# ISTANBUL TECHNICAL UNIVERSITY
# FACULTY OF COMPUTER AND INFORMATICS


# Space-Themed 3D MOBA Game

**Graduation Project Final Report**


**Emre Altan**
**150160024**

**Department: Computer Engineering**
**Division: Computer Engineering**

**Advisor: Assoc. Prof. Dr. Feza Buzluca**



June 2021

# Statement of Authenticity

I hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,
2. and all the remaining sections, especially the theoretical studies and implemented software/hardware that constitute the fundamental essence of this study is originated by my individual authenticity.

İstanbul, Haziran 2021

Emre Altan

# Space-Themed 3D MOBA Game

## (SUMMARY)

MOBA means multiplayer online battle arena and it is a highly favored game genre nowadays. Players fight in a restricted area and generally starts from level 1 to highest level with earning experience during the game or similar game mechanic. In my project, there would be a space-themed fight arena with planet colonizing. Teams consist of 3 people and there are 2 teams. Each player has the character to choose before the game starts and the weapon assigned to the character. There are 5 different characters with different features and 3 different weapons. Every player spawns on their home planet with a spacecraft. There are another planets other than home planets for the colonizing. This planets produces elements to create new protection units on the colonized planets. Neutralizing the enemy and colonizing the planet provides different amounts of experience points and contributes to leveling up. Opposite players could land on the same resource planet to colonize. There would be no progress on colonizing until there is only 1 team on the colonizing center of the planet. Therefore, players would need to neutralize the enemy players. Players have weapons that could be used when they are not in the spacecraft. If players are using a spacecraft, they can use spacecraft's weapon. There is only one fight restriction and that is the players cannot damage enemies who are not inside the spacecraft with a spacecraft's weapon. Planets have planet shields to protect planet from enemies. When a player dies, re-spawns after a certain time on the home planet. If a home planet is destroyed, then player of that planet will not spawn when he/she died. For the victory one team must destroy all home planets of the enemy team.

Unity game engine is used to make the game. Unity contains all aspects that a game needs and has large amount of resources. Therefore, it is a commonly used game engine. For the network side, Photon network service would be used and it is a Unity integrated system. The multiplayer feature was postponed so that it could be added later because it was a complex project and not enough time was available. Game can now be played with AIs. Krita software is used for 2d graphics. For the 3d graphics sci-fi space and sci-fi city assets of Synty Studios are used which is available on Unity Asset Store.

# Contents

# 1 Introduction and Project Summary

A 3d game contains many different aspects of technical works. Such as game mechanics, AI navigation, user interface, rendering, lighting, etc. In this game, I will explain according to this technical aspects and before that i will explain the game flow. Game starts with menu where weapon, character and spawn place can be selected. Then, the players spawns on their home planet and they can navigate through the planet. Due to the constant rotation change, I focused the game camera on the main character and turned off the up and down look while it was possible to look left and right with the mouse due to the constant rotation change that happens while navigating the planet. Players can get in the spacecrafts which spawns with players where they spawns. I also added a toggle pointing arrow to make it easier to find the spacecraft. Spacecrafts can transport players to target planets and if the spacecraft is landed on a planet then player can get out. Then, player goes to the target planet and colonizes the planet. As more planets are colonized, more material is gained. Materials can be used to add more guards on the planets to defend. Each material can be used to add one type of guards as in the character selection.

In the game, it was necessary to add gravitational forces as the system in the game includes planets. I tried realistic gravitational force at first but it was not suitable for the gameplay. Gravitation system will be explained in detail later.

There is also AI players. For now, other than the main player are artificial intelligence. AIs need two types of path-finding algorithm which are used for the walking on the planet and traveling with spacecraft in the space. In Unity there is a path finding system called Navigation. This system allows to find paths for the agents without going into details. Nevertheless behaviors of the agents must be defined and coded. Navigation works with a ground and baking the navigation gives us a walkable area. In my project there would be no flat ground as Unity expects. Still I managed to bake navigation surface on the planets which is explained later. AIs use spacecrafts to go other planets and they need to know the path to the target planet. I have considered 3 different approach to the path-finding in the space which are explained later. Although pathfinding issues were resolved, a complex AI behavior with multiple states had to be created. This behavior consists of three layers: priority states, continuous states, and decision making. Due to the fact that it is a complicated behavior tree, I attempted to cover for every scenario.

There is also a level system. When players kill enemies or colonize planets, they gain experience and can level up to level 5. When they level up, their health, shields and attack power increase. The faster they level up, the more powerful they can become and the easier they can kill their enemies. After killing enemies a certain number of times, the enemy's home planet's shield changes from invincible to normal and becomes attackable. The game is won if all the enemies' home planets are destroyed.

# 2 Comparative Literature Survey

In my project there should be AI players who can navigate in space. Therefore, I had to find a path finding algorithm which works on the 3D space. There is a lot methods that operable on 2D but 3D methods are lesser. Because of the amount of the working area, it is harder to construct methods but in the game development, with the progress on the 3D games it turned into a need. I found out three different way to implement. One of them is the combination of the hierarchical pathfinding algorithm and the octrees [1]. It is faster compared to other algorithms. Figure 2.1 shows that the running times of an A* algorithm on uniform grids, octree and octree with hierarchical pathfinding algorithm (COP). Running time of COP is has a steady rate compared to others. In Figure 2.2, nodes of the constructed abstract graph is shown. It is a clustered octree which defines the available routes. I left this solution as it is difficult to implement this system and I do not need to find such a detailed way around.

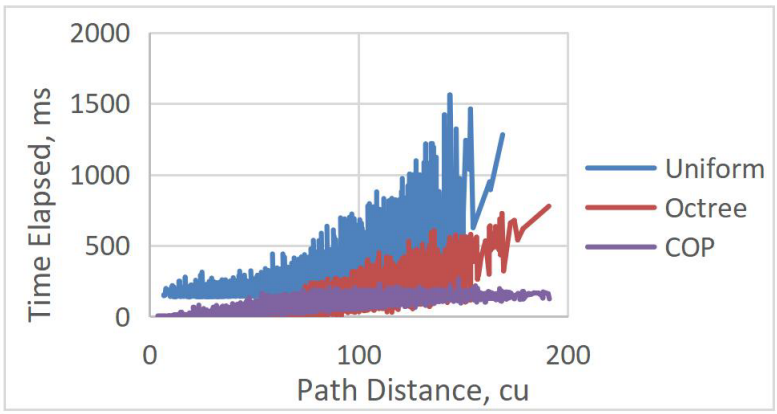


**Figure 2.1:** Running times of A* algorithm on different grids. [2]

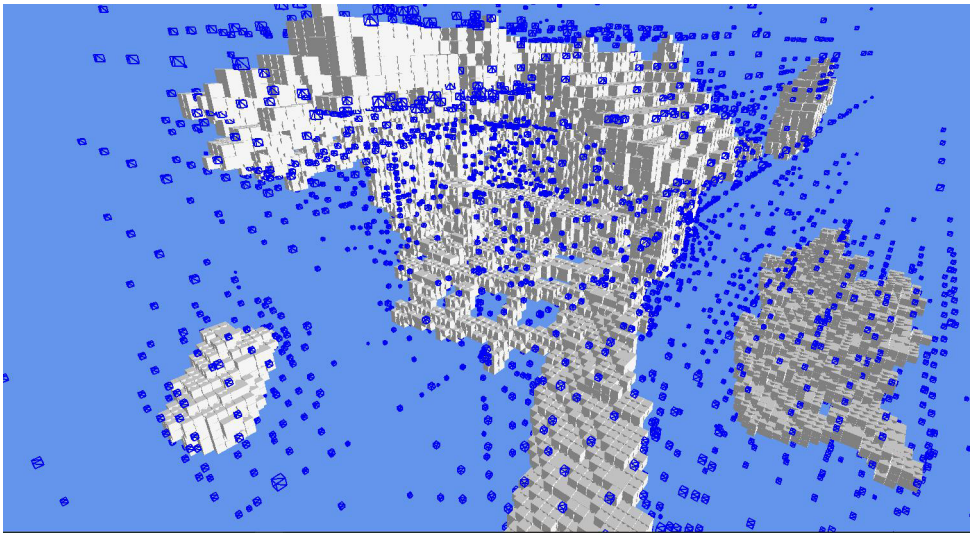

**Figure 2.2:** Nodes of constructed abstract graph (blue). [3]

Another approach on this subject was using the potential fields to find a path to target location [4]. In this method, when our spacecraft head towards to target location it calculates the potential fields and forces related to obstacles. In my project, there is

planets that have gravity zones affects spacecrafts and characters. Figure 2.3 shows a potential field graph. At the start point force is higher related to distance from the final point and it decreases as it come closer. Also when it approaches to an obstacle, force increases related to distance of the obstacle and it decreases in reverse. This approach is more suitable than the first one to my situation and considering the gravity zone in the movement of AI is affects game a lot. In Figure 2.4, an example system is shown for this algorithm which has many different sizes of obstacles. But in the game there is nearly a dozen planet and six spacecrafts for now. Spacecrafts with AI does not encounter obstacles often. Therefore, a continuous computing for the spacecrafts with all planets are unnecessary workload.

