30th September 2020

Gliding in Space

Introduction

My implementation of Gliding in Space covers stage b and stage c. In this document I will outline the planning process I went through to come up with my final design including the initial problems I solved and the different iterations of my design. I will then discuss the robustness of my design, based on multiple tests of each stage. Finally, I will reflect on my design.

Initial Planning Problems:

Globe Location

The first problem I solved was how ships would find the globe in part b, and globes in part c. The simplest solution I came up with was for ships to scan for globes, and if they find one then broadcast its location in a message and follow it.

Sending and Receiving Messages

The next problem I solved was how to send and receive messages, and what those messages would need to communicate. First, I wanted vehicles to send messages when they found a globe, so I wanted them to include the globe's location in the message. I realised they would also need to forward on these messages, as not all vehicles would be in range of the one that found the globe. In order to ensure ships weren't receiving old messages, meaning the globe would be in a different position, I added a time component to the message. I also decided to include the ships ID, in case I would need to use it later.
I then thought about when a ship would need to receive messages. This would be if the ship didn't have a globe near it, so wouldn't be able to find a globe without messages. If this was the case, I also wanted the ship to ignore old messages and only pay attention to the newest. Finally, I wanted the ship to forward on that message, so other ships would be able to find the globe.

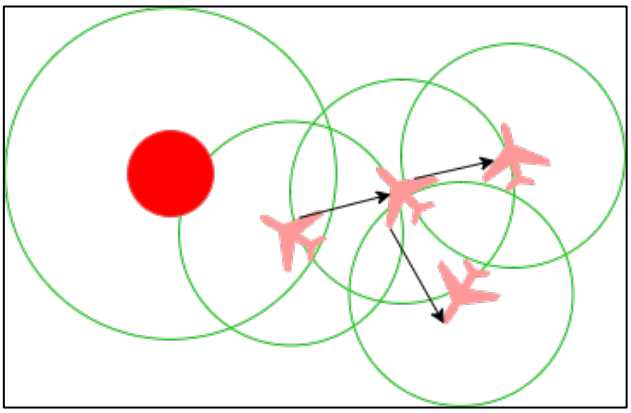

Figure 1 – Inter-Vehicle Message

As can be seen in Figure 1, if a vehicle finds a globe it sends a message to the vehicles around it. As they cannot see the globe, they receive the message. They then forward the message on to the surrounding vehicles.

Vehicle Actions

The final problem I solved was what actions the vehicles should make. The first idea I had was for a vehicle to follow any globe it found. I then realised they would also have to get close to the globe to charge. Therefore, I based the vehicles actions around what charge level they had. These actions were to follow the globe at a low speed if they had a high battery level, to go to the globe to charge at a high speed of they had a low battery level, and to return to following at a low speed if they had finished charging to a high battery level.

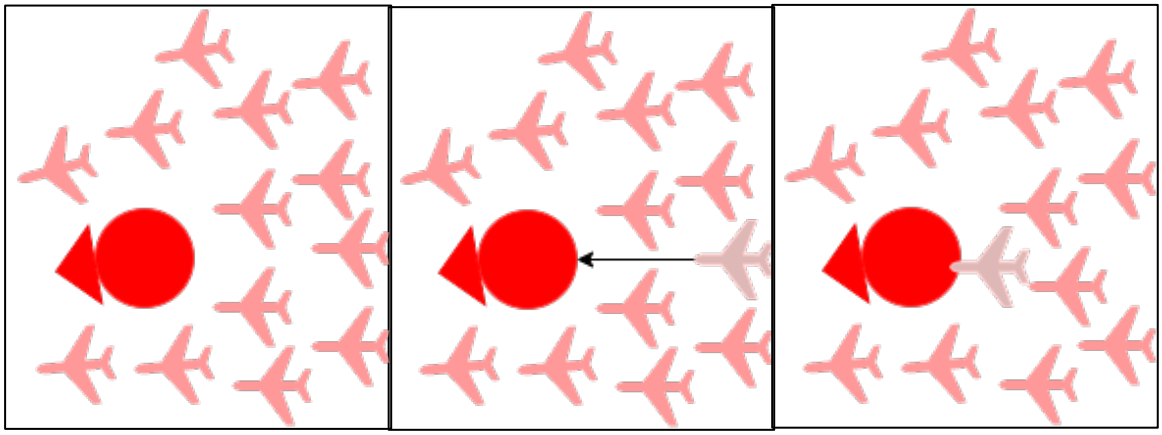

Figure 2 - Vehicle Charging

In the Figure 2, when the battery of the vehicle towards the back becomes low, it goes to the globe at a high speed to charge. Due to the collision mechanics, the other planes move out of the way for it. The other vehicles continue to follow the globe, as their battery has not reached the low threshold.

