**DELIVERY DRONE SIMULATOR**

****

**A Project by:**

**Muhammad Shahmeer Shaheedi (565007) – Logic Designer & Tester**

**Shubham Kumar (571672) – Programmer & Git Manager**

**Project Overview:**

The code presented deals with a delivery drone, allowing it to make its own decisions based on weather patterns and other, later stated conditions. To be more specific, it will choose whether to delay the expedition, conduct the delivery normally, recharge, or cancel the mission and mark it as a “failure” altogether. It seems to be an interesting solution to optimize delivery businesses by introducing automated machines that make choices on their own to ensure the safety of the package, and even save time, considering the fact that the drone will be able to avoid the daily on-ground traffic. Along with this, it will save the company money that usually goes into hiring and paying manpower, allowing them to cut on labor costs (operational costs might not reduce significantly, though).

**Program Design/Logic:**

We allowed the system to pick random numbers to simulate real-world problems in this case. For example, we had one random number picker to simulate weather conditions (sunny, windy, or rainy) and also one to determine if there will be an obstacle in the drone’s regular route, causing the device to reroute and drain more battery for one trip.

Then, as for the main function, where we let nature and chance run their due time, we used a for loop for 3 deliveries. The loop was nested with if-else statements, each depicting the steps the drone takes automatically. For clear, sunny weather, the delivery goes normally, giving us a battery drain of 10-25 percent per trip. If the conditions are windy, and the battery is below 40 percent, the drone goes back to the station to recharge. If the weather’s rainy, the delivery gets delayed. Furthermore, the presence of an obstacle in the pre-determined route causes the drone too reroute, which of course results in additional 5% battery loss.

The flowchart for each weather-based decision made is presented below. Keep in mind that this is only one iteration of 3, which is why there is an arrow going back from “END” to “START”. This indicates the end of the prior loop/iteration and the start of a new one. Once again, this is purely based on environmental conditions. Had we made a flowchart representing all the possibilities in the code (e.g the presence of obstacles, system malfunctions, and user-commanded recharging session), it would have been a large and messy flowchart with several junctions of crisscrossing arrows and blocks so small it would be difficult to read their contents. However, we are open to the possibility of a larger, but easy to follow flowchart. The current flowchart, though, is present on the next page:

START

“Delivery

Successful”

10<Battery rain < 25

Weather =

Sunny?

If battery<40, recharge (+10%)

10< Battery Drain< 25

Weather =

Windy?

?

“Delivery Successful”

Battery Drain = 0

Weather =

Rainy?

“Delivery Delayed”

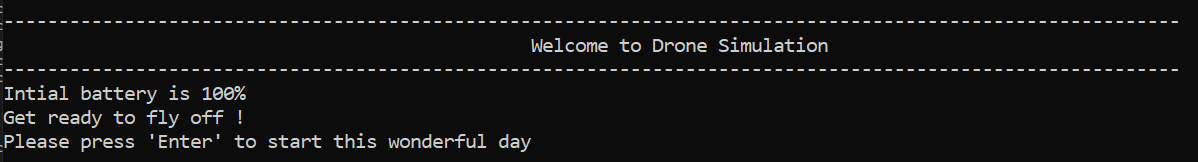
END

**Execution Instructions:**

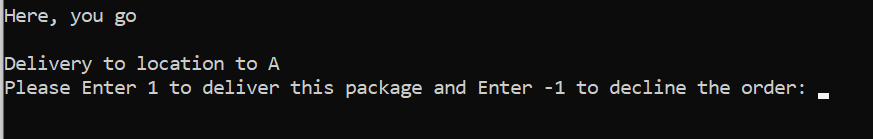
To run the code/commands, we obviously resorted to C++ and the C++ compiler, the use of which was taught during the course prior to the assignment.

