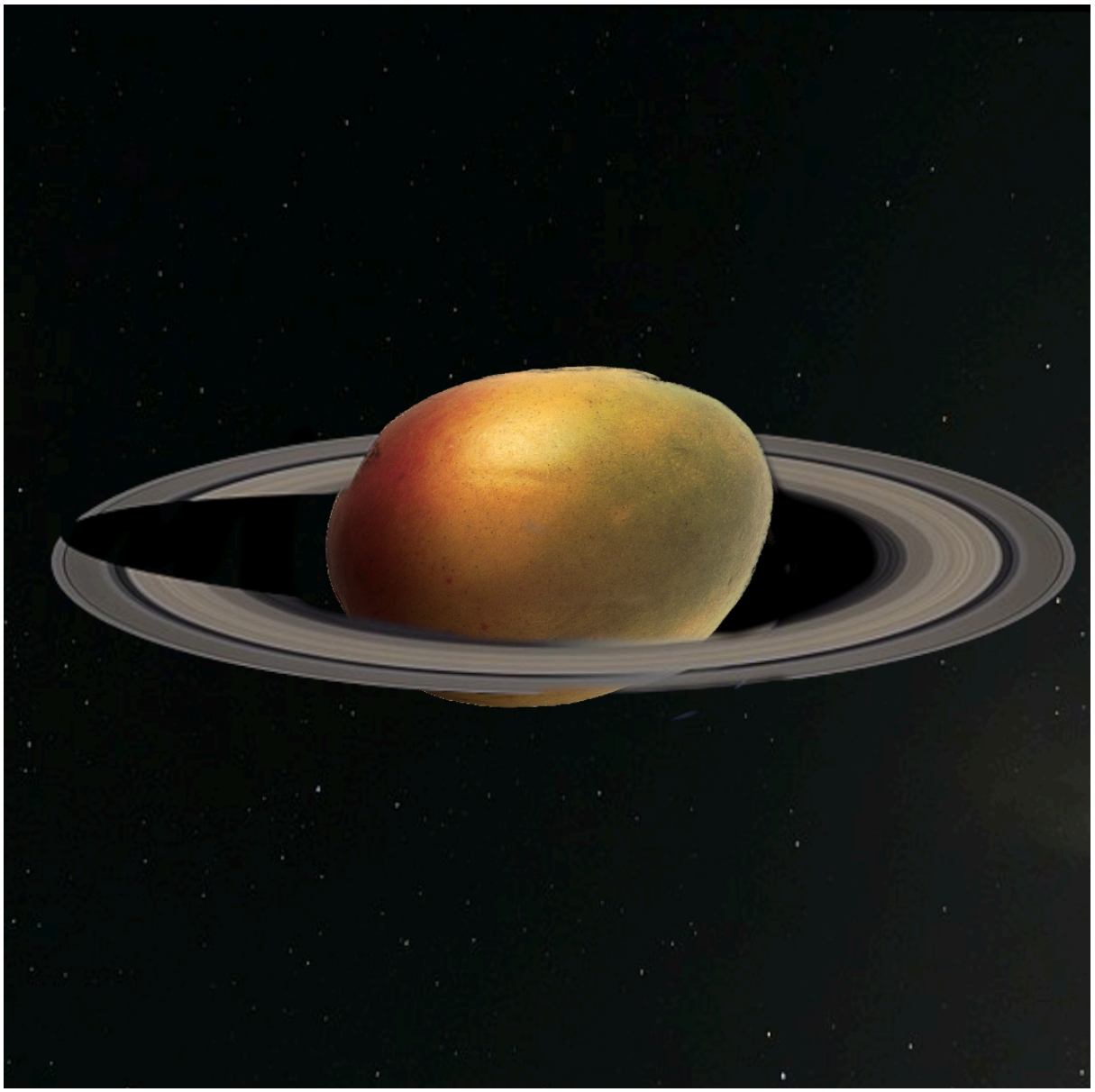
Project Notebook

Terminal Rim

Team Space Mango



Master Practical Course Games Engineering - “Games Lab” - SS 2025

Team Members

Thorya Aadland Jonas Hack Philipp Havemann

Formal Game Proposal

Game Description

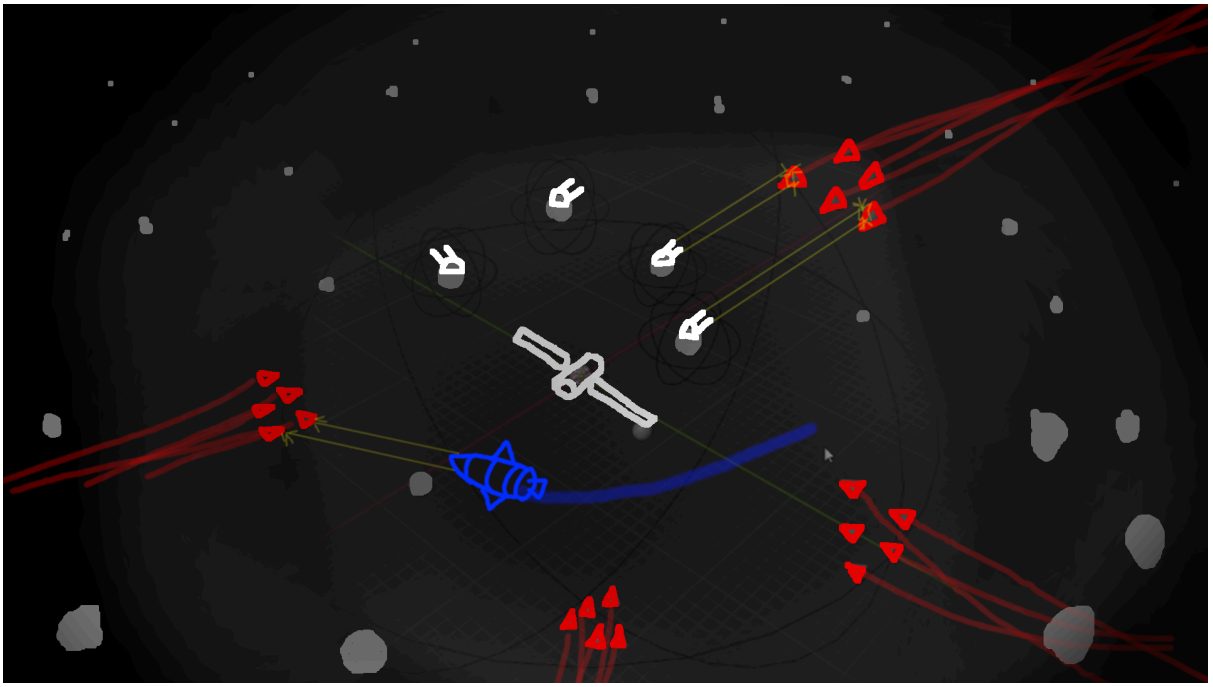
Overview

Terminal Rim is a 3D tower defense space shooter game set in deep space. It takes inspiration from both classic arcade games such as *Space Invaders* and *Asteroids*, as well as modern space simulation games such as *Elite: Dangerous.*

In the game, the player is tasked with defending a space station, which is being bombarded by alien spaceships. They take control of a lone starfighter and place sentry turrets. Their goal is to survive as long as possible, upgrading defenses as the alien onslaught becomes larger and more frequent.

This ties into the theme “Road to Nowhere”, as the player is isolated in deep space - the middle of nowhere, where they are forced into a last stand. As the number of opponents steadily increases, the aliens will eventually overwhelm the player. How long the player lasts is a matter of careful placing of turrets with tower-defense mechanics, and of quick decision-making skills, as part of action packed space shooter mechanics.

The gameplay mechanics are meant to be approachable by laymen, without extensive experience in the space simulation genre. As such, the systems will have a comparatively low complexity, such as the flight model, in order to not overwhelm the player. The technological and gameplay focus is on the large and ever-increasing number of enemies.



Birds Eye View of the Entire Play Area

Gameplay

Flight System

The flight system is designed to mimic that of an airplane during the first prototype phase, leaving out 6 degrees of freedom (DoF), or the x, y, and z coordinates to the ship's position and orientation. In addition, we will also try to get a 6 DoF flight mode that works without the use of two flight sticks and still offers a comfortable flight experience without feeling too difficult to control or learn.

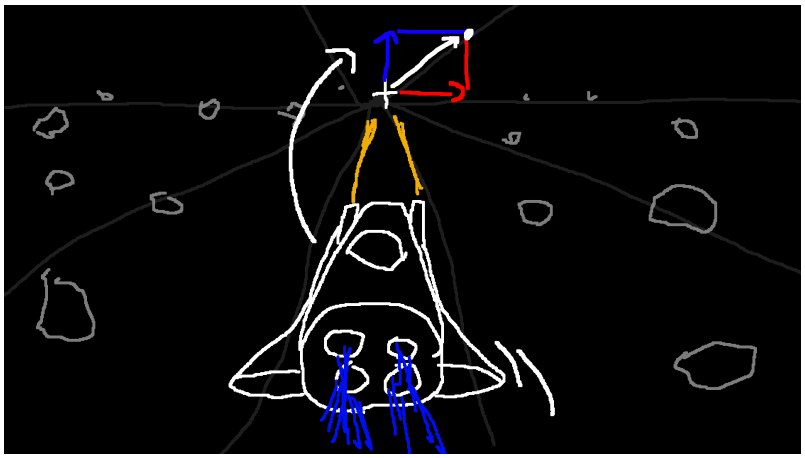
For a more realistic feeling, a simple physics drag and flight stabilization will be used to make the steering feel more engaging. While trying to follow the example of other games like *Halo*, *Arma,* and *Warthunder*, we are going to test three different steering methods for the player such Direct Controls, a Virtual Joystick and a Fly by Wire system. The direct controls work the same as any other first person controller, Mouse movement (mouse delta) is directly being applied as rotation to the player's spaceship. A Virtual Joystick, similar to the Direct controls will steer the ship according to the direction the cursor's normalized screen space position. Rather than rotating the ship only by a small amount the ship will stay angled until the mouse is moved back to the center of the screen. Finally, the "Fly by Wire" system uses a steering indicator positioned on a sphere around the ship that is controlled by the mouse. As long as the steering indicator is moved away from the center of the screen, the ship will steer in that direction until the ship is pointing in that direction.

Shooting

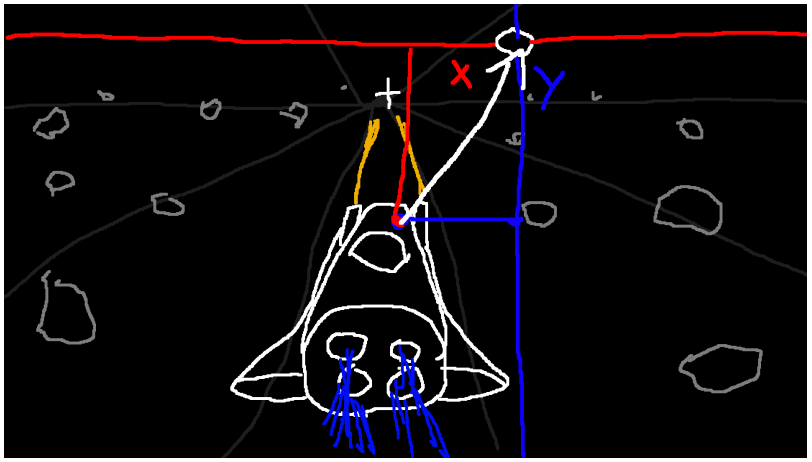
No space shooter is complete without guns, for which we aim to test three different control systems. By default the ship will have simple forward facing guns that follow the main crosshair. The player will then have to move the ship towards the enemy lead indicator to strike a hit when firing. A lead indicator is a visual representation, typically just a circle or a dot in other games, that shows the target's predicted trajectory, so that a physical bullet when shot will hit the enemy. To allow for better flight and firing control during combat, Gimbaled mode allows the guns to rotate towards the flight indicator rather than the main crosshair of the ship. This way, the weapon control is more independent of the flight direction and thus makes it easier to hit the enemies while trying to steer the ship.

Finally, to make the game more accessible, we also aim to build an auto-target system that allows the guns to follow the target lead automatically to make ship steering and gun controls fully separate. Auto-target will not cover a full 360 degree field around the player, but rather a larger area in front of the player while still limiting the guns in the general flight direction.