Implementations for Stage B:

Design 1:

My first attempt at a solution included all the answers to the problems discussed above. When a globe is located, vehicles send a message containing the position of the globe, the time the globe was seen, and their ID. The vehicle then follows the globe. If a vehicle cannot see a globe, it checks to see if it has received a message(s). It then finds the most recent message and goes to the globe location contained in the message. If its charge drops below a low threshold (set to 50%), it goes to the globe's location with a maximum throttle. When it has charged to a high charge (90%), it returns to following the globe with a low throttle.

Figure 4a

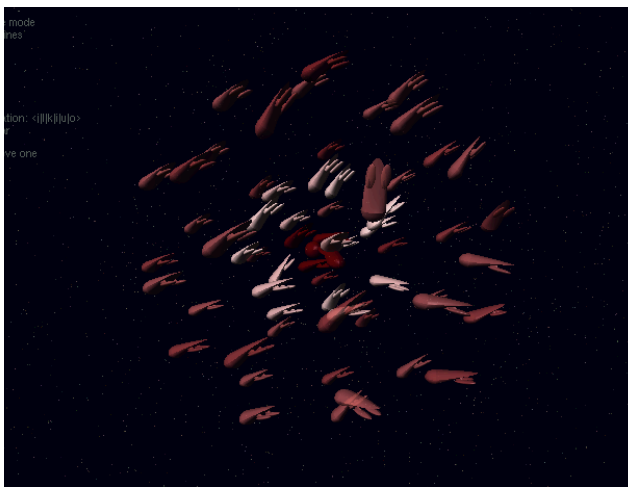

Figure 3b

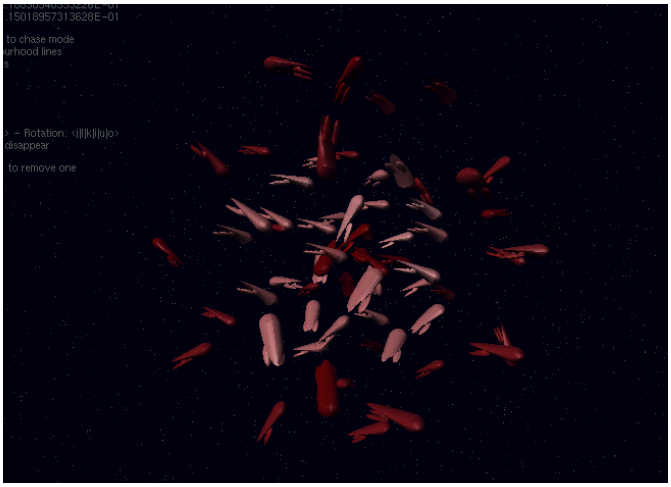

Figure 6c

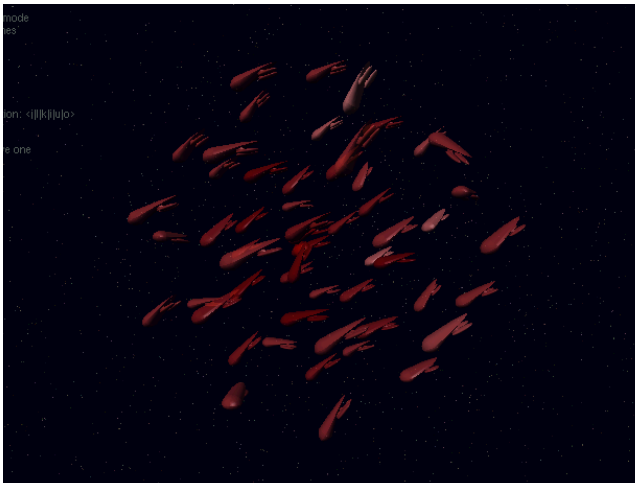

Figure 5d

Outcomes:

As can be seen in Figure 3a, the vehicles follow the globe happily at first. However, after a short amount of time many vehicles have to charge at once, so they all rush to the globe (Figure 3c). This causes vehicles to become stuck and not able to reach the globe, causing a large percent to die (around 17% for a population of 64, 25% for a population of 200). For a small population after the initial die off the population can sustain itself fairly well as they do not get stuck in rushes, and the charging becomes more staggered. However, for larger populations, vehicles easily get stuck in large rushes and so more ships die off until the population becomes small enough for the rushes to be smaller scale and for charging to become staggered.

