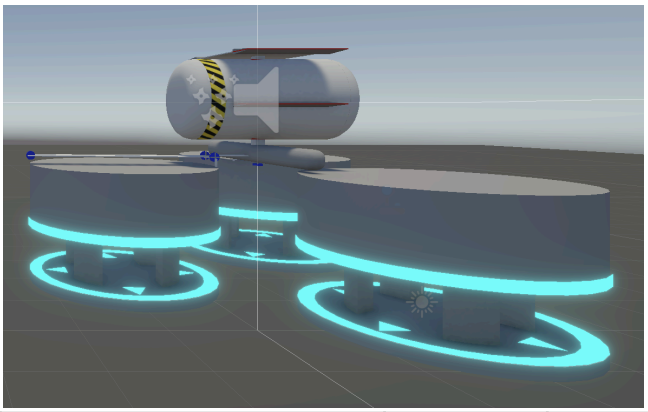
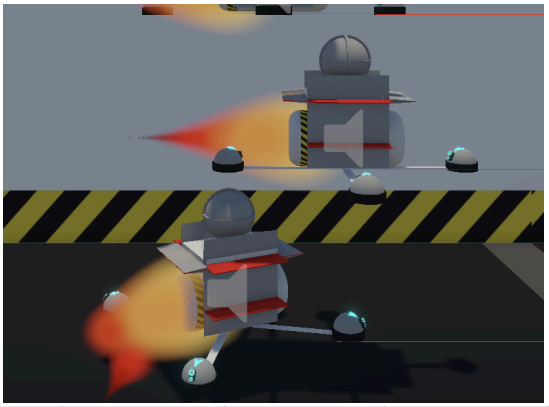
Introduction to Robotics: Project 3 Report

Team 9
Name: 송우영, 양성지, 박정훈

1. Performance Record

	Map1	Map2	Map3	Map4	Map5
Time [s]	41.48	40.13	37.74	37.30	35.24
Number of Collisions	0	2	1	0	1
Number of Collected Items	7	7	7	7	7
Final Time [s]	6.476994	15.13234	7.744949	2.295471	5.238724

Final Average Time[s]: 7.3776956

2. Participation Details

A. Drone, Mothership Design

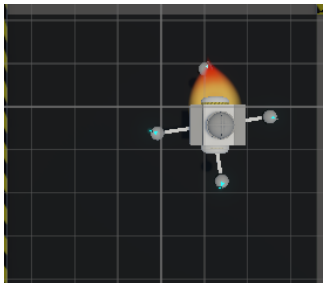
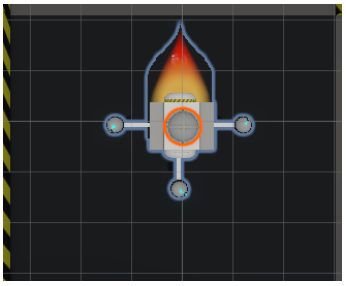
Iteration 1) Drone Design & Starting Process (Seong-Ji Yang)

Design Objective:
The goal of this iteration had 5 parts of the following :

1. **Implementation of Gyroscopic Effect**

Motivation: When tested with a fixed preset index 0 in MazeManager, we found that the first cube that the drone encountered when coming out of the starting position had rotational force, which kept pushing the engine, thus making the movement hard to control. This made us wish for a more stable rocket, guiding us the way we wish while ignoring the external force.

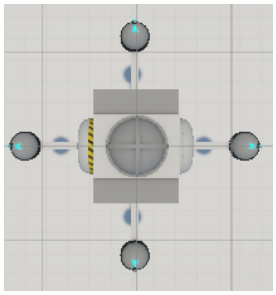
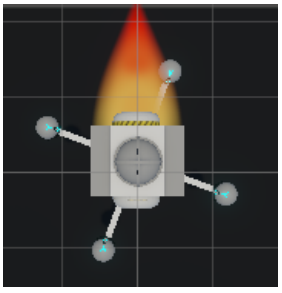
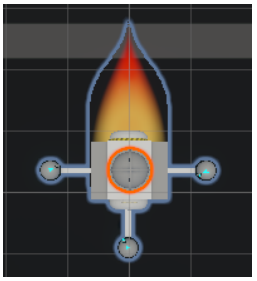
Verification Process: We decided to use a servomotor for this problem. Since the drone's Y-axis is fixed, we only needed to keep the x and z-axis stable. To calculate the rotational force, we placed the IMU at the center of the drone body to achieve the exact rotational force. From the calculation of the IMU, the servomotor turns the opposite direction, with the same speed as the IMU, making the control easier.

