Project Proposal for Aero-Fighters: AI-Based Interactive Plane-Fighting Game

TP3 Update

Changed the randomly generated terrain to a continuous hill range, with towers that form on top of the hills. The terrain is randomly generated each time instead of changing in visibility during the game. Added graphic elements, such as a starry sky background, a graphic for the title of the game, and made some visual improvements to the game. Added functionality for the enemy to be able to collect packages the same way as the player. Changes the enemy's colour when it is hit and moves randomly, indicating that the player now has time to fly away. Edited the code for checking collisions with the updated terrain elements.

TP2 Update

Added randomly generated terrain elements, sidescrolling, weapon package behavior, etc. to reach MVP successfully. I have also added an advanced scoring system based on collisions with the plane, bullets, bombs, and the edge of the world and included a 'Game Over' setting for the completion of the game logic. No other design changes were made.

TP1 Update

As of now, I have not made any significant design changes. I have specified the spread angle for the machine gun weapon mode and added some specifics for the behavior of the different weapon packages.

Project Description:

In the thrilling plane-fighting game Aero Fighters, users take control of fighter aircraft and engage in fierce aerial combat against an artificial intelligence opponent. To keep players on their toes, the game has features like care packages, challenging terrain and a variety of shooting techniques. This game will be based on the Wings.io game, a .io game, but it will have a different set of features and entertaining interactive gameplay.

Similar Projects:

It is evident from looking up related projects online that many plane-fighting games are available. In particular, Aero Fighters will be modelled after the video game Wings.io. But in order to differentiate itself, Aero Fighters includes AI players with sophisticated behaviors, surprise attacks to enhance unpredictability, and a dynamic care package system. This combination of features differentiates Aero Fighters from other traditional plane-fighting games.

Structural Plan:

The final project will be organized as follows:

- main.py: The main driver program - initializes the game and has the main game loop
- graphics.py: The CMU Graphics MVC Module for handling graphical elements - includes all the shapes to be drawn for a given state of the game
- player.py: Player-related functionalities - player controls, maneuvring the plane
- enemy.py: AI opponent behaviors - includes the enemy player controls using AI algorithms
- weapons.py: Different shooting methods and surprise attack logic
- helpers.py: Helper functions and classes - for any of the previous functions

Algorithmic Plan:

The trickiest part lies in implementing advanced AI behaviors for enemy planes. A potential approach is to use decision trees or neural networks to simulate realistic decision-making for AI opponents. Additionally, handling surprise attacks requires careful consideration of player position and timing. The algorithmic plan involves thoroughly testing and refining these features to ensure a challenging yet enjoyable gaming experience.

Ideas for Additional Features:

I have planned the following features in addition to the ones mentioned before, listed in order of increasing expected complexity.

- Dynamic Terrain and Obstacles:
    - Incorporate dynamic changes in the terrain, such mountains or shifting barriers.

    - Use pathfinding methods to enable AI enemies to move through dynamic terrain.
- Formation Flying:
    - Create algorithms that will enable AI planes to fly in formation.
    - Create algorithms for communication and coordination so that AI planes can work together to launch attacks or defend against threats.
- Weather Conditions:
    - Introduce weather elements that could alter the plane's trajectory, such as wind, rain, or storms.
    - Design algorithms to mimic how weather affects weapon accuracy, control, and visibility.
- Adaptive AI Difficulty:
    - Develop an AI difficulty system that modifies itself according to the player's skill.
    - Use algorithms to evaluate player performance and modify AI opponents' abilities and conduct in real-time.
- Resource Management:
    - Establish a system in which AI opponents and players alike are restricted by scarce resources, such as ammo or fuel.
    - Create algorithms that make the most use of available resources by taking into account variables such as target distance, enemy proximity, and ammunition left.
- Multi-Agent AI Interactions:
    - Allow AI rivals to engage in conflict and building alliances with one another.
    - Use algorithms to influence AI agents' behavior in the game by allowing them to cooperate and compete.
- Random Events:
    - During gameplay, introduce random occurrences like air raids, reinforcements, or updates to the mission's objectives.
    - In order to give the game more unpredictability, implement algorithms to manage and dynamically adjust to these random events.
- Advanced Physics Simulation:
    - Improve the physics simulation to simulate realistic plane movement by taking gravity and aerodynamics into account.
    - Use sophisticated collision detection and reaction algorithms, particularly in complex situations.
- Persistent World and Campaign Mode:

    - Construct a persistent universe in which the player's choices will have a lasting impact.
    - Use algorithms to create missions on the fly, monitor player progress, and maintain the state of the game world.

