.vscode/extensions/ms-python.debugpy-202
5.18.0-darwin-arm64/bundled/libs/debugpy/adapter/../debugp
y/launcher 508
04 -- /Users/school/Documents/GitHub/1-hour-crossy-road-with-
AI
-challenge/m
ain.py
Traceback (most recent call last):
  File "/Users/school/Documents/GitHub/1-hour-crossy-road-with-
AI-challenge/main.py", line 2, in <module>
    import pygame
ModuleNotFoundError: No module named 'pygame'
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge % cd /
Users/school
```

```
/Documents/GitHub/1-hour-crossy-road-with-AI-challenge ; /usr/bi  
n/env           /usr/          local/bin/python3.14  
/Users/school/.vscode/extensions/ms-pyth on.debugpy-202  
5.18.0-darwin-arm64/bundled/libs/debugpy/adapter/../../debugp  
y/launcher 508  
28 -- /Users/school/Documents/GitHub/1-hour-crossy-road-with-  
AI  
-challenge/m  
ain.py  
Traceback (most recent call last):  
  File "/Users/school/Documents/GitHub/1-hour-crossy-road-with-  
AI-challenge/main.py", line 2, in <module>  
    import pygame  
ModuleNotFoundError: No module named 'pygame'  
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge % cd /  
Users/school  
/Documents/GitHub/1-hour-crossy-road-with-AI-challenge ; /usr/bi  
n/env           /usr/          local/bin/python3.14  
/Users/school/.vscode/extensions/ms-pyth on.debugpy-202  
5.18.0-darwin-arm64/bundled/libs/debugpy/adapter/../../debugp  
y/launcher 508  
93 -- /Users/school/Documents/GitHub/1-hour-crossy-road-with-  
AI  
-challenge/m  
ain.py  
Traceback (most recent call last):  
  File "/Users/school/Documents/GitHub/1-hour-crossy-road-with-  
AI-challenge/main.py", line 2, in <module>  
    import pygame  
ModuleNotFoundError: No module named 'pygame'  
school@nabilas-mbp 1-hour-crossy-road-with-AI-challenge % c
```



The error logs show exactly what is happening: VS Code is still using Python 3.14 to run your code.

As we discovered, Python 3.14 cannot run Pygame on a Mac right now. You need to force VS Code to stop using 3.14 and start using Python 3.12.

Here is how to fix this in 3 steps:

## Step 1: Switch the Interpreter in VS Code

1. Open your main.py file.
2. Press Command + Shift + P on your keyboard to open the Command Palette.
3. Type Python: Select Interpreter and click it.
4. In the list that appears, look for Python 3.12.x. Click it.
  - o If you don't see 3.12 in the list, you still need to download it from [python.org](https://python.org).

## Step 2: Open a Fresh Terminal

This is crucial. The old terminal panel at the bottom still remembers the old Python version.

1. Click the trash can icon <i class="codicon codicon-trash"></i> on your current terminal to kill it.
2. Press Ctrl + ~ (or go to Terminal > New Terminal) to open a fresh one.

## Step 3: Install Pygame on the Correct Version

Now that you are in a new terminal using the correct Python version, install Pygame again (since it was never successfully installed on 3.14).

Type this in your new terminal:

```
bash  
pip3 install pygame
```

Once that finishes successfully, click the "Play" button (triangle icon) in the top right to run your game.

You Asked 

now change the code to play more like crossy roads



To make this feel like *Crossy Road*, we need to change the fundamental logic of the game.

The Major Changes:

1. Infinite Scrolling: Instead of the frog moving *up* the screen towards a finish line, the whole world will move *down* when you step forward.
2. Procedural Generation: The game will endlessly create random lanes (Grass, Road, or River) at the top of the screen.

3. Score: Your score is how many steps you take forward.

Here are the updated files. Replace the code in your existing files with these new versions.

### File 1: settings.py

We need to add definitions for the different types of terrain.

python

```
# settings.py
import pygame

# Screen dimensions
WIDTH = 600
HEIGHT = 600
FPS = 60

# Grid setup
TILE_SIZE = 50
GRID_WIDTH = WIDTH // TILE_SIZE
GRID_HEIGHT = HEIGHT // TILE_SIZE

# Colors
WHITE = (255, 255, 255)
BLACK = (20, 20, 20)
GREEN = (100, 200, 100) # Grass
GRAY = (100, 100, 100) # Road
BLUE = (50, 150, 255) # Water
RED = (200, 50, 50) # Car
BROWN = (139, 69, 19) # Log
YELLOW = (255, 215, 0) # Chicken color

# Lane Types
LANE_GRASS = 0
LANE_ROAD = 1
LANE_WATER = 2
```

### File 2: sprites.py

We are adding a `Lane` class to handle the background strips and updating the `Player` to look a bit more like a voxel chicken.

python

```

# sprites.py import
pygame import random
from settings import
*

class Player(pygame.sprite.Sprite):
def __init__(self, x, y):
super().__init__()
    # Make it a yellow square (Chicken-ish)           self.image =
pygame.Surface((30, 30))           self.image.fill(YELLOW)
self.rect = self.image.get_rect()       self.rect.center = (x, y)
self.starting_y = y # Keep track of where we should be on screen

    def move(self, dx, dy):          #
Calculate new position           new_x =
self.rect.x + (dx * TILE_SIZE)      new_y
= self.rect.y + (dy * TILE_SIZE)

    # Allow side to side movement
if 0 <= new_x <= WIDTH - TILE_SIZE:
    self.rect.x = new_x

    # Backward movement (down) is allowed but doesn't scroll screen
if dy > 0 and new_y < HEIGHT:
    self.rect.y = new_y
return False # We did not move forward

    # Forward movement (up)
if dy < 0:
    return True # Signal that we moved forward (to scroll the world)

return False

class Obstacle(pygame.sprite.Sprite):      def
__init__(self, x, y, width, color, speed):
    super().__init__()           self.image =
pygame.Surface((width, TILE_SIZE - 10))
self.image.fill(color)           self.rect =
self.image.get_rect()           self.rect.topleft = (x, y + 5) #
+5 for a little padding         self.speed = speed

    def update(self):
        self.rect.x += self.speed
        # Loop around screen

```

```

        if self.speed > 0 and self.rect.left > WIDTH:
            self.rect.right = 0
        if self.speed < 0 and self.rect.right < 0:
            self.rect.left = WIDTH

class Lane(pygame.sprite.Sprite):
    """Represents a horizontal strip of ground (Grass, Road, or Water)"""
    def __init__(self, y, type):
        super().__init__()
        self.type = type
        self.image = pygame.Surface((WIDTH, TILE_SIZE))

        # Set color based on type
        if self.type == LANE_GRASS:
            self.image.fill(GREEN)
        elif self.type == LANE_ROAD:
            self.image.fill(GRAY)
        elif self.type == LANE_WATER:
            self.image.fill(BLUE)

        self.rect = self.image.get_rect()
        self.rect.topleft = (0, y)

```

### File 3: main.py

This is where the magic happens. We need a function `generate_lane()` that randomly decides what the next row is and adds cars or logs automatically.

`python`

```

# main.py
import pygame
import sys
import random
from settings import *
from sprites import Player, Obstacle, Lane

# Initialize
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Crossy Road Clone")
clock = pygame.time.Clock()
font = pygame.font.SysFont("Arial", 30)

# Groups
all_sprites = pygame.sprite.Group()
lanes = pygame.sprite.Group() # Holds the ground strips

```



```

cars = pygame.sprite.Group()
logs = pygame.sprite.Group()

def create_lane(y_pos, is_start=False):
    """
    Generates a new lane at a specific Y position.
    is_start: If True, forces the lane to be Grass (safe zone).
    """
    if is_start:
        lane_type = LANE_GRASS
    else:
        # Randomly choose lane type: 40% Grass, 30% Road, 30% Water
        lane_type = random.choices([LANE_GRASS, LANE_ROAD, LANE_WATER],
                                   weights=[4, 3, 3])[0]

    # Create the background strip
    lane = Lane(y_pos, lane_type)
    lanes.add(lane)
    all_sprites.add(lane)

    # Spawn obstacles based on lane type
    if lane_type == LANE_ROAD:
        direction = random.choice([-1, 1])
        speed = random.randint(2, 5) * direction
        # Randomly spawn 1 or 2 cars
        for i in range(random.randint(1, 2)):
            # Random X start position
            start_x = random.randint(0, WIDTH)           car =
            Obstacle(start_x, y_pos, 40, RED, speed)
            cars.add(car)                  all_sprites.add(car)
    elif lane_type
    == LANE_WATER:
        direction = random.choice([-1, 1])
        speed = random.randint(2, 4) * direction
        # Spawn logs      for i in
        range(random.randint(1, 2)):           start_x =
        random.randint(0, WIDTH)           log =
        Obstacle(start_x, y_pos, 80, BROWN, speed)
        logs.add(log)                  all_sprites.add(log)

    # --- INITIAL SETUP ---
    # Create the player slightly lower on screen
    player = Player(WIDTH // 2, HEIGHT - 3 *
                    TILE_SIZE)

    # Fill the screen with initial lanes
    for i in range(GRID_HEIGHT + 1):

```



```

# Calculate Y from bottom to top
y = HEIGHT - (i * TILE_SIZE)
    # First 5 lanes are always grass (safe start)
create_lane(y, is_start=(i < 5))

# Add player LAST so he is drawn on top of lanes
all_sprites.add(player)

score = 0

# --- GAME LOOP --running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                player.move(-1, 0)
            elif event.key == pygame.K_RIGHT:
                player.move(1, 0)
            elif event.key == pygame.K_DOWN:
                player.move(0, 1)
            elif event.key == pygame.K_UP:
                # If move(0, -1) returns True, it means we moved
                # forward!                                         # Instead of moving player UP, we move the
                # World DOWN.                                     did_move = player.move(0, -1)
                if did_move:
                    score += 1