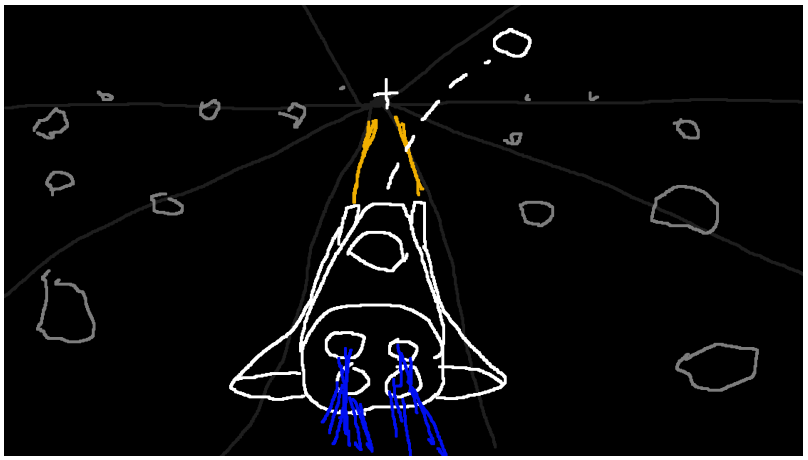
Mouse Control Schemes:



Direct Input

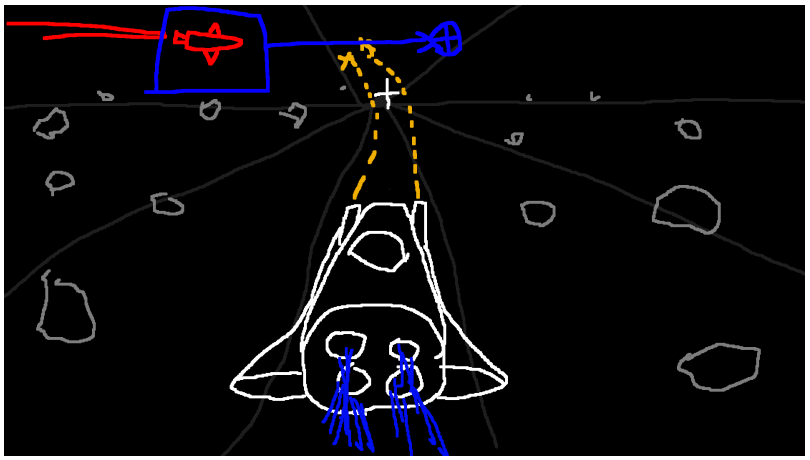


Virtual Joystick

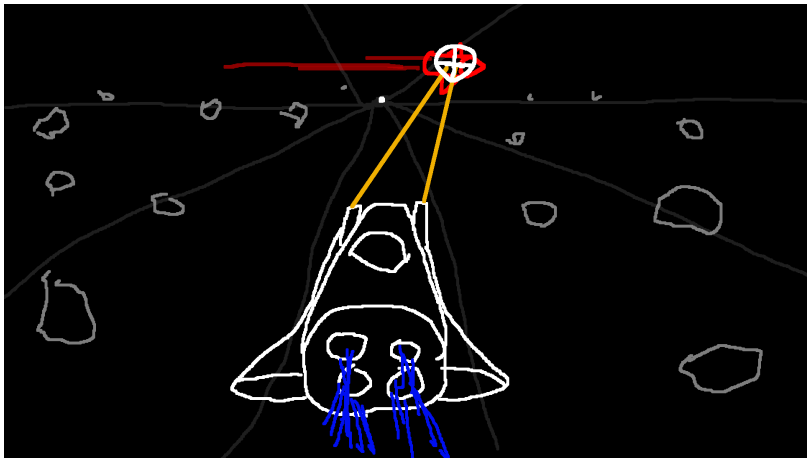


Fly by Wire

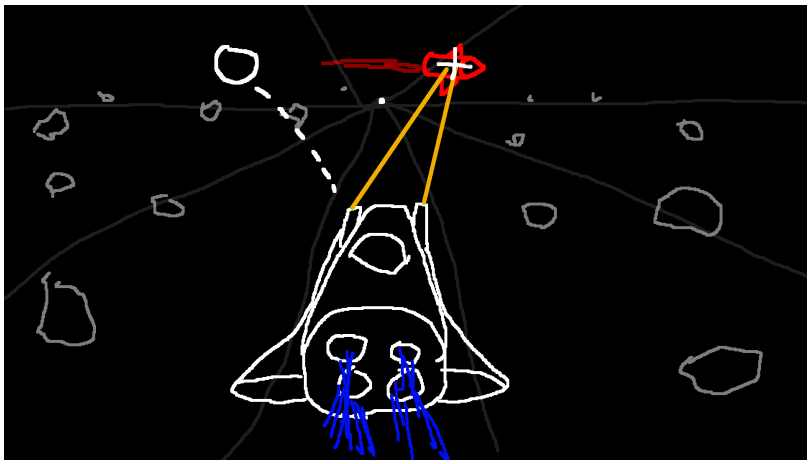
Shooting Considerations:



Manual Target Leading



Gimbaled Aim



Automatic Aim

Swarm of Enemies

Most of the enemies are built from a simple base enemy, adding variety by changing health, firing speed and flight speed. A main technical point of the game is the swarming behaviour of the enemies, for which we will use a BOID algorithm, simulating the flocking behaviour of birds and thus making a large group of enemies move in a more lifelike, natural way. Each wave of enemies that spawns will be in a group, approaching the center of the play area and attacking buildings along their path.

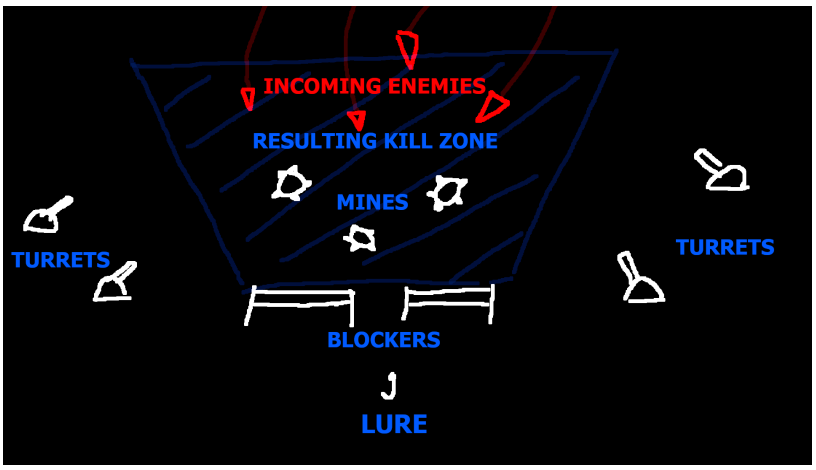
Enemy groups consist of different types of ships to make gameplay more varied and increase tactical depth for the player. For example, enemy ships can have a higher speed with lower health, making them a primary target due to approaching the center faster. On the flip side, lower speed ships with higher health approach later, making them the next target. These ship types keep the player constantly engaged. Other possible ship variations include Bombers that deal more damage to buildings and healing drones that can replenish the health of enemy ships to make strategic priority targets in a group of ships.

Tower Defense

For the tower defense part of the game, defensive turrets are an ideal feature. These can be placed by the player within the game area, around the central space station. Other optional buildings might bring some different functions: from area of effect structures, which repair surrounding buildings and resource gatherers, as well as lures for crowd control and defensive shields.

Surrounding the central structure are asteroids, which offer pre-defined placement locations for static buildings, like some of the aforementioned buildings. Additional asteroids can be bought and placed as needed near the station, offering more building placement options for the player to defend their base. Turrets and other smaller buildings, such as Lure devices and cloaks, can be freely placed in the surrounding space. Lures are used to steer groups of enemies into an area. Cloaks help the player hide buildings from enemy ships. Both buildings add an additional layer of strategic gameplay. Energy shields that have to be enabled by the player manually, lasting only a certain amount of time, also help in controlling enemy movement and defending player-built structures. All buildings offer different upgrades, such as rate-of-fire upgrades for the turrets, more armor on almost all buildings, cooldown timer upgrade for time-restricted building effects, different bullet types for the turrets, area of effect upgrades, and so on. All of these upgrades are endless, in order to emphasize the theme of road to nowhere and to make the entire game an endless tower defense game.

Strategic Combination of Towers



Resource Management

All structures and upgrades cost some form of currency. In our game, the currency the player can gather and spend is called ”scrap”. Scrap can be mined passively using the resource gathering buildings or earned by killing enemies. Having a resource the player has to manage adds another layer of strategy to the gameplay, as not everything can be upgraded all the time. Therefore, the player has to decide which buildings need upgrades and which buildings can be neglected for a while. Upgrades and building costs scale over time, so players need to earn more and destroy “higher level” enemies to afford more structures and upgrades. Having the option to also upgrade the mining speed of the resource gatherers adds a more casual side to the game and makes it playable as an Idle game.

Enemy Waves

The main focus of this game is the hordes of enemies which spawn on the outside of the play area and try to reach the center, attacking buildings on their way. Over time, enemy density will increase, making it more and more difficult for the player to defend their base. Every few waves a mini boss can be fought, earning the player some extra resources when defeated. Additionally, to the increase in enemy density over time, some variations of enemies will also spawn over time, making gameplay more varied and interesting.

As a stretch goal for players that would like to keep a more active play style an option might be added to manually increase the difficulty. This would allow action focused players to gather more resources and engage more actively with the game. For more casual players the game can be played like an idle game.

Ideally, engaging actively with the game also rewards the player after a while with a reinforcement ship in the form of a friendly “titan”, a large battle ship that can be controlled by the player, positioned within the play area and function as a mobile turret and defensive position. The titan ability, if implemented, would work on a cooldown timer and would not be constantly active. The titans cooldown is dependent on its remaining health. Therefore, it would be important for the player to keep an eye out for its health and strategically place it so that it is covered by other structures or healed by nearby repair stations.

Technical Achievement

The main technical achievement of this project will be handling massive amounts of enemy objects, their data, and their enemy AI. We plan to utilize either Compute Shaders or principles of data oriented programming to optimize the handling of many enemy objects at once, in parallel.

Both approaches to parallelization have their own strengths and drawbacks. Data oriented programming, as opposed to object oriented programming, separates functionality from data and stresses memory-efficiency and cache-locality. This programming paradigm is now natively supported in the Unity game engine by means of the Data Oriented Technology Stack (DOTS). DOTS consists of three pillars. Firstly a memory-efficient Entity Component System, which takes the place of game objects and components. Secondly a Jobs System for multithreading of the scripting pipeline. And lastly the Burst compiler, which is designed to achieve optimal performance of the Jobs System, by making best use of multiple CPU cores. This comes with the drawback of only supporting a subset of C#. DOTS provides a convenient framework, but entails a vastly different workflow than the tried and tested methodology. This requires a shift in development practices, which is likely to slow down development considerably.

The alternative method of parallelization is the extensive usage of compute shaders. The high number of GPU cores lends itself to the handling of a high number of objects, even though the added overhead might entail a noticeable performance cost. Previous experience in graphics programming makes this approach feasible. It is however, severely limited by the information available on the GPU, requiring custom scene serialization and object lifetime handling for complex game logic. Utilizing compute shaders as a custom scripting pipeline requires laboriously replacing much of the functionality of the existing game engine.

Consequently, Compute shaders can be effectively used for select, low coupling systems but are likely to cause unforeseen issues when used for game logic on a large scale. We plan on prototyping both the DOTS and the compute shader approach. However, data oriented programming seems to be a better fit for this use-case. Should this work better than expected, then it would leave development time for further experimentation.The game’s setting lends itself to the creation of custom visual effects. These include screen space lens flares or volumetric nebula rendering.

The “Big Idea” Bullseye

Development Schedule

Layered Breakdown

Functional Minimum

- Basic Tower Defense Mechanics
- Basic 3D Space Shooter Mechanics

Low Target

- Very High Number of Enemies
- Enemy Swarm System

Desired Target

- Resource Management Element
- Tower Upgrade System
- Arcade Flight System

High Target

- Enemy Wave System
- Ship Upgrade System
- Varied Enemy Types
- Varied Tower Types
- Boss & Reinforcement System
- Wayfinding GUI

Extras

- Manual Tower Control
- Twitch Integration for Enemy Waves
- Vfx & Shaders for Space Environment
- First Person View
- Multi-Monitor Support
- Flightstick Support

Task List

Milestone 1 - Game Idea Pitch

Task	Description	Who	Hrs
1	Brainstorming	All	3 h
2	Document Setup & Sketches	Jonas	2 h
3	Idea Finalization	All	4 h
4	Formal Game Proposal	All	8 h
5	Pitch Slides	Jonas	1 h
6	Presentation	Philipp	1 h
7	Mutual Project Critiques	All	TBD.

