Term Project – Elite Spaceship Optimizer Carl Buford, Andrew ID: cbuford

# Project Description

Elite Spaceship Optimizer will be a tool that a player of the popular online game ‘Elite: Dangerous’ will be able to use by choosing some basic parameters, like their ship and how many shield booster slots they will have available and the expected enemy they will fight, and the program will determine the best defensive loadout for such a scenario. In addition to being an assistance tool to Elite: Dangerous, there will also be a game where the user will be able to control their optimal ship loadout in a 2d space side scrolling environment and engage in combat with the enemy who will have the loadout they optimized against, demonstrating the superiority of the algorithms decision in a simplified environment. As per the real-life game, the ships will have access to four weapon types, thermal, kinetic, absolute, and explosive in order to reduce the enemy’s shield hit points to zero, which will be the win condition for this simplified version. The game will feature ship rotation and strafing as means of movement, and true to space flight the player will control their thrust vector and the ship will move accordingly.

# Evaluating the Competition

My inspiration for this project was the work done by player ‘DownToEarthAstronomy’ or ‘DTEA’ and his work on creating his own shield tester program using PowerShell. His original implementation however only has one ship available in the game. Additionally, it is designed primarily for hardcore players of the game and therefore is lacking any serious UI elements, severely limiting the user experience. As such, I will be building my own testing algorithm from scratch in Python, but due to the nature of optimizing for the game, the results should be largely similar for the defensive side. One other popular tool is Coriolis.io which allows players to build and design their own ship loadouts while viewing various stats for the loadout, but it has no optimizing capability, it is merely a loadout manager. The biggest attribute both lacks, however, is that they only intend to be used as a companion to the original game, whereas I will have a 2d unique recreation allowing the player to test their new loadout entirely within my app.

# Identifying Comparison Dimensions

The five attributes and features to compare with other similar projects would be:

1. Computer Assistance in the choices
2. Player loadout choice
3. Ease of use / attractive user interface
4. Player engagement within the app
5. Test loadout feedback

# Comparison Table

The following dimensions are ranked on a scale of one to ten

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Dimension 1** | **Dimension 2** | **Dimension 3** | **Dimension 4** | **Dimension 5** |
| DTEA | 10 | 5 | 1 | 1 | 5 |
| Coriolis | 2 | 10 | 8 | 3 | 4 |

# Summary

An area that both Coriolis and DTEA’s tool lacked was an ability for the player to be engaged by the app itself and to test the loadout within the app. In order to remedy this, I will have a 2d testing environment using the optimal stats generated so that the player can see how their newly optimized loadout performs.

# Structural Plan

My project will be structured with the top-level having folders for documents and images, images will contain the different images for the possible spaceships that the player can control. There will also be a folder for the shield tester which will contain the shield tester class and shield generator related classes (shield generator; shield generator upgrade; shield booster; simpleship – for tester only). Returning to the top level, there will be the main \_\_init\_\_ file to be run which will execute the entire app properly, along with displayApp which will hold the modal app with all the MVC elements of the program. The modes will be the tester / optimizer mode, the actual game mode, the help mode, and the game over mode. The ships will be represented in the shield tester folder also with the super class for a ship class and will then be sub classed into the player and enemy ships when the game is ready to be run.

# Algorithmic Plan

There are two difficult algorithms to accomplish in this program.

*Algorithm Challenge 1:*

The first part of this project is to calculate the optimal loadout for a set of player selected constraints (Ship, Enemy DPS in terms of Thermal, Kinetic, Explosive, and Absolute, and the number of shield boosters) for a total of 6 constraints. The algorithm plan I have in mind is as such

1. Create a base shield generator for those constraints
2. Make a copy of the base and apply a modification
3. Check to see if the modification increased the survivability with the specific damage type the user inputted (this step requires knowledge of how the original game does calculations, DTEA published the formulas the original game uses in his program, I will therefore use his formula to calculate, although his general algorithm is different, he calculates all possible combinations and checks the best rather than checking as he goes)
4. If it did increase, good, try another modification
5. If it decreased, remove that modification and try the next modification

When this is done for shield generators, the next level will then go to shield boosters, which the same basic algorithm will apply to, but it will be done before sending that result back up to its master shield generator.

*Algorithm Challenge 2:*

The second challenging algorithm is in programming the ai that will control the enemy ship. Spaceships will be controlled by the player changing the thrust vector through rotating the front of the ship, adjusting the engine power (speed) and strafing with side thrusters. These are multiple methods of control and present a complicated problem to ensuring that the enemy ai can stay on target while also trying to avoid the player’s attacks. My rough algorithm outline is as such

1. Calculate the current angle from the front of the enemy ship to the center of the player ship
2. Rotate the ship the direction which will decrease the angle
3. If the angle indicates that the enemy is facing away from the player, it should strafe the opposite way to set up a favorable angle to attack while turning
4. If the angle indicates that the enemy is facing the player, it should continue to turn while also increasing its speed to close the distance
5. Once the distance is small, it should make a choice between continuing to attempt to get behind the player or strafing to keep the player in view based on whether the player has a favorable angle to it once it crosses the distance threshold

The AI problem is complicated to solve without being able to simulate how it performs and what exploits appear in the algorithm until it is more fully implemented and is therefore subject to revision.

