Flight Simulator Code Structure and Logic

Introduction

This document outlines the structure and logic behind the Flight Simulator project, detailing the primary functions, auxiliary functions, and the reasoning behind certain design choices. The simulator is designed to be scalable and adaptable, with a focus on physics-based control and AI behavior.

Primary Functions

Controls

- **Pitch**: Controlled by W (nose down) and S (nose up)
- **Yaw**: Controlled by A (turn left) and D (turn right)
- **Roll**: Controlled by Q (left roll) and E (right roll)

Throttle

- **Increase Speed**: Up Arrow Key
- **Decrease Speed**: Down Arrow Key

Auxiliary Functions

- **Deploy AI Companion**: SPACE
- **AI Companion Stats**: TAB to toggle
- **Change View**: Page Up / Page Down

Code Hierarchy and Initialization

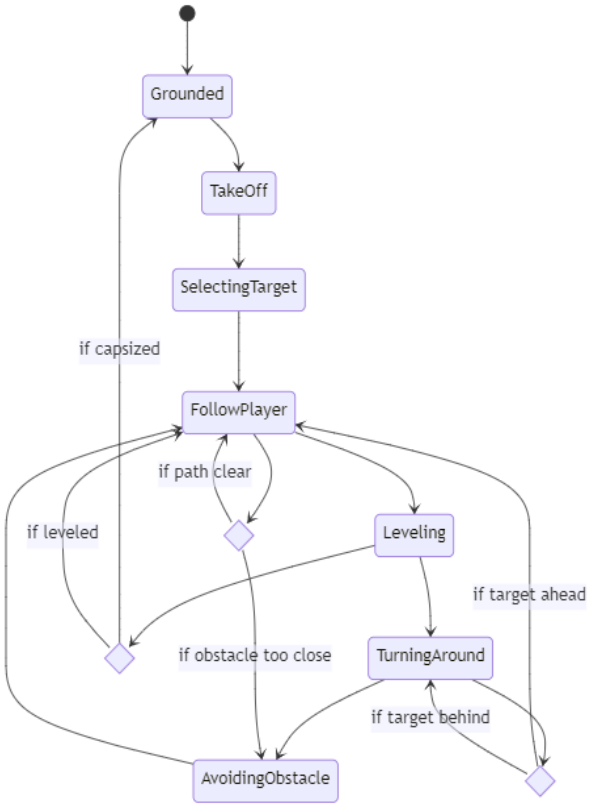
The code hierarchy is designed to ensure that objects and their fields are spawned and initialized in a structured manner. Scriptable objects are utilized to maintain a clean and scalable architecture.

Scriptable Objects

The use of scriptable objects, such as SO_ShipConfiguration, allows for easy modification and scaling of ship attributes without the need to alter the core game logic. This approach enables a modular design that can be expanded upon in future developments.

AI Companion Logic

The AI companion is governed by a state machine, which attempts to maintain the player within its cone of vision. The followPlayer() function is central to this logic, dictating the AI's behavior based on various states.



State Machine

Grounded: The initial state of the plane, presumably before takeoff.
TakeOff: Engaged when the AI is instructed to take off. It increases throttle until the plane is airborne.
SelectingTarget: Once airborne, the AI selects a target to follow.
FollowPlayer: The AI follows the player or a designated target, adjusting its throttle and checking for obstacles.
Leveling: If the plane is capsized or not level, this state will attempt to level the plane.
TurningAround: If the target is not in front of the plane, it will turn around.
AvoidingObstacle: If an obstacle is detected, the AI will attempt to avoid it by adjusting its direction.

Physics-Based Control

Drawing from robotics experience, the physics-based control system was developed to accommodate different aircraft models. The inputs are normalized between -1 and 1, akin to a joystick, allowing for potential real-world applications.

Future Plans

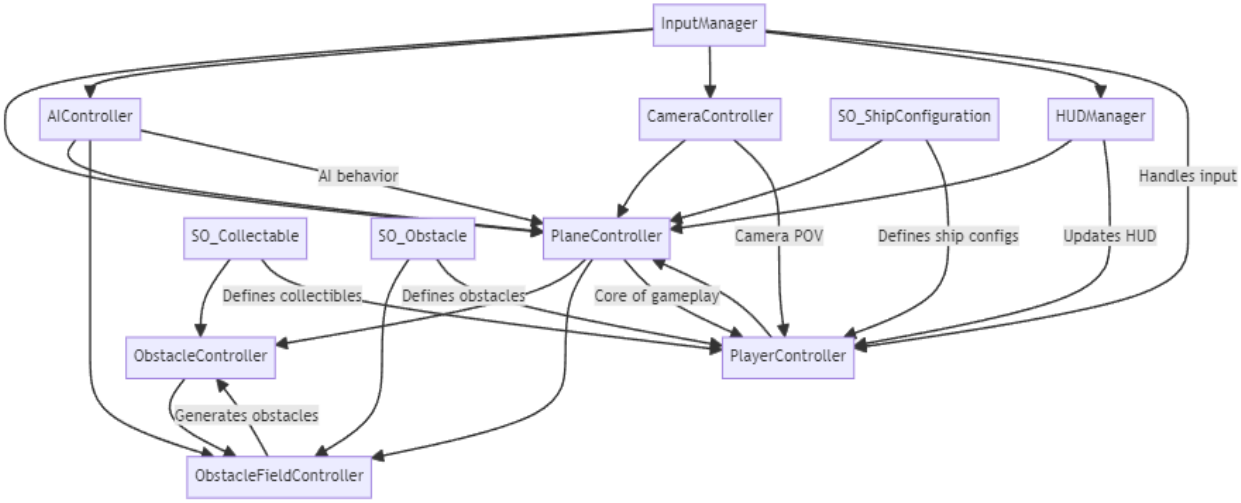
Given more time, the implementation of a fighting system and damage model would be the next steps. Additionally, the use of Unity MLAgents could be explored to enhance AI behavior.