                # 1. Move player back to original spot visually (cancel
                # t he up movement)                                player.rect.y += TILE_SIZE

                # 2. Move everything else
                for sprite in all_sprites:
                    if sprite != player:
                        sprite.rect.y += TILE_SIZE

                # 3. Create a new lane at the very top (off-screen
                area)                                         create_lane(-TILE_SIZE)

                # 4. Cleanup: Remove stuff that fell off the
                bottom                                         for sprite in all_sprites:
                if sprite.rect.top > HEIGHT:
                    sprite.kill() # Removes from all groups

```



```

# Update Logic
cars.update()
logs.update()

# Collision Check

# 1. Car Hit if
pygame.sprite.spritecollide(player, cars, False):
    print("SPLAT! Game Over. Score:", score)
running = False

# 2. Water Logic
# Check what kind of lane the player is standing on      # We
look for collisions with the "Lanes" group      current_lanes =
pygame.sprite.spritecollide(player, lanes, False)      if
current_lanes:
    current_lane = current_lanes[0]
    if current_lane.type ==
LANE_WATER:
        # Check if on a log          on_log =
pygame.sprite.spritecollide(player, logs, False)      if
on_log:
        # Move with the log
player.rect.x += on_log[0].speed
else:
    print("SPLASH! Game Over. Score:", score)
running = False

# Keep player on screen (Left/Right)      if
player.rect.right > WIDTH: player.rect.right = WIDTH
if player.rect.left < 0: player.rect.left = 0

# Drawing
screen.fill(BLACK)

# Draw lanes first (background)
lanes.draw(screen)      # Draw obstacles
logs.draw(screen)      cars.draw(screen)
# Draw player
screen.blit(player.image, player.rect)

# Draw Score      score_text = font.render(f"Score:
{score}", True, WHITE)      screen.blit(score_text, (10,
10))

pygame.display.flip()

```

```
clock.tick(FPS)
```

```
pygame.quit()  
sys.exit()
```

## How to play this new version:

1. Don't uninstall anything.
2. Just copy the code above into your existing 3 files ( settings.py , sprites.py , main.py ).
3. Run main.py again.

What to expect: Now, when you press UP, you will see the grass and roads slide down, and new random terrain will appear at the top. You can go on forever!

You Asked 

the player is not staying in the screen



Gemini 3 Pro

It sounds like your chicken is doing a "moonwalk"—instead of the world scrolling past you, the chicken is actually moving backwards off the bottom of the screen every time you press UP.

This is happening because of a small logic error in sprites.py . The player tells the game "I moved forward!" but doesn't actually step forward, so when the game tries to reset the camera, it accidentally pushes the player backwards.

Here is the fix. You only need to update one file.

Fix sprites.py

Replace your entire sprites.py with this correct version. The key change is inside the move function.

python

```
# sprites.py  
import pygame  
import random  
from settings import *  
  
class Player(pygame.sprite.Sprite):  
    def __init__(self, x, y):
```

```

super().__init__()
# Make it a yellow square (Chicken-
ish)           self.image = pygame.Surface((30,
30))           self.image.fill(YELLOW)
self.rect = self.image.get_rect()
self.rect.center = (x, y)

def move(self, dx, dy):          #
Calculate new position          new_x =
self.rect.x + (dx * TILE_SIZE)      new_y
= self.rect.y + (dy * TILE_SIZE)

# 1. Side to side movement check
# Ensure we don't walk off the left or right edge
if 0 <= new_x <= WIDTH - 30: # 30 is player width
self.rect.x = new_x

# 2. Backward movement (Down key)
# We only move down if we don't fall off the
screen        if dy > 0:          if new_y < HEIGHT:
                                self.rect.y = new_y          return False # We
did not move forward (no score increase)
# 3. Forward movement (Up key)      if dy <
0:          self.rect.y = new_y # <--- THIS WAS
MISSING!          return True # Signal that we
moved forward

return False

class Obstacle(pygame.sprite.Sprite):
    def
__init__(self, x, y, width, color, speed):
        super().__init__()          self.image =
pygame.Surface((width, TILE_SIZE - 10))
self.image.fill(color)          self.rect =
self.image.get_rect()          self.rect.topleft = (x, y +
5)          self.speed = speed

    def update(self):
        self.rect.x += self.speed          # Loop
around screen          if self.speed > 0 and
self.rect.left > WIDTH:
            self.rect.right = 0          if
self.speed < 0 and self.rect.right < 0:
            self.rect.left = WIDTH

```

```

class Lane(pygame.sprite.Sprite):
    """Represents a horizontal strip of ground"""
    def __init__(self, y, type):
        super().__init__()
        self.type = type
        self.image = pygame.Surface((WIDTH, TILE_SIZE))

        if self.type == LANE_GRASS:
            self.image.fill(GREEN)
        elif self.type == LANE_ROAD:
            self.image.fill(GRAY)
        elif self.type == LANE_WATER:
            self.image.fill(BLUE)

        self.rect = self.image.get_rect()
        self.rect.topleft = (0, y)

```

## Why this fixes it

In your old code, when you pressed UP, the code said return True immediately. It skipped the part where the chicken actually takes a step!

But main.py *thought* you took a step, so it moved the whole world (and you) down one tile to compensate. Since you never took the step up, you just got pushed down. Now, the chicken steps up first, and the camera correction keeps you in the center.

You Asked 

the game closes when I hit an obstacle, I want a fail screen with the score and high score. then upon pressing enter, the game should restart.



This is a great addition! To do this, we need to stop the game from closing ( running = False ) and instead switch to a "Game Over" state.

We will mostly modify main.py .

### 1. Update main.py

Replace your entire main.py with this version. It adds:

- A reset\_game() function to clear the board and start fresh.

- A high score system that saves to a file ( `highscore.txt` ).
- A "Game Over" screen loop.

`python`

```
# main.py
import pygame
import sys
import random
import os # Needed for file handling
from settings import *
from sprites import Player, Obstacle, Lane

# Initialize
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Crossy Road Clone")
clock = pygame.time.Clock()
font = pygame.font.SysFont("Arial", 30)
game_over_font = pygame.font.SysFont("Arial", 60)

# Groups
all_sprites = pygame.sprite.Group()
lanes = pygame.sprite.Group()
cars = pygame.sprite.Group()
logs = pygame.sprite.Group()

# Global Variables
score = 0
high_score = 0
game_state = "playing" # Can be "playing" or "game_over"

def load_high_score():
    """Reads high score from a file"""
    if os.path.exists("highscore.txt"):
        with open("highscore.txt", "r") as f:
            try:
                return int(f.read())
            except:
                return 0
    return 0

def save_high_score(new_score):
    """Writes new high score to file"""
    with open("highscore.txt", "w") as f:
        f.write(str(new_score))

# Load the high score when game starts
```



```

high_score = load_high_score()

def create_lane(y_pos, is_start=False):
    """Generates a new lane and obstacles"""
    if is_start:
        lane_type = LANE_GRASS
    else:
        lane_type = random.choices([LANE_GRASS, LANE_ROAD, LANE_WATER],
weights=[4, 3, 3])[0]

    lane = Lane(y_pos, lane_type)
    lanes.add(lane)
    all_sprites.add(lane)

    if lane_type == LANE_ROAD:
        direction = random.choice([-1, 1])           speed
        = random.randint(2, 5) * direction           for i in
        range(random.randint(1, 2)):                 start_x =
        random.randint(0, WIDTH)                   car =
        Obstacle(start_x, y_pos, 40, RED, speed)
        cars.add(car)                         all_sprites.add(car)
    elif lane_type
    == LANE_WATER:
        direction = random.choice([-1, 1])           speed =
        random.randint(2, 4) * direction           for i in
        range(random.randint(1, 2)):                 start_x =
        random.randint(0, WIDTH)                   log =
        Obstacle(start_x, y_pos, 80, BROWN, speed)
        logs.add(log)                         all_sprites.add(log)

def reset_game():
    """Clears everything and restarts"""
    global score, player

    # 1. Empty all groups
    all_sprites.empty()
    lanes.empty()
    cars.empty()
    logs.empty()

    # 2. Reset Score
    score = 0

    # 3. Create Player      player = Player(WIDTH //
    2, HEIGHT - 3 * TILE_SIZE)

```



```

# 4. Generate starting world
for i in range(GRID_HEIGHT + 1):
    y = HEIGHT - (i * TILE_SIZE)
    create_lane(y, is_start=(i < 5))

    all_sprites.add(player)

# Start the game for the first time
reset_game()

# --- MAIN LOOP ---
running = True
while running:
    # 1. Event Handling      for
    event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
            if event.type ==

        pygame.KEYDOWN:           if game_state ==
            "playing":             # Movement
            controls:              if event.key ==
                pygame.K_LEFT:
                    player.move(-1, 0)
                elif event.key == pygame.K_RIGHT:
                    player.move(1, 0)
                elif event.key == pygame.K_DOWN:
                    player.move(0, 1)
                elif event.key == pygame.K_UP:
                    did_move = player.move(0, -1)           if
                    did_move:                           score += 1
                    player.rect.y += TILE_SIZE           for
                    sprite in all_sprites:               if
                    sprite != player:                  if
                    sprite.rect.y += TILE_SIZE          for
                    create_lane(-TILE_SIZE)           if
                    Cleanup off-screen sprites       for
                    sprite in all_sprites:           if
                    sprite.rect.top > HEIGHT:
                        sprite.kill()

        elif game_state == "game_over":
            # Restart control           if
            event.key == pygame.K_RETURN: # 'Enter' key
            reset_game()                 game_state = "playing"
            # 2. Game Logic (Only run if playing)

```



