Team No. 11

Jarod Davis, Chase Stump, Zachary Atkins, Minye Wu, William Burdick

Frontier Unknown

Synopsis

A medium sized multiplayer 3D spacecraft shooter game focusing on utilizing high quality graphics and ray tracing technology built with the unity engine.

Description

• Why is the project being undertaken?

Our intentions for this project are to gain experience working with the Unity Engine in-depth and develop a professional-level product. Further, we aim to have loads of fun, while creating a polished game that could end up on the marketplace in order to pass that fun on to others.

• Describe an opportunity or problem that the project is to address

We seek to address the cabin fever produced from people being locked up without social interaction due to Covid-19. We will also expand the co-operative/competitive ship battle genre established by games like Guns of Icarus with new mechanics and features and unique setting. Further, we will bring new graphics technology to the genre through the use of DirectX 12 RTX raytracing.

. • What will be the end result of the project?

We will produce a medium scale, team-based multiplayer game that encourages effective cooperation and large-scale tactics in a space combat environment. Ideally, this will be a professional-level game that could potentially be sold for a profit or released online for free.

Project Milestones

First semester:

1. Use-Case and UML diagrams completed(10/26)
2. Networking choice and online multiplayer establishment (11/4)
3. Gameplay logic and fundamentals (11/23)
4. Physics and collisions (12/11)

Second semester:

5. Prototype models completed (2/28)
6. Shader and raytracing implemented (3/15)
7. Models & SFX Integration (4/15)
8. Full gameplay completed (5/1)
9. Presentation and final documentation completed (5/14)

Gantt Chart included in the submission.

Project Budget

- Hardware, software, and/or computing resources
- Unity 2020.1.9f1 LTS (free)
- Perforce's Helix Core (free tier, may upgrade)
- Unity assets (tbd, depending on amount of time for asset development)
- Estimated cost: $200-400
- Vendor: Unity Asset Store, Perforce
- Special training: Unity, DOTS, network coding
- When they will be required: Spring semester

Preliminary Project Design

How the Software Works

There are two major components of how our game works. First, users will be able to customize their ships through the use of various weapons, shields, and miscellaneous attachments. Each of these will have a number of strengths and weaknesses, encouraging teams to cooperate in order to have a well-balanced loadout. Ships will be balanced by means of a weapon slot system. In this system, different ships will have different capacities to carry certain weapons. For example, small ships will not be capable of handling heavy duty weapons but may carry a few medium weapons. Larger ships will be capable of handling heavy weapons but may have limitations on the number of heavy weapons they can carry. See the attached Game Mechanics document for a full enumeration of the components we intend to support.

The second main component are the player roles. There will be three roles: Pilot, Engineer, and Gunner. Within these roles, players will be able to select different perks which either provide increased specialization or some boon which provides value to their team. Additionally, players are most effective when doing tasks that relate to their roles. For example, a Gunner may have access to a number of ammunition types, be able to reload the guns faster, and have better accuracy when firing. On the other hand, a Pilot may be able to shoot the guns, but will excel in maneuvering the ship through the use of speed boosts and will also be able to issue commands to the other crewmates in order top prioritize particular issues during the game.

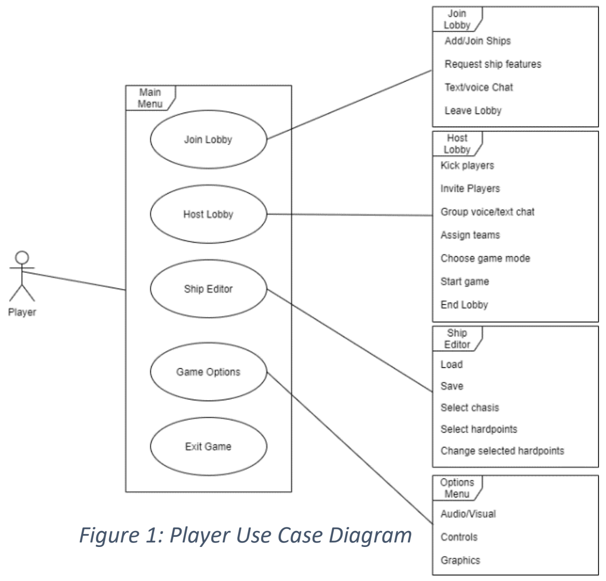


Figure 1: Player Use Case Diagram

From the player point of view, we intend to support a Ship Editor, as well as various Lobby related features. These include: create a new lobby, add/request certain ship types to a lobby, text/voice chat, and game mode controls. Further, the ship editor will allow customizations of armor, guns, chassis, and shields. A more complete description of the intended components is included in the attached Team 11 Game Mechanics v1 document in the submission. Moreover, a use case diagram for the player is given in Figure 1.

