Orbit Mania

Adriel Imaran Santoso (C2TB1701)

May 7, 2024

I Introduction

"Orbit Mania" is a space-themed arcade game where the player controls a spacecraft, dodging asteroids while managing fuel consumption. This report provides an overview of the game's objectives, background inspiration, theoretical concepts, and code explanation.

This game was inspired by the concept of the Hohmann transfer, which I encountered last spring break while exploring my curiosity about orbital mechanics. It refers to a space maneuver used to transfer a spacecraft between two orbits of different radii about a central body while conserving fuel.

II How to Play

II.1 Objective

The goal is to navigate the spacecraft through an asteroid field while avoiding collisions and managing the fuel bar to prevent depletion. Players can also switch between two circular orbits, where energy packets are scattered to provide slight fuel refills.

II.2 Controls

The game makes use of the following keys:

- **ESC**: Quit the game
- **SPACE**: Switch between orbits
- **s**: Activate shield (must be held pressed)

II.3 Objects

The game features the following elements:

- **Planet**
- **Player**
- **Obstacle**
- **Energy packet**
- **Shield**

Refer to Figure 1 and 2 for in-game snapshots of these objects.

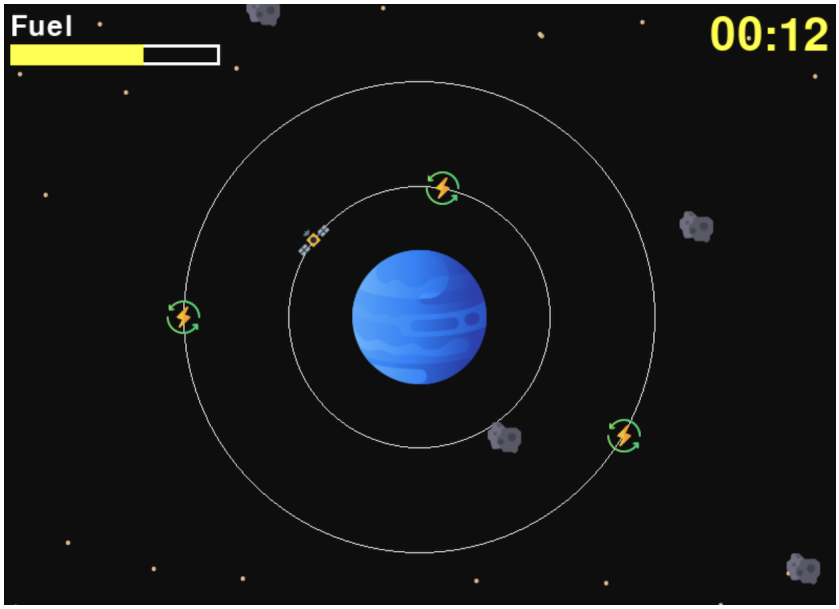

Figure 1: Game objects: planet, player, obstacles, and energy packets

III Theoretical Background

III.1 The Polar Representation of a Circle

A circle can be represented parametrically using a vector-valued function

$$f(\theta) = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix}$$

where r is the radius and θ varies from 0 to 2π radians for a full cycle.

III.2 Circular Orbits

Circular orbits happen when the gravitational force equals the centrifugal force,

$$\frac{GMm}{r^2} = \frac{mv^2}{r},$$

where G represents the gravitational constant, M is the mass of the central body (such as a planet or star), m is the mass of the orbiting object (e.g., a satellite), r denotes the radius of the orbit, and v signifies the orbital velocity of the object. This results in

$$v = \sqrt{\frac{GM}{r}}.$$

III.3 The Hohmann Transfer

In astronautics, the Hohmann transfer enables a spacecraft to switch from one circular orbit to another through two consecutive fuel thrusts. The first thrust initiates an elliptical orbit, followed by the second to enter the new circular orbit. Moreover, the ellipse transition orbit completes after π radians.

IV Code Explanation

IV.1 orbit_mania.py

This file contains the game's core functionality, including object classes, player movement, and game mechanics.