Milestone 2 - Interim Demo

Task	Description	Who	Hrs
1	Learning DOTS	All	15 h
2	Learning Unity Compute Shader Integration	All	8 h
3	Porting BOIDS to DOTS	Jonas	8 h
4	Porting BOIDS to Compute Shader	Philipp	8 h
5	Deciding on Technology	All	2 h
6	Building a Bullet and Health System	Philipp	2 h
7	Building a Basic Tower Defense System	Philipp	4 h
8	Building a Basic Spaceship Controller	Jonas	8 h
9	Make Concept Art	Thorya	10 h
10	Game State Management System	Thorya	3 h
11	Menus and GUI	Thorya	8 h
12	Putting it All Together	All	16 h
13	Prepare Playable Demo	All	2 h
14	Write Interim Results Chapter	All	10 h
15	Create Interim Demo Slides	All	4 h

Milestone 3 - Alpha Release

Task	Description	Who	Hrs
1	Improve Flight-Model	Jonas	4 h
2	Add Enemy Variation	Thorya	8 h
3	Add Building Variations	Thorya	8 h
4	Implement Building Upgrades	Philipp	4 h
5	Implement Resource Management	Philipp	4 h
6	3D Model Creation	Jonas	32 h
7	Texture Creation	Jonas	16 h
8	Sound Effects	Jonas	4 h
9	Particle Effects	Jonas	4 h
10	Putting it All Together	All	16 h
11	Prepare Playable Demo	All	4 h
12	Write Alpha Release Chapter	All	10 h
13	Create Alpha Release Slides	All	4 h

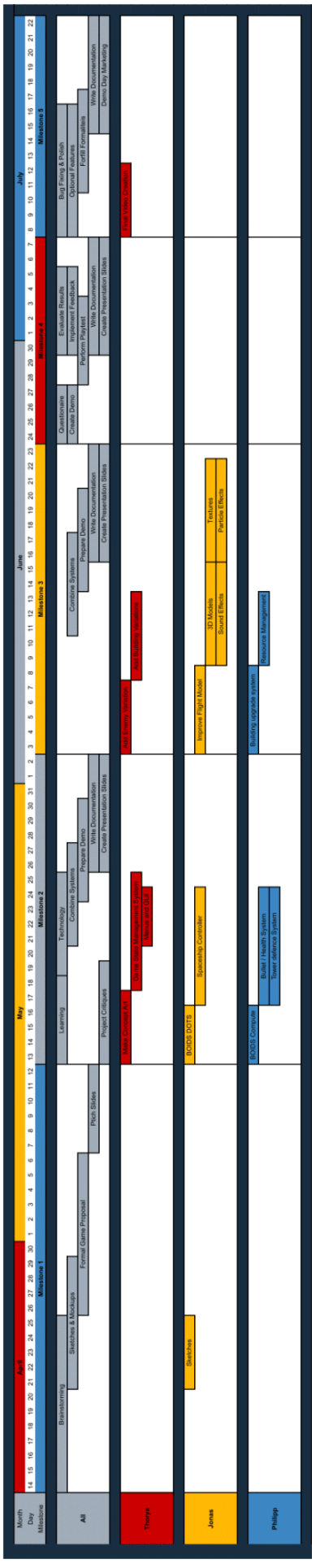
Milestone 4 - Playtesting

Task	Description	Who	Hrs
1	Implement Feedback from Interim Demo	All	16 h
2	Prepare Playable Demo	All	4 h
3	Create Questionnaire	All	8 h
4	Perform Playtest	All	8 h
5	Evaluate Notes	All	8 h
6	Implement Feedback	All	16 h
7	Write Playtesting Chapter	All	10 h
8	Create Playtesting Slides	All	2 h

Milestone 5 - Final Release

Task	Description	Who	Hrs
1	Implement Feedback from Playtest	All	16 h
2	Add Optional Features	All	X
3	Bug Fixes	All	16 h
4	Add Final Polish	All	16 h
7	Write Final Chapter	All	10 h
8	Final Release Slides	All	2 h
9	Final Video Creation	Thorya	4 h
10	Demo Day Poster	TBD.	2 h
11	Demo Day Slide	TBD.	1 h
12	Individual Contribution Emails	All	X

Timeline


Month
Day
Milestone
April
May
June
July
Milestone 1
Milestone 2
Milestone 3
Milestone 4
Milestone 5
All
Brainstorming
Sketches & Mockups
Formal Game Proposal
Pitch Slides
Learning
Project Critiques
Technology
Combine Systems
Prepare Demo
Write Documentation
Create Presentation Slides
Questionaire
Create Demo
Evaluate Results
Implement Feedback
Perform Playtest
Write Documentation
Create Presentation Slides
Bug Fixing & Polish
Optional Features
Forfill Formalities
Write Documentation
Demo Day Marketing
Thorya
Main Concept Art
Game Data Management System
Texture and GUI
Add Enemy Variation
Add Building Variations
Add Video Content
Jonas
Sketches
BODIS DOTS
Spaceship Controller
Improve Flight Model
3D Models
Sound Effects
Textures
Particle Effects
Philipp
BODIS Compute
Bullet / Health System
Tower defence System
Building upgrade system
Resource Management

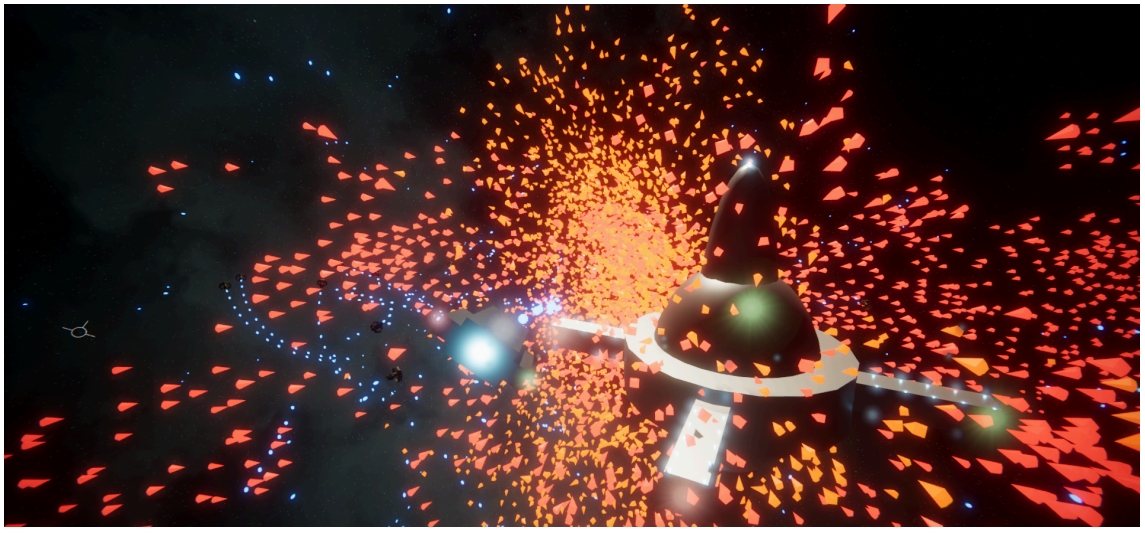
Assessment

The main strength of the game is the sense of a large-scale battle against impossible odds, conveyed mainly by the enemy swarm system, and made possible by the technical achievement. This scenario is reinforced by both gameplay layers: The space shooter element places the player in the middle of a one-on-many scenario, fighting a swarm of enemies directly. The tower defense element provides the necessary big-picture perspective.

The appeal of the game is based on the fantasy of being the last man standing, heroically holding off a numerically superior opponent through skill and cunning. This fantasy is made accessible to a wide range of players, due to the low skill floor of the arcade based mechanics. The sci-fi genre provides additional, formerly niche, now mass-market appeal.

The criteria for success consists mostly of effectively portraying the last-man standing scenario. It is the main artistic design goal to convey the intended emotional effect of a pinch of dread with euphorically glorious defiance. This does not necessarily require all of the gameplay mechanics discussed in this proposal. Some are, however, critical. The technical achievement is the primary one. A large number of enemies is necessary as part of the depicted scenario. As none of the team members are dedicated game designers, the technical aspect of the game takes precedence over the design and artistic aspects. A final product, with a functional technological implementation but lacking in polished design or emotional impact, is still considered a successful proof-of-concept.

Interim Progress Report



Typical Gameplay Situation

Layered Project Plan

The current status of the game is best described as a tech-demo. The main technical challenges are overcome. The technical achievement is fully functional and key gameplay systems are in place, but there is not yet a clear game-loop with interesting player decisions or a win-/lose-condition. The current progress is very satisfactory, building a solid base on which the next milestone can expand upon with gameplay variety and art assets.

Finished:

- Minimum Target:
  - Tower spawning, Shooting, Targeting Enemies
  - Player Ship with Flight Controls and Shooting
- Low Target:
  - Enemy Ships with BOIDs Algorithm for Steering
  - High Entity Count (Bullets and Enemies)
- Partial High Target:
  - Support for Enemy Variants
  - Support for Turret Variants

Not finished:

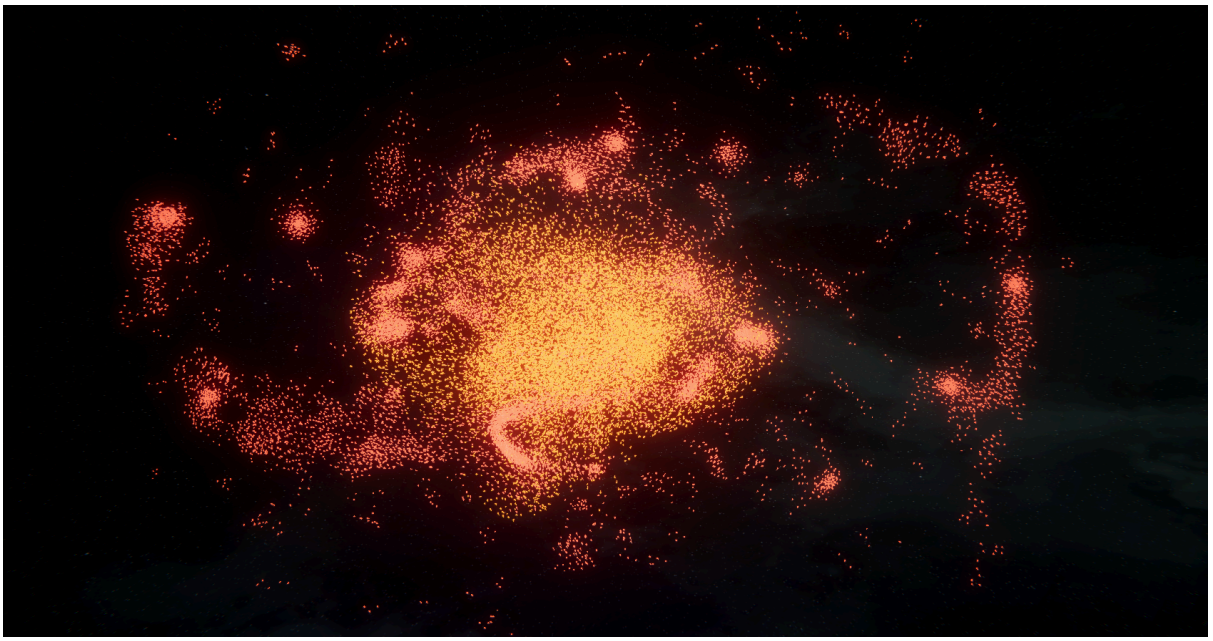
- Desired Target:
  - Resource Management
  - Upgrade System for Player and Turrets

Specific Systems

BOID System

The BOIDs system is fully implemented, as defined by Reynolds' paper. Additional steering behaviours are planned to be implemented, in the next sprint, when creating enemy variations. An underlying system for different enemy types, with varying steering behaviour weights is already in place.

The implementation heavily benefits from DOTS. A reference implementation with the traditional game object approach can handle around 300 BOIDs at fluent framerates. The entity component system (ECS) improves this number to around 500, with the highly optimized Burst compiler improving performance further. The most critical optimization is multi-threading by the use of the parallel Jobs-System, allowing enemy numbers as high as 30 thousand, surpassing our goals considerably. When integrated with other gameplay systems, this number drops to around 8 thousand, and in gameplay on low-end hardware two thousand is more practical. This still fulfills our technical achievement, but is planned to be further improved by the use of a spatial partitioning system, such as a hash grid, which allows efficient queries about close enemies.



BOIDs Simulation with 30000 Units and two Unit Variations

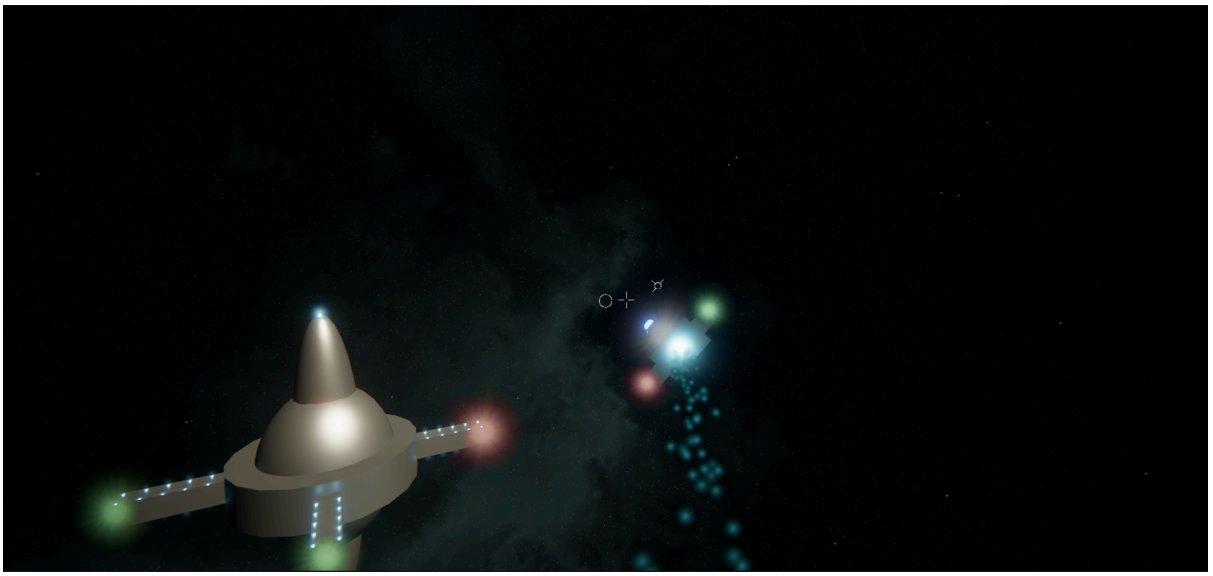
Spaceship Flight

All three planned flight modes are implemented. There is no obvious best choice, with personal preference varying based on the player's background. Consequently none of the flight-modes will be discarded, with a settings menu allowing the player to choose being planned for the following milestone. The current default mode is Fly-by-wire, because of its beginner friendliness, although it is the most unorthodox and therefore unfamiliar to veterans of the space-sim genre.