The win conditions of the game are based on a ticket system. Each team is assigned an equal number of tickets. Destroying opposing ships or completing objectives reduces the opponent’s ticket count. Upon ship destruction, that player’s team immediately loses tickets corresponding to the ship size. Also, ships are expected to take damage in the game without being destroyed. Ship size and armor dictates how much punishment a ship can take before destruction. Ships may lose guns or have a damaged hull. In this case the ships can be repaired at the cost of that team’s ticket count. A timer will be implemented to encourage more aggressive play and prevent a stall or slow game. The team with the most amount of tickets at the end of the match will win if each team has tickets remaining.

Design Constraints

The majority of our constraints come from the use of the Unity game engine. First, we consider networking for our game. We must use one of Unity’s multiplayer networking frameworks; either the depreciated UNet framework or the new connected games stack. The latter also requires the use of the DOTS framework, part of Unity’s new “Performance by Default” design paradigm. For a further analysis of our multiplayer constraints, we consider the flowchart released by Unity for determining the best multiplayer solution for Unity projects, given in Figure 2. First, in order to deliver the cooperative experience we desire for our game, we need to support at least 32 player games. Following the flowchart, this means that we need a dedicated game server (DGS) solution. This leaves us with two main options: utilizing the preview DOTS-Netcode stack or writing our own custom networking stack.

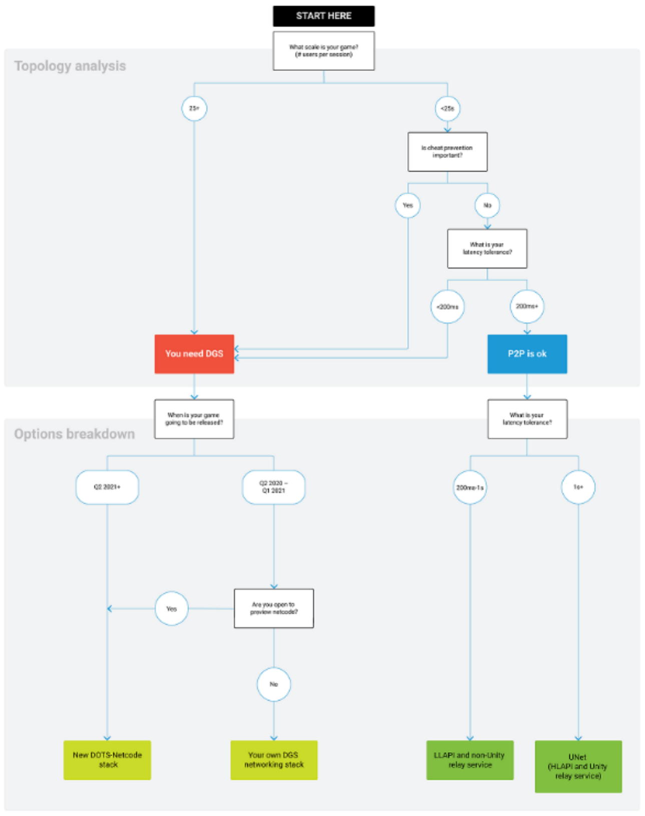
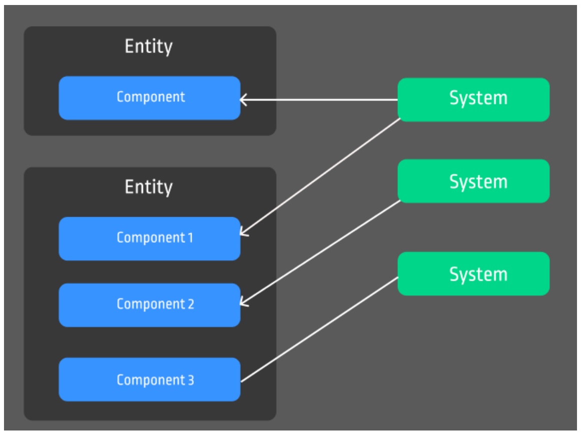

START HERE
What scale is your game? (# users per session)
Topology analysis
25+
<25s
Is cheat prevention important?
Yes
No
What is your latency tolerance?
<200ms
200ms+
You need DGS
P2P is ok
When is your game going to be released?
What is your latency tolerance?
Options breakdown
Q2 2021+
Q2 2020 – Q1 2021
200ms-1s
1s+
Are you open to preview netcode?
Yes
No
New DOTS-Netcode stack
Your own DGS networking stack
LLAPI and non-Unity relay service
UNet (HLAPI and Unity relay service)