# Timeline Plan

My current timeline plan is as follows:

Wednesday, Nov 20: Implement the groundwork for the shield tester algorithm

Friday, Nov 22: Have the basic game world environment running with shields and weapons on the ships

Saturday, Nov 23: Complete the backend for the shield tester algorithm

Sunday, Nov 24: Have a working demo for the AI battling in space

Tuesday, Nov 26: Have at least one ship available to fully optimize and battle against rudimentary ai

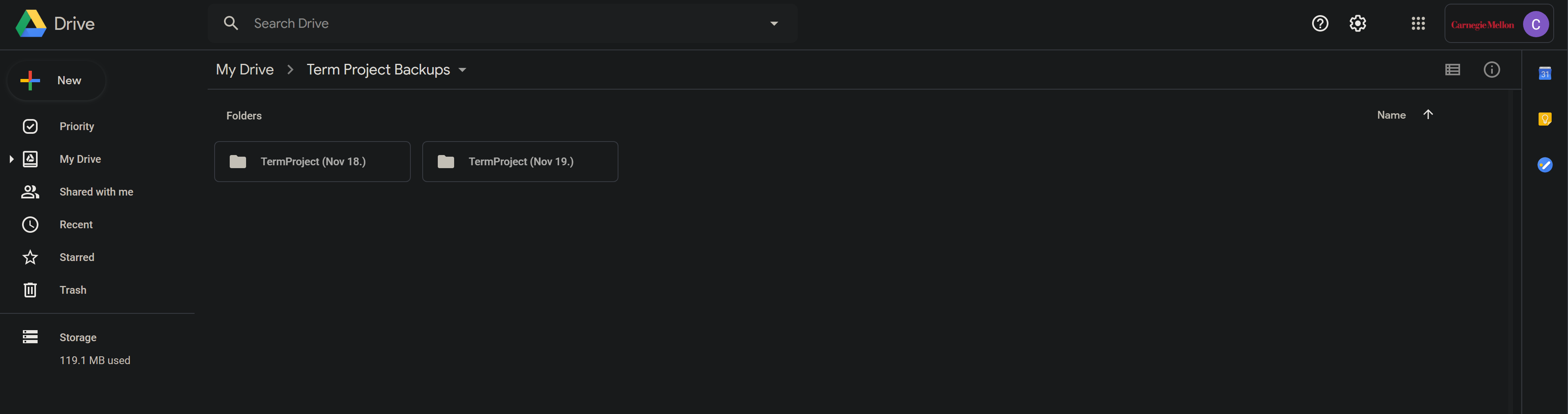
Saturday, Nov 30: Implement rest of ships and improve the AI battling scenario, adding other interest to the battle scape if the AI proved to be too rudimentary still

Tuesday, Dec 3: Finish polishing AI battle scenario, ready for play test

Thursday, Dec 5: Final feedback changes, project deadline at 4:30pm

# Version Control Plan

My version control plan involves backing up my TermProject folder to my university google drive and labeling the date as shown.



# Module List

I currently plan on using no external modules outside of the standard python modules covered in the course

# TP2 Update

*Structure:*

Files representing different classes were compressed into files of related classes for simplicity and grading sake at the advice of an autolab comment. The structure for my project now looks like the top level folder holding the actual \_\_init\_\_ file to be run, the displayApp file holding the gui and battle simulator, and the projectile and ship classes to be used in the battle simulator along with the cmu\_112\_graphics animation framework. A separate folder images now holds the ship sprite and background. The shield tester folder holds the shield classes file and the data used to calculate optimal shield loadouts.

*Algorithm 1:*

In terms of the backend shield tester algorithm implementations, no major design changes were made.

*Algorithm 2:*

The AI is very rudimentary right now, and strafing still needs to be implement, but it follows the rough plan I outlined previously, and the AI can fight back, and even win if the player is reckless.

*Miscellaneous Items:*

* Physics: Implemented a rudimentary ‘physics’ engine handling collisions between the player and the enemy, needs much refinement, but it at least has some implementation for now.
* Completed Backend: While the backend works for all ships in terms of optimizing, there is currently only one ship sprite represented graphically.
* Exploding Explosives: I decided it would be good to implement actual exploding for explosive projectiles, but I have not implemented this yet as it was not part of the original design.
* UI Plans: The UI currently is fully functional, but is not very user friendly, as they must know every possible ship name, and what values would be reasonable for different damage types, etc. With my project being at (hopefully) MVP, I can now work on a more user friendly UI that ensures that the user has all the relevant info to make reasonable selections, and achieve the outcome that they desire without having to look at the data files.

# TP3 Update

*General Overview:*

No significant design changes were made other than implementation of the UI features previously discussed. All features previously mentioned only saw tweaking and polishing to help bring them towards a better end state for a final project.

*Miscellaneous Items:*

* Physics: I updated the speed management on some of the physics so that it responds better with some bouncing mechanics so that ships no longer come to a complete stop after colliding, greatly improving close quarters combat interaction.
* AI: Tweaked responsiveness for shooting, speed control in close quarters to combine with the new physics changes, and updated solution tracking with dot product mathematics.
* Exploding Explosives: This seemingly minor feature proved somewhat counter intuitive under further consideration for implementation as the only ways for it to occur are once the projectile has made contact (and therefore is removed from the world) or once the projectile is beyond the world (and therefore not going to be drawn). Consequently, I decided not to implement this feature.
* UI: I added a splash screen with an overview of the project, along with general instructions and labels to the relative screens for general user experience. As per request of our user-study-a-thon, I opted for a separate mode that gives the user a complete list of ships so that they can make their selection instead of just presenting the info to them in the same screen they are trying to enter they choice into.