```

if game_state == "playing":
    cars.update()
logs.update()

        # Check for Car Crash           if
pygame.sprite.spritecollide(player, cars, False):
game_state = "game_over"

        # Check for Water/Logs         current_lanes =
pygame.sprite.spritecollide(player, lanes, False)           if
current_lanes:
    current_lane = current_lanes[0]
if current_lane.type == LANE_WATER:
    on_log = pygame.sprite.spritecollide(player, logs, False)
if on_log:
    player.rect.x += on_log[0].speed
else:
    game_state = "game_over"

        # Check Screen Boundaries      if
player.rect.right > WIDTH: player.rect.right = WIDTH
if player.rect.left < 0: player.rect.left = 0
    # Update High Score immediately if we beat
it        if score > high_score:
high_score = score
save_high_score(high_score)

        # 3. Drawing
screen.fill(BLACK)
    if game_state ==
"playing":
        lanes.draw(screen)
logs.draw(screen)           cars.draw(screen)
screen.blit(player.image, player.rect)

        # Draw Score HUD            score_text =
font.render(f"Score: {score}", True, WHITE)
screen.blit(score_text, (10, 10))
    elif game_state ==
"game_over":
        # Draw Dark Overlay          overlay =
pygame.Surface((WIDTH, HEIGHT))
overlay.set_alpha(200) # Transparency
overlay.fill(BLACK)           screen.blit(overlay,
(0, 0))

```

```

# Draw Game Over Text
game_over_text = game_over_font.render("GAME OVER", True, RED)
text_rect = game_over_text.get_rect(center=(WIDTH//2, HEIGHT//2 - 5
0))
screen.blit(game_over_text, text_rect)

# Draw Scores
score_msg = font.render(f"Final Score: {score}", True, WHITE)
score_rect = score_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 20))
screen.blit(score_msg, score_rect)

high_msg = font.render(f"High Score: {high_score}", True, YELLOW)
high_rect = high_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 60))
screen.blit(high_msg, high_rect)

restart_msg = font.render("Press ENTER to Restart", True, WHITE)
restart_rect = restart_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 12
0))
screen.blit(restart_msg, restart_rect)

pygame.display.flip()
clock.tick(FPS)

pygame.quit()
sys.exit()

```

## What's New?

1. Game State: The game now remembers if it is playing or in game\_over . If it's game over, the cars stop moving and the controls freeze (except for the restart button).
2. High Score File: The game will automatically create a file called highscore.txt in your folder. If you close the game and open it tomorrow, it will remember your best run!
3. Reset Logic: Pressing Enter runs reset\_game() , which wipes the board clean and places the player back at the start without needing to restart the Python script.

You Asked 

now its a black screen



A "Black Screen" usually means the game is running but nothing is being drawn, OR the game actually crashed (stopped working) due to a hidden error, and you are staring at a frozen window.

Here is how to fix it.

## 1. Check the Terminal

Look at the bottom of VS Code (the terminal panel). Do you see a lot of red text or an error message?

- If you see `NameError: name 'LANE_GRASS' is not defined` :
  - This means your `settings.py` is still the "Old Frogger" version. You need to update it to the "Crossy Road" version below.
- If you see `ImportError: cannot import name 'Lane'` :
  - This means your `sprites.py` is incomplete. You might have pasted the "Moonwalk Fix" but accidentally deleted the `Lane` class.

---

## 2. The Solution: Ensure all 3 files are correct

Since we have changed the code a few times, the easiest fix is to make sure all three files match exactly.

Copy and paste these blocks into your files to ensure they are 100% synced.

File 1: `settings.py` (Must have Lane Types)

```
python

# settings.py
import pygame

# Screen dimensions
WIDTH = 600
HEIGHT = 600
FPS = 60

# Grid setup
TILE_SIZE = 50
GRID_WIDTH = WIDTH // TILE_SIZE
GRID_HEIGHT = HEIGHT // TILE_SIZE

# Colors
WHITE = (255, 255, 255)
BLACK = (20, 20, 20)
GREEN = (100, 200, 100) # Grass
```

```

GRAY = (100, 100, 100) # Road
BLUE = (50, 150, 255) # Water
RED = (200, 50, 50) # Car
BROWN = (139, 69, 19) # Log
YELLOW = (255, 215, 0) # Chicken

# Lane Types
LANE_GRASS = 0
LANE_ROAD = 1
LANE_WATER = 2

```

## File 2: `sprites.py` (Must have Player, Obstacle, AND Lane)

python

```

# sprites.py
import pygame
import random
from settings import *

class Player(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = pygame.Surface((30, 30))
        self.image.fill(YELLOW)
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)

    def move(self, dx, dy):
        new_x = self.rect.x + (dx * TILE_SIZE)
        new_y = self.rect.y + (dy * TILE_SIZE)

        if 0 <= new_x <= WIDTH - 30:
            self.rect.x = new_x

        if dy > 0: # Moving Down
            if new_y < HEIGHT:
                self.rect.y = new_y
            return False

        if dy < 0: # Moving Up
            self.rect.y = new_y
            return True

        return False

class Obstacle(pygame.sprite.Sprite):
    def __init__(self, x, y, width, color, speed):

```

```

super().__init__()
self.image = pygame.Surface((width, TILE_SIZE - 10))
self.image.fill(color)
self.rect = self.image.get_rect()
self.rect.topleft = (x, y + 5)
self.speed = speed

def update(self):
    self.rect.x += self.speed
    if self.speed > 0 and self.rect.left > WIDTH:
        self.rect.right = 0
    if self.speed < 0 and self.rect.right < 0:
        self.rect.left = WIDTH

class Lane(pygame.sprite.Sprite):
    def __init__(self, y, type):
        super().__init__()
        self.type = type
        self.image = pygame.Surface((WIDTH, TILE_SIZE))

        if self.type == LANE_GRASS:
            self.image.fill(GREEN)
        elif self.type == LANE_ROAD:
            self.image.fill(GRAY)
        elif self.type == LANE_WATER:
            self.image.fill(BLUE)

        self.rect = self.image.get_rect()
        self.rect.topleft = (0, y)

```

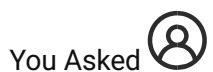
### File 3: main.py (The one with High Score)

No changes needed here if you copied the code from the last step, but if you are unsure, just re-paste the `main.py` from my previous response.

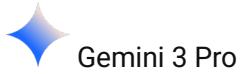
### 3. Restart

1. Save all files ( Ctrl+S or Cmd+S ).
2. Click the trash can icon in the terminal to kill the old process.
3. Run `main.py` again.

Does it work now, or do you still see an error message in the terminal?



make it so I can move to the side and back using the arrow keys.  
also make the obstacles and the player look more like the original



To make the game look more like *Crossy Road* without needing you to download image files, we can use Pygame's drawing tools to "paint" the Chicken, Cars, and Trees using simple shapes (rectangles and circles).

We will also double-check the movement code to ensure you can move in all directions (Left, Right, Down) freely.

Here are the updated files. Replace all code in your files with these new versions.

### File 1: settings.py (Better Colors)

We updated the colors to match the pastel "voxel" look of the original game.

python

```
# settings.py

# Screen dimensions
WIDTH = 600
HEIGHT = 600
FPS = 60

# Grid setup
TILE_SIZE = 50
GRID_WIDTH = WIDTH // TILE_SIZE
GRID_HEIGHT = HEIGHT // TILE_SIZE

# Colors (Crossy Road Palette)
WHITE = (255, 255, 255)
BLACK = (20, 20, 20)
# Pastel Colors
GRASS_GREEN = (109, 190, 48)    # Voxel Grass
ROAD_GRAY = (80, 80, 80)        # Asphalt
WATER_BLUE = (80, 180, 255)     # River
LOG_BROWN = (120, 70, 20)       # Wood
TREE_GREEN = (30, 100, 30)       # Dark Green Tree
CHICKEN_WHITE = (240, 240, 240)
BEAK_ORANGE = (255, 165, 0)
COMB_RED = (220, 40, 40)

# Lane Types
```

```
LANE_GRASS = 0
LANE_ROAD = 1
LANE_WATER = 2
```

## File 2: sprites.py (Voxel Art Update)

This file is much larger now because we are manually drawing the Chicken, Cars, and Trees using code.

- **Chicken:** Now has a body, beak, red comb, and wings.
- **Cars:** Now have wheels and a roof (looks like a sedan).
- **Trees:** New static obstacles that spawn on grass!