before : without gyroscopic design	after : with gyroscopic design
	
heading to unwanted direction (not exactly -z direction)	heading to the wished direction (exactly -z direction)

2. **Proper Lidar Attachment & Fixed Direction**

Motivation: if the lidar was attached to the top of the drone body, the cube was 12 * 12 size, and lidar had only 5 radius. Therefore, we needed more lidars to clearly identify the next path when the drone was at the center of the cube to choose the next direction. Also, each lidar had to have a globally fixed position to have no confusion.

Verification Process: we make a cross with frames with length of 1.5. since the wall had 1.0 thickness, and it overlapped, making 0.5 distance from the border of the cube to each side, the drone could now identify that the wall existed when it could only see 4.0 ahead. Also, each lidar had rotational value for (0, 90, 180, 270 degrees) since the lidar could only sense direction when it is first set, and can't sense other directions. We used the same gyroscopic design as the engine to make it have a globally fixed direction.

Initial lidar direction	4 lidars + no gyroscopic method	4 lidars + gyroscopic method
		
lidar direction to where it is stretched out	lidars turn in unwanted direction	lidars having their own direction

3. **Starting Delay**

Motivation: When we first used the lidar to detect the wall and use it to move, the drone always crashed to the wall.

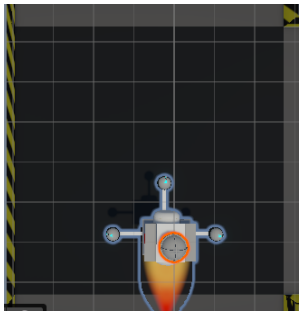
Verification Process: To find out why we used debugging to see what was exactly going on. To have no variation with the testing, we used fixed preset index 0 in MazeManager. It turns out that the lidar sensed before all the map was loaded, thus the lidar sensed correctly at that time, but when the map was completely loaded, it turned out to be false information. Therefore, we used time as a buffer to load all the map, then sense with the lidar.

Without starting delay	With starting delay
[16:37:14] [LIDAR Scan] Pos:(0.00, 0.00, 0.00) | -Z: 5.00m | -X: 5.00m | +Z: 4.00m | +X: 4.00m | UnityEngine.Debug:Log (object)	[16:35:10] [LIDAR Scan] Pos:(0.00, 0.00, 0.00) | -Z: 5.00m | -X: 4.00m | +Z: 4.00m | +X: 4.00m | UnityEngine.Debug:Log (object)
say -x direction has no wall, even if there is	can sense wall correctly

4. **Arrival Padding Space**

Motivation: When we used 12.0 as the difference between current destination point and the next destination point, it turned out that the drone kept oscillating around the destination point. We found that due to the speed of the drone and the frame's time stamp, the GPS could not sense that the drone arrived, thus making the drone turn around.

Verification Process: We added padding radius around the destination point, making it more easier to detect that the drone arrived at the destination point desired.

without padding space	with padding space
	
the drone oscillate	the drone acknowledges its destination

5. **ServoBehave Modification (extension of Lidar Fixed Direction)**

Motivation: When we tried to align the lidars perpendicular to the X or Z-axis, we encountered an issue where the lidars rotated unintentionally as the drone's engine turned. This happened because the lidar's servo motor is structurally dependent on the engine's orientation. Thus, as the engine rotates, the lidar's frame of reference rotates with it.