The spaceship's controls are physics based. They do not directly change the velocity, as is common in arcade-style systems, but apply limited forces. With this approach it is more

challenging to create a responsive system, but enables external influences such as knockback from explosions. The physical approach also leads to yet unsolved difficulties with gimbal lock, due to euler rotation representation of angular velocity of Unity's physics engine.

So far only the implemented weapon fire-mode is forward facing. This makes the most sense from a user interaction perspective, when considering the flight mode, focus on deliberate heading changes. Other fire-modes will possibly be implemented in the next milestone, though this has a low priority.

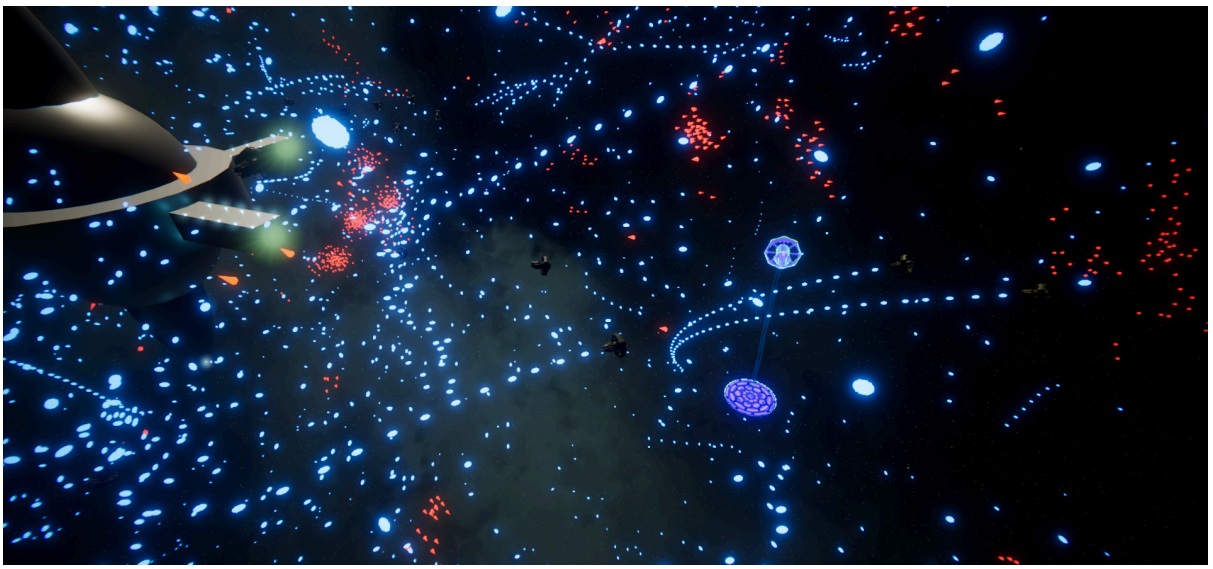


Player Spaceship in Flight - with HUD, including Fly-by-Wire Target Direction, Weapon Crosshair and Heading Indicator

Turret System

Defense turrets are an essential part of a tower-defense game. To that end, a system for creating different turrets with individual settings was created. Each turret shares the same table of base parameters, making it easy to introduce new variants to the system by simply adjusting values, such as: firing rate, multi shot, bullet spread, and so on.

While in "strategy mode", players are able to place turrets by first selecting the type of turret they want. The selected turret will then appear as a hologram at the cursor location on the "ground" plane, at the mouse cursor's location, with a vertical offset of 0. Using the Left-Shift key and moving the mouse up or down, players are able to offset the turret on the vertical axis for placement options in 3d space. With the Left mouse button, the turret's location is set, and it will spawn where the hologram preview was located. From there, the turret will pick targets and try to destroy them. The bullets, like all projectiles, have a travel time, so the turret has to slightly lead the target by aiming ahead of the enemy. The occasional misses add to the cinematic look and feel of the game when a lot of enemies are on the screen, and the player's turrets try to defend the station from a neverending onslaught of incoming ships.

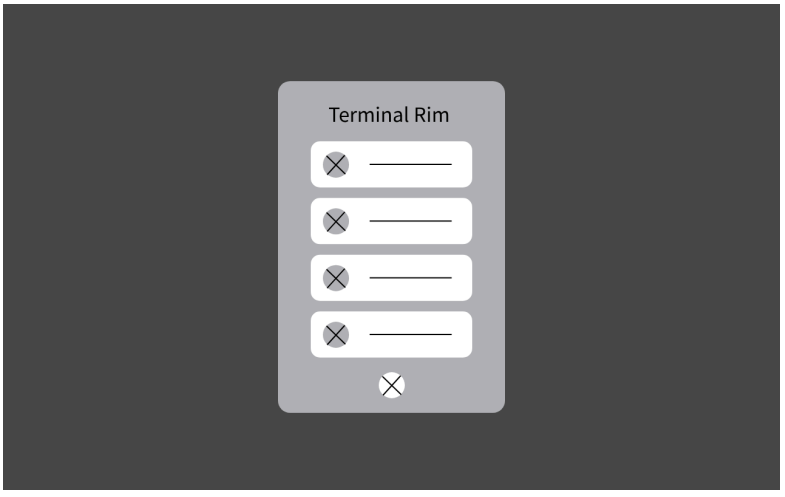


Chaotic Gameplay Situation in Strategy-Mode, Featuring a Large Number of Turrets and Projectiles

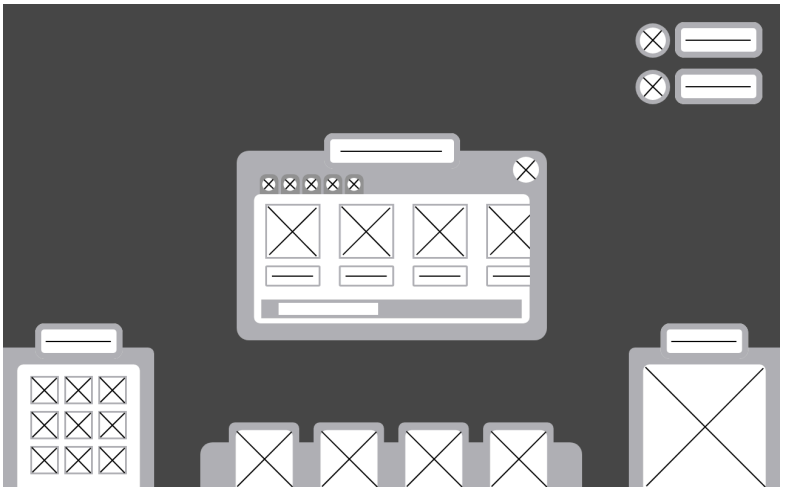
Graphical User Interface

Between the spaceship flight mode, building mode, menu screens, and other features, the GUI has to be carefully planned to organize all the information and functionalities necessary for the user to play. Because the main features of the gameplay are space flight and combat, it's important to incorporate GUI designs that are not only practical, but compact, in order to allow as much visibility of 3D space as possible.

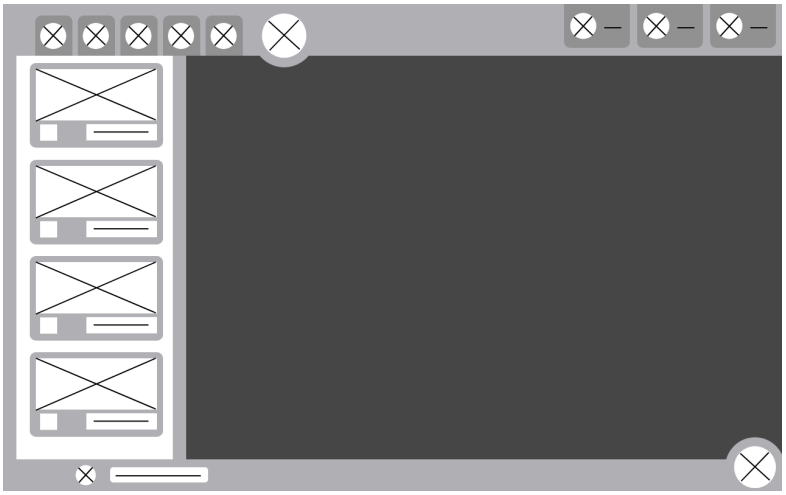
Before creating GUI blockouts in the game itself, we studied the GUIs of existing space games, such as *Spore*, and games with compact building edit modes, such as *Stronghold*. From these, we created several wireframes that closely follow the tried-and-true layout of these successful games. After compiling these designs, we assessed the strengths and weaknesses of each to determine exactly what our users should expect from our GUI layout and the specific traits from each wireframe that we would like to recompile into a new, unique design.



Wireframe for simple main menu design

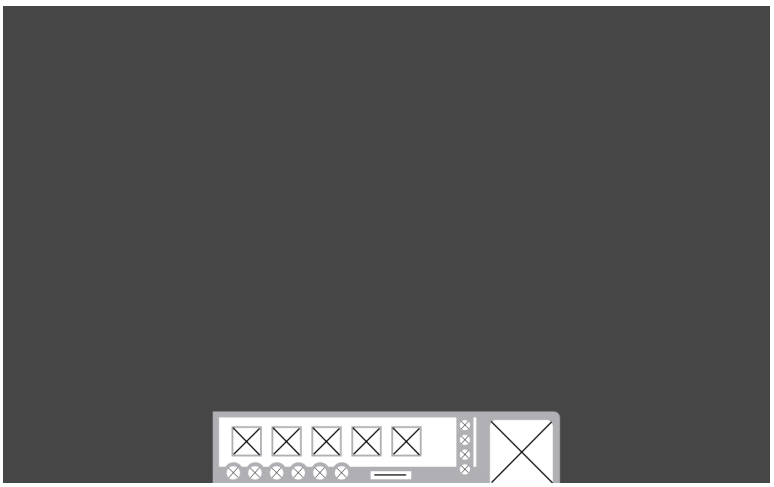


Wireframe for an edit mode and “shop” menu.



Wireframe derived from the Spore building edit mode

Wireframe derived from the Spore space travel mode



Wireframe derived from the Stronghold building edit mode

In the game itself, we created basic blockouts for the scene menus, such as the main menu and pause menu. Basic buttons allow us to start the game, pause it, return to the main menu, and end the current session, and exit the application. Additional management scripts also serve as the foundation for the game-state management, UI management, and session life cycles.

Challenges of Unity DOTS

To realize our space tower defence game, Unitys DOTS (Data-Oriented Technology Stack) have proven to be the perfect choice for implementing and handling a large number of entities. Unity's ECS & Jobs system offers multiple ways of handling and working with a large number of entities in a multithreaded way. This allows for a performant implementation of the BOID algorithm used for the enemies and the high number of bullets fired by the numerous towers defending the player's base. While DOTS has been around for some years now, documentation is both lacking and often outdated, as the framework is both unfinished and subject to change. This made it difficult during the development to find the functionality needed for certain systems, often resulting in some "work arounds" as documented functions simply don't exist or do not work as expected. With the official documentation proving unreliable, the most useful and somewhat up-to-date source of information on this topic are community posts on forums.

To aid the performance, multithreading using the Jobs system was used, resulting, unknowingly, in many resource contention issues and race conditions. Unfortunately, the error logging in Burst compiled code is not all too informative when it comes to relaying the sources of errors. This resulted in an extensive search following an error resulting from potentially unsynchronized access to data. At the time, this could have been almost anything. After some trial and error, the issue was tracked down to the Jobs and their scheduling. For updating entities in a multithreaded way, a EntityCommandBuffer was used. At first it was assumed that mixups in order of execution should have been automatically avoided, as commands are issued into this buffer to bec executed later. However, some of the assumed error handling on Unity's side was not provided, resulting in illegal read and write access errors. This was fixed by manually setting the update order of the system

components by hand, ensuring up-to-date data is used in each process, including entity life-cycle handling, preventing operations on already destroyed entities.

While adding functionality for physical collisions Unity's physics library for ECS is the official way of adding physics interactions. Collisions, triggers, rigid bodies and many of the by default supported physics interactions should be compatible with ECS and get API access in DOTS. This library is however also missing functionality or missing the related documentation. Trying to get the library to work only resulted in the default Unity physics breaking. As a large physics library is not required for the game to function, a custom collision detection system was implemented to handle simple collision checks.
As performance is very important in the project, DOTS was one of the two proposed options to deal with this issue. The alternative was compute shaders. After some deliberation, without needing to extensively test compute shaders, it was decided to only use the DOTS system as it offers more functionality in an easy to use fashion then compute shaders. Additionally due to the time constraints and the time needed to read up on DOTS and ECS a compute shader variant was impractical to implement, due to the additional effort of integrating it into other gameplay systems.