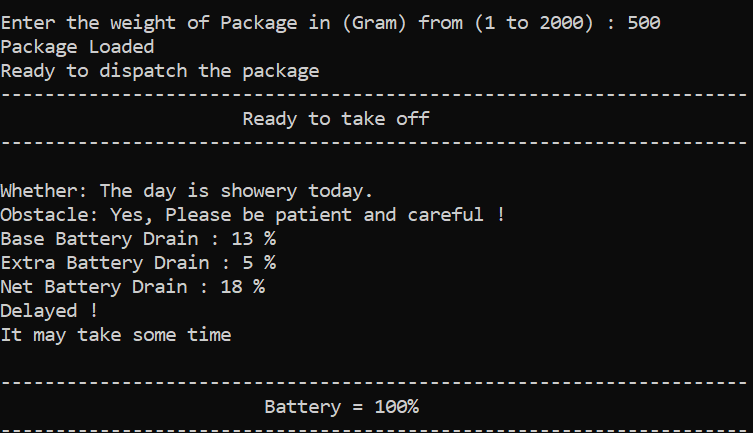
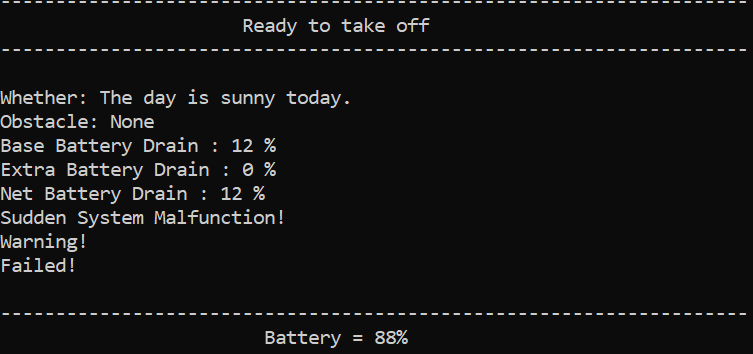
We made a few assumptions while carrying out this program. For example, whenever the user gave the instruction of battery recharging, the battery would always fill up/ reach 100 percent. Moreover, we assumed that the presence of an obstacle will always add an extra battery drain of 5%. Also, we made the random seed use the current time (time since epoch).

When the program is run, the viewer initially sees a welcome message like so:

