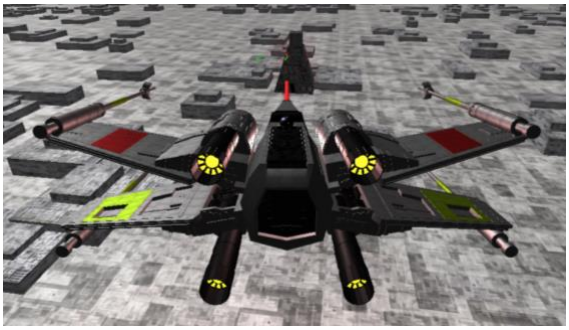
Death Star Assault

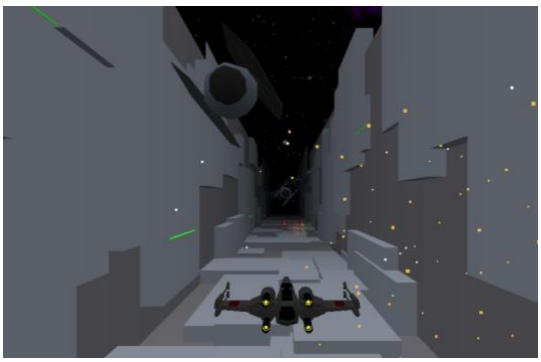
Antony Toron (atoron) and Ajay Penmatcha (ajayp)

1. Introduction

For our game, we decided to immerse the user in the experience of the rebel alliance attack on the Death Star in Episode IV, *A New Hope* by placing them in the cockpit of an X-Wing fighter and giving them the chance to lead the alliance to victory over the galactic empire. We hope Star Wars fans everywhere will enjoy playing the two parts of our game, the surface battle with the Death Star's laser turrets, and the trench battle where players must dodge and destroy enemy TIE fighters before finally shooting their proton torpedoes down the thermal exhaust port. Previous approaches to Star Wars games have included Lego Star Wars, Battlefront and Rogue Squadron. While these games were hours of fun to play, we feel our fan made game adds something special since it wasn't made by a big studio and is also a game that can be played on any computer, without having to buy an expensive console or game disc. Also, since it is open source, other people could develop on top of what we have built.



Initially, we had a different game idea entirely where you were just dodging obstacles while flying over water, but we quickly realized that the game, while fun for a bit, just lacked that spark that we were looking for. Since we were both passionate about Star Wars, as soon as the idea was floated, we quickly ran with it. Our approach then became to create a game that would both be fun to play, but also bring back memories and hopefully make you feel like you were actually fighting on the Death Star. Our game is not one of those games which is addicting that you play over and over again trying to beat your high score. That was never our intention, even though we could have easily used what we developed to build an endless game like that. Instead, there is a clear end to the game that takes you through the two main parts of the battle and hopefully leaves the player satisfied at the end. This approach works for gamers who enjoy campaign games where there a story that you are playing through, rather than gamers who enjoy endless "high-score" games. You don't get a score at the end of the game, either the alliance wins, or they lose.



2. Methodology

Since our game was developed as a two part game, we will discuss what both parts had in common and then discuss the unique implementation challenges of each. As our game library framework, we used ThreeJS to do all of the graphics. We found an X-Wing 3D model online, licensed under CC Attribution that we used for both of our games. Both parts used the same approach to generating explosions by using animating a particle system originating from a source and shooting out in random directions. Both parts also used the same code to check for collisions

by checking for the intersections of bounding boxes and/or bounding spheres. For collisions, there was a more complex ray tracing algorithm that could more precisely determine collisions of the underlying geometry rather than the approximate bounding boxes but in most cases, this was too expensive and the added collision accuracy was determined to not be worth the drop in FPS. We also used the same status dialog box (with health, etc.) to maintain consistency.

2.1 Surface Game

The most significant pieces of the surface game were the generation of the Death Star surface, the generation and functioning of the laser turrets, and the flight control of the X-Wing.