Alpha Release

Layered Overview

The game is now a playable prototype. It is fully feature-complete. The planned gameplay elements are implemented, but require polish and some balancing adjustments for a fair gameplay experience. The placeholder programmer art is now fully replaced by finished assets.

Finished:

- Desired Target:
  - Resource Management
- High Target:
  - Varied Enemy Types
  - Varied Tower Types
  - Enemy Wave System
- Extras:
  - Visual Effects

Change Management

Some tasks from the previous milestone were not yet completed, notably the menus and concept art. As they need to be completed before others, some tasks had to be reassigned for efficient work-load balancing. While Thorya worked on catching up these tasks to the target timeline, Philipp and Jonas worked on further implementing tower and enemy variations, alongside their originally planned tasks. The 3D art was in danger of also being delayed, because it was dependent on the concept art. So Philipp also took over the creation of the particle effects, allowing Jonas to put additional focus on the 3D assets to finish on schedule. As a result of this reevaluation of task priorities, the planned but optional upgrade system and advanced flight model were cancelled.

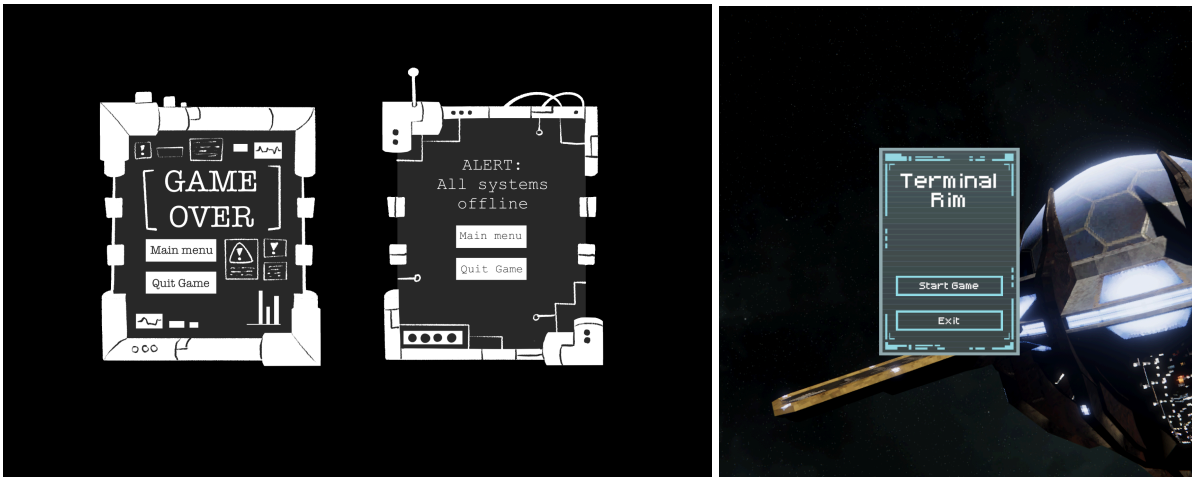
Art

Graphical User Interface

The tasks left over from the previous milestone for the GUI are complete; the user interfaces are now polished and fully functional. They include a main menu, pause menu, game over menu, edit mode menu, and an asteroid info menu.

After analyzing the wireframes created from existing game interfaces, we weighed the pros and cons of each example. Then, we created newer, higher fidelity mockup designs using the traits that would prove advantageous to our game UI’s specific features and

functionalities. After a couple of iterations and reviews, we finalized our design and adapted our other interfaces to match the style, colors, and overall look.



Higher fidelity mockup designs, and the resulting design implemented into the game

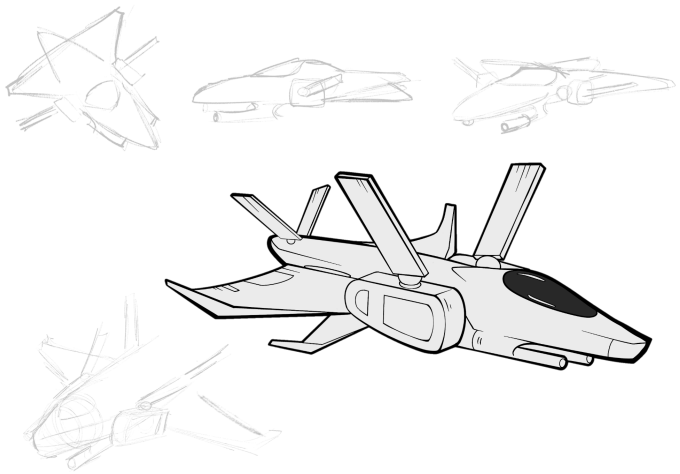
Example interfaces: the Edit Mode for placing turrets, and the asteroid info display.

Concept Art

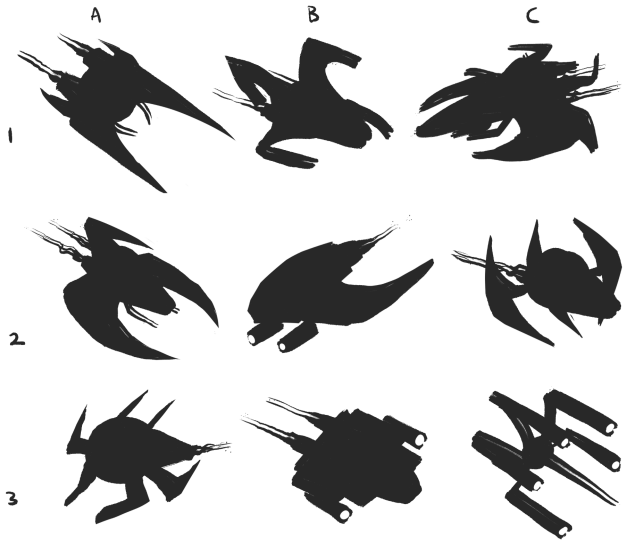
All concept art, up to the High target items, is complete. Designed assets include the player spaceship, turret variations, enemy ship variations, and building variations.

Color theory was taken strongly into consideration when designing assets. Enemy ships, projectiles, and visual effects are strictly warm colors (red, orange, pink); in contrast, the player's ship, buildings, projectiles, and visual effects are cool and neutral colors (blue, steel blue, gray, white). This is advantageous to the user, as it allows the user to easily discern which entities amidst the chaos are friendly or harmful.

In addition, the shapes of the ships and turrets differ wildly, depending on friend or foe; enemy ships are sharper with narrower edges. Friendly entities are more rounded, with smooth edges and corners. This was accomplished with silhouette ideation, which is an ideation method that focuses on unique, easily discernible shapes, rather than fine details.



Player-controlled spaceship ideation



Enemy ship silhouette ideation

3D Assets

All the game's art is custom made. To facilitate easy visual identification of enemies and allies, each turret and swarm has its own 3D model and corresponding textures. The chosen art style consists of low polygon count meshes and physically based materials with pixelated textures. This style is both comparatively time efficient to produce and is evocative of the arcade games, which inspired this project, whilst still making use of modern high-fidelity rendering techniques.

Custom Made Assets in the Chosen Low-Resolution, High-Fidelity Style

Sound Design

Audio is critical for the player's sub-conscious perception of a game. Therefore, every common player and enemy action is accompanied by a custom sound effect. Special weight is given to high-impact situations, such as the arrival of a new enemy swarm, to direct the player's attention. Especially often played sounds, such as the player-ship's main gun or engines are additionally procedurally adjusted to provide interactive feedback.

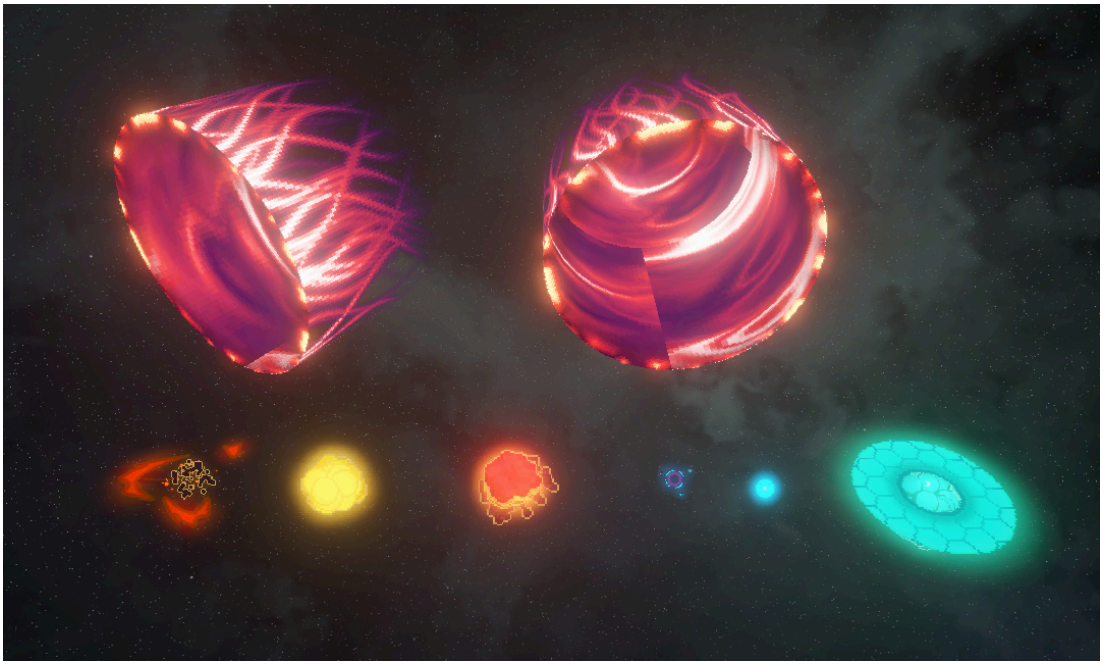
The chosen style for the audio follows the same design philosophy as the 3D-assets: it is a combination of realistic samples with 8-bit synthesised sound bites. This also partially mitigates audio mixing issues introduced by limitations of the Unity game engine. Its limit on the maximum number of sound sources and the inability of *DOTS* to interact with the engine's sound system, proved an unexpected obstacle.

Visual Effects

Visual feedback is critical for the player's actions, such as successfully defeating an enemy. To achieve this, several different visual effects provide the player with the necessary information of what is going on. For example, a simple explosion particle effect is played when enemy ships die. When they are hit, a blue impact effect is played at the point of impact, giving some information and visual weight to the successful hit, which is absent if the projectile simply disappears. To distinguish when one of the player's structures has been destroyed, a different explosion effect, also in blue, is played, making the situation more obvious to the player.

It is also important for the player to see when and where new enemies have appeared. For the enemy spawn in effect, a portal appears in the asteroid belt surrounding the space station. This portal effect is large enough for the player to see it when facing in the general direction of the spawn point.

At the end of the game, when the space station has taken a critical amount of damage, the death animation is triggered. During this animation, several smaller explosions appear around the space station, followed by one large, final explosion before the user is redirected to the Game Over screen.



A few stills of some of the visual effects, explosions, impacts and enemy warp in effect

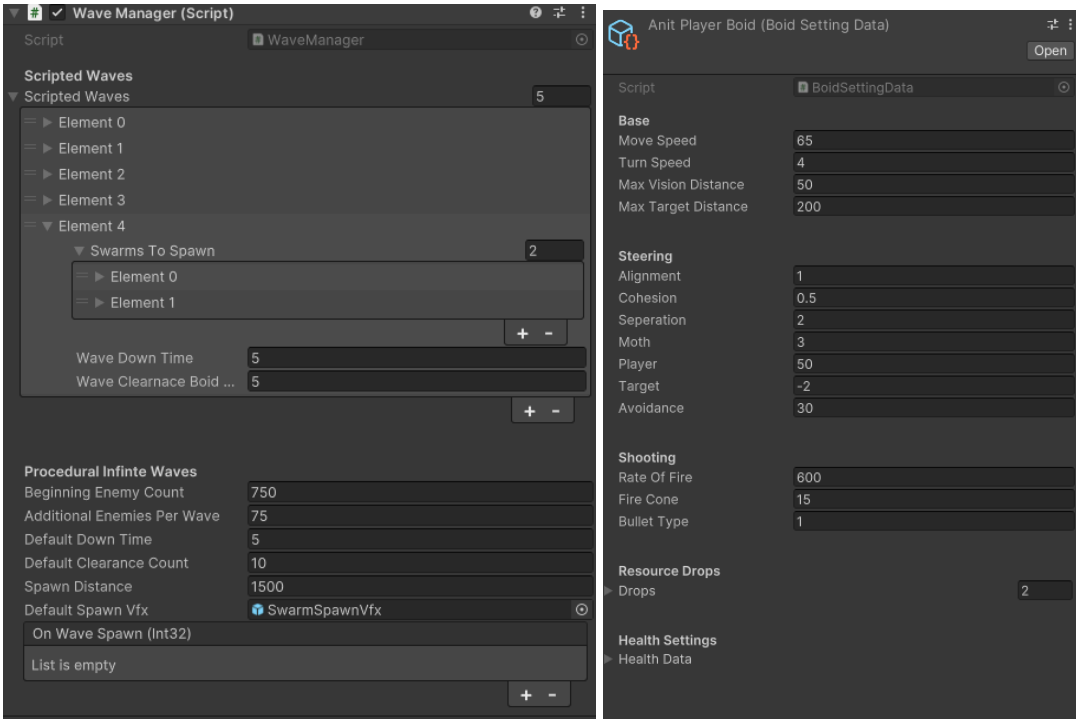
Specific Systems

Enemy Variations

The enemy swarming system now supports additional steering behaviours. They mostly consider target selection. Different enemy types prefer to attack different targets, such as the turrets, the main station or even the player. This allows for gameplay variety, designed to target specific weaknesses in the player's defenses. The other AI improvements concern the avoidance of colossians with scenery and the shooting of projectiles aimed at the respective targets.

