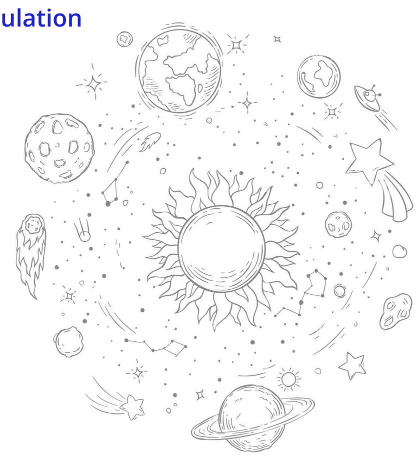
Break the Universe: A Mini Space Simulation

Intro to physics and motion in games

Art Jørstad Olson

The Gathering 2026



Introduction

Mini Solar System Challenge

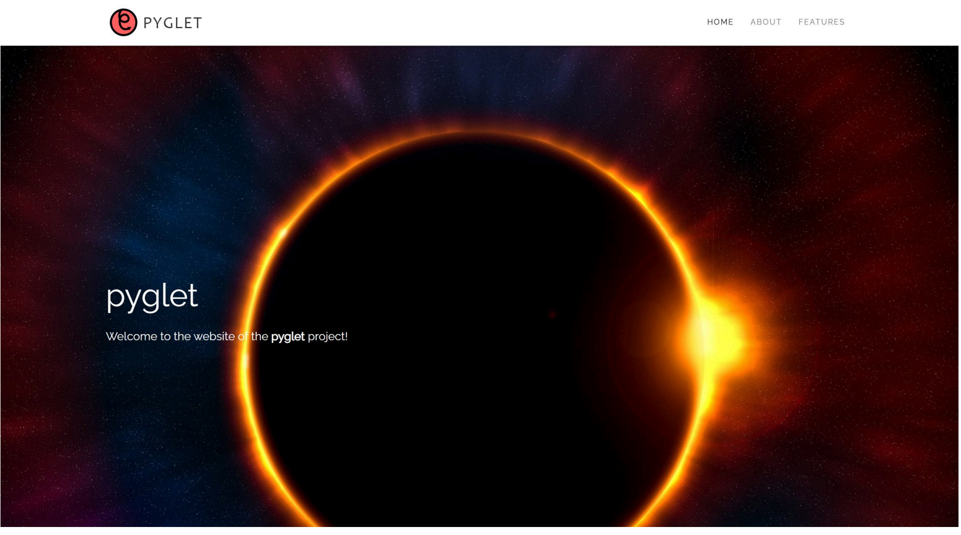
In this quick hands-on session, you'll build your own tiny solar system using simple code. By changing just a few numbers, you'll make planets orbit, speed up, slow down, and create your own unique space scene.

No advanced math needed—just experiment, tweak, and watch what happens.

By the end, you will code
:

- 🪐 *Multiple planets orbiting a star*
- ⚡ *Control over speed, size, and distance*
- 🎮 *A feel for how movement works in games*

👉 *Your mission: make the coolest (or weirdest) solar system you can.*



pyglet

Welcome to the website of the **pyglet** project!

What We're Using: **Pyglet *in* Python**

Pyglet is a lightweight Python library for making games and visual apps.

- 🎮 *Opens a game window*
- 🟢 *Draws shapes (like our planets)*
- ⏱️ *Runs a game loop (updates every frame)*

Perfect for:

- *beginners*
- *quick experiments*
- *visual projects*

👉 ***GitHub: pyglet*** ***https://github.com/pyglet/pyglet***

Setup

Open terminal

Type: `pip install pyglet`

Press Enter

You're ready 🚀

Try pip3 install pyglet if needed
Make sure Python is installed

Starter file => ***https://github.com/Artomic2020/Break_the_Universe***

starter_planets.py

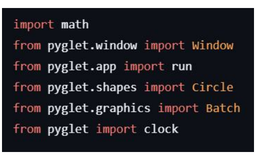
1. Window (the game screen)
Creates the space where everything happens.

2. Sun + Planets
Simple shapes (circles) we draw on screen.

3. Update Loop
Runs ~60 times per second
→ moves the planets

4. Math (the magic part)
cos + sin = circular motion

libraries



```
import math
from pyglet.window import Window
from pyglet.app import run
from pyglet.shapes import Circle
from pyglet.graphics import Batch
from pyglet import clock
```

`import math`
Provides trigonometric functions (like sine and cosine).