```
python
```

```
# sprites.py
import pygame
from settings import *

class Player(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        # 1. Create a transparent surface for the chicken
        self.image = pygame.Surface((40, 40), pygame.SRCALPHA)

        # 2. Draw the Chicken (Voxel Style)
        # Body (White Cube)
        pygame.draw.rect(self.image, CHICKEN_WHITE, (5, 10, 30, 25))
        # Wing (Grayish)
        pygame.draw.rect(self.image, (200, 200, 200), (5, 20, 30, 10))
        # Comb (Red thing on head)
        pygame.draw.rect(self.image, COMB_RED, (15, 0, 10, 10))
        # Beak (Orange)
        pygame.draw.rect(self.image, BEAK_ORANGE, (30, 15, 10, 5))
        # Legs (Orange)
        pygame.draw.rect(self.image, BEAK_ORANGE, (10, 35, 5, 5))
        pygame.draw.rect(self.image, BEAK_ORANGE, (25, 35, 5, 5))

        self.rect = self.image.get_rect()
        self.rect.center = (x, y)

    def move(self, dx, dy):
        """Handles movement logic"""
        new_x = self.rect.x + (dx * TILE_SIZE)
        new_y = self.rect.y + (dy * TILE_SIZE)
```



```

    # Left / Right Movement
if 0 <= new_x <= WIDTH - 40:
    self.rect.x = new_x

        # Flip image if moving left/right
Facing Right           self.image =
pygame.transform.flip(self.original_image, False, False)
dx < 0: # Facing Left
        self.image = pygame.transform.flip(self.original_image, True,
False)

    # Down (Backwards)
Movement      if dy > 0:
if new_y < HEIGHT:
    self.rect.y = new_y
return False

    # Up (Forward) Movement
if dy < 0:
    self.rect.y = new_y
return True

return False

    # Store original image for flipping
@property      def
original_image(self):
    # We recreate it to avoid degradation from multiple
flips      img = pygame.Surface((40, 40), pygame.SRCALPHA)
pygame.draw.rect(img, CHICKEN_WHITE, (5, 10, 30, 25))
pygame.draw.rect(img, (200, 200, 200), (5, 20, 30, 10))
pygame.draw.rect(img, COMB_RED, (15, 0, 10, 10))
pygame.draw.rect(img, BEAK_ORANGE, (30, 15, 10, 5))
pygame.draw.rect(img, BEAK_ORANGE, (10, 35, 5, 5))
pygame.draw.rect(img, BEAK_ORANGE, (25, 35, 5, 5))
return img

class Car(pygame.sprite.Sprite):
def __init__(self, x, y, speed):
    super().__init__()           width = 60           height = 30
self.image = pygame.Surface((width, height), pygame.SRCALPHA)
    # Pick a random car color
color = (200, 50, 50) # Red default

```



```

        # Draw Car Body           pygame.draw.rect(self.image,
color, (0, 10, width, 15))
        # Draw Roof/Windows       pygame.draw.rect(self.image, (150,
200, 255), (10, 0, width-20, 10))
        # Draw Wheels            pygame.draw.rect(self.image, BLACK,
(5, 20, 10, 10))          pygame.draw.rect(self.image, BLACK,
(width-15, 20, 10, 10))

self.rect = self.image.get_rect()
self.rect.topleft = (x, y + 10)
self.speed = speed

def update(self):
    self.rect.x += self.speed      if
self.speed > 0 and self.rect.left > WIDTH:
        self.rect.right = 0      if
self.speed < 0 and self.rect.right < 0:
self.rect.left = WIDTH

class Log(pygame.sprite.Sprite):
def __init__(self, x, y, speed):
    super().__init__()      width = 100
height = 30           self.image =
pygame.Surface((width, height))
self.image.fill(LOG_BROWN)

        # Draw wood texture (lighter lines)
pygame.draw.rect(self.image, (140, 90, 40), (0, 0, width, 5))
pygame.draw.rect(self.image, (140, 90, 40), (0, 25, width, 5))

self.rect = self.image.get_rect()
self.rect.topleft = (x, y + 10)
self.speed = speed

def update(self):
    self.rect.x += self.speed      if
self.speed > 0 and self.rect.left > WIDTH:
        self.rect.right = 0      if
self.speed < 0 and self.rect.right < 0:
self.rect.left = WIDTH

class Tree(pygame.sprite.Sprite):
    """A static obstacle that blocks movement"""
def __init__(self, x, y):      super().__init__()
self.image = pygame.Surface((40, 40), pygame.SRCALPHA)

```

```
# Trunk
pygame.draw.rect(self.image, LOG_BROWN, (15, 25, 10, 15))
# Leaves (Pyramid shape)
pygame.draw.polygon(self.image, TREE_GREEN, [(5, 25), (35, 25), (20,
0)])]

self.rect = self.image.get_rect()
self.rect.topleft = (x, y + 5)

class Lane(pygame.sprite.Sprite):
    def __init__(self, y, type):
        super().__init__()
        self.type = type
        self.image = pygame.Surface((WIDTH, TILE_SIZE))

        if self.type == LANE_GRASS:
            self.image.fill(GRASS_GREEN)
        elif self.type == LANE_ROAD:
            self.image.fill(ROAD_GRAY)
            # Add lane markers
            pygame.draw.line(self.image, WHITE, (0, 0), (WIDTH, 0), 2)
            pygame.draw.line(self.image, WHITE, (0, TILE_SIZE-2), (WIDTH, TIL
E_SIZE-2), 2)
        elif self.type == LANE_WATER:
            self.image.fill(WATER_BLUE)

        self.rect = self.image.get_rect()
        self.rect.topleft = (0, y)
```

File 3: main.py

Updated to handle the new Obstacle classes ( Car , Log , Tree ) and verify the input keys.

python

```
# main.py
import pygame
import sys
import random
import os
from settings import *
from sprites import Player, Car, Log, Tree, Lane

# Initialize
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Crossy Road Voxel Clone")
```

```

clock = pygame.time.Clock() font =
pygame.font.SysFont("Arial", 30, bold=True) game_over_font
= pygame.font.SysFont("Arial", 60, bold=True)

# Groups all_sprites = pygame.sprite.Group() lanes
= pygame.sprite.Group() cars =
pygame.sprite.Group() logs = pygame.sprite.Group()
trees = pygame.sprite.Group() # New group for
trees

# Variables score = 0
high_score = 0
game_state =
"playing"

def load_high_score():
    if os.path.exists("highscore.txt"):
        with open("highscore.txt", "r") as f:
            try: return int(f.read())
            except: return 0
    return 0

def save_high_score(new_score):
    with open("highscore.txt", "w") as f:
        f.write(str(new_score))

high_score = load_high_score()

def create_lane(y_pos, is_start=False):
    if is_start:
        lane_type = LANE_GRASS
    else:
        lane_type = random.choices([LANE_GRASS, LANE_ROAD, LANE_WATER],
weights=[4, 3, 3])[0]

    lane = Lane(y_pos, lane_type)
    lanes.add(lane)
    all_sprites.add(lane)

    # Spawn Objects based on Lane Type
    if lane_type == LANE_ROAD:
        direction = random.choice([-1, 1])
        speed = random.randint(3, 6) * direction
        for i in range(random.randint(1, 2)):
            start_x = random.randint(0, WIDTH)
            car = Car(start_x, y_pos, speed)
            cars.add(car)

```



```

        all_sprites.add(car)
    elif lane_type
== LANE_WATER:
    direction = random.choice([-1, 1])
speed = random.randint(2, 4) * direction
for i in range(random.randint(1, 2)):
start_x = random.randint(0, WIDTH)
log = Log(start_x, y_pos, speed)
logs.add(log)           all_sprites.add(log)
    elif lane_type ==
LANE_GRASS:           # 30% Chance to
spawn a tree           if
random.random() < 0.3:
    start_x = random.choice(range(0, WIDTH,
TILE_SIZE))          tree = Tree(start_x, y_pos)
trees.add(tree)         all_sprites.add(tree)

def reset_game():
    global score, player,
game_state      all_sprites.empty()
lanes.empty()    cars.empty()
logs.empty()     trees.empty()
    score = 0
game_state = "playing"
    player = Player(WIDTH // 2, HEIGHT - 3 *
TILE_SIZE)
    for i in range(GRID_HEIGHT +
2):                  y = HEIGHT - (i *
TILE_SIZE)          create_lane(y,
is_start=(i < 4))

    all_sprites.add(player)

reset_game()

# --- MAIN LOOP --running = True
while running:      # 1. Inputs
for event in pygame.event.get():
if event.type == pygame.QUIT:
    running = False

```



```

    if event.type == pygame.KEYDOWN:
if game_state == "playing":

    # Store potential new position
original_x = player.rect.x
original_y = player.rect.y

moved_forward = False
    if event.key
== pygame.K_LEFT:
        player.move(-1, 0)
elif event.key == pygame.K_RIGHT:
        player.move(1, 0)
elif event.key == pygame.K_DOWN:
        player.move(0, 1)
elif event.key == pygame.K_UP:
if player.move(0, -1):
moved_forward = True

    # Tree Collision Logic (Solid objects)
# If we hit a tree, undo the move
pygame.sprite.spritecollide(player, trees, False):
    player.rect.x = original_x
player.rect.y = original_y
moved_forward = False #
Cancel the world scroll if we hit a tree moving up

    # If the move was valid and was "UP", scroll the
world
    if moved_forward:
score += 1
        player.rect.y += TILE_SIZE
for sprite in all_sprites:
    if sprite != player:
        sprite.rect.y += TILE_SIZE
create_lane(-TILE_SIZE)

    # Cleanup
for sprite in all_sprites:
if sprite.rect.top > HEIGHT:
    sprite.kill()
    elif
game_state == "game_over":
    if
event.key == pygame.K_RETURN:
reset_game()

    # 2. Logic
if
game_state == "playing":
    cars.update()

```



```

        logs.update()

        # Car Hit           if
pygame.sprite.spritecollide(player, cars, False):
game_state = "game_over"

        # Water Logic       current_lanes =
pygame.sprite.spritecollide(player, lanes, False)           if
current_lanes:           if current_lanes[0].type == LANE_WATER:
                        on_log = pygame.sprite.spritecollide(player, logs, False)
if on_log:
                        player.rect.x += on_log[0].speed
else:
                        game_state = "game_over"