Verification Process: To solve this, we needed the absolute angle of the engine to compensate for its rotation. Therefore, we added the GetCurrentAngle() function to the ServoBehave script to accurately retrieve the child object's current rotation at the global level, and now the lidars worked as we wanted.

```
public float GetCurrentAngle()
{
    if (childTf != null)
        return childTf.localEulerAngles.y;

    return transform.localEulerAngles.y;
}
```

The newly added code for neutrality of the Lidars

Iteration 2) Physics-Based Inertia Control (Jeonghoon Park)

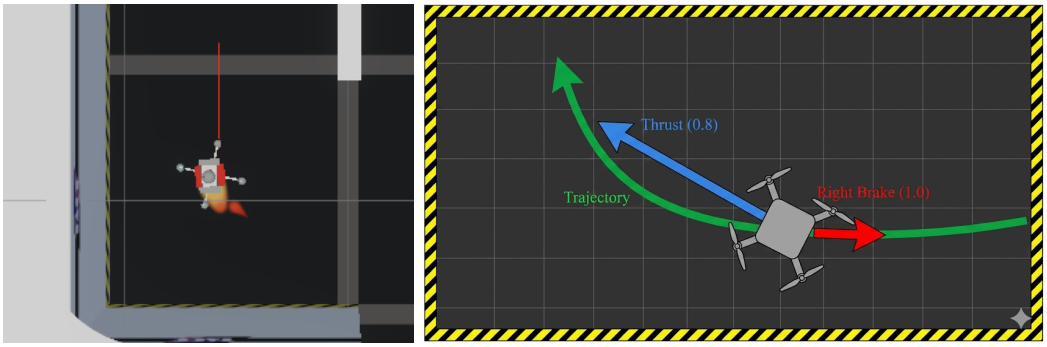
Design Objective:

To overcome the time inefficiency of the previous "Stop-and-Go" movement, we implemented a Dynamic Aerodynamic Braking System.

1. Lateral Stability: Side airbrakes are permanently engaged (1.0f) to prevent lateral drift during forward motion.
2. Cornering: During 90° turns, thrust is throttled to 0.8 to prevent overshooting, while the servo directs the drone smoothly.
3. Rapid Deceleration: critically, for Dead-end reversals (180°) or entering a Wait state, the Front and Rear Airbrakes are also fully engaged at maximum intensity (1.0f). This executes a "Full Brake," allowing the drone to stop instantly or turn around without drifting.

Verification Process:
First, we verified the cornering mechanics using a Force Vector Analysis (as shown in the figure), confirming that the differential braking generates the necessary centripetal force. Second, to quantify the performance gain, we measured the Total Time to Reveal 'Map ID=2', comparing the previous "Stop-and-Go" logic against the new "Continuous" logic.

Results:



The transition to continuous navigation drastically improved the overall mission speed. By eliminating the need to stop at every cell, the drone could maintain momentum throughout the complex layout of Map ID=2.
Outcome: The total time required to fully reveal the map was reduced from 46.23 s to 27.48s, representing a significant improvement in exploration efficiency.

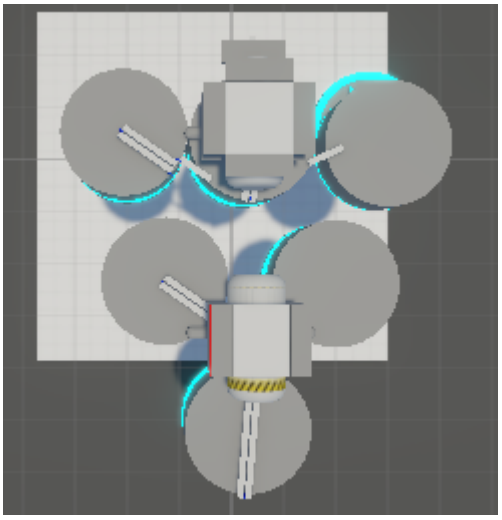
Iteration 3) Mothership & Dronesilo Design (Wooyoung Song)

Design Objective: The goal of this iteration had 2 parts of the following :

1. **Circular Design (Symmetric structure)**

Motivation: Due to the size and number of the drone silos, we needed to make the mothership densely to avoid collision with the wall as much as possible. In conclusion, we thought of placing each drone silo taking place at each 120 degrees from the center of the drone body, creating a form between triangle and a circle.

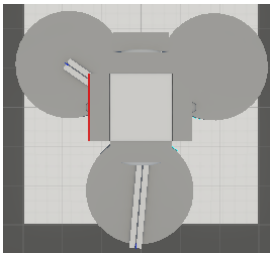
Verification Process: Also, when we used the same gyro mechanism as the drone design, as the Mothership body turned, then the silos would also turn. Therefore, linear design would increase the possibility of collision. And when we used padding space to always stay a bit of a distance away from the wall, this circular design would need less padding, allowing more speed due to momentum.