1. The `World` class is defined with an `__init__` function that initializes the game environment, including setting the width and height of the screen, loading images and background music, and setting up sounds.

2. The `Planet` class represents the celestial body around which the player orbits. It includes functions for drawing the orbit path and updating its position.

3. The `Player` class represents the player-controlled object in the game. It includes functions for movement control, orbit switching, and interaction with other game elements. Methods for activating and deactivating the shield are also included.

4. The `Shield` class represents the protective barrier around the player. It includes functions for activation, deactivation, and visual representation on the screen.

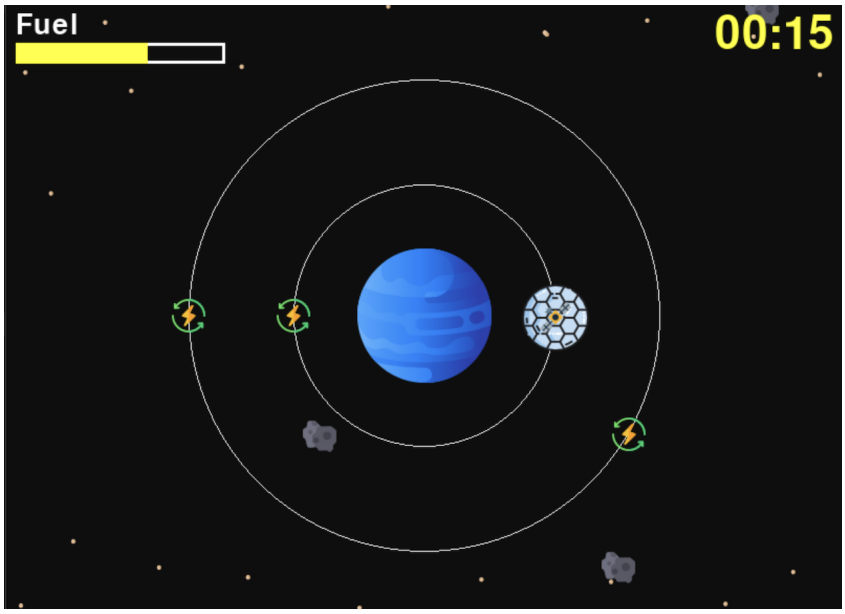

Figure 2: Shield functionality

5. The `FuelBar` class represents the player's fuel level. It includes functions for updating the fuel level based on player actions and collisions with energy packets.

6. The `EnergyPacket` class represents the collectible items that replenish the player's fuel. It includes functions for spawning at random locations and interacting with the player's fuel bar.

7. Lastly, the `Obstacle` class is defined, responsible for generating obstacles within the game environment. It includes attributes such as position and speed, along with methods for updating and drawing obstacles on the screen.

IV.2 main.py

This file initializes the game environment and manages the game loop, event handling, and game over conditions.

1. The `AppMain` class is defined with an `__init__` function that initializes the game environment, including creating instances of the `World`, `Planet`, `FuelBar`, and `Player` classes, as well as initializing sprite groups for obstacles and energy packets.
2. Inside the `__init__` function of the `AppMain` class, various attributes such as the game screen, fonts, images, sounds, and time-related variables are set up.
3. The `format_time` function formats the elapsed time into minutes and seconds for display purposes.
4. The `add_obstacle` function generates random obstacles at intervals, updating the game environment with new challenges for the player.
5. The `add_energy_packet` function adds energy packets to the game environment at random locations, providing opportunities for the player to replenish their fuel.
6. The `run` function controls the main game loop, handling user input events, updating game objects, checking for collisions, and drawing elements on the screen.
7. Inside the `run` function, conditions for ending the game are checked, such as collisions with obstacles or depletion of the fuel bar.
8. When the game ends, appropriate actions are taken, including stopping background music, displaying a game over message with an explosion effect, and waiting for a few seconds before quitting the game loop.

Figure 3: Game over screen