        # Update High Score      if
score > high_score:
high_score = score
save_high_score(high_score)

        # 3. Drawing
screen.fill(BLACK)
        if game_state ==
"playing":
            lanes.draw(screen)
logs.draw(screen)           cars.draw(screen)
trees.draw(screen) # Draw trees
screen.blit(player.image, player.rect)
            # UI           score_bg = pygame.Surface((150,
40))           score_bg.set_alpha(100)
score_bg.fill(BLACK)           screen.blit(score_bg, (5, 5))
score_text = font.render(f"Score: {score}", True, WHITE)
screen.blit(score_text, (10, 10))
        elif game_state == "game_over":
# Draw transparent overlay           overlay =
pygame.Surface((WIDTH, HEIGHT))
overlay.set_alpha(180)
overlay.fill(BLACK)           screen.blit(overlay,
(0,0))
            game_over_text = game_over_font.render("GAME OVER", True,
WHITE)           text_rect = game_over_text.get_rect(center=(WIDTH//2,
HEIGHT//2 - 5

```

```

    0)) )
    screen.blit(game_over_text, text_rect)

    score_msg = font.render(f"Score: {score}", True, WHITE)
    score_rect = score_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 20))
    screen.blit(score_msg, score_rect)

    high_msg = font.render(f"Best: {high_score}", True, BEAK_ORANGE)
    high_rect = high_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 60))
    screen.blit(high_msg, high_rect)

    restart_msg = font.render("Press ENTER to Restart", True, WHITE)
    restart_rect = restart_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 12
0))
    screen.blit(restart_msg, restart_rect)

    pygame.display.flip()
    clock.tick(FPS)

    pygame.quit()
    sys.exit()

```

You Asked 

make sure the lanes are separate, the player shouldn't be able to stand in the middle of two lanes

 Gemini 3 Pro

This is a very common issue in grid-based games! The problem is that when you ride a log, your character moves "smoothly" (e.g., to pixel x=312.5), which misaligns you from the grid.

We need to fix this by adding a "Grid Snap" logic. Whenever you press a key, instead of just moving 50 pixels over, the game will calculate the nearest valid "lane square" and snap you exactly into the center of it.

You only need to update `sprites.py`.

Update `sprites.py`

Replace your entire `sprites.py` with this code. I have updated the `move` function to force the player into perfect alignment every time they jump.



```

python
# sprites.py import
pygame from settings
import *

class Player(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()

        # 1. Create a transparent surface for the chicken
        self.image = pygame.Surface((40, 40), pygame.SRCALPHA)

        # 2. Draw the Chicken (Voxel Style)
        # Body (White Cube)           pygame.draw.rect(self.image,
CHICKEN_WHITE, (5, 10, 30, 25))
        # Wing (Grayish)            pygame.draw.rect(self.image, (200,
200, 200), (5, 20, 30, 10))
        # Comb (Red thing on head)  pygame.draw.rect(self.image, COMB_RED, (15, 0, 10, 10))
        # Beak (Orange)             pygame.draw.rect(self.image,
BEAK_ORANGE, (30, 15, 10, 5))
        # Legs (Orange)            pygame.draw.rect(self.image,
BEAK_ORANGE, (10, 35, 5, 5))
        pygame.draw.rect(self.image, BEAK_ORANGE, (25, 35, 5, 5))

        self.rect = self.image.get_rect()

        # CENTER THE PLAYER IN THE TILE
        # We assume x, y are the top-left of the tile.
        # We add a 5px offset to center the 40px chicken in the 50px tile.
        self.rect.topleft = (x + 5, y + 5)

    def move(self, dx, dy):
        """
        Handles movement with GRID SNAPPING.

        """
        # 1. Find which grid column/row we are currently closest to
        # (This fixes the drift from riding logs)
        current_grid_x = round(self.rect.x / TILE_SIZE)
        current_grid_y = round(self.rect.y / TILE_SIZE)

        # 2. Calculate the Target Grid Cell
        target_grid_x = current_grid_x + dx
        target_grid_y = current_grid_y + dy

        # 3. Convert Grid Coordinates back to Pixel Coordinates

```

```
# (target * 50) + 5px padding to center it
new_x = (target_grid_x * TILE_SIZE) + 5
new_y = (target_grid_y * TILE_SIZE) + 5
```



```

# --- Check Boundaries ---

# Left / Right Check
# We check grid columns 0 to 11 (WIDTH // TILE_SIZE)
if 0 <= target_grid_x < (WIDTH // TILE_SIZE):
    self.rect.x = new_x

        # Flip image if moving left/right
        if dx > 0: # Facing Right
            self.image =
pygame.transform.flip(self.original_image, False, False)           elif
dx < 0: # Facing Left
            self.image = pygame.transform.flip(self.original_image, True,
False)

        # Down (Backwards)
Movement      if dy > 0:
if new_y < HEIGHT:
    self.rect.y = new_y
return False

        # Up (Forward) Movement
if dy < 0:
    self.rect.y = new_y
return True

return False

@property      def
original_image(self):
    img = pygame.Surface((40, 40), pygame.SRCALPHA)
pygame.draw.rect(img, CHICKEN_WHITE, (5, 10, 30, 25))
pygame.draw.rect(img, (200, 200, 200), (5, 20, 30, 10))
pygame.draw.rect(img, COMB_RED, (15, 0, 10, 10))
pygame.draw.rect(img, BEAK_ORANGE, (30, 15, 10, 5))
pygame.draw.rect(img, BEAK_ORANGE, (10, 35, 5, 5))
pygame.draw.rect(img, BEAK_ORANGE, (25, 35, 5, 5))
return img

class Car(pygame.sprite.Sprite):
def __init__(self, x, y, speed):
    super().__init__()           width = 60           height = 30
self.image = pygame.Surface((width, height), pygame.SRCALPHA)
    # Pick a random car color

```



```

color = (200, 50, 50) # Red default

        # Draw Car Body           pygame.draw.rect(self.image,
color, (0, 10, width, 15))
        # Draw Roof/Windows       pygame.draw.rect(self.image, (150,
200, 255), (10, 0, width-20, 10))
        # Draw Wheels            pygame.draw.rect(self.image, BLACK,
(5, 20, 10, 10))          pygame.draw.rect(self.image, BLACK,
(width-15, 20, 10, 10))

self.rect = self.image.get_rect()
self.rect.topleft = (x, y + 10)
self.speed = speed

def update(self):
    self.rect.x += self.speed      if
self.speed > 0 and self.rect.left > WIDTH:
        self.rect.right = 0      if
self.speed < 0 and self.rect.right < 0:
self.rect.left = WIDTH

class Log(pygame.sprite.Sprite):
def __init__(self, x, y, speed):
    super().__init__()      width = 100
height = 30      self.image =
pygame.Surface((width, height))
self.image.fill(LOG_BROWN)

        # Draw wood texture      pygame.draw.rect(self.image,
(140, 90, 40), (0, 0, width, 5))
pygame.draw.rect(self.image, (140, 90, 40), (0, 25, width, 5))

self.rect = self.image.get_rect()
self.rect.topleft = (x, y + 10)
self.speed = speed

def update(self):
    self.rect.x += self.speed      if
self.speed > 0 and self.rect.left > WIDTH:
        self.rect.right = 0      if
self.speed < 0 and self.rect.right < 0:
self.rect.left = WIDTH

class Tree(pygame.sprite.Sprite):      def __init__(self, x,
y):      super().__init__()      self.image =
pygame.Surface((40, 40), pygame.SRCALPHA)

```

```

# Trunk
pygame.draw.rect(self.image, LOG_BROWN, (15, 25, 10, 15))
# Leaves
pygame.draw.polygon(self.image, TREE_GREEN, [(5, 25), (35, 25), (20,
0)])]

self.rect = self.image.get_rect()
self.rect.topleft = (x + 5, y + 5) # Centered in tile

class Lane(pygame.sprite.Sprite):
    def __init__(self, y, type):
        super().__init__()
        self.type = type
        self.image = pygame.Surface((WIDTH, TILE_SIZE))

        if self.type == LANE_GRASS:
            self.image.fill(GRASS_GREEN)
        elif self.type == LANE_ROAD:
            self.image.fill(ROAD_GRAY)
            pygame.draw.line(self.image, WHITE, (0, 0), (WIDTH, 0), 2)
            pygame.draw.line(self.image, WHITE, (0, TILE_SIZE-2), (WIDTH, TIL
E_SIZE-2), 2)
        elif self.type == LANE_WATER:
            self.image.fill(WATER_BLUE)

        self.rect = self.image.get_rect()
        self.rect.topleft = (0, y)

```

Why this fixes it:

I added math in the move() function that rounds your position: round(self.rect.x / TILE\_SIZE)

This means even if you are floating on a log at position 312.4 , as soon as you press UP, the game rounds that to 300 (Column 6) before moving you. You will now always land perfectly in the center of the squares!

You Asked 

to make it more like the original game, make the screen scroll diagonally so the player has to move or die

This is a fantastic request. Adding a "forced scroll" is exactly what creates that tension in arcade games!

To make this work, we need to change two things:

1. The Move Logic ( sprites.py ): We must stop "snapping" the Y-axis to the grid. Since the world will be moving smoothly (e.g., a lane might be at Y=53 instead of Y=50), the player needs to move *relative* to their current spot (moving exactly 50px up) rather than snapping to a fixed screen grid.
2. The Game Loop ( main.py ): We need to constantly push every object down the screen. If the player touches the bottom, they die.

Here are the updated files.

#### File 1: sprites.py (Relative Movement)

I have updated the move() function. It still snaps Left/Right to the grid (to keep you centered), but for Up/Down, it now just jumps 50 pixels from wherever you currently are. This allows you to stay on a moving lane perfectly.