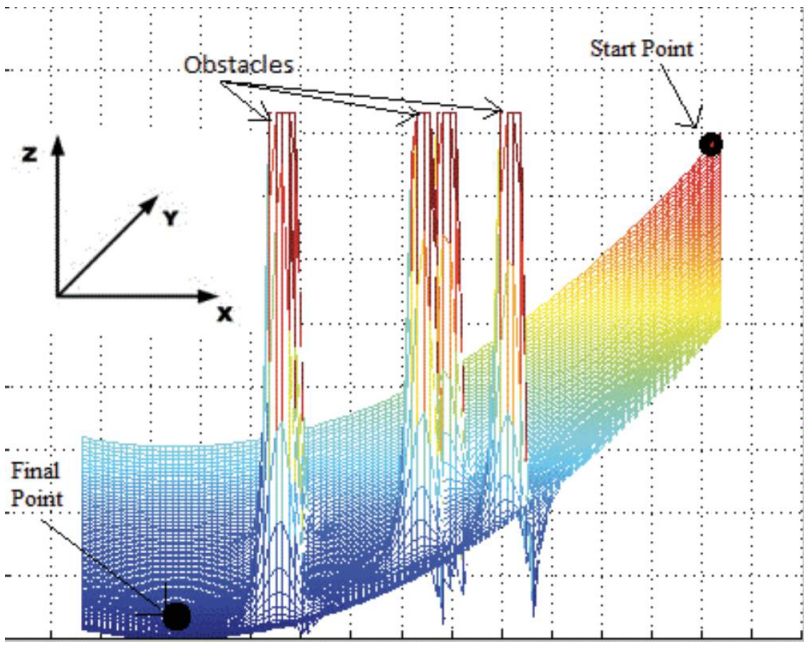


**Figure 2.3:** Forces from the potential fields of the obstacles. [5]

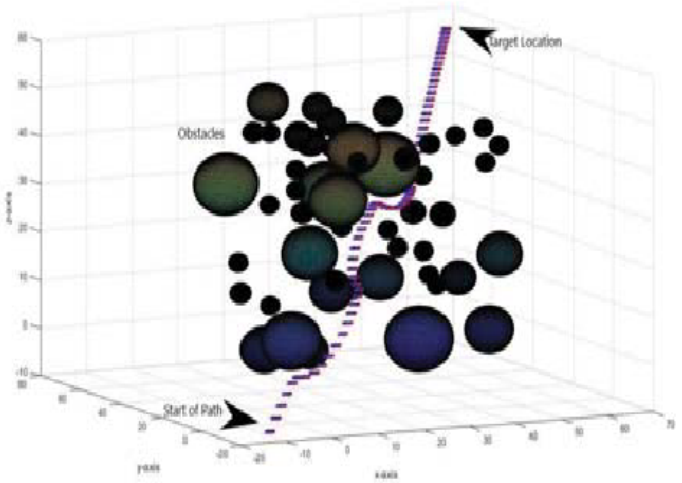


**Figure 2.4:** Collision-free path with existing of fifty fixed obstacles. [6]

Last approach is checking around the vehicle. There is 14 different checking point used around the vehicle as shown in Figure 2.5 and implementation of that explained in

detail later. If one of this checking points encounters an object, a planet in the game, then spacecraft moves on the opposite direction until there is no obstacles that collides with the checking points.

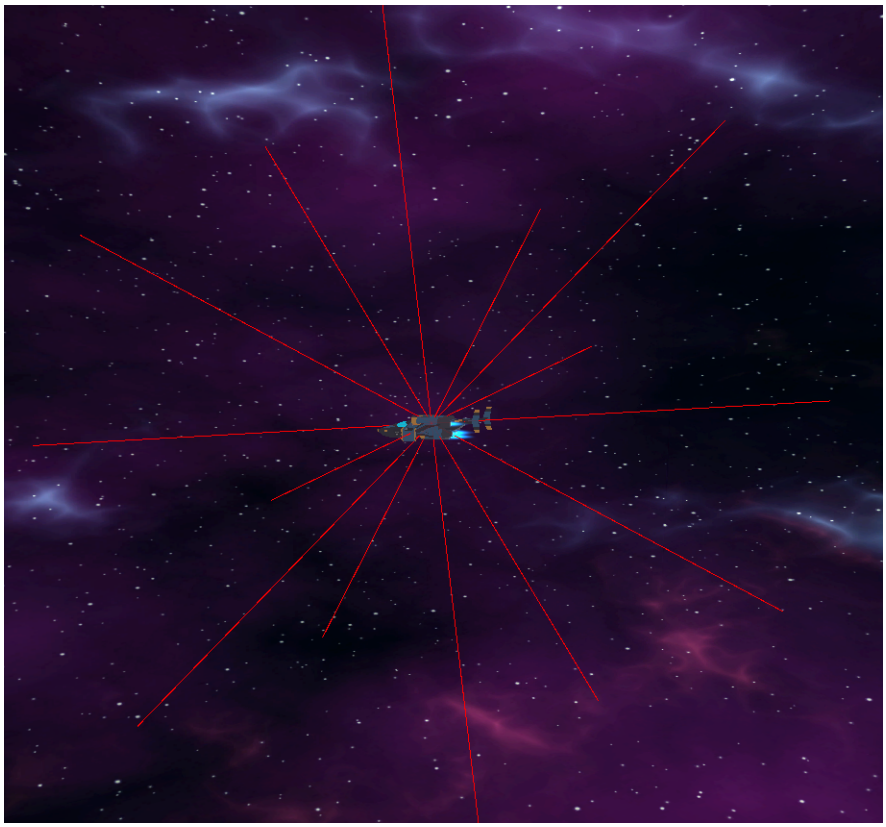

**Figure 2.5:** A spacecraft shown with checking rays in debug mode.

# 3 Developed Approach and System Model

## 3.1 Data Model

Strategy pattern is used in two behaviours: Attack and Move. In Figure 3.1, attacking strategy factory can be seen. It has two different strategy which are attacking with character's weapon and vehicle's weapon. Also in Figure 3.2, moving strategy factory is shown. This two different move strategies are walking on a planet and using vehicle. Strategy pattern that used here is a proper design pattern for the situation since there could be new attacking types or new movement types to be added in the future. Therefore, it is easier to add new types with strategy pattern.

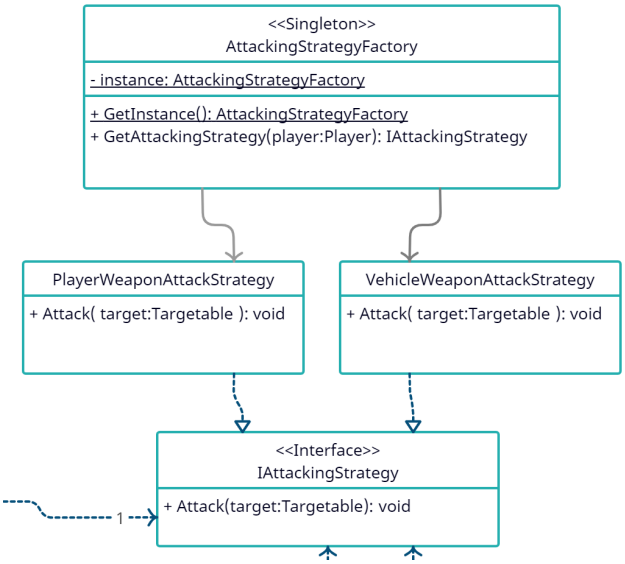


**Figure 3.1:** Attacking strategy factory.

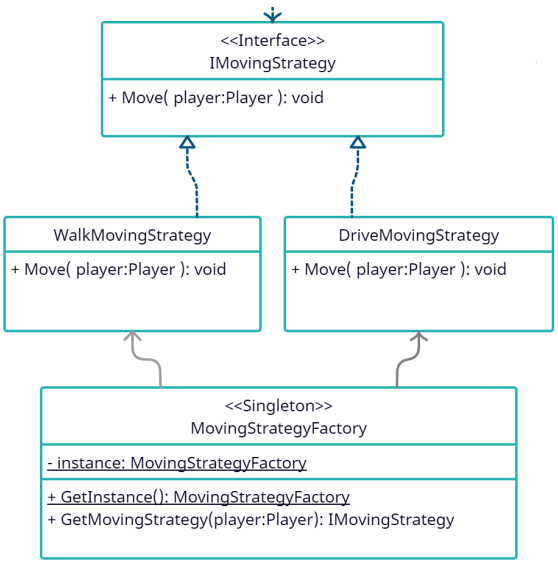


**Figure 3.2:** Moving strategy factory.

There are 3 types of characters in the game: player character, AI character and AI guard character. This 3 different character class inherits a base class called "CharacterBase", as shown in Figure 3.3. Player character class is used for our character when we playing, shown in Figure 3.4. AI character is stands for the main AI characters which can go other planets, conquer and attack all enemies, shown in Figure 3.5. Lastly, AI guard character is used for guard characters of the planets, shown in Figure 3.6. After conquering a planet, guards spawn on the planet and later with spending materials, guards can be added to planets.