Design 2:

My second design attempted to fix the biggest flaw in the first design – all the vehicles rushing to charge at the same time. I tried to do this by including if a vehicle was charging or not in the message they sent. This would allow other vehicles to know if others are already charging. Therefore, vehicles can only go to charge if the vehicle who sent the message they received wasn't charging. This enforces vehicles to take turns charging. I also had to increase the threshold to go and charge, so vehicles had a longer chance to make it to the globe to charge before dying. Otherwise, they would die before it was their turn to charge.

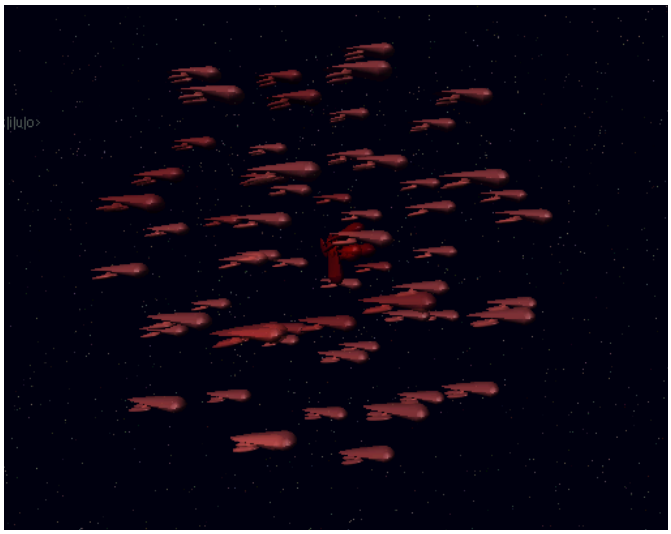

Figure 4a

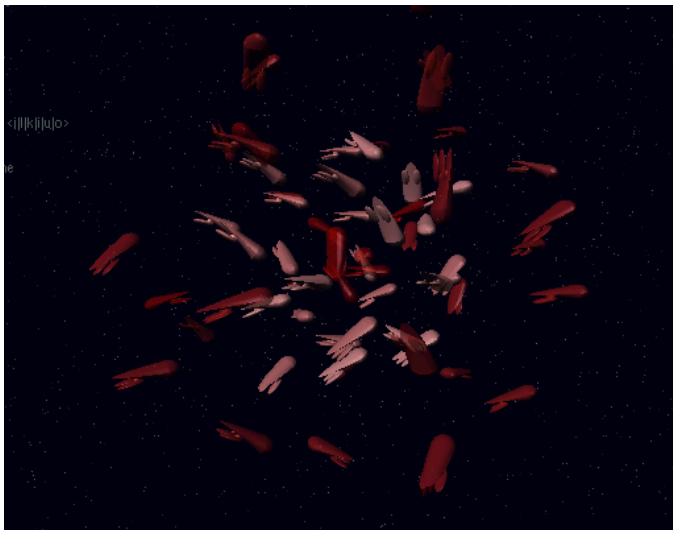

Figure 4b

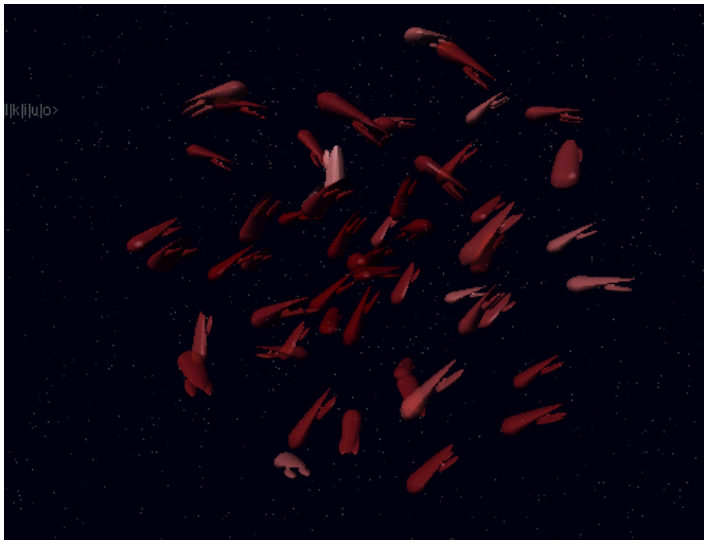

Figure 4c

Outcomes:
Again, there was a bit of a drop off in population after the initial charge rush. This was caused by a couple of vehicles who were unmoving from the globe, making it impossible for more than a few vehicles to get close enough to charge at once. However, after this initial drop off, those ships moved, which can be observed in figure 4c. After they moved there was a slight increase in vehicle survival, however this change didn't end up improving much upon design 1 in survivability.