To generate the Death star surface, originally, the Death star was modelled as a sphere and the X-Wing was contained to some altitude off of the sphere's surface. However, the math for the laser turret targeting on the surface of a sphere as well as the math for controlling the X-Wing in an intuitive way for a player on the sphere proved to be extremely complex and it was determined that the time spent figuring out the algorithms for gameplay on a sphere would be better spent on other aspects of the game. Thus, the Death Star in the current version is modelled as a large plane, on which the X-Wing is contained in some bounding box in cannot go outside of.

The laser turrets in the game turn towards the X-Wing when they are within a specified targeting radius. Once the laser turret is aimed within some delta angle of the X-Wing, it begins to fire on the ship. Originally, we debated not having the turret turn at all, but just shoot directly at the ship, but we decided it was a better experience to make the laser turrets have to turn and aim, and this also gives the ability to the player to move fast enough around the turrets such that they cannot aim at you fast enough to fire. The implementation of figuring out how to turn the laser turrets towards the X-Wing took a little more math than anticipated but works quite well. It was also debated whether we should have the turrets anticipate the X-Wing's position and fire leading shots to better hit the ship, but this made gameplay too hard as the turrets could shoot down the X-Wing too easily.

The final large implementation challenge was figuring out the X-Wing flight controls. Because you are flying in 3D space but only using the arrow keys, it was a bit tricky to figure out exactly how to give 3D range of motion with 2 keyboard degrees of freedom. What we decided on was having up and down control pitch and right and left control yaw, with roll automatically calculated based on whether the player is yawing right or left. The math for translating the yaw, pitch, and roll into 3D space movement and rotation took a bit of time to figure out.

2.2 Trench Game

The second major portion of the game is when you enter the trench of the Death star to take out a certain number of TIE fighters before you shoot the proton torpedoes into the Death Star to destroy it. There were multiple implementations that we could have gone with for this section of the game, but we opted for a format that ended up paying homage to our original game idea. The goal is to fly through the trench, dodging lasers of TIE fighters and the TIE fighters themselves, while destroying enough of them to get to the portion where the player must destroy the Death Star.

One implementation difficulty with this was giving the sense of motion. Since the player could technically play this section infinitely long, generating the map infinitely did not seem to be the most efficient way to approach it. We opted for an approach where we would keep the player stationary (in the plane going in and out of the screen) and allow them to move only up/down and left/right. To give the illusion of motion, the surroundings instead move around the player (e.g. the walls zoom past the player), and so do the TIE fighters that approach. To further give the illusion of "infiniteness", when the meshes go out of view of the screen (behind the player), we simply place them back far enough away from the player that no one can see them, and they consequently fly back into the scene when close enough. Using this approach, as opposed to removing the meshes and regenerating new ones, gave a boost in optimization (on top of others, some of which we will enumerate in the results section).

There were many things that could have been implemented in this game that we opted for not creating. For example, we thought about making lasers track the player's position in the game, and also having TIE fighters move more randomly as opposed to just approaching the screen. These were not calculation intensive changes, but we decided not to implement them after receiving feedback from peers. The game may have been more challenging, but it would be an unfair challenge not based on player skill, because of the added randomness. The other components we considered adding in were more textures on objects and shadows, but the small change in visual appearance did not warrant the FPS slowdown that this came with.

3. Results

3.1. Measuring Success

Since we made a game, some primary measures of success included playability (intuitive controls and handling), efficiency (speed of game), and overall fun factor. We discuss each of these three factors separately, noting that overall fun factor generally drives the other two factors, since the ultimate goal of making a game is for the user to enjoy it.

When thinking about playability of the game, we wanted to ensure that it was easy for someone to start playing the game and understand how to interact with it. This is an important factor to consider because this is the only link between the game and the user, which allows them to interact with the code we've written. For the two different stages of the game, we considered making the control as fluid as possible for the game format. For example, we opted for an inverted camera scheme for the more flight-simulator type game format for the first part of the game, and a regular 2D rail shooter camera scheme for the second stage.

For efficiency, we felt that creating a game that ran smoothly (consistently high in FPS) was important for the player's experience. In the process of writing our code, we attempted to optimize where we could, and we made this easier for ourselves by modularizing the code early on into the work. Some prime examples of optimizations that we made during the work are: merging geometries on static meshes, using consistency of enemy design to efficiently "generate" new enemies, and precomputation when possible, etc. Measuring success for this component is more quantitative in that we would like average FPS throughout gameplay to be high, and optimize any drops in FPS away.