**Figure 3.3:** CharacterBase class.

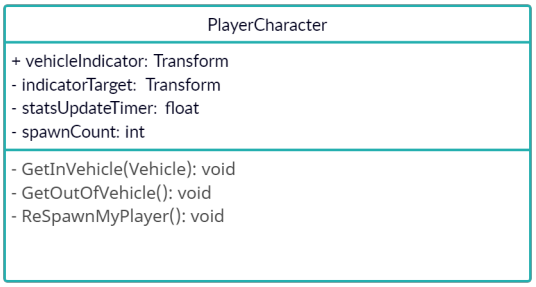


**Figure 3.4:** PlayerCharacter class.

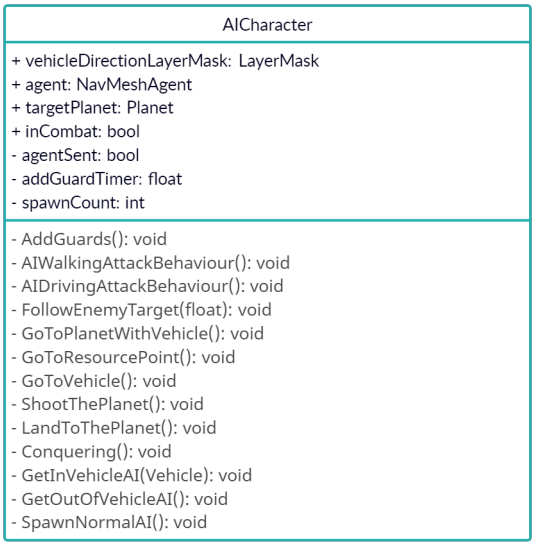


**Figure 3.5:** AICharacter class.

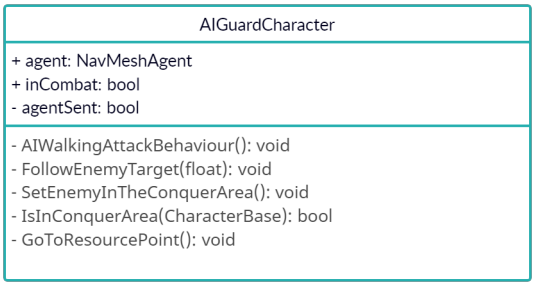


**Figure 3.6:** AIGuardCharacter class.

In Unity, there is also a system called prefab. Prefab is a prearranged game objects with components. Inheritance is the basis of the prefab system. A prefab can be used to create variant of that prefab and it is called prefab variant. This variants can have different values from the original prefab and when original prefab changes, values that are not changed anytime in the variant change accordingly. With this prefab system, I have created 3 character prefabs with 3 different character classes. Then, using prefab variants, I created 4 additional character types for each character classes.

In Figure 3.7, Planet class is shown. There are 2 functional types of planets: home planets and resource planet. And 5 different resource types of planets: Promerium, Prometid, Seprom, Terbium and Xenomit. Prefab variant system is used here to differentiate the planets according to resource type and productivity values.

Weapon class is included in the Figure 3.8. This class is a base class for the PlayerWeapon class shown in Figure 3.9 and VehicleWeapon class shown in Figure 3.10.

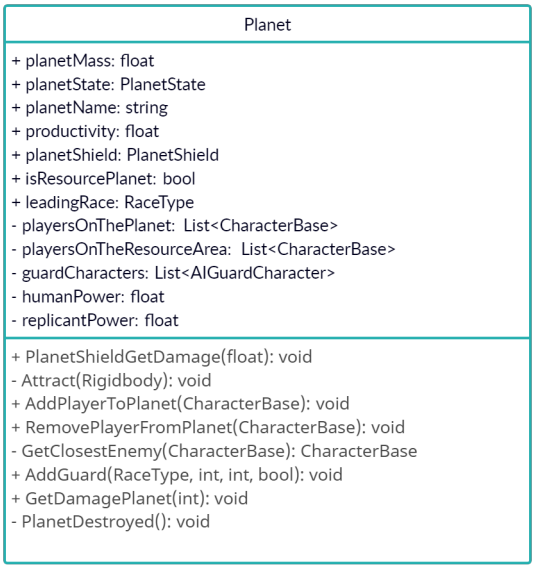


**Figure 3.7:** Planet class.

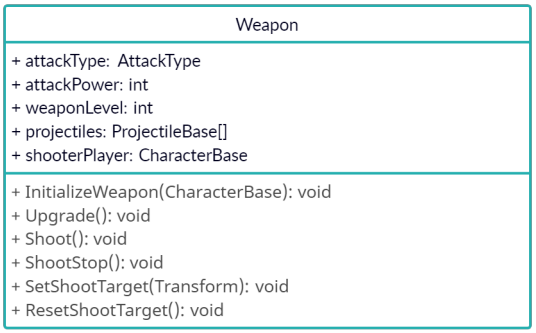


**Figure 3.8:** Weapon class.

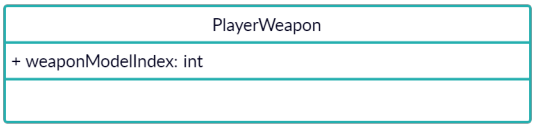


**Figure 3.9:** PlayerWeapon class.

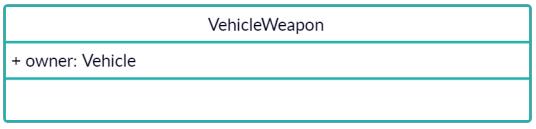


**Figure 3.10:** VehicleWeapon class.

## 3.2 Structural Model

### 3.2.1 Pathfinding

Since it is a 3d space game with walkable planets and usable spacecrafts, it needs a complex AI behaviour and pathfinding. Before the AI behaviour, I will explain the pathfinding. Unity has a pathfinding system called Navigation as mentioned earlier. Navigation system works with a surface called "NavMesh". NavMesh is pre-baked and used as AI's movement space. To be able to use this system, AI characters also need a component called NavMeshAgent. When this component is enabled it is automatically tries to go to a NavMesh surface which shown in Figure 3.11 with blue.

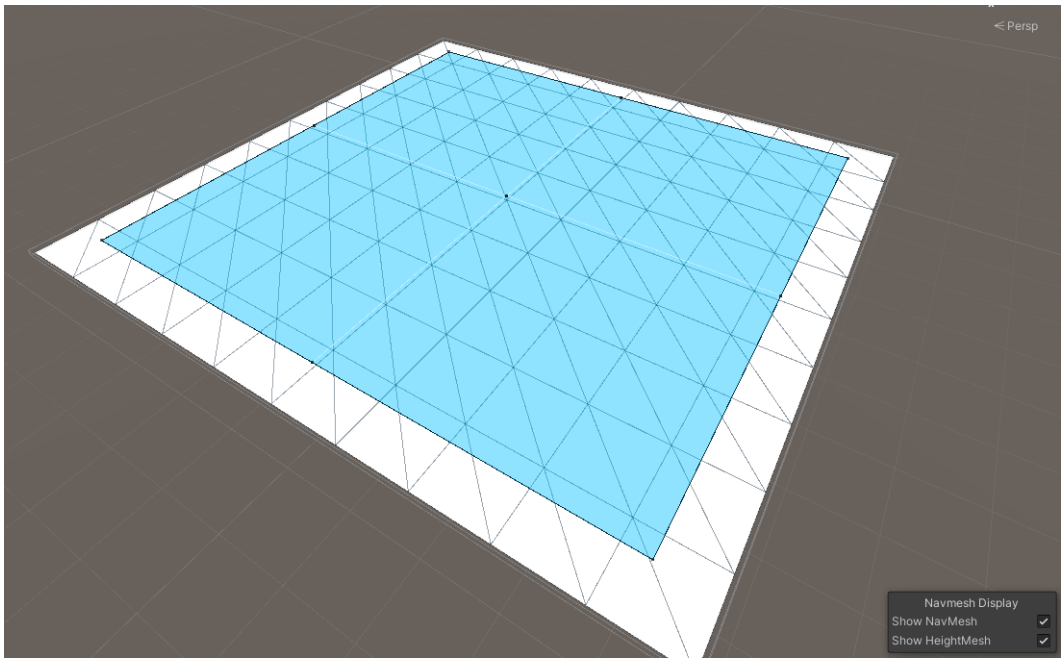


**Figure 3.11:** NavMesh surface highlighted in blue on a plane.

In Unity, it is expected to use NavMesh on a flat ground with no upside down. Because of this, pathfinding on a planet was not able to with Navigation system of Unity and creating my own system to form a grid and implement A* pathfinding algorithm was hard and it would be time consuming. At first I implemented an algorithm to find next position between AI character and target character. It was a straightforward method. I found the next position according to speed of AI character using spherical interpolation around the planet. To be able to have a correct result planet must be smooth round without hills. I have changed the meshes of planets to one type with a lesser hills. But actually this was not solution because it does not look for any obstacle, another character or walkable area. Therefore, I have tried to find a way to use Unity's navigation system and I found a solution on the Unity's forum. Solution was basically baking 6 different NavMesh surfaces by turning the planet. With these 6 different NavMesh surface, planet was almost covered with the surface. To fill the areas between the surfaces I used a built-in component called NavMeshLink. NavMeshLink creates a connection between close NavMesh surfaces. An example NavMesh surface can be seen in Figure 3.12. There was still a problem when using NavMesh to find a

path. Since it was used for flat surfaces, it finds a path using 2 axis which are x and z. When we target the AI to far pole of the planet, it goes to near pole without looking to y axis and it assumes that other pole is the closest point to the position that we give. To overcome this issue, I used spherical interpolation to give a closer point to the NavMeshAgent. The closer the interpolation is to the AI character, the more accurate pathfinding will be.