```
python

# sprites.py
import pygame
from settings import *

class Player(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = pygame.Surface((40, 40), pygame.SRCALPHA)

        # Draw Chicken
        pygame.draw.rect(self.image, CHICKEN_WHITE, (5, 10, 30, 25))
        pygame.draw.rect(self.image, (200, 200, 200), (5, 20, 30, 10))
        pygame.draw.rect(self.image, COMB_RED, (15, 0, 10, 10))
        pygame.draw.rect(self.image, BEAK_ORANGE, (30, 15, 10, 5))
        pygame.draw.rect(self.image, BEAK_ORANGE, (10, 35, 5, 5))
        pygame.draw.rect(self.image, BEAK_ORANGE, (25, 35, 5, 5))

        self.rect = self.image.get_rect()
        self.rect.topleft = (x + 5, y + 5)

    def move(self, dx, dy):
        """
        Handles movement.
        X-Axis: Snaps to Grid (to fix drift).
        Y-Axis: Relative movement (to allow smooth scrolling).
        """
```

```

"""
# --- X AXIS (Horizontal) ---
# Snap to the nearest column to fix any drift
current_grid_x = round(self.rect.x / TILE_SIZE)
target_grid_x = current_grid_x + dx           new_x =
(target_grid_x * TILE_SIZE) + 5

# Check Left/Right Boundaries      if 0
<= target_grid_x < (WIDTH // TILE_SIZE):
    self.rect.x = new_x

# Flip image
if dx > 0:
    self.image = pygame.transform.flip(self.original_image,
False, False)           elif dx < 0:
    self.image = pygame.transform.flip(self.original_image,
True, False)

# --- Y AXIS (Vertical) ---
# Move exactly 1 tile up/down from current position
# We DO NOT snap to grid here, because the grid is scrolling!
new_y = self.rect.y + (dy * TILE_SIZE)

# Down (Backwards)
if dy > 0:           if
new_y < HEIGHT:
    self.rect.y = new_y
return False

# Up (Forward)
if dy < 0:
    self.rect.y = new_y
return True

return False

@property      def
original_image(self):
    img = pygame.Surface((40, 40), pygame.SRCALPHA)
    pygame.draw.rect(img, CHICKEN_WHITE, (5, 10, 30, 25))
    pygame.draw.rect(img, (200, 200, 200), (5, 20, 30, 10))
    pygame.draw.rect(img, COMB_RED, (15, 0, 10, 10))
    pygame.draw.rect(img, BEAK_ORANGE, (30, 15, 10, 5))
    pygame.draw.rect(img, BEAK_ORANGE, (10, 35, 5, 5))
    pygame.draw.rect(img, BEAK_ORANGE, (25, 35, 5, 5))
    return img

```



```
# --- THE OBSTACLES (Unchanged) ---
class Car(pygame.sprite.Sprite):
    def __init__(self, x, y, speed):
        super().__init__()           width = 60; height = 30
        self.image = pygame.Surface((width, height), pygame.SRCALPHA)
        color = (200, 50, 50)         pygame.draw.rect(self.image, color, (0, 10,
        width, 15))                 pygame.draw.rect(self.image, (150, 200, 255), (10, 0,
        width-20, 10))               pygame.draw.rect(self.image, BLACK, (5, 20, 10,
        10))                      pygame.draw.rect(self.image, BLACK, (width-15, 20, 10, 10))
        self.rect = self.image.get_rect()      self.rect.topleft = (x, y + 10)
        self.speed = speed

    def update(self):
        self.rect.x += self.speed          if self.speed > 0 and
        self.rect.left > WIDTH: self.rect.right = 0           if self.speed < 0
        and self.rect.right < 0: self.rect.left = WIDTH

class Log(pygame.sprite.Sprite):
    def __init__(self, x, y, speed):
        super().__init__()           width = 100; height = 30
        self.image = pygame.Surface((width, height))
        self.image.fill(LOG_BROWN)     pygame.draw.rect(self.image,
        (140, 90, 40), (0, 0, width, 5))
        pygame.draw.rect(self.image, (140, 90, 40), (0, 25, width, 5))
        self.rect = self.image.get_rect()      self.rect.topleft = (x, y
        + 10)                     self.speed = speed

    def update(self):
        self.rect.x += self.speed          if self.speed > 0 and
        self.rect.left > WIDTH: self.rect.right = 0           if self.speed < 0
        and self.rect.right < 0: self.rect.left = WIDTH

class Tree(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()           self.image = pygame.Surface((40, 40),
        pygame.SRCALPHA)             pygame.draw.rect(self.image, LOG_BROWN, (15, 25,
        10, 15))                   pygame.draw.polygon(self.image, TREE_GREEN, [(5, 25),
        (35, 25), (20,
        0)])                  self.rect =
        self.image.get_rect()       self.rect.topleft = (x + 5, y + 5)
```

```
class Lane(pygame.sprite.Sprite):
    def __init__(self, y, type):
        super().__init__()
        self.type = type
        self.image = pygame.Surface((WIDTH, TILE_SIZE))
        if self.type == LANE_GRASS:
            self.image.fill(GRASS_GREEN)
        elif self.type == LANE_ROAD:
            self.image.fill(ROAD_GRAY)
            pygame.draw.line(self.image, WHITE, (0, 0), (WIDTH, 0), 2)
            pygame.draw.line(self.image, WHITE, (0, TILE_SIZE-2), (WIDTH, TILE_SIZE-2), 2)
        elif self.type == LANE_WATER:
            self.image.fill(WATER_BLUE)
        self.rect = self.image.get_rect()
        self.rect.topleft = (0, y)
```

## File 2: main.py (Auto-Scroll & Death)

I've added a SCROLL\_SPEED variable. Every frame, the game pushes everything down by that amount. If the player touches the bottom edge, it triggers

python

```
# main.py
import pygame
import sys
import random
import os
from settings import *
from sprites import Player, Car, Log, Tree, Lane

# Initialize
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Crossy Road Clone")
clock = pygame.time.Clock()
font = pygame.font.SysFont("Arial", 30, bold=True)
game_over_font = pygame.font.SysFont("Arial", 60, bold=True)

# Groups
all_sprites = pygame.sprite.Group()
lanes = pygame.sprite.Group()
cars = pygame.sprite.Group()
logs = pygame.sprite.Group()
trees = pygame.sprite.Group()

# Variables
game_over .
```



```

score = 0 high_score
= 0 game_state =
"playing"
SCROLL_SPEED = 0.5 # <-- NEW: Speed of the camera movement
scroll_accumulator = 0.0 # To handle smooth float movement

def load_high_score():      if
os.path.exists("highscore.txt"):
with open("highscore.txt", "r") as f:
    try: return
int(f.read())           except:
return 0      return 0

def save_high_score(new_score):      with
open("highscore.txt", "w") as f:
    f.write(str(new_score))

high_score = load_high_score()

def create_lane(y_pos, is_start=False):
if is_start:
    lane_type = LANE_GRASS
else:
    lane_type = random.choices([LANE_GRASS, LANE_ROAD, LANE_WATER],
weigh ts=[4, 3, 3])[0]

    lane = Lane(y_pos, lane_type)
lanes.add(lane)
all_sprites.add(lane)

    if lane_type == LANE_ROAD:
        direction = random.choice([-1, 1])
speed = random.randint(3, 6) * direction
for i in range(random.randint(1, 2)):
    car = Car(random.randint(0, WIDTH), y_pos,
speed)           cars.add(car)
all_sprites.add(car)      elif lane_type == LANE_WATER:
    direction = random.choice([-1, 1])
speed = random.randint(2, 4) * direction
for i in range(random.randint(1, 2)):
    log = Log(random.randint(0, WIDTH), y_pos,
speed)           logs.add(log)
all_sprites.add(log)      elif lane_type == LANE_GRASS:
if random.random() < 0.3:
    tree = Tree(random.choice(range(0, WIDTH, TILE_SIZE)), y_pos)
trees.add(tree)

```



```

        all_sprites.add(tree)

def reset_game():
    global score, player, game_state, scroll_accumulator, SCROLL_SPEED
    all_sprites.empty(); lanes.empty(); cars.empty(); logs.empty(); trees.empty()
    score = 0
    scroll_accumulator = 0.0
    SCROLL_SPEED = 0.5 # Reset speed
    game_state = "playing"
    player = Player(WIDTH // 2, HEIGHT - 3 * TILE_SIZE)

        # Generate initial lanes      for i
    in range(GRID_HEIGHT + 2):          y =
    HEIGHT - (i * TILE_SIZE)
    create_lane(y, is_start=(i < 4))

    all_sprites.add(player)

reset_game()

# --- MAIN LOOP --running = True
while running:      for event in
pygame.event.get():           if
event.type == pygame.QUIT:
    running = False
    if event.type ==
pygame.KEYDOWN:           if game_state
== "playing":           original_x
= player.rect.x
original_y = player.rect.y
moved_forward = False
    if event.key == pygame.K_LEFT:
player.move(-1, 0)           elif event.key ==
pygame.K_RIGHT: player.move(1, 0)           elif event.key
== pygame.K_DOWN: player.move(0, 1)           elif
event.key == pygame.K_UP:           if player.move(0, -
1):
    moved_forward = True

        # Tree Collision           if
pygame.sprite.spritecollide(player, trees, False):
    player.rect.x = original_x
    player.rect.y = original_y
    moved_forward = False

```



```

# If moved forward, scroll world faster (instant
hop)           if moved_forward:                      score +=

1               player.rect.y += TILE_SIZE

for sprite in all_sprites:                         if sprite !=

player:                                              sprite.rect.y += TILE_SIZE

                     elif game_state
== "game_over":                                     if event.key ==

pygame.K_RETURN:                                    reset_game()

if game_state == "playing":
    # 1. AUTO SCROLL LOGIC
    # Slowly increase speed as score gets higher
current_scroll_speed = SCROLL_SPEED + (score * 0.01)

    # We use an accumulator to handle float speeds (e.g. 0.5 pixels per
frame)          scroll_accumulator += current_scroll_speed
pixels_to_move = int(scroll_accumulator)
    if
pixels_to_move > 0:
    scroll_accumulator -= pixels_to_move
    # Move everything DOWN
for sprite in all_sprites:
    sprite.rect.y += pixels_to_move