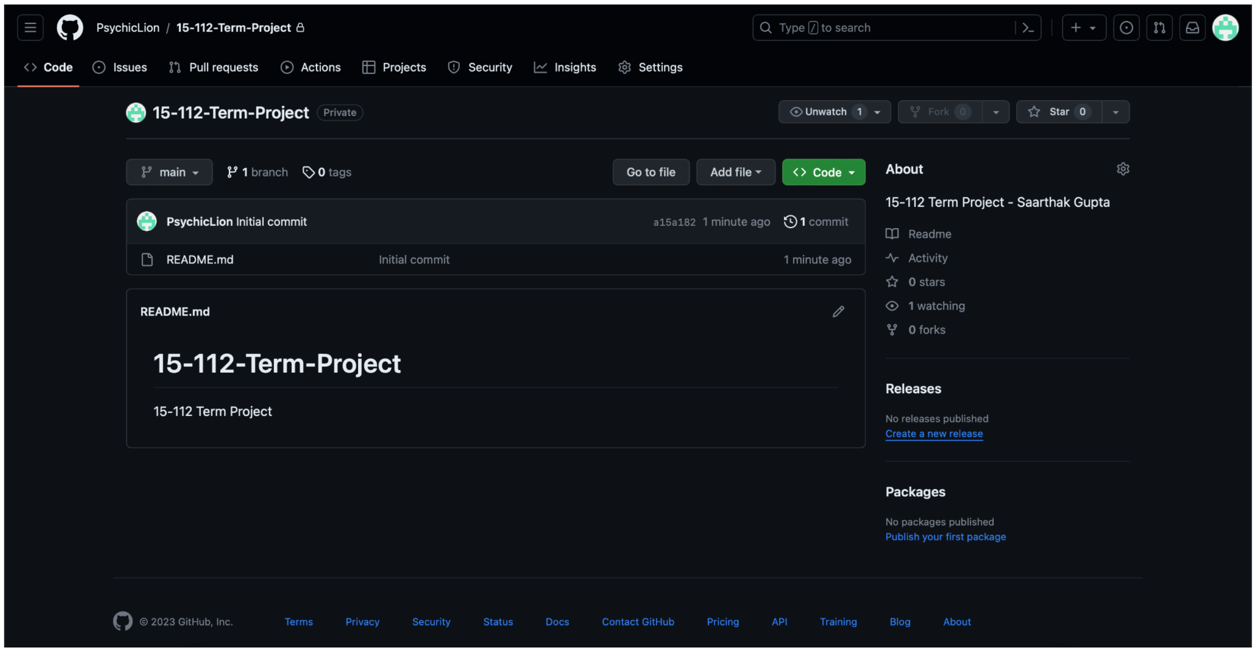
Timeline Plan:

- TP1: Set up the project structure, integrate CMU Graphics, and implement basic player controls. Write at least 600 lines of code.
- TP2: Develop and test AI behaviors for enemy planes, implement surprise attacks and shooting methods, introduce features mentioned above and refine gameplay mechanics. Have an MVP ready at this point.
- TP3: Bug fixing, testing, and final optimizations. Record video demo.

Version Control Plan:

I will use Git for version control, with a repository hosted on GitHub. Commits will be made regularly, and major milestones will be tagged. I will ensure an organised repository with clear segregation between the different files and their purposes. This guarantees code backup on the cloud and lets me revert changes if necessary.

My Github Repository Image:



PsychicLion Initial commit a15a182 1 minute ago 1 commit

README.md Initial commit 1 minute ago

README.md

15-112-Term-Project

15-112 Term Project

Module List:

- CMU Graphics
- Python standard library (for basic functionalities)
- Git for version control
- No other external modules