Waves of Swarms

Additionally a potentially endless swarm spawning system incrementally increases the difficulty. This works first and foremost by adjusting the type and number of enemies. The system allows sufficient down-time to repair defenses through the procedural adjustment of the timing of waves by detecting the current number of active enemies. The procedurally spawned waves are randomized within tightly designed constraints to provide a fair challenge. These automated endless waves are preceded by a few manually scripted waves, acting as a tutorial introduction.

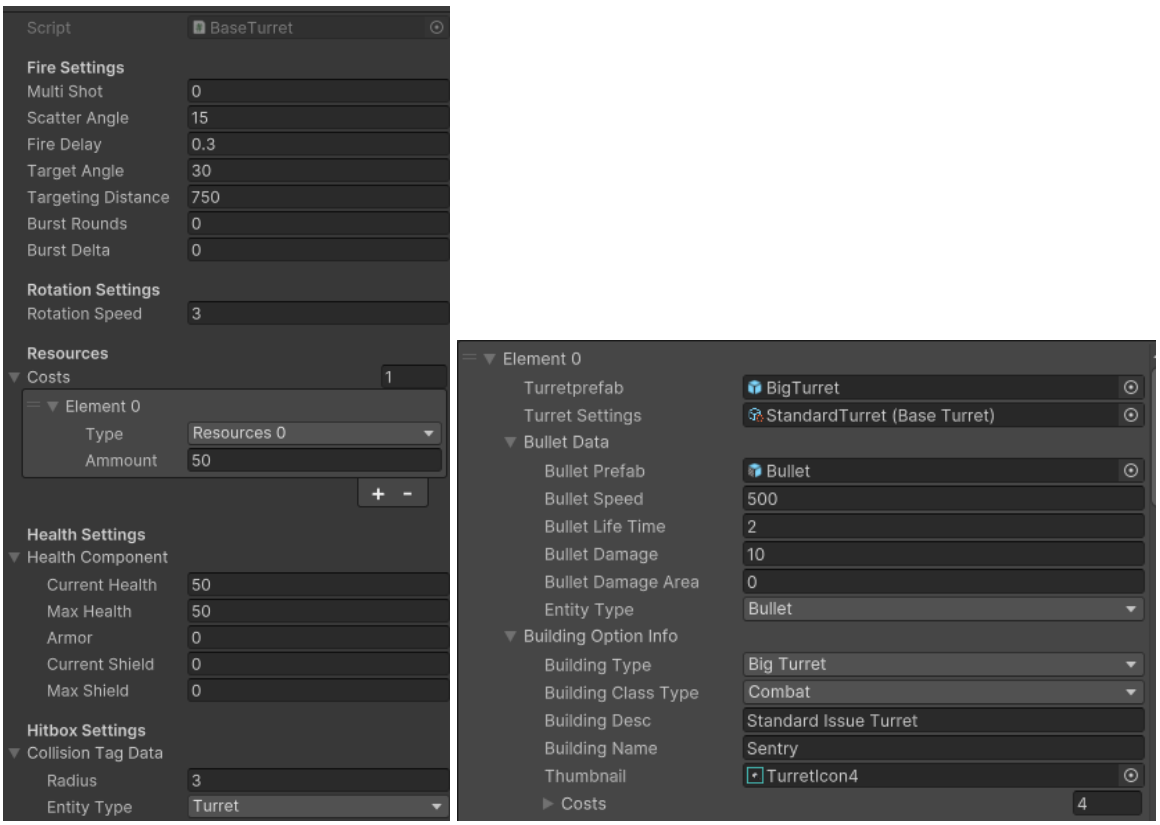


Settings of the Wave System with Both a Scripted Tutorial and Endless Mode

Turret Variations

The implemented turret system allows us to easily configure new turret variants when needed. This makes gameplay more varied, as each turret can specialize in a different area, such as long range targeting or a more wide spread attack. Since we have multiple enemy types, having specialized turrets to deal with each variant makes the gameplay experience feel more layered and fleshed out. Each turret can be configured from a range of parameters. On top of changing the settings for the turret behaviour itself, each turret can have its own bullet type with adjustable projectiles. Combining these two factors gives an even greater amount of variation. To place turrets, the player has to be in the strategy mode, or the edit mode the user enters upon landing on the space station docks. From there, they can pick a turret from the Edit menu and place it in the world.

Turrets will take damage when enemies fire at them or crash into them. This will require the player to replace destroyed turrets after a while, keeping the player on edge as they constantly look out for the integrity of the defense line. Each turret also has an associated resource cost, only allowing turrets to be placed when enough resources of the required type are in the resource bank.



Overview of turret settings as well as bullet settings for the turret

Resource Management

Placing defensive or utility structures, i.e. turrets and resource miners, is not free of charge for the player, as it would otherwise make the core gameplay loop trivial. For this purpose, a resource system is in place to slow down the speed at which buildings can be placed and add an additional layer of strategy to the gameplay. The resource system is built to be easily expandable with additional resource types that can be added or changed as needed. The whole system is built around a central resource bank that holds the current resources and a ledger for future transactions. When costs are generated, they are written on the ledger and processed at the end of each frame. This makes communication between systems clean and simple.

Placing a turret works the same way as mining resources from an asteroid; both generate an entry on the ledger for their respective resource type that gets summed up at the end and added to the central resource bank. During gameplay, the player can view their current balance in strategy mode via the Edit menu at the bottom of the screen. Alongside resource costs for buildings, costs are also displayed when the turrets are selected.

Example of placing a turret, with the cost display

Playtest

Preparatory Work

Before the start of the playtest, some final changes were made to improve the user experience. These are not major design decisions, but mostly minor adjustments for better stability and polish.

Performance Improvements

To support a high number of enemies, a supporting data structure is needed to help in checking for collisions and distance checks for the boids. For this reason, a hash grid was implemented. Each entity that is supposed to be inserted into the hash grid gets a “GridEntityComponent” with the required information for it to be inserted correctly. At the beginning of each update cycle, all the tagged entities are inserted by the hash grid system and can later be accessed in other systems. Using the hash grid allows collision checks only to consider entities in their immediate vicinity instead of checking against all other entities. This drastically increased the performance from around and below 60fps to a constant 60fps and above (only rough measurements taken from development pc).
Another avenue for improved performance is a limit on the maximum number of active audio sources. The processing of multiple audio clips at once is fairly processing heavy, slowing down the already CPU limited game. Lowering the limit allows us to prioritize gameplay functionality in the frame budget, with marginally worse audio visual cohesion.

Bug Fixes

To properly evaluate the gameplay during the user study, the game has to be free of distractingly game-breaking bugs. To that end, we fixed a multitude of technical issues. Some stemmed from systems which were newly reworked for performance, causing various unexpected behaviour. Some concern the user interface not correctly reflecting the current world state. Most issues however, can be traced back to the Unity Engine’s DOTS systems. Specifically, its handling of the lifecycle of entities, singletons and buffers, is difficult to debug, because its editor behaviour is different to the one in the exported executable.

Gameplay Adjustments

Various minor adjustments to gameplay are introduced in this milestone. Most serve a more streamlined user experience, allowing people other than the player to play and hopefully enjoy the game without major hurdles. The simplest among these improvements is basic branding on the games webpage, launch screen, and most importantly, a launcher icon, making the executable easy to identify by laymen.
Up to this point, the properties of the enemies, turrets and wave system have been designed to allow for quick testing and showcase of features in the least amount of time possible. This is not conducive to a fun player experience. Therefore, these values are balanced for a gradual increase in difficulty.
The amount of enemies is reduced drastically at the beginning of the game, allowing the player to acclimatise to the controls and their objective. The resource types are assigned

degrees of rarity, in order to rank both turrets and enemies in a hierarchy by their cost. Furthermore, the cost of turrets and the resource-drop rate of enemies has been tuned to sequentially unlock the different turret types during the early enemy waves.
To give a feeling of achievement, a score screen at the top left corner gives an overview of the most important gamesters. Furthermore, not only scores are tracked, but also some relevant information about the current state of the game. This includes, most importantly, the current wave the player finds themselves in and other stats like lost turrets or destroyed enemies.

User Study

Method

Prior to playtesting the game, a testing plan document was drafted to specify the research goals, research questions we wanted to have answered, key performance indicators (KPIs), the methodology, a script, and the target audience for sourcing participants.

Key Performance Indicators

We had three primary key performance indicators (KPIs): drop-off rates, or the abandonment rate of tasks; the system usability scale, or the user-specified rankings for survey questions about usability and gameplay; and time on task, or the amount of time users spend to accomplish certain tasks.
We addressed these KPIs with the following research questions, which were the basis of the tasks we assigned to the user:

1. How easily does the user adjust to the control scheme when first playing the game?
2. How easily does the user navigate through the menus?
3. Are there any noticeable frustrations with the core mechanics of the game?
4. Are there features that the user would like to see included in the revisions?
5. Are there parts of the user flow where the user gets stuck?

The Script

Our script, or the instructions for the user's gameplay and corresponding prompts for the user to answer, was designed to address our research questions and provide demonstrable KPIs, leveraging both qualitative and quantitative results.
The script eventually was migrated to a user-facing google form, which transcribed the script into seven sections: introduction and privacy policy, starting the game, reading the instructions, playing a first full gameplay loop, playing a second full gameplay loop, closing the game, and a post-testing survey.
For example, the prompts from the first gameplay were as follows:

1. What were your first impressions of the game?
2. Are there any immediate issues that you noticed, based on your first play-through? Were some mechanics confusing, too easy, etc.? If so, please explain.
3. Did you encounter any bugs in this section? If so, please describe the bug(s) and what happened leading up to them. If you did not encounter any bugs, leave this question blank. (Optional)

4. How easy or difficult was it to complete this section?

It's imperative that these questions remain as neutral as possible and encourage the user to explain themselves, rather than risk leaving it as a simple yes or no.

The Participants

Participants consisted of nine people, with six following the formal research plan. Three users tested informally, with no recordings or user data taken. When gathering participants, we made sure to find people who are within the target demographic of our game. Namely, adults with an interest in video games. We also made sure to incorporate a couple of users with vision impairments, which helps in accessibility assessments, such as preventing issues with cybersickness and low contrast ratio in colors of the UI.

The Playtesting Phase

Our playtesting consisted of both moderated and unmoderated sessions. Both session types were provided with a privacy policy, which disclosed the use of user data, what data will be recorded, when the data will be erased, who will have access to this data, and a disclaimer that the user can opt out at any time by emailing us.

We had exactly one unmoderated study, and the user emailed screen recordings of their gameplay afterwards. Per the privacy policy, This user was instructed beforehand to avoid recording personally identifiable or sensitive data on their screenoid recording audio of their voice.

The testing itself followed the google form and its instructions; users were instructed to read each section in order, complete the assigned tasks, return to the form to provide feedback, then continue to the next section. Users were highly discouraged from skipping ahead.

At the end of the form, we introduced a questionnaire survey to gather quantifiable data, based on user opinions. We gave them a set of five options, ranging from "Strongly Disagree" to "Strongly Agree", and ten blanket statements about the gameplay experience. For example, one of the blanket statements was "I think this game is too easy", and the users responded with one of the five options, which corresponds to how much they agree or disagree with the statement. The results of this questionnaire were exported as pie charts.

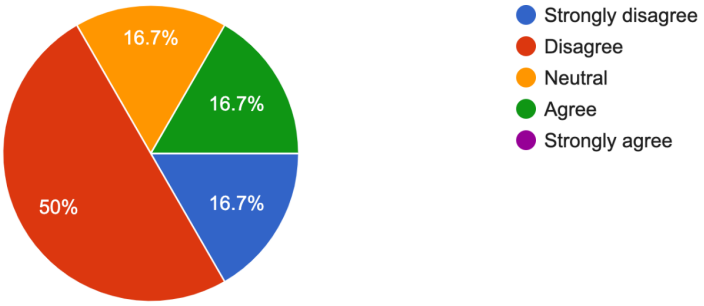
Compiling Feedback

After testing concluded, all user feedback was compiled, section by section, by addressing each prompt and any corresponding trends or complaints. For qualitative data, we analyzed each user's opinion and grouped them together by common concerns. If two or more users had similar recommendations or complaints, this was noted as a possible opportunity to improve.