    # 2. Check for Infinite Generation
    # Find the highest lane          min_y = min(lane.rect.y for lane
in lanes)          if min_y > -TILE_SIZE: # If the top is visible, add a
new lane above          create_lane(min_y - TILE_SIZE)
    # 3. Cleanup old
lanes          for sprite in
all_sprites:          if
sprite.rect.top > HEIGHT:
    sprite.kill()

    # 4. PLAYER OFF SCREEN DEATH
if player.rect.top >= HEIGHT:
print("Too Slow!")
game_state = "game_over"

    # 5. Standard Updates
cars.update()
logs.update()

```



```

# Car Hit           if
pygame.sprite.spritecollide(player, cars, False):
game_state = "game_over"

# Water Logic       current_lanes =
pygame.sprite.spritecollide(player, lanes, False)           if
current_lanes:           if current_lanes[0].type == LANE_WATER:
on_log = pygame.sprite.spritecollide(player, logs, False)
if on_log:
    player.rect.x += on_log[0].speed
else:
    game_state = "game_over"

# Update High Score      if
score > high_score:
high_score = score
save_high_score(high_score)

# Drawing
screen.fill(BLACK)           if
game_state == "playing":
    lanes.draw(screen)
logs.draw(screen)           cars.draw(screen)
trees.draw(screen)
screen.blit(player.image, player.rect)
    score_bg = pygame.Surface((150, 40));
score_bg.set_alpha(100); score_bg.fill(BLACK)
screen.blit(score_bg, (5, 5))           score_text = font.render(f"Score: {score}", True, WHITE)
screen.blit(score_text, (10, 10))
elif game_state ==
"game_over":
    overlay = pygame.Surface((WIDTH, HEIGHT)); overlay.set_alpha(180);
overlay.fill(BLACK)           screen.blit(overlay, (0,0))
    game_over_text = game_over_font.render("GAME OVER", True, WHITE)
text_rect = game_over_text.get_rect(center=(WIDTH//2, HEIGHT//2 - 5
0))           screen.blit(game_over_text,
text_rect)
    score_msg = font.render(f"Score: {score}", True, WHITE)
score_rect = score_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 20))
screen.blit(score_msg, score_rect)

```

```
        restart_msg = font.render("Press ENTER to Restart", True, WHITE)
        restart_rect = restart_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 12
0))
        screen.blit(restart_msg, restart_rect)

    pygame.display.flip()
    clock.tick(FPS)

pygame.quit()
sys.exit()
```

You Asked 

make the scroll slower, and make the graphics more 3d and smooth



To make the game feel premium and "smooth," we are going to do three things:

1. Slow Down the Scroll: We will lower the speed so you have more time to think.
2. Add "Pseudo-3D" (Voxels): We will draw every object with a "side face" (a darker shadow) so they look like 3D blocks popping out of the screen.
3. Smooth Polish: We will adjust the colors and shapes to be cleaner.

Here are the updated files. Replace the code in your files again.

#### File 1: settings.py (New "Shadow" Colors)

We need darker versions of every color to draw the 3D sides of objects.

python

```
# settings.py

# Screen dimensions
WIDTH = 600
HEIGHT = 600
FPS = 60

# Grid setup
TILE_SIZE = 50
GRID_WIDTH = WIDTH // TILE_SIZE
```

```

GRID_HEIGHT = HEIGHT // TILE_SIZE

# --- COLORS ---
WHITE = (255, 255, 255)
BLACK = (20, 20, 20)

# Voxel Palette (Main Color + Shadow Color)
# Grass
GRASS_TOP = (109, 190, 48)
GRASS_SIDE = (80, 150, 30)

# Road
ROAD_TOP = (90, 90, 90)
ROAD_SIDE = (60, 60, 60)

# Water
WATER_TOP = (80, 180, 255)
WATER_DEEP = (40, 120, 200)

# Objects
LOG_TOP = (139, 69, 19)
LOG_SIDE = (100, 50, 10)

TREE_TOP = (34, 139, 34)
TREE_SIDE = (20, 100, 20)
TRUNK_TOP = (160, 82, 45)

CHICKEN_WHITE = (240, 240, 240)
CHICKEN_SHADOW = (180, 180, 180)
BEAK_ORANGE = (255, 165, 0)
COMB_RED = (220, 40, 40)

# Lane Types
LANE_GRASS = 0
LANE_ROAD = 1
LANE_WATER = 2

```

## File 2: sprites.py (The 3D Graphics Engine)

This is a big update. Every object (Car, Log, Tree, Chicken) now draws a "Shadow/Side" rectangle first, and then the "Top" rectangle slightly higher. This creates the 3D block effect.

python

```

# sprites.py
import pygame
from settings import *

```



```

class Player(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        # We make the surface taller (50px) to fit the 3D height
        self.image = pygame.Surface((40, 50), pygame.SRCALPHA)

        # --- DRAW 3D CHICKEN ---
        # 1. Shadow/Side (The 3D thickness)
        pygame.draw.rect(self.image, CHICKEN_SHADOW, (5, 20, 30, 25))
        # 2. Main Body (Top Face) - Shifted Up by 5 pixels
        pygame.draw.rect(self.image, CHICKEN_WHITE, (5, 15, 30, 25))
        # 3. Details          # Wing
        pygame.draw.rect(self.image, (200, 200, 200), (5, 25, 30, 10))
        # Comb (Red head)
        pygame.draw.rect(self.image, COMB_RED, (15, 5, 10, 10))
        # Beak                  pygame.draw.rect(self.image, BEAK_ORANGE,
        (30, 20, 10, 5))
        # Legs                  pygame.draw.rect(self.image,
        BEAK_ORANGE, (10, 40, 5, 5))
        pygame.draw.rect(self.image, BEAK_ORANGE, (25, 40, 5, 5))

        self.rect = self.image.get_rect()
        # Visual offset to align the "feet" with the grid
        self.rect.topleft = (x + 5, y - 5)

    def move(self, dx, dy):          # 1. Snap X to
        Grid           current_grid_x = round(self.rect.x /
        TILE_SIZE)           target_grid_x = current_grid_x +
        dx           new_x = (target_grid_x * TILE_SIZE) + 5
                        if 0 <= target_grid_x < (WIDTH //
        TILE_SIZE):
                        self.rect.x = new_x          # Simple Flip          if dx
        > 0: self.image = pygame.transform.flip(self.original_image, False, False)
        elif dx < 0: self.image = pygame.transform.flip(self.original_image, True,
        False)

        # 2. Free Movement Y (For scrolling)
        new_y = self.rect.y + (dy * TILE_SIZE)
                    if dy > 0: # Back          if
        new_y < HEIGHT: self.rect.y = new_y

```



```

        return False
    if dy < 0: #
Forward
self.rect.y = new_y
return True           return
False

@property      def
original_image(self):
    # Helper to redraw for flipping           img =
pygame.Surface((40, 50), pygame.SRCALPHA)
pygame.draw.rect(img, CHICKEN_SHADOW, (5, 20, 30, 25))
pygame.draw.rect(img, CHICKEN_WHITE, (5, 15, 30, 25))
pygame.draw.rect(img, (200, 200, 200), (5, 25, 30, 10))
pygame.draw.rect(img, COMB_RED, (15, 5, 10, 10))
pygame.draw.rect(img, BEAK_ORANGE, (30, 20, 10, 5))
pygame.draw.rect(img, BEAK_ORANGE, (10, 40, 5, 5))
pygame.draw.rect(img, BEAK_ORANGE, (25, 40, 5, 5))
return img

class Car(pygame.sprite.Sprite):
def __init__(self, x, y, speed):
    super().__init__()           width = 60           height = 40 #
Taller for 3D effect           self.image = pygame.Surface((width,
height), pygame.SRCALPHA)
                                base_color = (220, 60, 60) #
Bright Red           shadow_color = (160, 40,
40) # Dark Red

                                # 1. Draw Shadow (Side/Bottom)
pygame.draw.rect(self.image, shadow_color, (0, 15, width, 20))
                                # 2. Draw Top Body (Shifted up 8px)
pygame.draw.rect(self.image, base_color, (0, 7, width, 20))
                                # 3. Windows / Roof           pygame.draw.rect(self.image, (150,
200, 255), (10, 0, width-20, 12))
                                # 4. Wheels           pygame.draw.rect(self.image,
BLACK, (5, 28, 12, 12))           pygame.draw.rect(self.image,
BLACK, (width-17, 28, 12, 12))

                                self.rect = self.image.get_rect()
self.rect.topleft = (x, y + 5)
self.speed = speed

def update(self):

```



```

        self.rect.x += self.speed           if self.speed > 0 and
self.rect.left > WIDTH: self.rect.right = 0           if self.speed < 0
and self.rect.right < 0: self.rect.left = WIDTH

class Log(pygame.sprite.Sprite):
def __init__(self, x, y, speed):
    super().__init__()           width = 100           height = 40
self.image = pygame.Surface((width, height), pygame.SRCALPHA)
    # 1. Side (Darker Wood)
pygame.draw.rect(self.image, LOG_SIDE, (0, 10, width, 25))
    # 2. Top (Lighter Wood)
pygame.draw.rect(self.image, LOG_TOP, (0, 0, width, 25))
    self.rect =
self.image.get_rect()
self.rect.topleft = (x, y + 5)
self.speed = speed

def update(self):
    self.rect.x += self.speed           if self.speed > 0 and
self.rect.left > WIDTH: self.rect.right = 0           if self.speed < 0
and self.rect.right < 0: self.rect.left = WIDTH

class Tree(pygame.sprite.Sprite):      def __init__(self, x,
y):           super().__init__()           self.image =
pygame.Surface((50, 60), pygame.SRCALPHA)