2. **Height Adjustment**

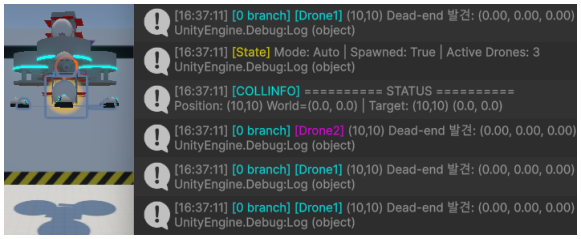
Motivation: if the drones were at the same y axis, then it could mean collision between the drones. thus making control and detection with the lidar even more difficult. Therefore, as in the best rocket model in project 1, we used servo motors and frames to lower the drone silos so that each drone will be located at different angles.

Verification Process: We used logging to see if the lidar and drones were creating collision or false alarm of dead end. Then, we calculated the length of the drone, then provided each drone silo with no frame, frame with the height of one drone, and the frame with the height of two drones, which were divided to minimize the width of the Mothership.



After this, we checked the log to see if our implementation was working properly and the drones were not overlapping.

Drone collision & disruption of lidar	no collision & disruption

	
console log shows lidar malfunction	console log shows no problem

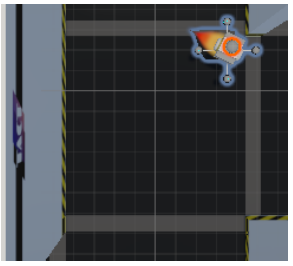
B. Drone Algorithm

Iteration 1) One Drone Automatic Design (Wooyoung Song)

Design Objective: The goal of this iteration had 2 parts of the following :

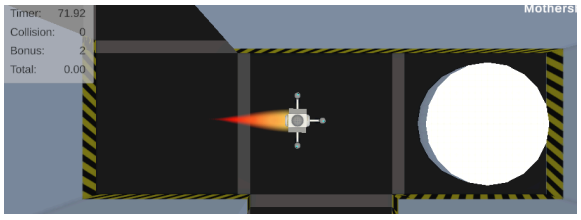
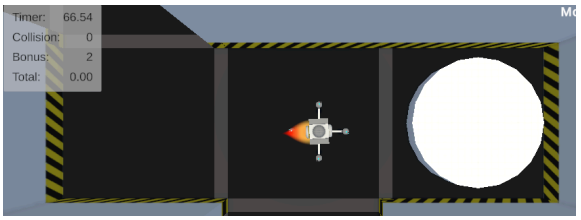
1. **Direction Decision Making**

Motivation: As we thought about minimizing the travel time of the drone, we thought the drone should turn before it reaches the next destination (next cube's center). Therefore, we made the drone turn if the next cube had any opening at the side of the drone's sides.
Verification Process: The drone needs to have a distance of less than 1.0 to know the opposite side, whether it is an open path or a wall, and less than 5.5 to know its sides. So we used the drone's GPS and lidars to check if it is in the next cube, and is open to at least one of its sides, then the drone will move towards it.

before : less efficient movement	after : more efficient movement
	
drone changing position at the center	drone changed its direction as it entered

2. **Weighted score system**

Motivation: We needed to prevent the drone from heading to the dead end again, or going back the way it came. Therefore, we used a scoring based process and increased a lot of scores to make sure of dead ends, or the path that the drone encountered.
Verification Process: From debugging, we found that the problem was due to little difference in the score, so we made the drone mark a bigger score the way it passed, and maximum score at the dead end.