Finally, for overall fun factor, the primary measure was asking friends what they felt about the game, and iterating based on their results and how we saw they were interacting with the game.

3.2 Experiments Executed

In terms of experiments run, playability and overall fun factor "success" were both measured by continually play testing our own work, but also gauging feedback and sentiment from friends. We also calibrated a few of the controls based on what would make the game more fun; we would limit the amount of lasers that a user could shoot if this made the game more challenging and fun. Also, we would continually test each other's work to sanity check it as an end user of the game as opposed to being the developer.

To experiment and determine whether the game was more efficient or not, we were constantly checking resulting FPS via the stats screen (the canonical stats screen often paired with ThreeJS), and we would use the developer console in Google Chrome to our advantage. Whenever we would feel the game slowing down, or would want to know bottlenecks in the code, we would do a performance analysis via the developer console on the order of ~50 to 100 seconds, and then check functions being spent most time in. We could tell if progress had been made to speed up the game by checking this before and after changes.

3.3 Results

We received critical feedback from peers on whether the game was intuitive and fun to play, ultimately resulting in more and more positive feedback over time. This component of determining the level of interest of users was very important for success, because people often play games in different ways than the developers imagine, which gives a different perspective on how to re-examine the code.

The results of our optimization throughout the code's lifespan were positive as well. We display the results of one of the performance analyses in the developer console before and after an optimization here, as an example (the top image of FPS is before optimization). Specifically, for this optimization, we merged geometries of objects in the scene which were all static (everything non-moving on the Death Star surface) and created one huge mesh from this, which is known to have positive results for ThreeJS's rendering portion. This was definitely confirmed in the results. We see that the average FPS stayed more consistent after optimization (specifically during portions where the player was turning the X-Wing, where we see significant spikes in lag before optimizing). We also see below (right hand side is after optimization) that the render function (renderDS) is now lower on the list in terms of which functions took most time to execute (the collision detection code becomes the bottleneck there). We also determine empirically that the more consistent FPS after optimization leads to a better gaming experience.

Self Time		Total Time		Activity
1784.1 ms	2.1 %	83172.3 ms	99.9 %	Task
857.8 ms	1.0 %	80347.4 ms	96.5 %	Animation Frame Fired
970.4 ms	1.2 %	79511.4 ms	95.5 %	Function Call
27.4 ms	0.0 %	78528.1 ms	94.3 %	animateDS
23.9 ms	0.0 %	51971.7 ms	62.4 %	renderDS
293.9 ms	0.4 %	51946.9 ms	62.4 %	render
208.5 ms	0.3 %	25962.2 ms	31.2 %	updateDS
2264.1 ms	2.7 %	24332.2 ms	29.2 %	checkSceneForCollisionsDS
1715.4 ms	2.1 %	23390.8 ms	28.1 %	m
1713.8 ms	2.1 %	21781.0 ms	26.2 %	checkIfCollidedCheapDS

Self Time		Total Time		Activity
1779.7 ms	2.5 %	71442.4 ms	100.0 %	Task
1108.9 ms	1.6 %	67960.7 ms	95.1 %	Animation Frame Fired
1259.4 ms	1.8 %	66877.1 ms	93.6 %	Function Call
45.0 ms	0.1 %	65617.5 ms	91.8 %	animateDS
298.2 ms	0.4 %	47628.9 ms	66.7 %	updateDS
4716.9 ms	6.6 %	44991.2 ms	63.0 %	checkSceneForCollisionsDS
2999.3 ms	4.2 %	39908.8 ms	55.9 %	checkIfCollidedCheapDS
21.4 ms	0.0 %	36133.3 ms	50.6 %	setFromObject
8070.2 ms	11.3 %	35906.5 ms	50.3 %	(anonymous)
64.9 ms	0.1 %	16984.6 ms	23.8 %	renderDS