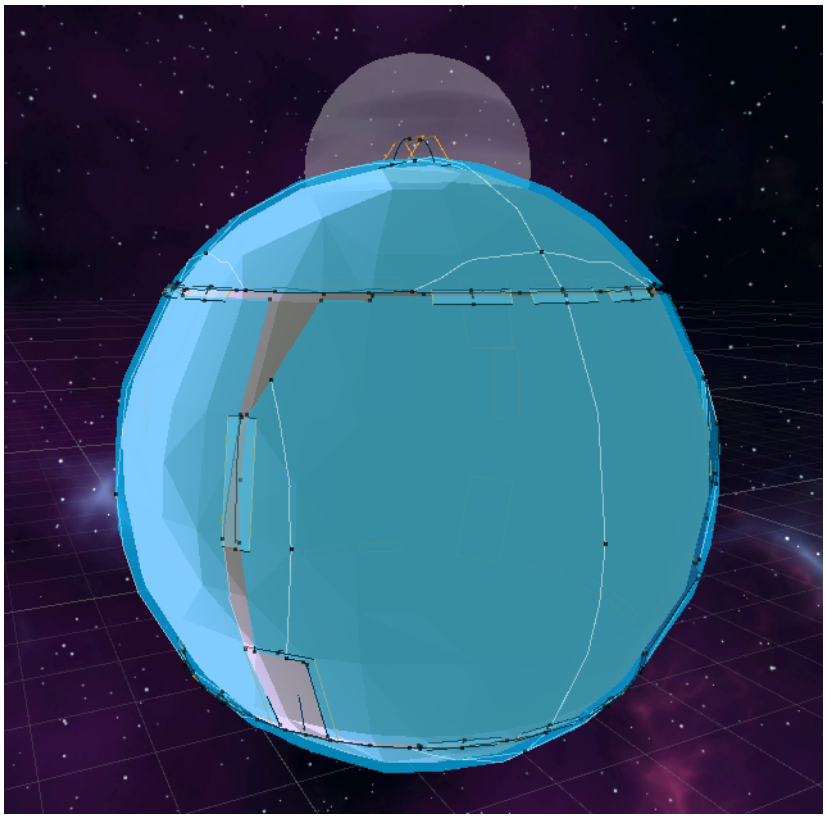

**Figure 3.12:** NavMesh surfaces connected with NavMeshLinks on a planet.

AI characters also use spacecrafts to go in space. Therefore another pathfinding was needed here. I did this pathfinding by checking around the vehicle as mentioned before. This checking is made by Unity's raycasting. Raycasting is a method for sending rays from an origin position with a direction and maximum distance. I have used 14 rays to check around the vehicle. This checking works on every frame while AI character drives the spacecraft. Sent rays from the vehicle can hit to another object such as a planet. Then I made it move in reverse of that collided ray. When there is no colliding with rays, then spacecraft can go to the target planet.

### 3.2.2 AI Character Behaviour

Since pathfinding is handled, there is also AI behaviour which is basically all work that AI character or AI guard character does. AI character is more complicated than AI guard character because it goes other planets to conquer, adds guards or fights with enemy everywhere. Therefore, AI character has many states and many decisions that needed to be handled. I have created a behaviour tree for that. It is basically a binary

tree with decisions at the non-leaf nodes and actions on the leaf nodes. This decisions checks every frame in the game with Update function. Update function is Unity's built in function to handle works at every frame to be rendered.

Behaviour tree can be divided to 3 segments: prior actions, continuous actions and decision actions. In Figure 3.13, prior actions are shown. This prior actions represent unexpected events such as enemy in range or enemy attacking to our planet. Since enemy's behaviour is not pre-defined and combat is the first priority, this actions must handled before other actions.

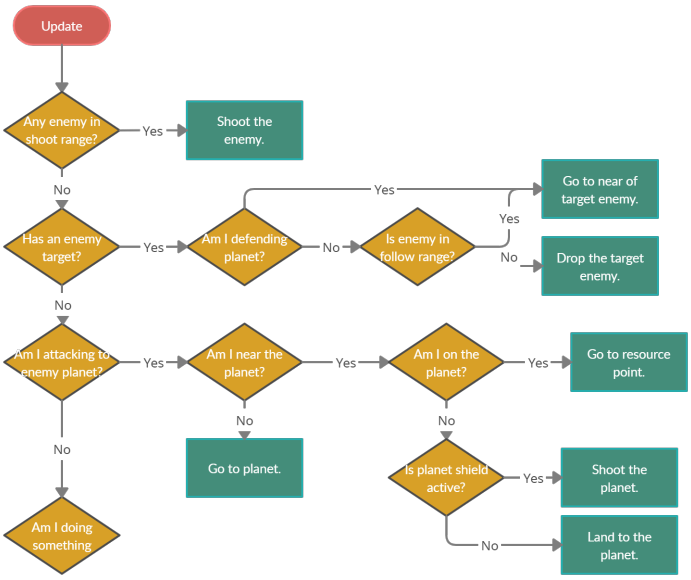


**Figure 3.13:** Prior actions of AI character on a behaviour tree.

In Figure 3.14, continuous actions are shown. Continuous actions differ from the others with the type of the action. This actions must be go on for a time such as going to vehicle or going to another planet. This actions cannot complete in one frame and each action corresponds to a state of the AI character.

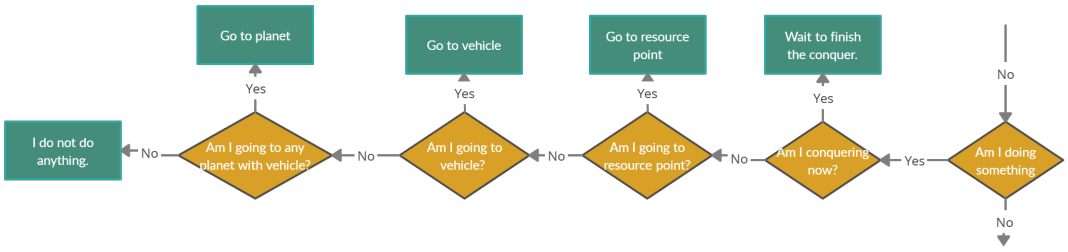


**Figure 3.14:** Continuous actions of AI character on a behaviour tree.

Lastly, in Figure 3.15, decision segment is shown. After completing a task from continuous actions or prior actions, state of the AI character becomes "DoingNothing" and starts to check decision segment. This part is the main strategical part of the AI character. Leaf nodes in here works once and then AI character starts to do something until the task is done.

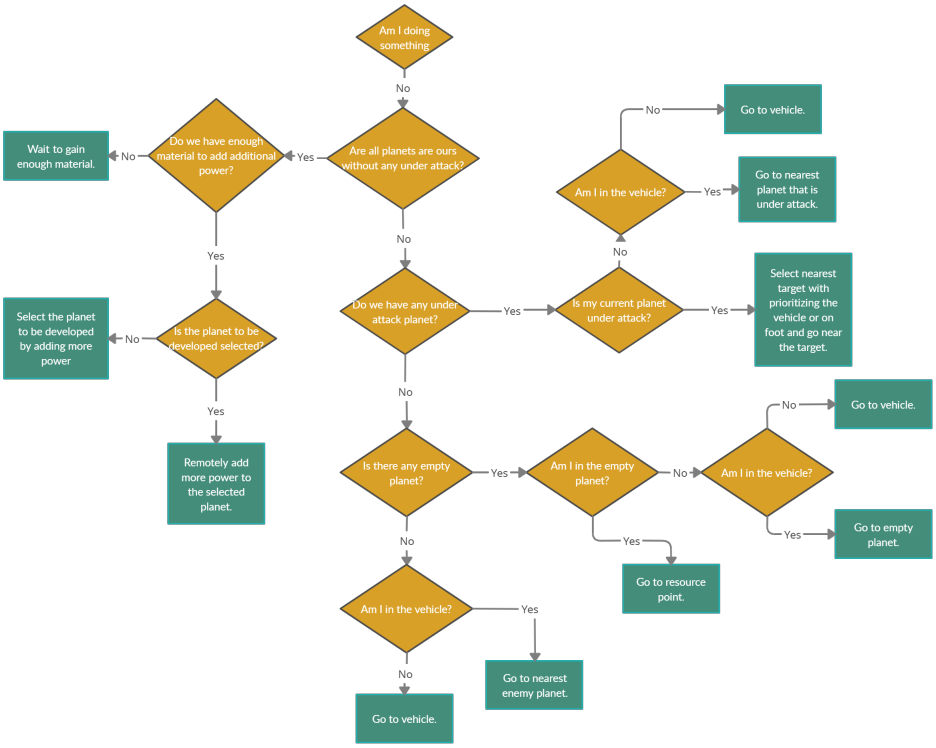


**Figure 3.15:** Decision actions of AI character on a behaviour tree.

### 3.2.3 AI Guard Character Behaviour

There is also AI Guard characters which are spawns on the planet and stays there for protecting the planet from enemies. As it shown in Figure 3.16, it has a similar behaviour of AI character but lesser.