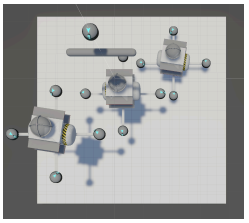
before : takes some time	after : less time taken
	
due to low score change	is sure of dead end now

Iteration 2) Expansion to 3 Drones & Map State Saving and Recording (Jeonghoon Park)

Design Objective:
Previously, the system operated with a single drone. Even when expanded to three, they moved as a cluster without sharing information, leading to inefficiency. If the leading drone moved toward a non-dead-end path, the others would follow blindly, missing "stars" (rewards) located in other branches. To resolve this, the goal was to implement a Multi-Agent System with a Shared Map. The design intention was for the first drone to record its path and discoveries on a global map, allowing subsequent drones to read this state and decide whether to follow or explore a different route.

```
public enum CellState
{
    Fog = 0,                // 미탐색 영역
    DiscoveredOpen = 1,     // 발견된 열린 공간
    VisitedUndecided = 2,   // 방문했으나 분류되지 않은 지점
    Branch = 3,             // 분기점 (갈림길)
    DeadEnd = 4,            // 막다른 길
    Trunk = 5               // 주 경로
}

// 맵 상수
public const int MAP_GRID_SIZE = 21;        // 그리드 크기 (0..20)
public const int MAP_CENTER_INDEX = 10;     // 중심 인덱스 (원점)
public const float MAP_CUBE_SIZE = 12.0f;   // DroneControlUnit과 동일한 셀 크기
```

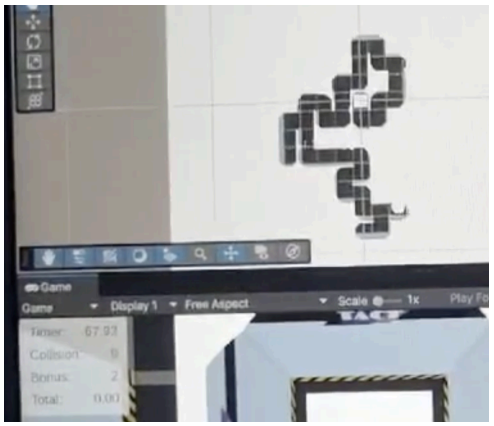
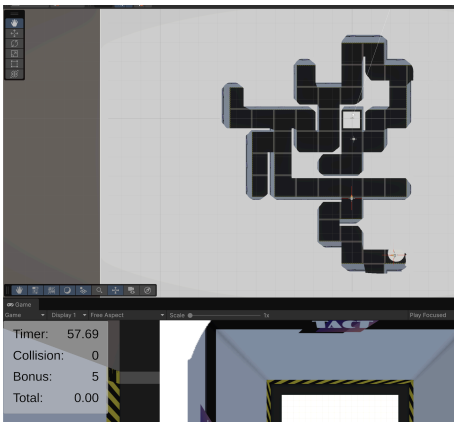


Verification Process:
Identity & Hierarchy Verification: We debugged the Unity Hierarchy view to ensure that Drone_0, Drone_1, and Drone_2 were instantiated with correct internal IDs and properly mapped to the global manager.
Cooperative Behavior Logic: We compared the behavior of a single drone versus a three-drone swarm. Specifically, we verified if the drones exchanged information via the shared map (e.g., one explores, others wait) rather than moving identically.

Results:
Through debugging logs, we confirmed successful information exchange. As shown in the logs, while Drone 1 explores "Right(+X)" and Drone 0 explores "Back(-Z)", Drone 2 correctly enters a "Wait" state because no new paths are available.

```
[03:30:51] [0 branch] [Drone1] (10,9) → 탐색: Right(+X)
UnityEngine.Debug:Log (object)
[03:30:51] [0 branch] [Drone0] (10,9) → 탐색: Back(-Z)
UnityEngine.Debug:Log (object)
[03:30:51] [0 branch] [Drone2] (10,9) → 대기 (Unknown/Trunk 없음)
UnityEngine.Debug:Log (object)
```

Quantitatively, the three-drone system demonstrated significant performance improvements over the single-drone system.
Time Efficiency: Reduced mission time from 67.93s to 57.69s (approx. 15% improvement).
Exploration Coverage: Increased collected stars (bonus) from 2 to 5 (2.5x improvement).