`pyglet.window import Window`
Creates the application window where the animation is displayed.

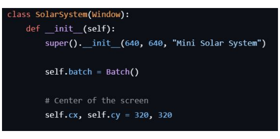
`pyglet.app import run`
Starts and runs the main event loop for the program.

`pyglet.shapes import Circle`
Used to draw circular shapes (sun and planets) on the screen.

`pyglet.graphics import Batch`
Groups multiple shapes together for efficient rendering.

`pyglet import clock`
Handles timing and updates, allowing smooth animation over time.

Window
(the Game screen)



```
class SolarSystem(Window):
    def __init__(self):
        super().__init__(640, 640, "Mini Solar System")

        self.batch = Batch()

        # Center of the screen
        self.cx, self.cy = 320, 320
```

Class Definition (SolarSystem)
Creates a window class for the simulation by extending Pyglet's Window.

init Method
Initializes the window with a fixed size and title.

Batch Creation
Sets up a batch to efficiently draw all objects together.

Center Coordinates
Defines the center point of the screen, used as the sun's position and orbit reference.

Update Loop

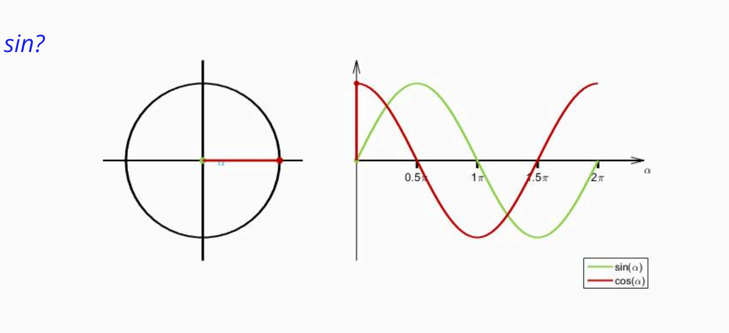
The Update Loop
Calls the update function 60 times per second to keep the animation smooth and continuously update the planets' positions.

```
game = SolarSystem()
clock.schedule_interval(game.update, 1 / 60)
```

Start Simulation
Creates the solar system instance and schedules the update function to run continuously for smooth animation.

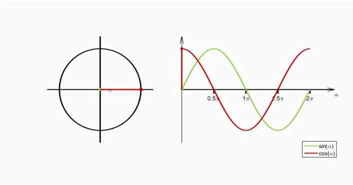
What's cos and sin?

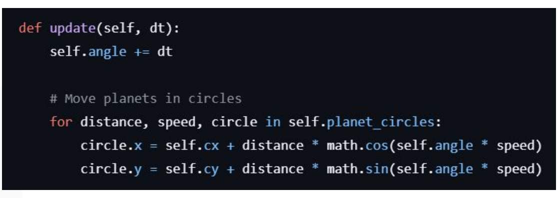
Imagine a point going around a circle.
cos gives the horizontal position, sin gives the vertical.



Put them together, and you get circular motion.

Coding with cos and sin





```
def update(self, dt):
    self.angle += dt

    # Move planets in circles
    for distance, speed, circle in self.planet_circles:
        circle.x = self.cx + distance * math.cos(self.angle * speed)
        circle.y = self.cy + distance * math.sin(self.angle * speed)
```

This function updates the animation by increasing the angle over time and recalculating each planet's position so they move in circular orbits around a central point using cosine and sine.

This is Trigonometry

Run

Run Command

```
python starter_planets.py
```

Type this line in your terminal to launch the simulation and start the program.

Experiment

shaky.py

A simple orbit simulation becomes far more dynamic when gravity, velocity, and randomness are introduced—turning predictable motion into a living system where planets can wobble, collide, or spiral unpredictably, much like a real chaotic universe.

Challenge: *Take this simulation further by exploring shader programs for visual effects, or add boundaries to see how planets react when they collide with limits of the system.*

Experiment

shader.py

Use randomness, obstacles, shaders, gravity or mouse controls in your univers

.

https://github.com/Artomic2020/Break_the_Universe