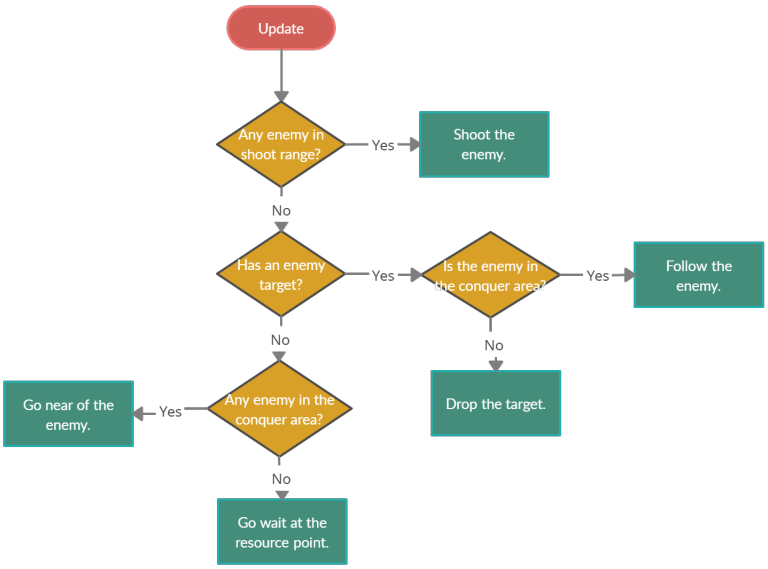


**Figure 3.16:** Behaviour tree of AI Guard character.

## 3.3 Dynamic Model

### 3.3.1 Unity's Object System

In Unity, a game is consists of scenes. A scene contains game objects and related informations about that scene. In Figure 3.17, build settings are shown. This window allows us to manage the scenes that are included in the final game. Also there is different platform options that can be selected and build easily.

Scenes has their own hierarchy which is shown in a its own window. Figure 3.18 shows the hierarchy of the Game scene. In hierarchy, game objects can be seen and managed. A game object can be parent to another and in Figure 3.18 parent objects has a little arrow on the left to indicate that its expandable.

Game objects are the main unit in a Unity game. All game objects stays in 3d space which Unity handles. Game objects has components that can be added to change behaviour of that object. Every component is essentially a script which derived from base class called "MonoBehaviour". To be able to add a class as component to game object it must derived from MonoBehaviour. Also all game objects has a unchangeable component called Transform. It handles the position, rotation and scale of object in 3d space. I have created a few game objects to add manager script to them to handle the game. As shown in Figure 3.19, GameManager is one of them and it controls game duration, level experience amounts, informations about two teams in the game and start countdown.

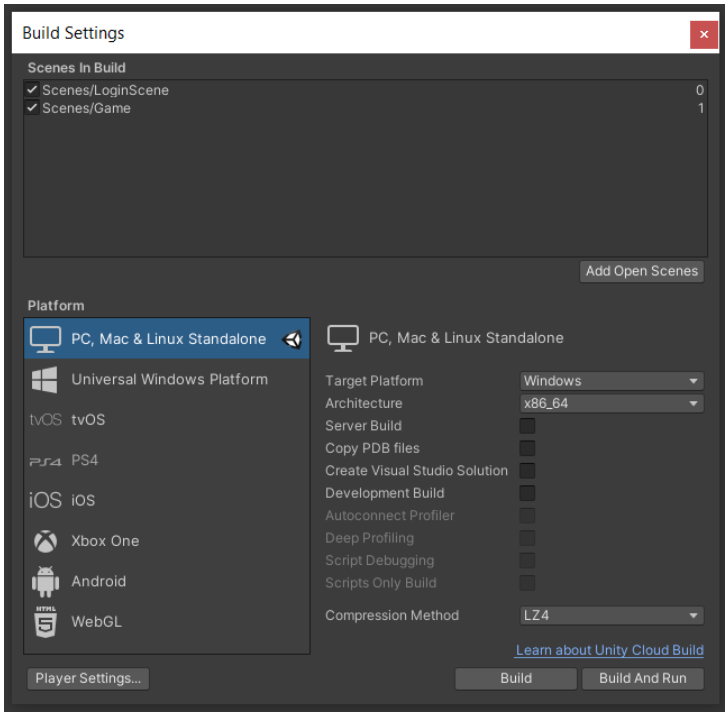


**Figure 3.17:** Build settings in Unity.

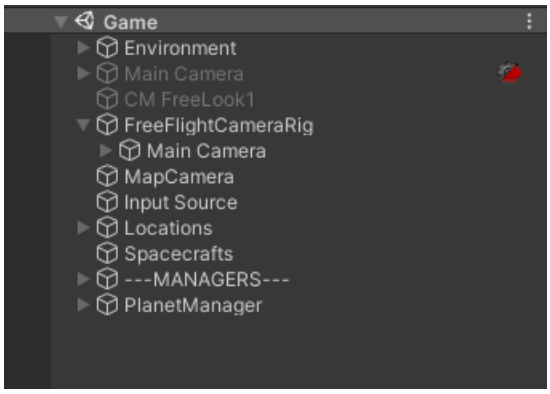


**Figure 3.18:** Hierarchy window in Unity.

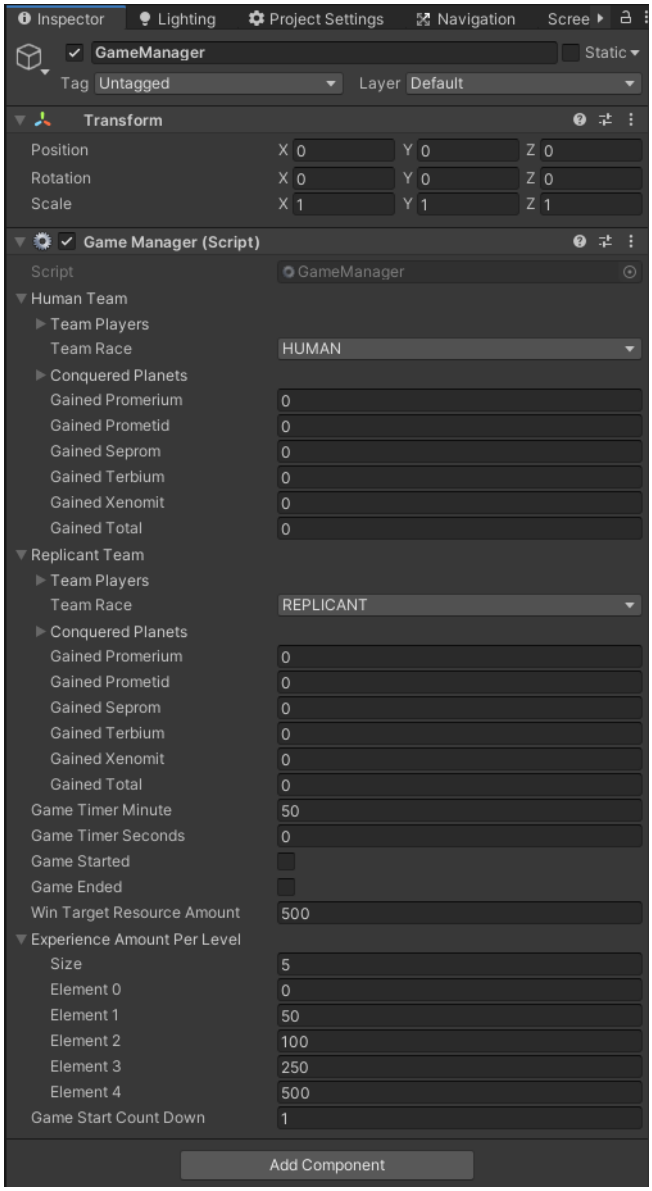


**Figure 3.19:** GameManager object with Transform and GameManager components.

### 3.3.2 Game Views

Game starts with a 3 button menu as shown in Figure 3.20. Play button opens room view which is shown in Figure 3.24. Options button directs to character edit screens, shown in Figure 3.21 and Figure 3.22. Figure 3.21 left and right buttons which changes character to be edited. Sword icon at right-left is also a button and it opens weapon change mode which shown in Figure 3.22. In those screens, weapons for characters can be selected specifically. Lastly before the game, there is a room screen shown in Figure 3.23 and Figure 3.24. In room screen, we select our character, we can change our place and fill AI players with random type to be able to start. Figure 3.24 shows filled room with AI players and our player.