	
One Drone (Before): 67.93s, resource: 2	3 Drone (After): 57.69s, resource: 5

Iteration 3) Rank-Based Algorithm for Two-Branch (Jeonghoon Park)

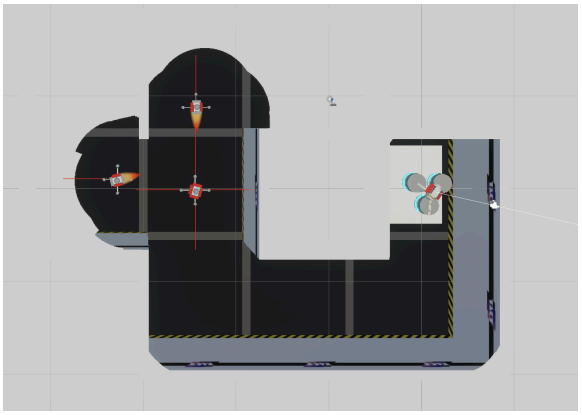
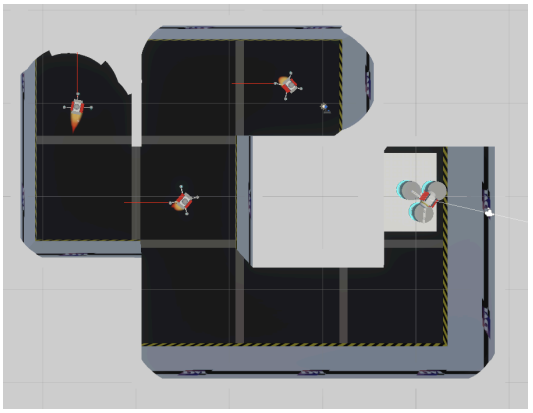
Design Objective:
We identified that most map structures split into two branches. In the existing "Greedy" approach, the third drone (Rank 3) would randomly follow one of the leaders. This posed a high risk of all three drones entering a dead-end simultaneously, wasting significant time.

Goal: To minimize this time risk, we designed a "Wait & Join" algorithm. The Rank 3 drone is programmed not to explore aggressively but to Wait at the branch. It only moves once the leaders identify the correct path (Trunk) or a new branch, ensuring efficient resource allocation.

Verification Process:
This algorithm was verified by integrating the Global Map Manager (Singleton) with the drone's State Machine. We conducted a simulation in a Two-Branch scenario to confirm two specific behaviors:

1. Station Keeping: Verified that the Rank 3 drone maintains the Wait state at the branch while Rank 1 and 2 explore.
2. Dynamic Response: Checked if the Rank 3 drone immediately switches to the Move state and joins the confirmed path upon receiving a DeadEnd report from a leader.

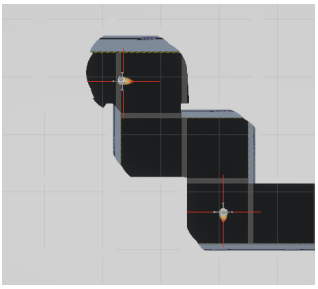
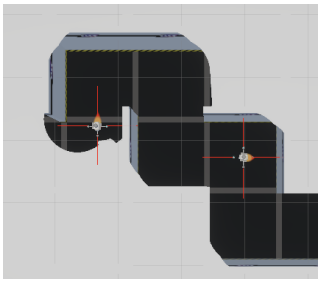
Results:

	
rank=1: Go Straight (Exploring) rank=2: Go Left (Exploring) rank=3: **Wait** (Standby at Branch)	rank=1: Finds Dead-end => rank=3 rank=2: On Trunk => rank=1 rank=3: Joins Trunk direction => rank=2

As shown in the figure, the rank-based logic functioned correctly, enabling dynamic role switching. Crucially, this mechanism successfully prevented the worst-case scenario where two drones simultaneously enter dead-ends, thereby significantly reducing wasted time and optimizing exploration efficiency.

Iteration 4) Transition to Node-Based Topological Map (Jeonghoon Park)

Design Objective:

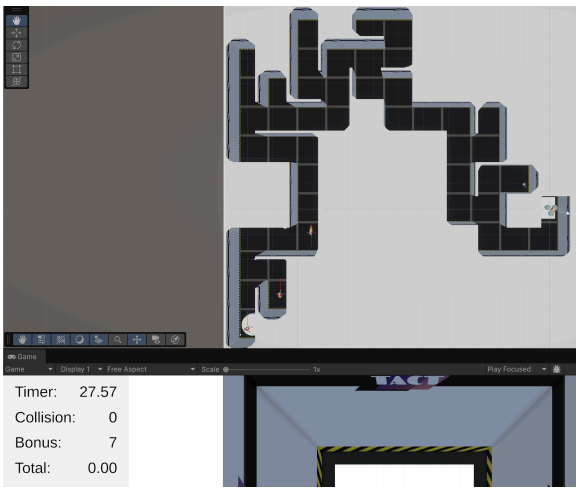
	
Rank=1 passes the branch first and records the instruction "Rank=2 must turn right." The marker is clearly visible on the map before Rank=2 arrives.	As Rank=2 arrives later, the wall detection data overwrites the branch navigation data in the shared memory. Consequently, Rank=2 loses the path and blindly follows Rank=1.

The goal was to resolve the Data Overwrite Conflict found in the Grid Map system, where wall detection in narrow corridors erased navigation instructions. To prevent this, we transitioned to a Node-Based Topological Map. This approach decouples navigation nodes from physical obstacle data, ensuring pathfinding instructions are preserved regardless of the terrain.

Verification Process:

We conducted Scenario-Based Testing on Map ID=2, which features narrow corridors prone to overwrite errors. The evaluation focused on Data Persistence: verifying if the Rank 2 drone correctly executed the stored "Turn Right" instruction instead of blindly following the leader, confirming that branch data remained intact.

Results:



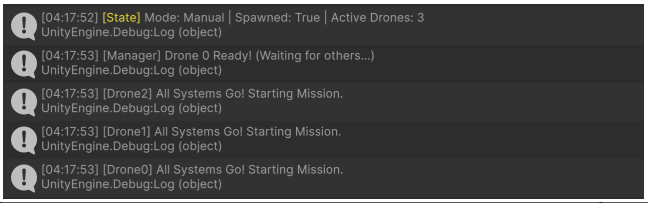
The transition successfully ensured data integrity. As a result, the "Blind Follow" issue was resolved, and the system achieved 100% map completion, increasing the number of collected resources from 5 to 7.

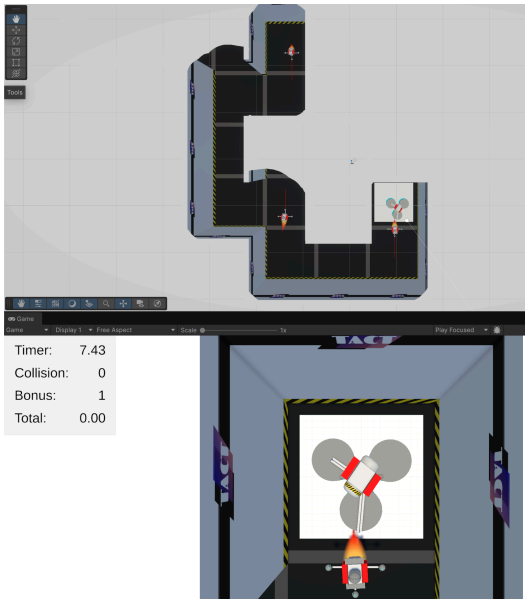
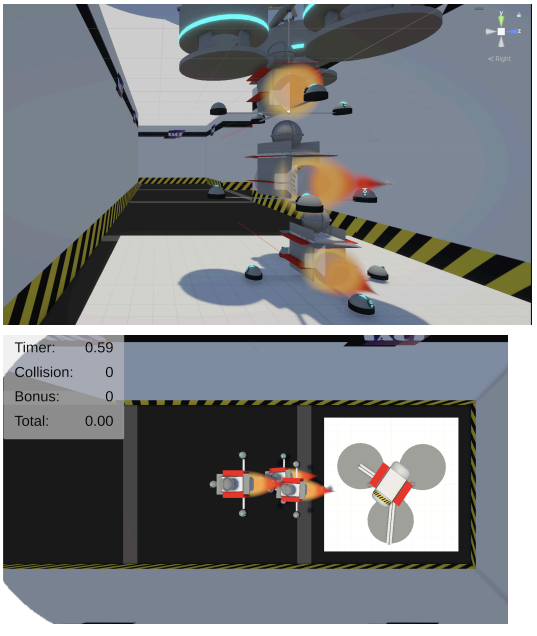
Iteration 5) Optimization of Deployment: Synchronized Departure Logic
(Idea: Seong-Ji Yang / Implement: Jeonghoon Park)