Figure 2: Unity Networking Flowchart
https://blogs.unity3d.com/2019/06/13/navigating-unitys-multiplayer-netcode-transition/

In regard to gameplay limitations, as we are using the DOTS framework, a reasonably large number of entities can be used in our game on a technical level (See in figure 3 and 4). The greater



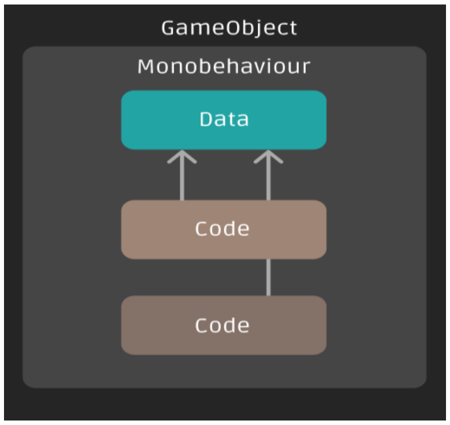


Figure 3 and 4: Illustration of the Entity-Component-System (ECS) Model
https://medium.com/@firatcetiner/unity-dots-and-beyond-a-brief-introduction-bce205e8bace

challenge outside of coding complexity would be producing and implementing assets. I would go so far as to say that when it comes to the game engine itself, the greatest limitation would be our skill with the engine and modeling. We likely will not be able to instantiate 20^{20} entities or anything to that degree, but overall, it is unlikely that we run into limitations due to the number of entities. If we put in the necessary work to implement the code and sufficiently polish the visuals, then our imagination and time are the limits for our game's design.

Next, we will consider the platform constraints of our project. Since we are designing a PC game, support for the Windows operating system is necessary. While support for Linux and MacOS would also be beneficial to the success of our game, the player bases on those platforms tends to be far smaller, so we would not consider supporting these platforms to be a design constraint.

Another constraint we must consider is the project management and version control platform we choose to use. Due to the nature of game development with Unity, tradition source control management platforms such as GitHub or BitBucket tend to provide insufficient granularity for file control, leading to a large amount of merge conflicts if multiple developer are working concurrently. As such, we have opted to utilize the Perforce Helix Core source control system. Perforce has built-in Unity integration, allowing individual files to be changed and submitted to source control seamlessly, preventing the aforementioned merge conflict issues. However, the use of Helix Core has also introduced a number of internal project constraints. For example, Helix Core does not come with any free hosting solutions, so we have opted to run our versioning server on a server owned by a team member. This means that server maintenance and management must be accounted for in development time and allocation moving forward with our project, presenting a time constraint. Additionally, in order to best utilize this version control system, we will need to commit time as a team to assess the best practices for Helix Core version control, as they differ in some aspects from traditional Git approaches.

Ethical Issues

One ethical issue we may have is ACM code of 1.6, which is 'Respect privacy.' Respect privacy is very important for our project since we are designing a medium sized online multiplayer game and there will be a lot of user data gathered. We will make sure only the minimum amount of personal information necessary will be collected in our system, and the retention and disposal periods for those data will be communicated clearly to data subjects. We will need to make sure those data won't be used for any purposes without the user's consent.

The other ethical issue we may have is ACM code of 2.6, which is 'Perform work only in areas of competence.' For this project, we are going to implement some challenging technologies that we might not have a large amount of experience before. Hence, we will make sure to do as much research as we can and spend enough time acquiring the necessary competencies before pursuing a certain assignment. If at any time during the game development we find ourselves lack a necessary expertise, we will pause our work assignment, try to obtain the necessary competencies, and get back to the work assignment afterward.

Intellectual Property Issues

The main intellectual property issue we may have is Patent and Copyright. We will make sure to ask for consent from the patent owners and give credit to them if we are going to use any patented technologies. We may need to pay extra attention to some games on the market that appear to have similar ideas or design to ours.

We will respect the copyright of technologies we are going to use, for example, Unity, DOTS framework, Blender, and Helix Core. We will use artworks and assets that are either created by us or licensed for our team.

Change Log

- Unity 2019.4.9f -> Unity 2020.1.9f

This change was due to the different preview packages available for the 2019 and 2020 releases. Many of the new features that we intend to use are only available on the 2020 release of Unity, so we changed our target version.