For quantitative data, the google form exported the questionnaire at the end into pie charts. For each blanket statement, we determined the "ideal" response and how many users deviated by two or more points from this response. For example, one such statement was "I think this game is unnecessarily complex". Our target responses are "Disagree" and "Strongly Disagree", and we counted how many users responded in range "Agree" to "Strongly Agree".

"I think that the game is unnecessarily complex."

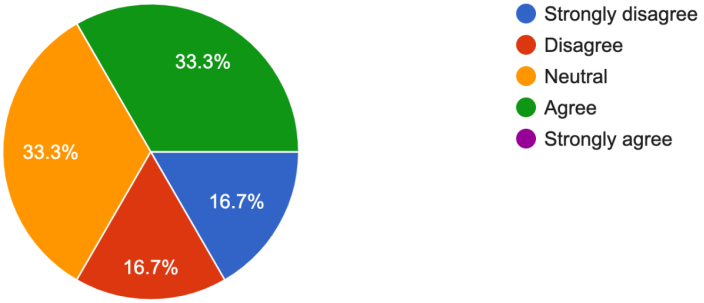
6 responses



Pie

"Searching and placing buildings and turrets was straightforward."

6 responses



charts exported by the Google form

Pattern Identification

After the feedback was compiled, a document was made for pattern identification. That is, when two or more users answered a prompt in a similar manner or chose answers that do not correspond to our target answer, we recorded this as a pattern. The following are our identified patterns, formatted with a statistic and corresponding takeaway:

1. It was observed that 4 out of 6 participants had severe issues with the turret system, specifically buying and placing turrets. This means that users are struggling to intuitively figure out or find information on the use of turrets within the game.

2. It was observed that 3 out of 6 participants cited dissatisfaction with the game's difficulty scaling in the Section 6 survey and post-gameplay recommendations. Specifically, the game is "too easy". This means that users are not satisfied with the rate at which the difficulty increases with each wave.

3. It was observed that 2 out of 6 participants felt that the instructions did not adequately prepare them for gameplay. Specifically, the shooting instructions are wrong, and very little was mentioned about turrets. This means that users require more accurate and detailed instructions in the modal.

4. It was observed that 4 out of 6 participants had severe issues with controlling the ship. Specifically, they cited issues with mouse sensitivity. This means that the mouse sensitivity will need to be significantly increased, or a settings page is needed for the user to adjust it themselves.

Actionable Insights

After identifying patterns and determining the largest pain points during testing, we were able to discern our actionable insights, or the key user habits and desires determined from playtesting that will serve as the foundation for future improvements to the game. The actionable insights were as follows:

1. Based on the theme that: the turret system is not straightforward, an insight is: users want to leverage the turret system but don't quite know how.

2. Based on the theme that: users are dissatisfied with the difficulty scaling, an insight is: users want to be challenged more during gameplay.

3. Based on the theme that: the instructions modal does not properly prepare users, an insight is: users heavily consider the contents of the instructions.

4. Based on the theme that: users are having issues with the ship due to mouse sensitivity, an insight is: users, especially during early game, rely heavily on the ship.

Results

Finally, after determining the actionable insights, we were able to pinpoint exactly what we need to do in order to improve the game and compiled these into a list of top priorities. There were many recommendations by users on how to improve the game, but of course, we do not have time to accommodate every single recommendation given by every single playtester. Therefore, using the pattern identifications and actionable insights, we determined our top priorities, which were as follows:

1. **Fix the user cues for the turret system.** Users need to know how to dock the space station, determine if they can afford a turret, know precisely what they can expect the turret to do, and place the turret in an obvious location.

2. **Adjust the game's difficulty scaling.** Once the turrets, ship controls, and instructions are fixed, we may find that early game is very easy. We should strive to make the game noticeably harder as time goes on.

3. **Fix the instructions modal.** Users rely heavily on its contents when they are first learning to play the game, and confusing/inconsistent instructions heavily frustrate them.

4. **Increase mouse sensitivity for flight, or allow the user to adjust it themselves.** Users entirely rely on the ship at the beginning of the game, as they start with 0 resources and can't buy turrets. It's imperative to make the ship flight experience intuitive and easy to manage.

Several bugs were noted by the players, namely a bug regarding the mouse cursor disappearing, resources persisting between playthroughs, and the turret edit mode purchase playing incorrect audio. Fortunately, because these bugs did not prevent any user from completing a full gameplay loop. We will still, however, fix these before release.

Everything considered, the results of testing were very promising. Users expressed great interest in continuing the game, with some voluntarily playing a third playthrough, even though it was not required by the playtesting.

Planned Changes

From the user a number of issues with a multitude of possible fixes emerge. While there is not time to address all of them, it is nonetheless important to document them for future reference.

Bug fixes

The most straightforward improvement is the fixing of the remaining bugs: the mouse cursor not properly focusing on the game's window, the GUI consuming input events during turret placement and the persistence of the resource bank through multiple playthroughs.

Rebalance Resources

Also concerning the resources is the need for a rebalancing of the late game. Currently the player acquires resources with every defeated enemy, thereby gaining power too quickly in a snowball effect, resulting in the enemy ceasing to be a threat and losing the need for the action flight mode. One possible fix to this issue is changing the resource acquisition, so defeated enemies only drop resources if destroyed by the player personally, not when destroyed by a turret. This limits the maximum amount of resources gained each wave and also incentivises a more balanced mix of action and strategy play-modes. Additionally the spawn-rate of enemies may increase exponentially, instead of linearly in order to pose more of a perceived threat in the late game. This has the side effect of reintroducing the game's main technical achievement into the gameplay.

Turret Placement System Update

Both play-modes should be streamlined in terms of user interaction. In strategy mode the turret placement menu interaction should be homogenized. The selection, buy and place actions should be in the same order using the same inputs, regardless of the building type. There should also be immediate feedback on the success or failure of such an action, which

might occur for example due to insufficient resources. Additional improvements to the building placement might include placing multiple turrets of the same type, without re-selecting the turret type each time and a pop-up window elaborating on the building's purpose and properties on mouse-hover.

Flight Control System Update

In action mode, the HUD and controls for the flight model should be simplified. The icon denoting the player ship's current velocity does not fulfill a major purpose and can be removed in order to not confuse players. The gliding should be improved for more plan-like movement, as playtesters vastly overused the lateral thrust feature. Alternatively the exclusion of a full 6 degrees of freedom might be reconsidered, adding vertical thrust and manual roll. More importantly all the different existing flight modes should be selectable in a settings menu, as well as adjustable mouse sensitivity and audio volume.

Addition of a Settings Menu

The addition of a settings menu with a mouse sensitivity slider would allow the user to control the sensitivity of the mouse during flight mode, which was a major pain point during early game. Fortunately, because we can already adjust this value via the Unity inspection panel, we should be able to easily map this value to an input.

Addition of Tutorial Hints

It was noted in playtesting that some users expected some form of tutorial when playing the game for the first time. It's one thing to read the instructions modal on the Main Menu; it's another to play the game itself. Tutorial information, integrated into gameplay when a new game starts, would help tremendously in introducing users to key gameplay mechanics in context of the game scene itself.

Final Release & Conclusion

Playtest Feedback, Fixes, and Final Changes

Following user feedback during playtesting, some bugfixes and quality of life changes were made to improve the user experience.

Tutorial

At the beginning of gameplay, we added a tutorial, which covers the basic controls, defeating enemy ships, and entering Build mode. The user is first instructed on the WASD ship movements, and after they provide the inputs, it teaches them about mouse rotation, shooting, and so on. Each tutorial step awaits user input and carries them to the next step of gameplay mechanics. This ensures that the user is fully informed and practiced on the controls and necessary features to progress in the game.

In-Situ Tutorial, Reacting to Player Actions

Turret System

Tying into user concerns about gameplay controls is the turret system. To improve the interface and make buying turrets more intuitive, we added a negative sound indicator to the turret selection to let them know audibly when they can't afford the transaction. When a user attempts to select a turret they can't afford, it will play a low-pitch rejection noise and refuse to preview the turret placement. When the user can afford the turret, however, it will play a much more cheerful, higher pitched sound and preview the shape of the turret as a holographic blue object in valid turret placement locations, which are now filtered using

collision checks with preexisting turrets. The user can also place multiple copies of a turret now, without having to repeatedly re-select the same turret type on the edit mode interface, streamlining the process. Additionally when selecting a turret type, an info-window pops open providing the player with the turret's stats, to facilitate informed decision making.

Rebalanced Difficulty Scaling

A major issue discovered during the playtest was the game balance. Particularly, the snowball-effect of the player's resources made it much too easy in the late game. Among numerous changes in the fine-tuning of the resources-costs, drop rates, effectiveness of turrets and enemies, some systemic changes are now also in place.
Notably, resources can now only be earned, if the player themselves eliminates an enemy, not if a turret does so. This limits the maximum amount of resources earned per wave to a fairly constant maximum, independent of turret or enemy count. Consequently, the player's resources and therefore number of turrets increases linearly, instead of exponentially as it did before.
Another systemic change is in the selection of the type of enemy waves. Previously, the procedural waves were created randomly, providing variation, but inconsistent challenge. Now the wave management system takes into account multiple factors, such as the number of active turrets or player and station health. The system then spawns a wave, fitting the player's tactical situation. For example, if there are a lot of active turrets, a wave targeting them is spawned. If on the other hand there are only very few turrets, the wave targets the player ship or station directly. This system also takes into account desired difficulty level, slowly increasing the number of enemies, but also overriding this value for individual waves, should the player be in an unexpectedly strong or weak position.

Ship Control Refinements

Another pain point during playtesting was the ship's flight controls. Players were unfamiliar with and overwhelmed by them. Their default behaviour was to strafe back and forth and fire continuously while nearly stationary. To simplify the HUD, the current velocity indicator is removed, limiting it to only the crosshair and heading. This should help remedy some of the confusion. Additionally, the strafe speed and drag are reduced, incentivising forward gliding, thereby encouraging the full use of the fly-by-wire flight system. Additionally, the mouse sensitivity can now be adjusted for a more comfortable user experience.

Settings Menu

A Settings menu was added to the Main Menu and game scene Pause Menu, where users can adjust mouse sensitivity and the volumes of each audio source in the game (ambient, enemy ships, UI, etc). The mouse sensitivity ties directly into ship controls and allows the user to customize this input to suit their needs. These settings persist between the Main Menu scene, Gameplay scene, and application sessions, so if the user quits the game and plays at a later time, they will not need to re-adjust their settings.

Additional Polish

Additionally, some other improvements were made, which were not a result of direct feedback from the playtest, but from our own observations and reasoning about the player experience.

High Score System

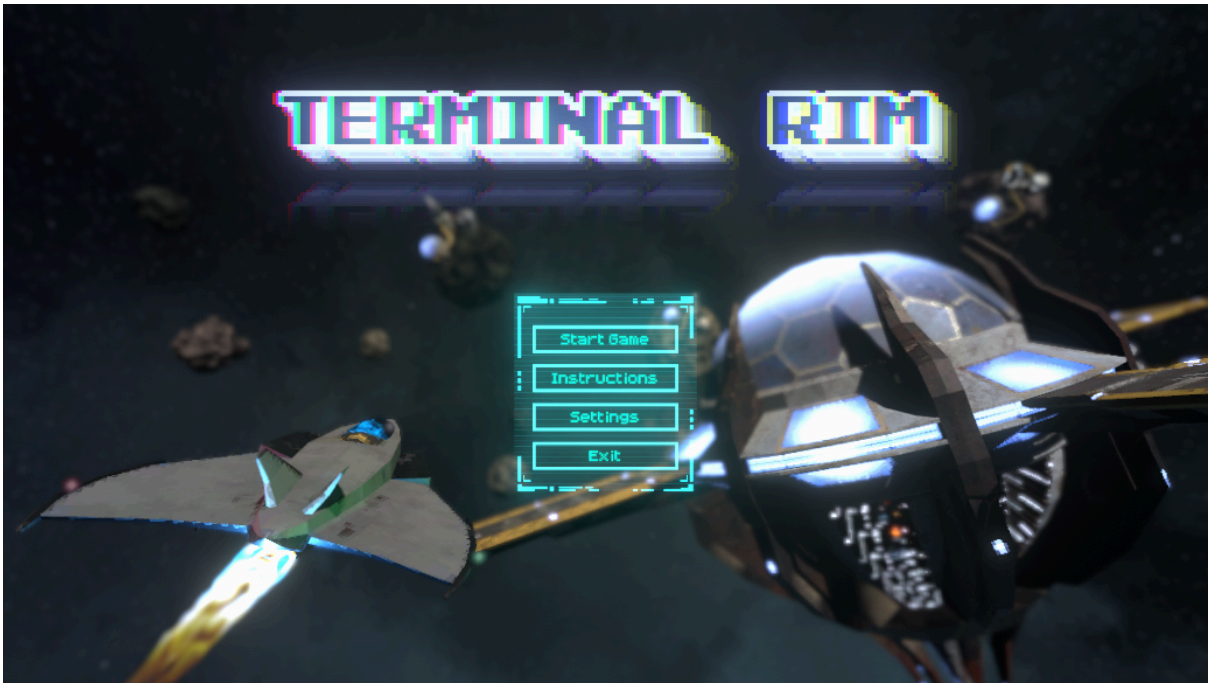
The change with the largest impact is the addition of a highscore system. It serves as external motivation, and validation. Its aim is to encourage multiple playthroughs by making the end of the gameplay loop more satisfying. To that end, the score display makes use of multiple animations, inspired by arcade and slot machines.