Design Objective:
After implementing the DroneSilo system, a latency issue occurred due to the sequential spawning process. Previously, each drone would depart immediately upon instantiation. Problem: The mission timer resources when the first drone leaves the starting cell. However, since the 3rd drone is still spawning, a significant time gap is created. This forces the Mothership to delay its operations until the last drone is fully deployed. Goal: To resolve this, we implemented a Global Synchronization Algorithm. Even if a drone spawns early, it enters a Standby phase at the center and holds its position until a global "All Ready" signal is broadcast by the Manager. This ensures that all three drones depart simultaneously in formation.

Verification Process:
The design was evaluated by comparing the "Time to Departure" of the last drone. We measured the timestamp when the 3rd drone exited the starting cell and observed whether the drones maintained a cohesive formation during the initial deployment phase.

Results:
After the last drone spawns and heads toward the center, three drones simultaneously take off.



	
Top: Progress of other drones when the last drone departs. **Bottom:** Departure time of the 3rd drone (7.43s).	**Top:** Simultaneous departure of three drones after gathering. **Bottom:** Departure time of the 3rd drone (0.59s).

C. Control system design

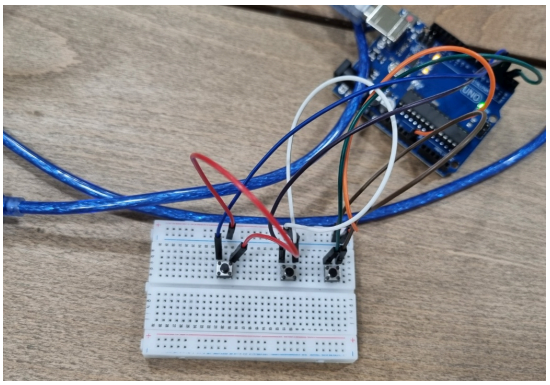
Iteration 1) Automatic & Final Manual Design (Wooyoung Song)

Design Objective:

After implementing the mechanism of the drone, we tried to implement the Mothership's algorithm. We found that at the beginning, the Mothership only needs to follow the path in which more than two drones passed by, which indicates that this path is not a dead end. This method ensured secure passage until the 7th branch. However, we encountered a problem where we could not know which path was the accurate destination. Therefore, until the 7th branch, we used automatic design, and from the point of the 7th branch, we implemented manual control.

Verification Process:

The design was verified with the following. We colored each drone differently, to know what drone was connected to what button. Then, if one of the drones found the end point, we pressed the button connected to that drone, which made the Mothership know the final destination.



3. Results

Ultimately, the system was finalized as a Multi-Agent System where three drones cooperate using a Global Shared Map. The flight control utilized a Dynamic Aerodynamic Braking System to automate continuous high-speed navigation and inertia management. Strategically, a Rank-Based Topological Map algorithm was implemented, allowing drones to autonomously synchronize their departure, preserve path data without conflicts, and dynamically switch roles (Explore vs. Wait) to efficiently cover the map without redundant movements.The Mothership is at the highest location including the drones. Thus, it spawns the drones by lowering the drone silo using frame & servo motors, and then retrieving them before the drones get started. Only after all three drones are spawned and begin to move, the Mothership moves based on the drone's detection of branch (intersection), following the path towards the next branch.

4. Discussion (Optional)

4.1 How did the work from Projects 1 and 2 help in Project 3?

Project1: A general understanding of Unity and design using frames proved helpful.
Project2: I was able to effectively control inertia using the brakes.

4.2 What would you improve if you had more time?

The rank=1 drone uses a heuristic algorithm to assess the map distribution, prioritizing exploration toward areas with a high probability of dead-ends, rather than a fixed priority algorithm of always advancing, left turn, right turn. This prevents scenarios where the rank=1 drone drifts away from rank=2 and rank=3 drones, reducing efficiency. Furthermore, it learns to prioritize dead-end paths by leveraging deep learning to understand the map structure.

Since the mothership and drones following already explored paths move toward the center and don't necessarily need to measure, they are optimized to turn more sharply by reducing their build size.

These were not addressed due to time constraints.