Conclusion

The Flight Simulator project represents a complex integration of systems developed within a short timeframe. While it remains a work in progress, the foundations laid out provide a robust starting point for future enhancements.

Flight Simulator Code Structure

This document provides an in-depth look at the key scripts within the Flight Simulator project, highlighting their functions and the overall hierarchy.

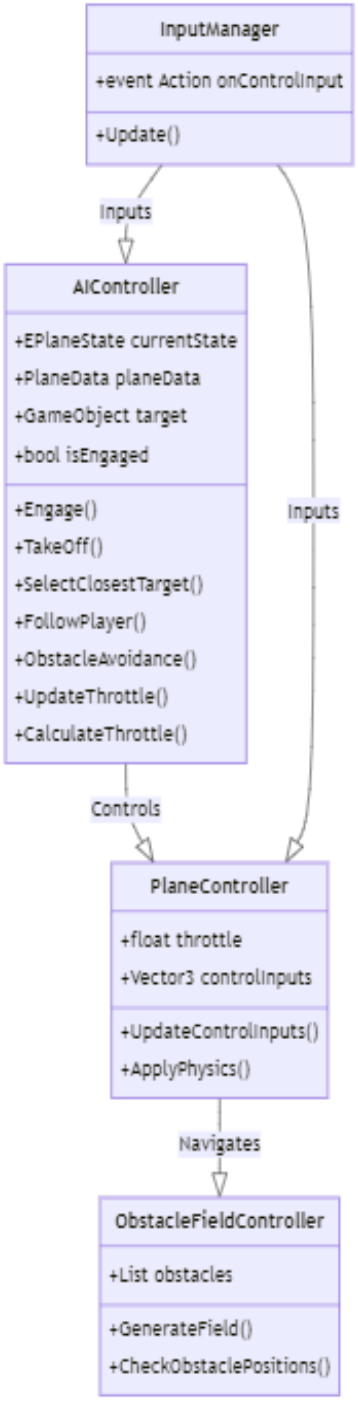


AIController.cs

- **State Management**: Handles the AI plane's state transitions between grounded, takeoff, target selection, and obstacle avoidance.
  - EngageAI(): Activates the AI control.
  - TakeOff(): Handles the takeoff procedure.
  - Update(): Main loop updating AI states.
  - ObstacleAvoidance(): Manages obstacle avoidance maneuvers.
  - FollowPlayer(): Logic for following the player.
  - LevelPlane(): Levels the plane if it's not upright.
  - TurnAround(): Turns the plane around to face the target.
- **Obstacle Detection**: Employs raycasting to detect potential collisions and changes state to avoid obstacles.
- **Throttle and Movement**: Adjusts throttle and movement to follow the player or avoid obstacles.

PlaneController.cs

- **Control Inputs**: Processes input for throttle, pitch, yaw, and roll.
- **Physics Application**: Applies physics to simulate realistic plane movement.



ObstacleFieldController.cs

- **Field Generation**: Creates a field of obstacles for navigation.
- **Position Checks**: Ensures obstacles are positioned correctly within the field.

InputManager.cs

- **Event Handling**: Utilizes the new input system to broadcast control events.
- **Input Updates**: Monitors and processes player inputs for plane control.

CameraController.cs

- **Update()**: Checks for input to switch camera POVs.
- **FixedUpdate()**: Smoothly transitions the camera to the target position.

HUDManager.cs

- **SetActivePlane(int index)**: Sets the active plane for the HUD.
- **UpdateHUD(PlaneController planeController)**: Updates the HUD with the current plane's status.

InputManager.cs

- **OnEnable()**, **OnDisable()**: Manages the enabling and disabling of input actions.
- **Update()**: Checks for various input actions and triggers events.

ObstacleController.cs

- **Start()**: Initialization of the obstacle size and rotation.

ObstacleFieldController.cs

- **GenerateObstacles()**: Generates a field of obstacles, coins or terrain.

PlaneController.cs

- **AdjustThrottle(float increment)**: Adjusts the plane's throttle.
- **SetPitchTorque(float torque)**: Sets the pitch torque for the plane's movement.

PlayerController.cs

- **Update()**: Handles player input and updates the plane's movement accordingly.

Scriptable Objects (SO)

- **SO_Collectable.cs**, **SO_Obstacle.cs**, **SO_ShipConfiguration.cs**: Define Scriptable Objects for collectibles, obstacles, and ship configurations, acting as data containers for game elements.