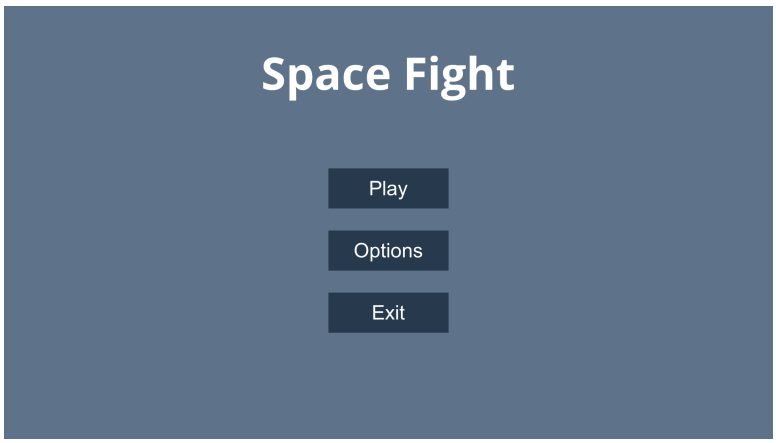


**Figure 3.20:** Initial view in Unity.

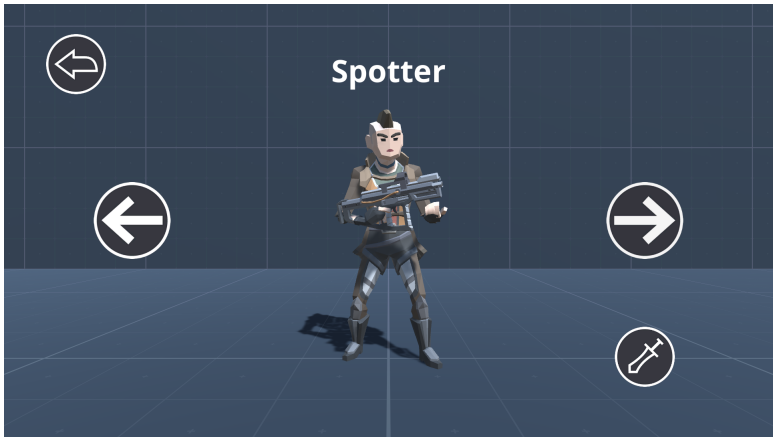


**Figure 3.21:** Character change view.

After filling the room and pressing to start button, scene changes to game scene and characters spawn on their planets related to their place in the room. Figure 3.25 shows an example initial view in the game. In the upper left corner, user interfaces of level, health and shield bars and material counts are shown. Health and shield bars are dynamically changes. In the middle upper corner, UIs of remained time of the game and balance of the gained materials of two teams are shown. In the upper right corner, there is a minimap which is a realtime camera that follows our character from the up related to y axis. Lastly there is a fire icon at the lower left corner of the screen. It

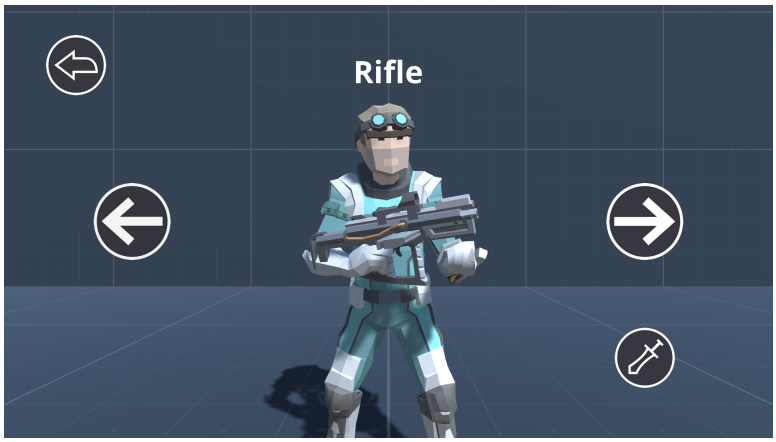


**Figure 3.22:** Weapon change view.

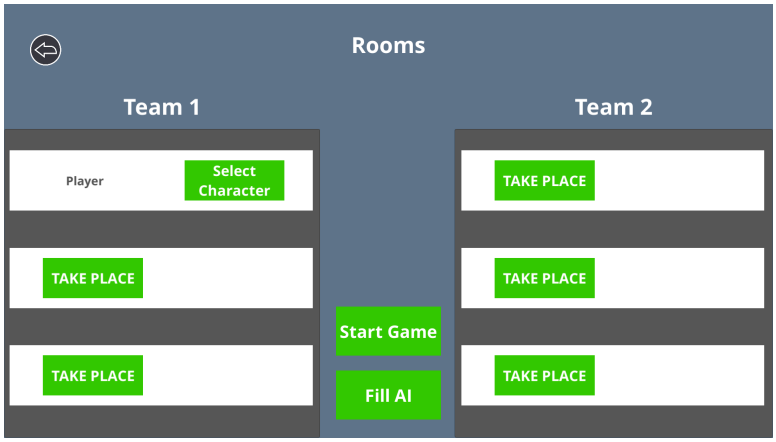


**Figure 3.23:** Room view before selections.

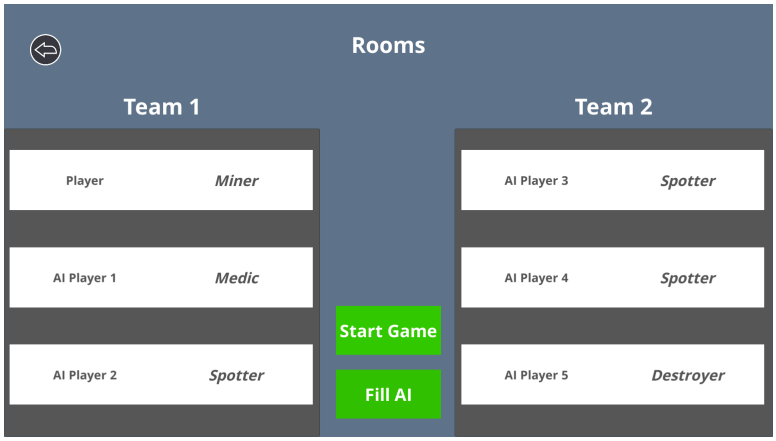


**Figure 3.24:** Room view after selections.

fills related to enemy kills and after fully filled it can be used and it gives boost to our character.

Figure 3.26 shows a spacecraft that we use to go some planet. Without using the spacecraft, we cannot leave the planet. Also in minimap it can be seen that a planet at the center has a green color and it is a result of being conquered by human team. In Figure 3.27, we are waiting on a planet to conquer it. There is no enemy in resource


**Figure 3.25:** Initial view in game.

area and so conquering process continues. Below the material balance bar, there is another bar fills from 0 to 100 percent to indicate the conquering process. After it reaches to 100, planet becomes conquered and planet shield gets activated by related color of the conqueror team. After the conquering, in Figure 3.28 shows that we can toggle an indicator arrow to find our spacecraft.


**Figure 3.26:** Using spacecraft.


**Figure 3.27:** Conquering a planet.


**Figure 3.28:** Indicator arrow points our spacecraft.

In Figure 3.29, there is a information table which gives statuses of the planets and related informations. Table contains this informations from left to right: planet name, planet status, conqueror race of the planet, main human players count, guard human players count, total human power, main replicant players count, guard replicant players count, total guard power and productivity. Also there is white arrows at right-most of some planets. This arrow is expandable as shown in Figure 3.30. It expands 5 buttons per type of characters. If there is enough material, then button of the type of character can clickable. After clicking the button, one guard character of selected type is added to planet.

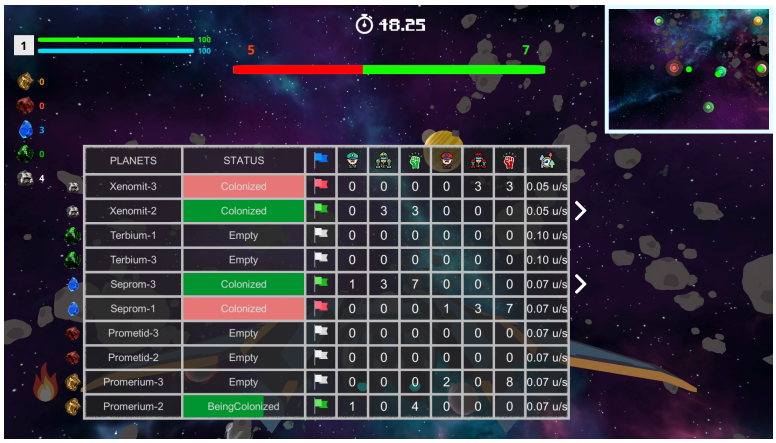


**Figure 3.29:** Information table in the game.

In Figure 3.31, there is a enemy spacecraft at lower left corner which is shooting at us. Spacecraft has its own health and shield values which are currently not shown in the user interface. After hitting to our spacecraft a few times, our spacecraft explodes and a spawn countdown appears as shown in Figure 3.32. We spawn on our home planet as out of the spacecraft with our character and our spacecraft stays at original spawn position. After the explosion, enemy AI character continues to its own action.