Health Bar Animations on Hit

Similarly, the health bars of the player ship and station now have animations. They shake, when the player receives damage. This is an additional visual indicator of change and draws the player's attention in a potentially chaotic situation. Where the player might have previously missed the visual queue, they now understand their reason for failure.

Visual Improvements

Finally the background of the main menu and game-over menu are now filled with cinematic scenery. This sets the mood, further communicating the setting and prepares the player for the gameplay situation. This change also makes the game look of higher quality than before, with a more rounded presentation over all.



Main Menu, with Background and Settings

Game-Over Menu with Background and Highscore

Release Day

For the final release on the Demo-Day, the presentation of our game was prepared, focusing mostly on branding. The game appears in a unified style and with consistent quality on the itch.io page, as well as the demo day poster and one-minute-madness slide.

One-Minute-Madness Slide

Demo-Day Poster

Conclusion

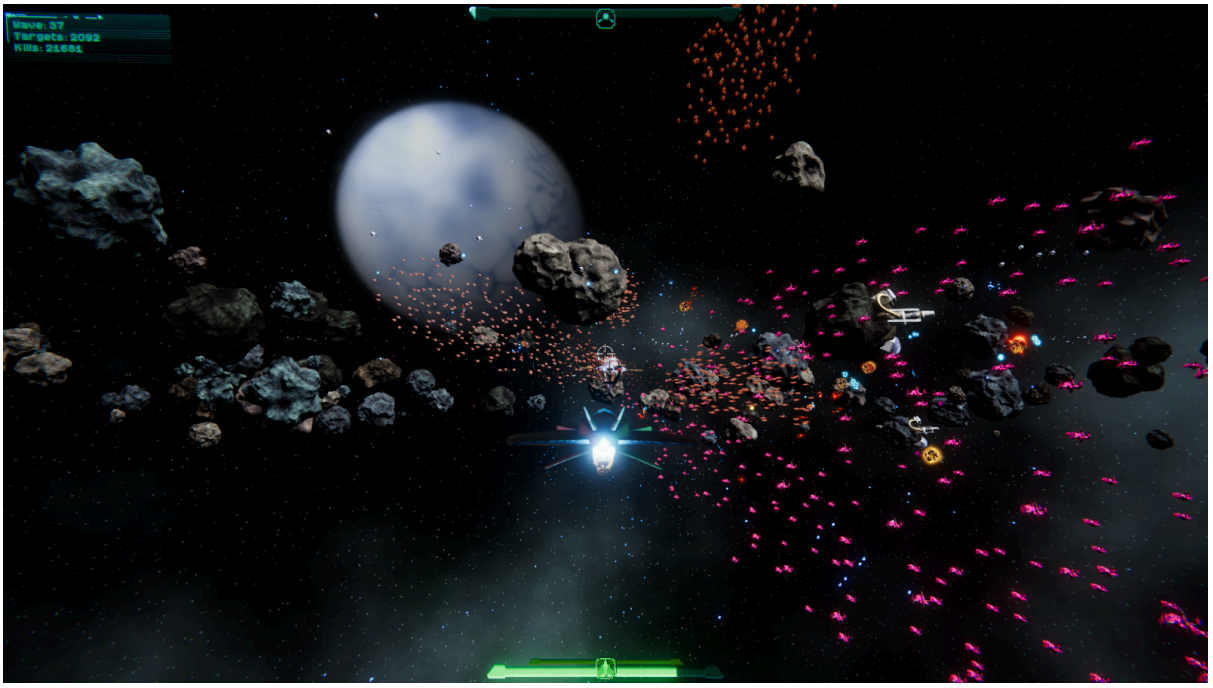
State of the Game

The state of the game is slightly better than expected. The initial design ideas materialized well into the final game. Our layered development schedule proved to be realistic. While some minor omission exists, notably the upgrade system, overall the scope is where we expected it to be and the level of polish is slightly higher than expected.

The development schedule was mostly adhered to, with only minor exceptions of tasks in milestone 2, which were completed early in milestone 3, requiring some rescheduling. Some tasks were reassigned to other team members and multiple tasks turned into a group effort. Particularly debugging systems where features of multiple team members intersected profited from pair-programming.

The project structure initially seemed overly strict, similar to the project management waterfall model, but later proved to be invaluable. It kept the team on track and provided a useful frame of reference for prioritizing tasks and deadlines. Particularly the inclusion of a pure prototyping phase gave the team the liberty to experiment with different workflows of DOTS, which would have been costly if done late in development. The playtesting phase proved useful as well, providing both humbling and encouraging feedback.

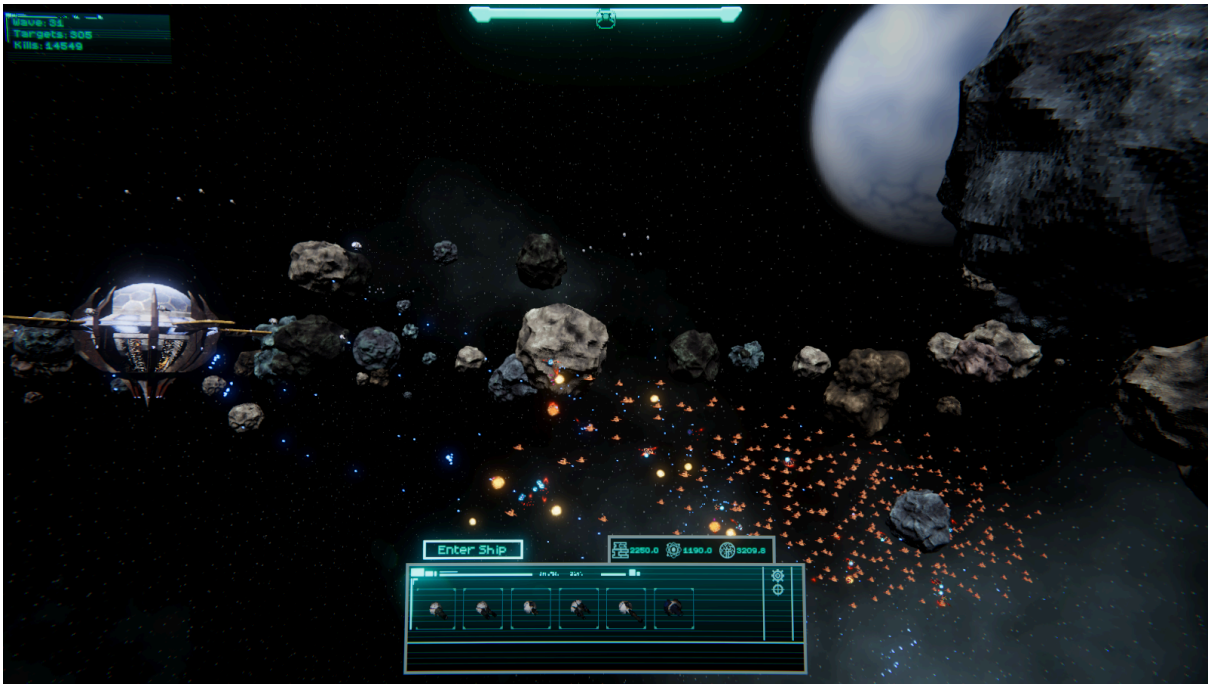
Screenshot Gallery



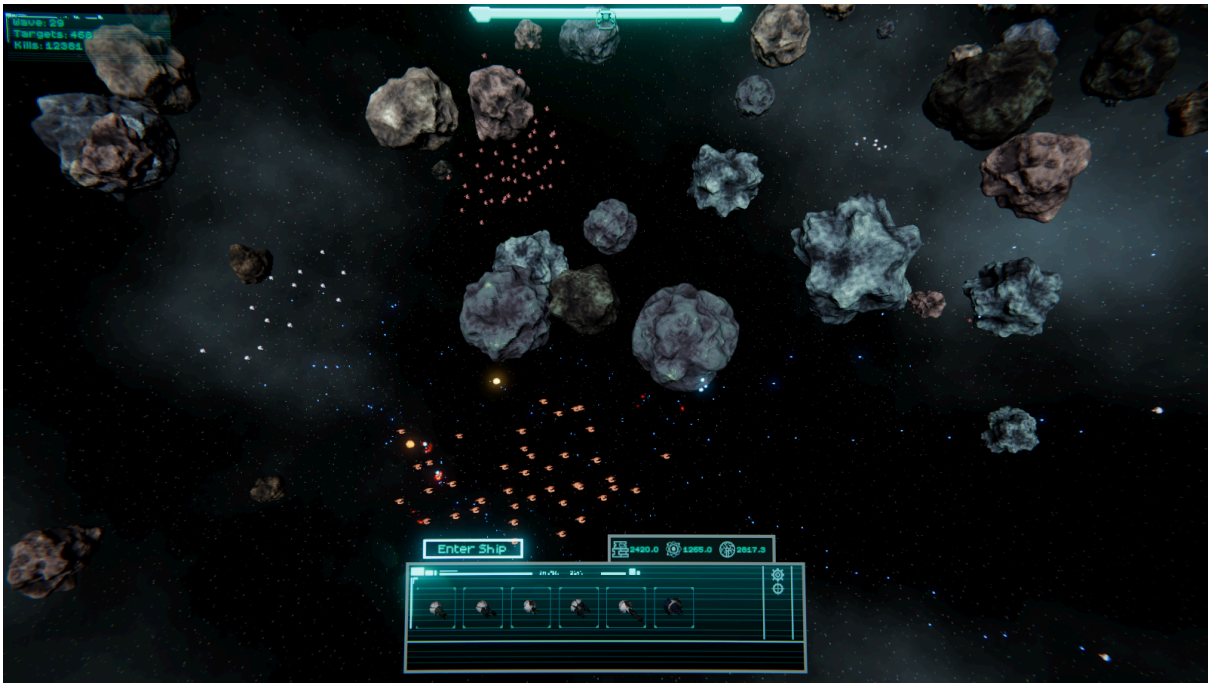
High Number of Enemies in the Late Game

Action Packed Dogfighting

Engaging an Entire Swarm With the Help Turretfire



Defending the Station in Tower Defense



Variety of Turrets and Enemies

Turrets Placed near the Station Defensively

Course Retrospective

Overall Impressions

We enjoyed the course very much. It met our expectations, though it was a bit more free than the course notebook would suggest, sometimes leading to minor confusion regarding the exact requirements of the deliverables. Mostly due to self imposed goals, the workload was noticeably higher than is to be expected from a 10 ECTS course.

We are very happy with the final state of the game. While there are some slight improvements still to be made, we are proud of our accomplishments as part of this game project, especially the technical achievement and overall quality of presentation. As it stands, the game is stable and complete, with a fully fleshed out gameplay life cycle.

We feel we accomplished as much as could reasonably be expected within the tight schedule of the course, which is already as optimized to the constraints of the length of a single semester as possible.

Specifics

What was the biggest technical difficulty during the project?
By far the biggest hurdle was the usage of Unity Engine's DOTS and the numerous issues associated with it, primarily the quality of documentation and incompleteness of features, requiring us to implement workarounds and custom reimplementation of core systems.

What was your impression of working with the theme?
The theme provided some useful inspiration during the ideation phase, but did not meaningfully influence the development of the game afterwards.

Do you think the theme enhanced your game, or would you have been happier with total freedom?
The theme is already broad enough to allow the justification of many game design choices, providing next to total freedom.

What would you do differently in your next game project?
Spend more time documenting the codebase and defining workflows early, to facilitate cooperation. Now that we are more experienced in working with DOTS, some of the early software engineering decisions seem ill advised.

What was your greatest success during the project?
Besides the technical achievement, we are particularly proud of our unique visual style consisting of a high number of custom handmade assets; including menus, models, textures, audio and particle effects, as well as a few shaders.

Are you happy with the final result of your project?
Despite a tendency to dream big, we planned for a consciously narrow scope from the beginning and are very happy to have achieved it. The final result is nearly exactly what we had envisioned at the start of the course.

Do you consider the project a success?
Yes, we do consider the project a success. Our main criteria for success was the technical achievement, which we implemented early in development. We then went on to also make the game visually appealing. Our weaknesses in game design were expected from the beginning and did not prevent us from successfully portraying the last-man standing scenario. Overall, we learned a great deal about the process.

To what extent did you meet your project plan and milestones (not at all, partly,mostly, always)?
We met all the milestones comfortably. The project timeline was not entirely accurate, as some tickets took longer than expected, but did not impede the project plan as a whole. We were able to meet the minimum requirements, and then some, for each milestone and presentation during the semester.

What improvements would you suggest for the course organization?
There were some inconsistencies between the course's project structure document, its webpage and the direct communication, particularly regarding the exact requirements of the deliverables.
We also would have welcomed more feedback during the course of the semester, although we understand this is due to the high number of participating groups this semester. Feedback was, after the first milestone, mainly delivered in one or two questions during and after presentations.