Design 3:

My final design improved upon design 2 by dividing ships into octants around the globe by using their ID mod 8. I then separated these octants from the globe by using a radius. Instead of directly following the globe, they would follow a new point defined by the x/y/z position of the globe plus or minus the radius depending on the octant. The octants are given by:

	Result of ID mod 8 (Plus or Minus Radius)							
	0	1	2	3	4	5	6	7
x	+	-	-	+	-	-	+	-

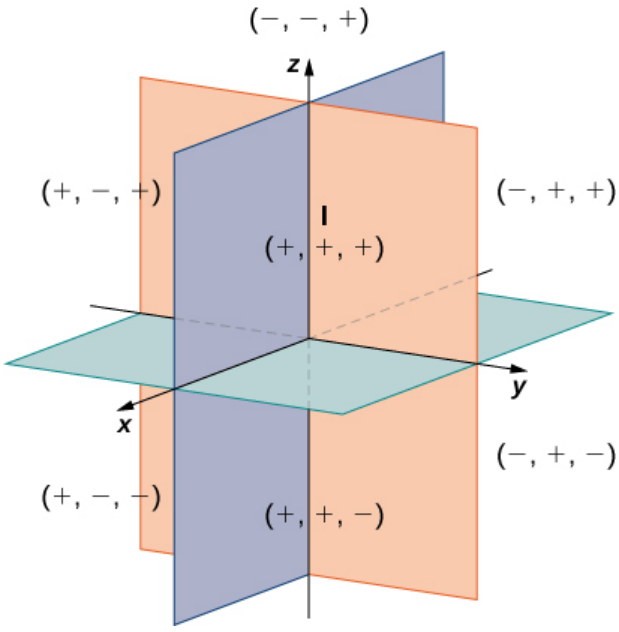

Figure 4 – Illustrations of Octants

Source: "Vectors in 3D", Strang, Gilbert and Herman, Edwin, 2020

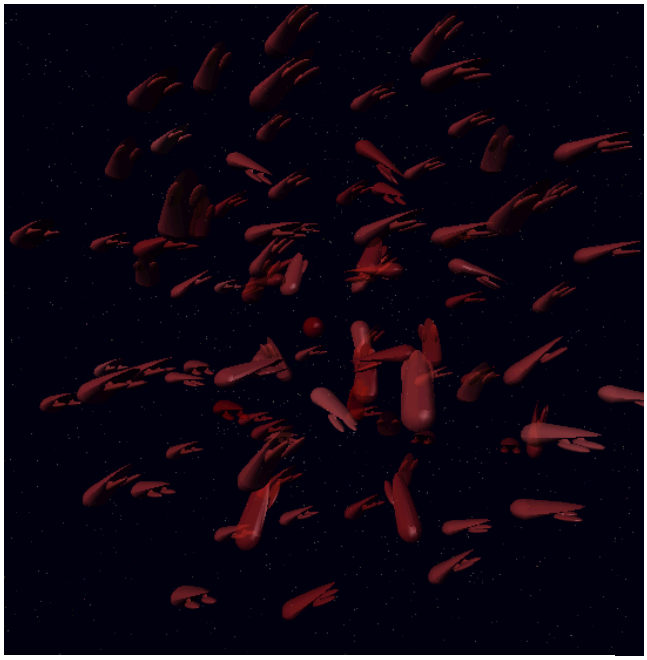

Figure 5a

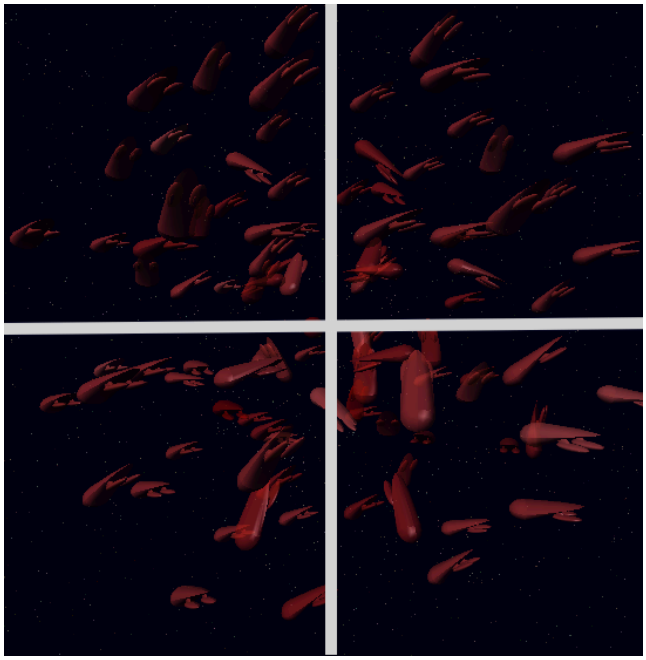

Figure 5b

y	+	+	-	+	-	+	-	-
z	+	+	+	-	+	-	-	-

Outcomes:
This successfully fixed the globe crowding problem after a bit of trial and error to find a suitable radius. The octant separation can be seen in Figure 5a, highlighted in 5b.

Implementations for Stage C:

Design 1:

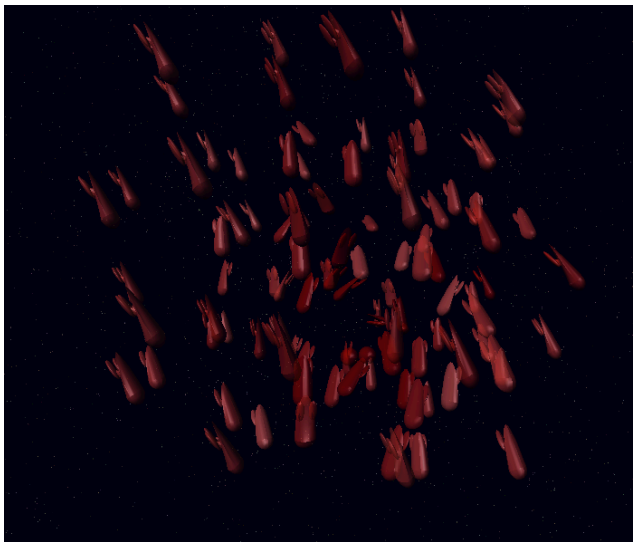

Figure 6a

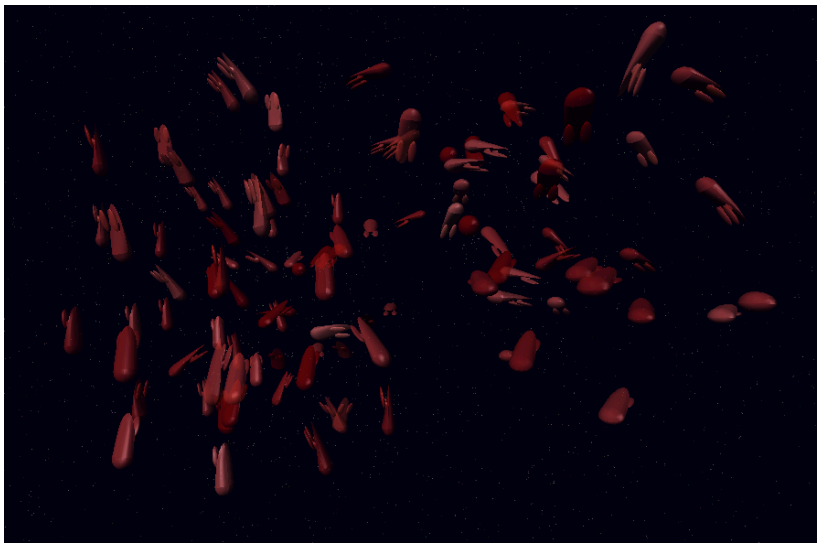

Figure 6b

My first design for stage C consisted of
me running my code for stage B with Random_Globes_In_Orbits.

Outcomes:

My code for part B worked really well for part C. Due to the nature of the vehicles, they follow which ever globe is closest to them, or was seen most recently if there isn't a globe nearby. This results in groupings such can be seen in Figure 6a, as different groups of

vehicles follow different globes. If their globe disappears, they can easily find another due to the incoming messages. However, occasionally a vehicle would lose a globe and due to the separation from the radius distribution it would be out of range to find a new one.

Design 2:

My second design included a slight tweak to the radius, so the groups are closer together while still leaving globes with some breathing room for vehicles needing to charge.