Figure 3.33 shows that an enemy AI character that interrupts our conquering process and attacks to us. It is enough close to us to being automatically targeted and showed in user interface. Another fight shown in Figure 3.34, two enemy AI characters can be seen with human AI guard characters and our character. One of the enemy character

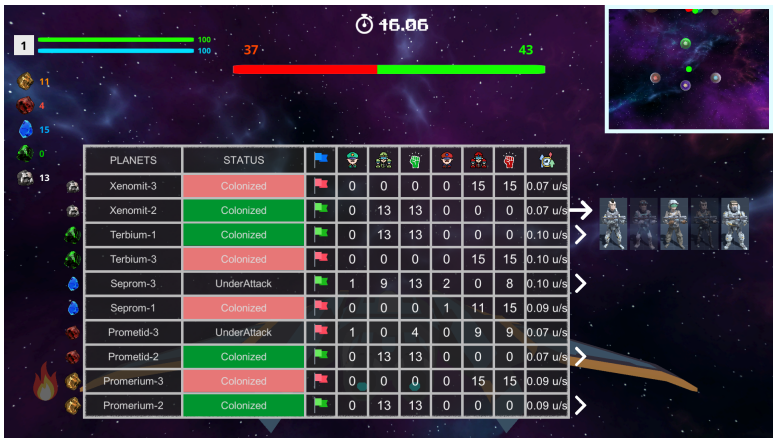


**Figure 3.30:** Adding guard characters.


**Figure 3.31:** Another spacecraft shooting at us.


**Figure 3.32:** Our spacecraft is exploded and waiting for spawn on home planet.

is dead and will be spawned in its home planet after the spawn countdown ends. Human AI guard characters are crowded because of collecting more materials. Two AI characters in our team adds guard characters when the material is enough and therefore, collecting material is effective in the game.

To be able to win the game, player must destroy all three planets of the enemy team. A home planet can be attacked only after its AI character dies 5 times. After that home

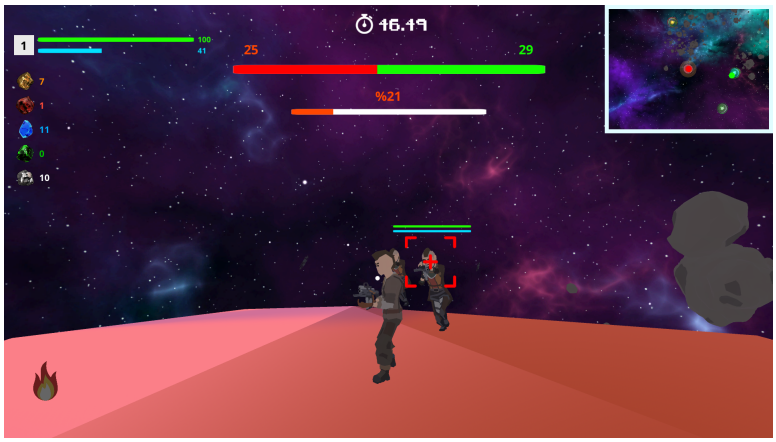


**Figure 3.33:** Enemy AI character stops to conquer process and shoots at us.

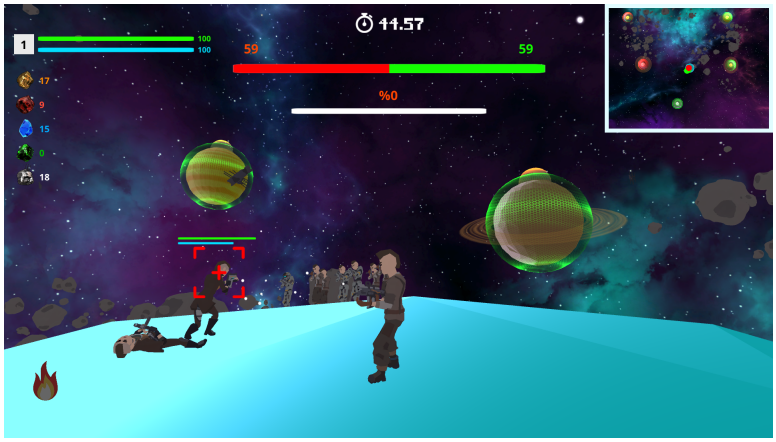


**Figure 3.34:** One dead and one alive replicant AI characters being attacked by human AI guard characters.

planet's planet shield becomes vulnerable and can be attacked. Figure 3.35 shows a home planet getting attacked to its planet shield. Later, in Figure 3.36, enemy home planet without a planet shield is being attacked. After the enough damage, planet destroy as shown in Figure 3.37. Enemy AI character of the destroyed home planet cannot spawn again after being dead.

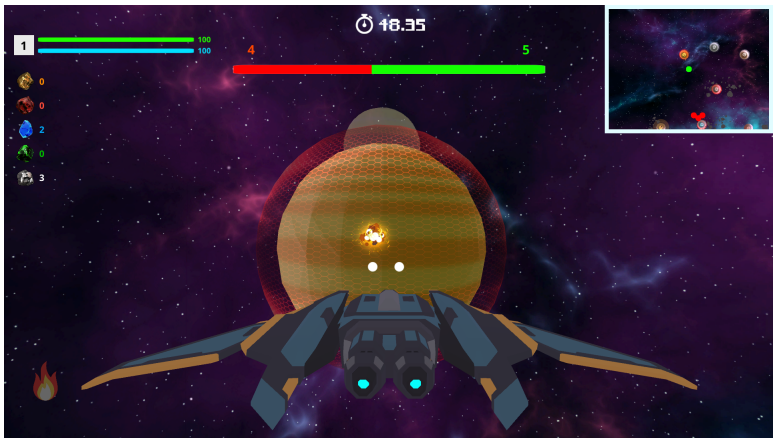


**Figure 3.35:** Enemy home planet with planet shield is being attacked.

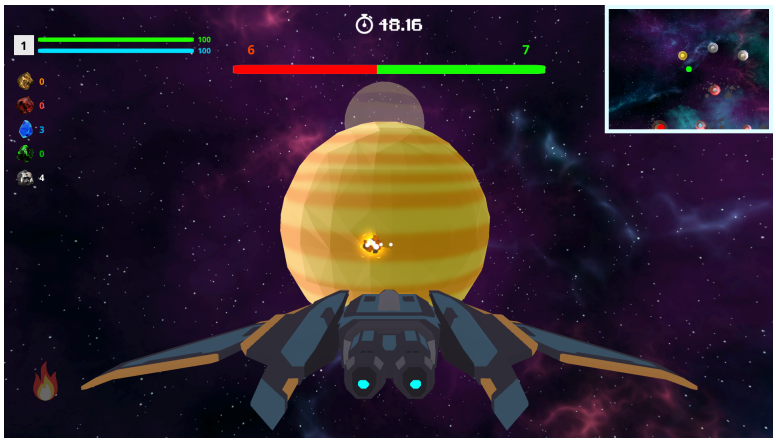


**Figure 3.36:** Enemy home planet is being attacked after planet shield is destroyed.

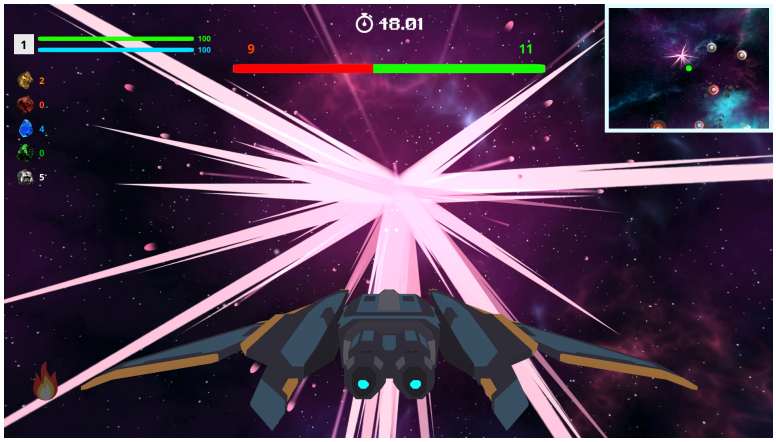


**Figure 3.37:** Enemy home planet is exploding.


**Figure 3.38:** Victory view.

# 4 Experimentation Environment

In this section, I will explain the statistics of the game like cpu usage, memory usage or related workloads while running on the device. In Figure 4.1, cpu usage at the start of the game is shown. Green area corresponds to Rendering, blue are to Scripts, dark green to various things like player loops (update functions of MonoBehaviour class) and

yellow area corresponds to VSync workload. At the upper limit, it can be seen that the cpu usage in one frame is almost 4 ms and it corresponds to 250 frame per second. In Figure 4.2, cpu usage is showed with after playing some time. In this graphic, Rendering and Scripts have more cpu time than others. Also Others (dark green area) is also has a high usage and there is spikes almost the entire time. I select a frame from the recorded interval by simply clicking to a spike as shown in Figure 4.3. There is also a hierarchy of profiler below the cpu usage window. This hierarchy shows the functions that are runned at selected frames. By sorting the used cpu time, we can detect the slow and not optimized codes. In the selected frame of Figure 4.3, update functions of MonoBehaviour based class are the most time consuming. After expanding to down, we can see our class names. In this frame, update function of AIGuardCharacter class was the most consuming function. If the update function of AIGuardCharacter class is nearly optimized, there would be still high cpu usage. That would happen because of the count of the objects that has AIGuardCharacter class. As the game progressed there would be high count of guard characters in the game. Therefore, call count for the update function will increase and game will become to have lesser frame per second. In Figure 4.3, there is also a Calls column that gives us the call count. As it can be seen, there is 130 call for update function of AIGuardCharacter in one frame and that usage corresponds to 0.29 ms. To overcome this problem, lesser game objects can be used or many objects can be controlled from one update function of one game object. To investigate further about the optimization, same way can be used with Profiler of Unity and can detect the time consuming parts.