4. Discussion

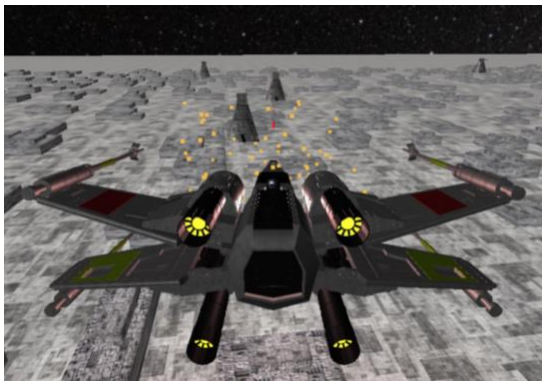
Overall, the approach we took to creating the game was promising - focusing on the numbers behind the game and looking at tools like the Google developer console for performance analysis (as well as the statistics module in ThreeJS) made determining whether we were on the right path easier. The feedback that we received from peers was also very useful to get outside perspectives. When working on a codebase for a long time and playing a game continuously for testing purposes, one can lose track of what it means for a game to be fun. For example, in an effort to create smooth motion and fast action in the second part of the game (in the trench), we lost track of whether the gameplay was fun at all. This was immediately obvious to us after receiving feedback from friends but also from each other double checking one another's work.

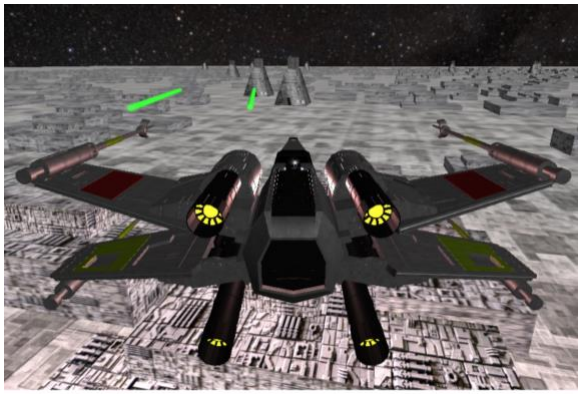
In future iterations we would focus more on having more test-driven development with a better infrastructure to determine the efficiency of the code, as well as a better way to gauge feedback from an audience (and/or utilizing the opportunities to get this feedback better). Being able to have statistics on the game gives a clear picture of the direction one should move in to improve overall.

We learned a significant amount about game development and the trials/tribulations that come with it throughout this process, and there is a significant amount of follow-up work that can be done next. Notably, there were many points in the development in the game that we had to make a decision between adding in more effects vs. keeping the game running at a reasonable frame rate. We would like to do more research in the future to come up with a robust way of writing games using ThreeJS that allows for many meshes on screen and smooth gameplay. That being said, we would also like to put more of an effort on play testing and getting feedback from users, as aforementioned, to ensure to never lose sight of the original purpose of the game: a fun experience for the players.

5. Conclusion

We very much enjoyed making this game, especially because we are both big Star Wars fans. It was so fun to see our ideas come to life and we were impressed with how quickly we could a basic version of our ideas working. Here are some more screenshots of our game.





References

We generally implemented the code on our own, but we were inspired by a few different code sources:

- ThreeJS, TGALoader, GLTFLoader, and stats.min.js
    - We used ThreeJS as our primary framework for this project, and also some ancillary tools that were helpful in loading textures and showing FPS, and memory statistics
- https://gamedevelopment.tutsplus.com/tutorials/creating-a-simple-3d-endless-runner-game-using-three-js--cms-29157
    - We used this initially when trying to implement the first version of the game, and ended up using it mostly as a learning experience for how ThreeJS can be used to create games
- https://discourse.threejs.org/t/how-to-optimize-objects-in-three-js-methods-of-optimization/2242/2
    - There are various discussions online about how to optimize ThreeJS code, and we used this one in particular to get ideas of how to make the code run faster.

Our X-Wing model was licensed under CC Attribution 4.0 and was taken from https://sketchfab.com/3d-models/star-wars-x-wing-a93f607a94d747568371b8910a81fb12

Github repository: https://github.com/ajayp15/DeathStarAssault
Github pages site: https://ajayp15.github.io/DeathStarAssault/