Outcomes:

Changing the radius to a smaller number fixed the issue in Design 1. Visually, they look very similar as it can be hard to see the octant grouping when vehicles are moving around different globes, however after numerous tests (see below) the infrequent occurrence of a vehicle losing a globe in Design 1 and dying didn't occur after the reduce of the radius for a size of 64.

Experimental Analysis

Part B: (3 trials)

Time Elapsed/ Initial Size	1 Min	2 Mins	3 Mins	4 Mins	5 Mins
2	2/2/2	2/2/2	2/2/2	2/2/2	2/2/2
5	5/5/5	5/5/5	5/5/5	5/5/5	5/5/5
10	10/10/10	10/10/10	10/10/10	10/10/10	10/10/10
50	50/50/50	50/50/50	50/50/50	50/50/50	50/50/50
100	100/100/100	100/100/100	100/100/100	100/100/100	100/100/100
150	150/150/150	150/150/150	150/150/150	150/150/150	150/150/150
200	200/200/200	200/200/200	200/200/200	200/200/200	200/200/200
250	250/250/250	250/250/250	250/250/250	250/250/250	250/250/250
300	300/295/297	296/290/295	292/289/295	291/287/293	286/287/291

Bad frame rate at 250: 15Hz
Even worse at 300: 12Hz

Other tests (one trial):
2 after 30 mins: 2
250 after 30 mins: 250

Discussion:
From the above data, it can be seen that my solution for part B is robust for a range of vehicles from 1 to 250. At 300 vehicles, there is some die-off caused by the large amount of ships needing to recharge at once, and the single globe not being able to cater for them all. This die off seems to stop with a population of 250, with no die off after a 30 min test run. This leads to my conclusion that the vehicles can be sustained from a population of 1 to 250.

Part C: (3 trials)

Time Elapsed/ Initial Size	1 Min	2 Mins	3 Mins	4 Mins	5 Mins
2	2/2/2	2/2/2	2/2/2	2/2/2	2/2/2
5	5/5/5	5/5/5	5/5/5	5/5/5	5/5/5
10	10/10/10	10/10/10	10/10/10	10/10/10	10/10/10
50	50/50/50	50/50/50	50/50/50	50/50/50	50/50/50
100	100/100/100	100/100/100	100/100/100	100/100/100	100/100/100
150	150/150/150	150/150/150	150/150/150	150/150/150	150/150/150
200	200/200/200	200/200/200	199/200/197	199/199/195	199/198/195
250	250/250/250	247/241/248	236/241/236	225/231/227	222/229/227

Bad frame rate at 250: 15Hz

Other tests (one trial):
2 after 30 mins: 2
150 after 30 mins: 149

Discussion:
From the data above, it can be seen that my solution for part C is robust from a range of 2 – 149. I was surprised that it could not handle a larger population than Part B, due to more globes so less crowding. However, I saw that this was not the case, due to groups following other globes colliding and causing vehicles to be unable to reach the globe. There was also infrequent die off from ships losing globes and being out of communication range although this happened much less frequently than in the first design before the radius was adjusted, and only for the larger groups (above 150).
Following my 30-minute test run on a population of 2 and 150, I conclude that the vehicles can sustain a population between 2 and 149.

Reflection

From my findings, I believe my designs were fairly robust for both a small population and a larger population. My design technique worked a lot better for the single globe world. This is most likely due to the initial design being formed around what was working for a single globe, and the design for the multiple random globe world being based entirely on it. In theory, the multiple random globes should be able to sustain a much larger population than the single globe, due to allowing more vehicles to charge at a given time. In my design, vehicles distribute themselves somewhat randomly, being attracted to whichever globe is the closest making even distribution between all globes unlikely. A potential solution to allow a larger population would be to distribute vehicles between globes evenly. Unfortunately, due to time constraints I was unable to test this solution. I also believe that the single globe would be able to sustain a greater population. One idea for this I also didn't have the time to implement would be to add more grouping locations further away from the globe. Vehicles would then cycle between the locations, going to closer locations when they would need to charge soon and further away locations when they had finished charging.

Overall, I am very pleased with the outcome of my solution. Although there is room for improvement, I think it is a simple solution which provides vehicle survivability for a large range of vehicles.

References

My submission was entirely my own work, apart from these references:

Figure 4: Strang, Gilbert and Herman, Edwin, 'Vectors in Three Dimensions', Mathematics, 2020, Accessed 30/09/2020, https://math.libretexts.org/Bookshelves/Calculus/Book%3A_Calculus_(OpenStax)/12%3A_Vectors_in_Space/12.2%3A_Vectors_in_Three_Dimensions