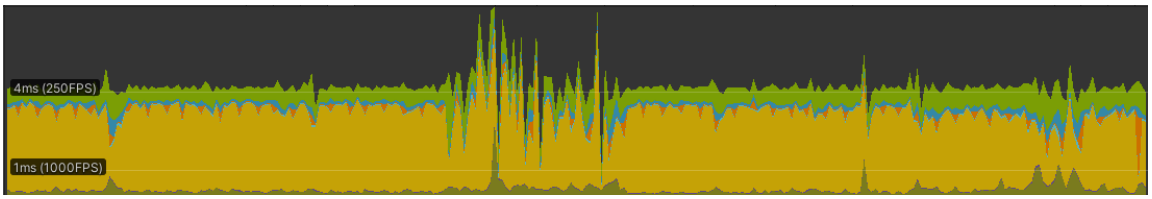


**Figure 4.1:** Cpu usage at start.

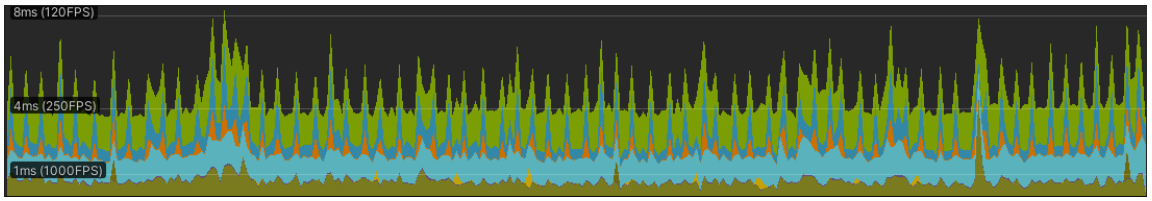


**Figure 4.2:** Cpu usage after a time.

There is also another tool called Frame Debugger. It shows the render operation one by one in one frame. Figure 4.4 shows us a frame with render operations. There is 183 different operations that are works in the frame. At left, there is another hierarchy which belongs to Frame Debugger. Render operation start with the camera since we see the game world by the camera. It start with the opaque objects and renders transparent objects over. There can be seen the planet names at Draw Mesh events. Shortly, if Rendering takes too much time in cpu usage, it can be inspected with Frame Debugger.

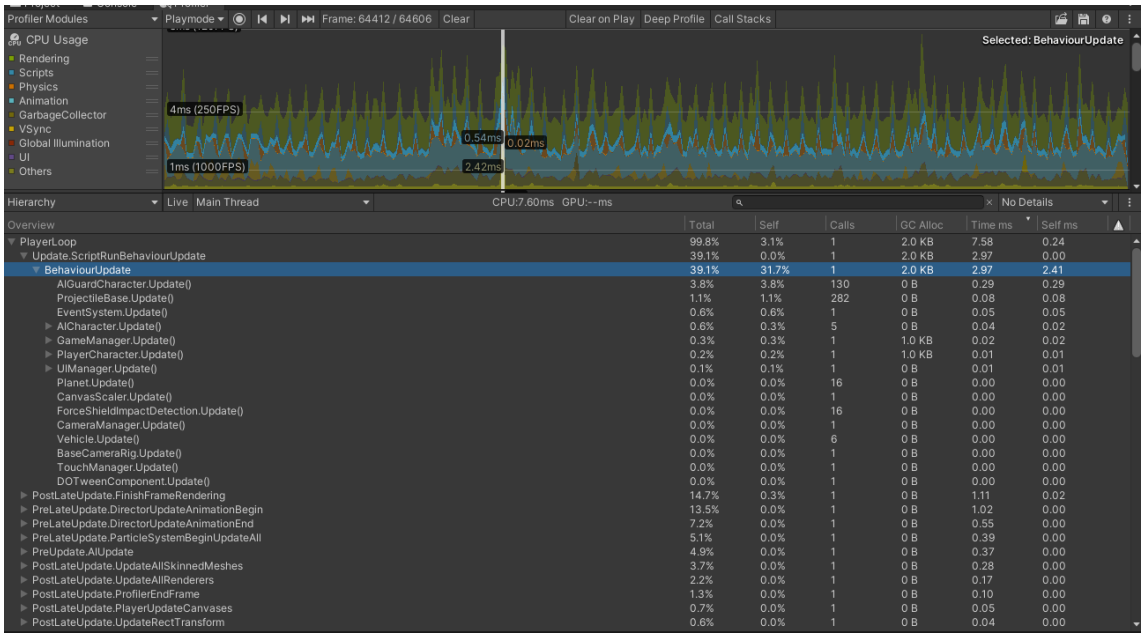


**Figure 4.3:** Executed functions on a selected frame.

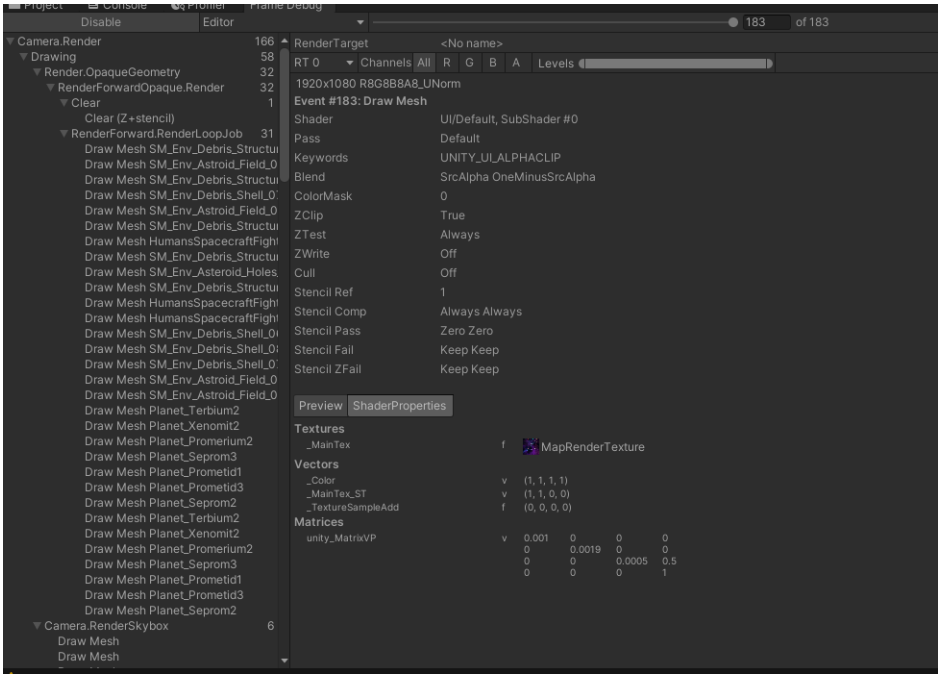


**Figure 4.4:** Frame debugger on a random frame.

# 5 Comparative Evaluation and Discussion

In the game, there is a gravity simulations with the many planets. Aim of the gravity simulation at the start of the project was a realistic gravity while couple of planets can affect at a time. After the project is being advanced, I realized that realistic gravity would not have much fun. It would have a ponderous and heavy effect on the other objects. Player would not have much fun with slow and boring movement. Therefore, I changed the gravity to much responsive and less gradual.

Network results were accurate when it tested at the beginning. After it decided that network would be make much complex and time consuming, it is removed as a future work.

Pathfinding in 3d space is explained in section 2.

# 6 Conclusion and Future Work

To conclude, I think this game project can be continued on. It is designed to be able to add new things to the game such as new characters, weapons, planets or game modes. Also, multiplayer feature was removed as a future work and it can be added after the game completed for single player. Game can be optimized by the design of the gameplay with counts of the AI characters or AI guard characters. Lastly, some game plays that are not affect the game balance but affects the game picture can be corrected.

# References

[1] T. Muratov and A. Zagarskikh, "Octree-Based Hierarchical 3D Pathfinding Optimization of Three-Dimensional Pathfinding", Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control, 2019. Available: 10.1145/3386164.3386181.

[2] T. Muratov and A. Zagarskikh, "Octree-Based Hierarchical 3D Pathfinding Optimization of Three-Dimensional Pathfinding", Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control, 2019. Available: 10.1145/3386164.3386181.

[3] T. Muratov and A. Zagarskikh, "Octree-Based Hierarchical 3D Pathfinding Optimization of Three-Dimensional Pathfinding", Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control, 2019. Available: 10.1145/3386164.3386181.

[4] A. Albaghdadi and A. Ali, "3D Path planning of fixed and mobile environments using potential field algorithm with Genetic algorithm", 2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON), 2019. Available: 10.1109/iemeconx.2019.8877086.

[5] A. Albaghdadi and A. Ali, "3D Path planning of fixed and mobile environments using potential field algorithm with Genetic algorithm", 2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON), 2019. Available: 10.1109/iemeconx.2019.8877086.

[6] A. Albaghdadi and A. Ali, "3D Path planning of fixed and mobile environments using potential field algorithm with Genetic algorithm", 2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON), 2019. Available: 10.1109/iemeconx.2019.8877086.