    # Trunk Side           pygame.draw.rect(self.image, (100, 50,
10), (15, 35, 20, 20))
    # Trunk Top           pygame.draw.rect(self.image,
TRUNK_TOP, (15, 30, 20, 20))
    # Leaves Side (Dark Green Shadow)
pygame.draw.rect(self.image, TREE_SIDE, (5, 10, 40, 25))
    # Leaves Top (Bright Green)
pygame.draw.rect(self.image, TREE_TOP, (5, 0, 40, 25))

    self.rect = self.image.get_rect()
# Offset to center on tile
self.rect.topleft = (x, y - 10)

class Lane(pygame.sprite.Sprite):
def __init__(self, y, type):
    super().__init__()
    self.type = type

```

```
self.image = pygame.Surface((WIDTH, TILE_SIZE))

if self.type == LANE_GRASS:
    # Main Grass
    self.image.fill(GRASS_TOP)
    # Bottom edge shadow (fake 3D height for the ground layer)
    pygame.draw.rect(self.image, GRASS_SIDE, (0, TILE_SIZE-5, WIDTH,
5))

elif self.type == LANE_ROAD:
    self.image.fill(ROAD_TOP)
    pygame.draw.rect(self.image, ROAD_SIDE, (0, TILE_SIZE-5, WIDTH,
5))
    # Markings
    pygame.draw.line(self.image, WHITE, (0, 5), (WIDTH, 5), 2)
    pygame.draw.line(self.image, WHITE, (0, TILE_SIZE-10), (WIDTH, TI
LE_SIZE-10), 2)

elif self.type == LANE_WATER:
    self.image.fill(WATER_TOP)
    # Water looks deeper with a dark bottom edge
    pygame.draw.rect(self.image, WATER_DEEP, (0, TILE_SIZE-10, WIDTH,
10))

self.rect = self.image.get_rect()
self.rect.topleft = (0, y)
```

---

### File 3: main.py (Smoother Speed)

Changed `SCROLL_SPEED` from 0.5 to 0.2. This makes the game much more playable and "chill," giving you time to admire the new graphics.

```
python
```

```
# main.py
import pygame
import sys
import random
import os
from settings import *
from sprites import Player, Car, Log, Tree, Lane

# Initialize
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Crossy Road 3D Voxel")
clock = pygame.time.Clock()
```

```

font = pygame.font.SysFont("Arial", 30, bold=True)
game_over_font = pygame.font.SysFont("Arial", 60, bold=True)

# Groups all_sprites =
pygame.sprite.Group() lanes =
pygame.sprite.Group() cars =
pygame.sprite.Group() logs =
pygame.sprite.Group() trees =
pygame.sprite.Group()

# Variables score = 0
high_score = 0
game_state =
"playing"

# --- CONFIGURATION ---
# Slower scroll for smoother feel
SCROLL_SPEED = 0.2
scroll_accumulator = 0.0

def load_high_score():
    if os.path.exists("highscore.txt"):
        with open("highscore.txt", "r") as f:
            try: return int(f.read())
            except: return 0
    return 0

def save_high_score(new_score):
    with open("highscore.txt", "w") as f:
        f.write(str(new_score))

high_score = load_high_score()

def create_lane(y_pos, is_start=False):
    if is_start:
        lane_type = LANE_GRASS
    else:
        lane_type = random.choices([LANE_GRASS, LANE_ROAD, LANE_WATER],
        weights=[4, 3, 3])[0]

    lane = Lane(y_pos, lane_type)
    lanes.add(lane)
    all_sprites.add(lane)

    if lane_type == LANE_ROAD:
        direction = random.choice([-1, 1])

```

```
speed = random.randint(2, 5) * direction
for i in range(random.randint(1, 2)):
```



```

        # Random X position
x_pos = random.randint(0, WIDTH)
car = Car(x_pos, y_pos, speed)
cars.add(car)
all_sprites.add(car)

        elif lane_type
== LANE_WATER:
    direction = random.choice([-1, 1])
speed = random.randint(2, 4) * direction
for i
in range(random.randint(1, 3)): # More logs
x_pos = random.randint(0, WIDTH)
log =
Log(x_pos, y_pos, speed)
logs.add(log)
all_sprites.add(log)

        elif lane_type == LANE_GRASS:
if random.random() < 0.4: # 40% chance of tree
    # Snap tree to grid
    x_pos =
random.choice(range(0, WIDTH, TILE_SIZE))
tree
= Tree(x_pos, y_pos)
trees.add(tree)
all_sprites.add(tree)

def reset_game():
    global score, player, game_state, scroll_accumulator, SCROLL_SPEED
all_sprites.empty(); lanes.empty(); cars.empty(); logs.empty(); trees.emp
ty()
    score = 0
    scroll_accumulator = 0.0
SCROLL_SPEED = 0.2
game_state = "playing"
    player = Player(WIDTH // 2, HEIGHT - 3 *
TILE_SIZE)
        for i in range(GRID_HEIGHT +
2):
            y = HEIGHT - (i *
TILE_SIZE)
            create_lane(y,
is_start=(i < 4))

    all_sprites.add(player)

reset_game()

# --- MAIN LOOP --running = True
while running:
    for event in
pygame.event.get():
        if
event.type == pygame.QUIT:
            running = False

```



```

        if event.type ==

pygame.KEYDOWN:                      if game_state
== "playing":                         original_x
= player.rect.x
original_y = player.rect.y
moved_forward = False

                                if event.key == pygame.K_LEFT:
player.move(-1, 0)                  elif event.key ==
pygame.K_RIGHT: player.move(1, 0)          elif event.key
== pygame.K_DOWN: player.move(0, 1)          elif
event.key == pygame.K_UP:             if player.move(0, -
1):
                                         moved_forward = True

# Tree Collision
if pygame.sprite.spritecollide(player, trees, False):
    player.rect.x = original_x
player.rect.y = original_y
moved_forward = False

# Camera Hop
if moved_forward:
score += 1
    # Push everything down by 1
tile           player.rect.y += TILE_SIZE
for sprite in all_sprites:
if sprite != player:
    sprite.rect.y += TILE_SIZE
    elif game_state
== "game_over":
        if event.key ==
pygame.K_RETURN:
reset_game()

if game_state == "playing":           # 1. SMOOTH SCROLL
current_scroll_speed = SCROLL_SPEED + (score * 0.005) # Speed up very
slowly      scroll_accumulator += current_scroll_speed

# Move pixels only when accumulator > 1 (sub-pixel rendering logic)
if scroll_accumulator >= 1.0:
    pixels =
int(scroll_accumulator)
scroll_accumulator -= pixels
for sprite in all_sprites:
sprite.rect.y += pixels
    # 2. Infinite Generation min_y =
min(lane.rect.y for lane in lanes)

```



```

    if min_y > -TILE_SIZE:
        create_lane(min_y - TILE_SIZE)

    # 3. Cleanup           for sprite
    in all_sprites:         if
    sprite.rect.top > HEIGHT:

        sprite.kill()
    # 4. Death Conditions
    if player.rect.top >= HEIGHT:
        game_state = "game_over"

    cars.update()
    logs.update()
    if pygame.sprite.spritecollide(player,
    cars, False):           game_state = "game_over"

    current_lanes = pygame.sprite.spritecollide(player, lanes,
    False)                 if current_lanes:           if current_lanes[0].type
    == LANE_WATER:
        on_log = pygame.sprite.spritecollide(player, logs, False)
    if on_log:
        player.rect.x += on_log[0].speed
    else:
        game_state = "game_over"
        if score > high_score:
            high_score = score
            save_high_score(high_score)

    # Drawing
    screen.fill(BLACK)      if
    game_state == "playing":
        lanes.draw(screen)
    logs.draw(screen)       cars.draw(screen)
    trees.draw(screen)
    screen.blit(player.image, player.rect)
        # Score           score_bg = pygame.Surface((150, 40));
    score_bg.set_alpha(100); score_bg.fill(BLACK)
    screen.blit(score_bg, (5, 5))           score_text = font.render(f"Score:
    {score}", True, WHITE)           screen.blit(score_text, (10, 10))
        elif game_state ==
    "game_over":
```

```

        overlay = pygame.Surface((WIDTH, HEIGHT)); overlay.set_alpha(180); overlay.fill(BLACK)
            screen.blit(overlay, (0,0))

        game_over_text = game_over_font.render("GAME OVER", True, WHITE)
        text_rect = game_over_text.get_rect(center=(WIDTH//2, HEIGHT//2 - 5
0))
        screen.blit(game_over_text, text_rect)

        score_msg = font.render(f"Score: {score}", True, WHITE)
        score_rect = score_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 20))
        screen.blit(score_msg, score_rect)

        high_msg = font.render(f"Best: {high_score}", True, BEAK_ORANGE)
        high_rect = high_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 60))
        screen.blit(high_msg, high_rect)

        restart_msg = font.render("Press ENTER to Restart", True, WHITE)
        restart_rect = restart_msg.get_rect(center=(WIDTH//2, HEIGHT//2 + 12
0))
        screen.blit(restart_msg, restart_rect)

    pygame.display.flip()
    clock.tick(FPS)

pygame.quit()
sys.exit()

```

You Asked 

[create pdf of this conversation](#)



Gemini 3 Pro

I cannot directly generate a downloadable PDF file. However, you can easily save this conversation as a PDF yourself using your browser.

How to Save this Chat as a PDF:

1. Windows: Press Ctrl + P
2. Mac: Press Cmd + P
3. Destination: Change the printer destination to "Save as PDF".