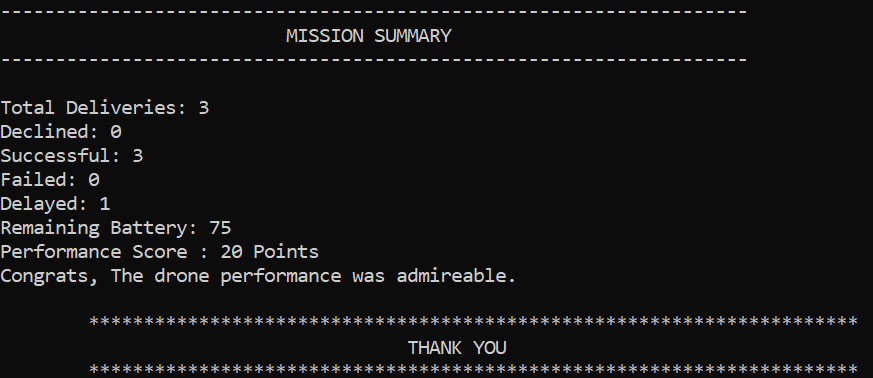


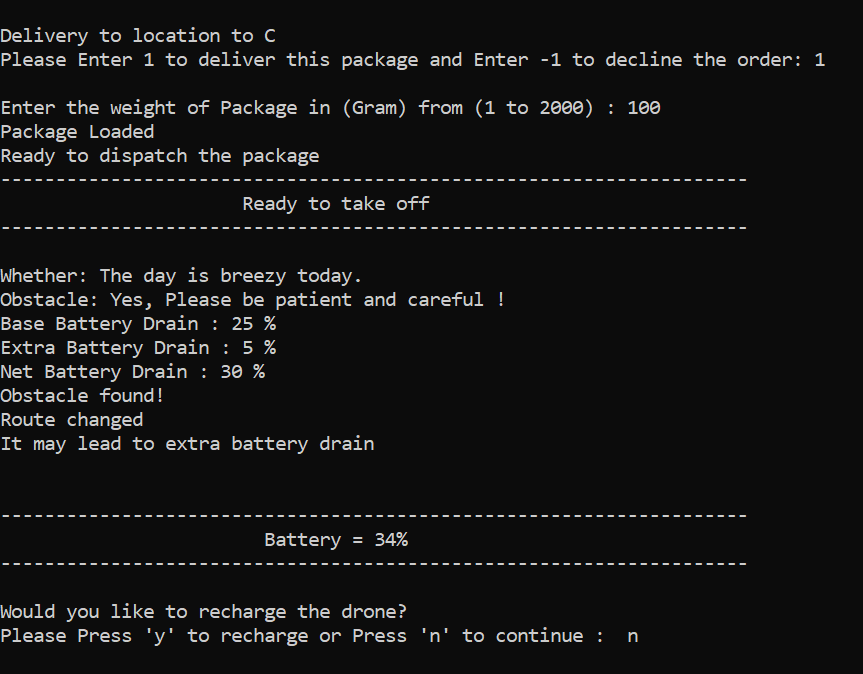
Then, when one presses “Enter”, he is asked if he wants to accept/deliver the order or decline it:

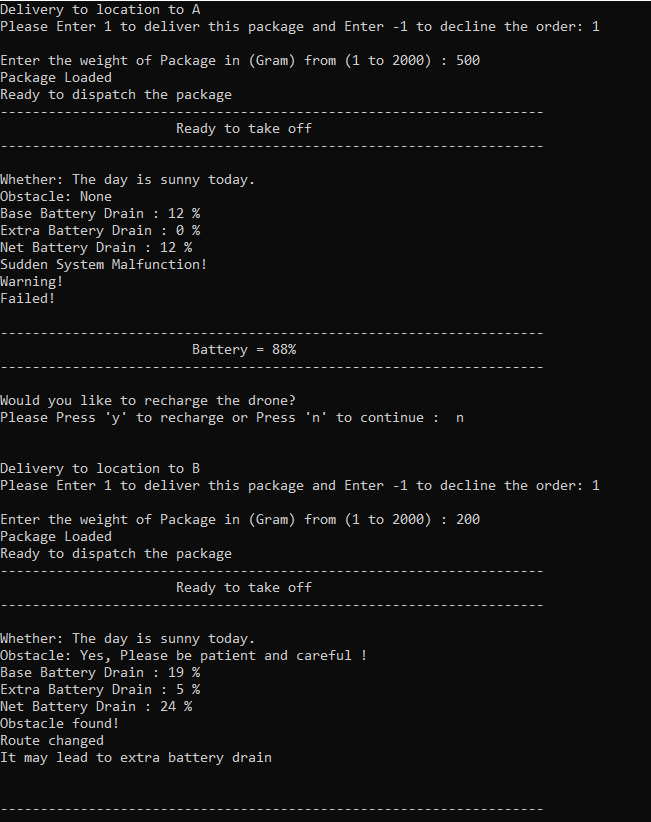


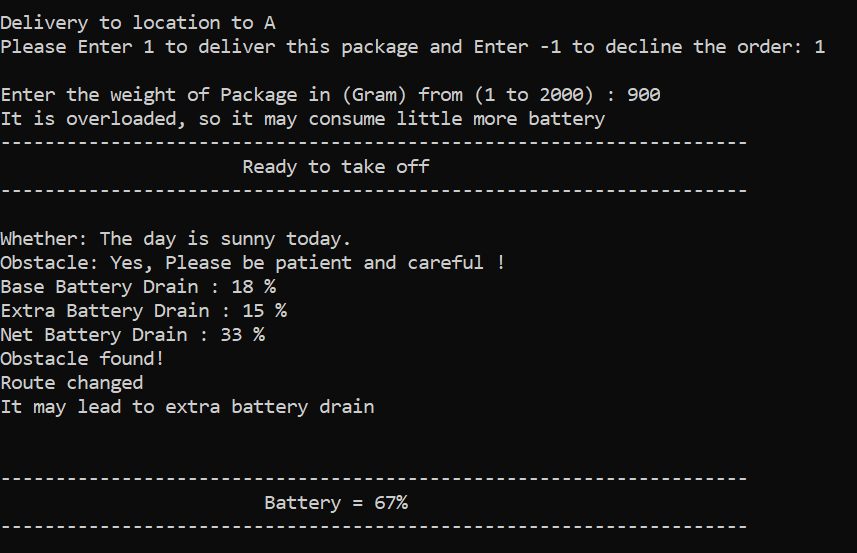
Upon entering 1, the program asks us to enter a weight (in grams) of the object (1-2000). Then, the weather, the presence of obstacles, and the battery drainage (both base drainage and extra drainage due to weight) is simulated and we are presented with a message like so:

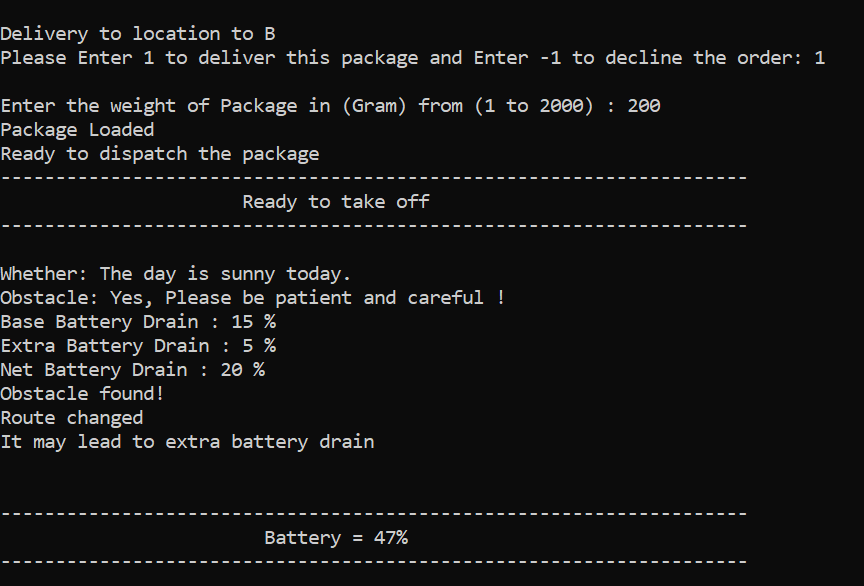
This process gets repeated until all the deliveries have either been successful or have failed. At the end, we see a performance evaluation summary, highlighting the number of successful, failed, and delayed deliveries. There’s also a performance score displayed at the end:

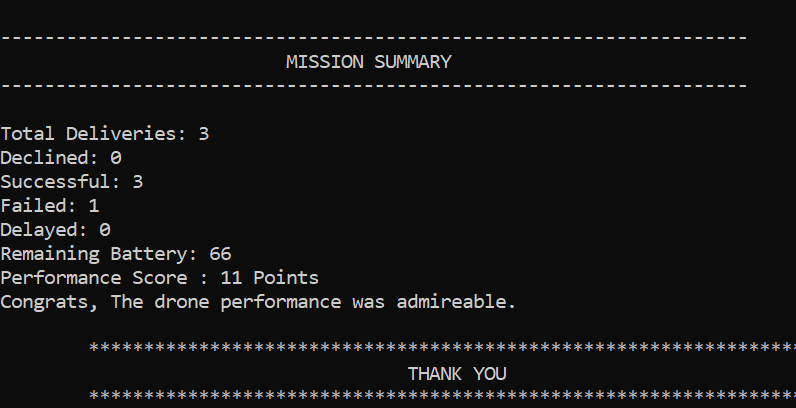
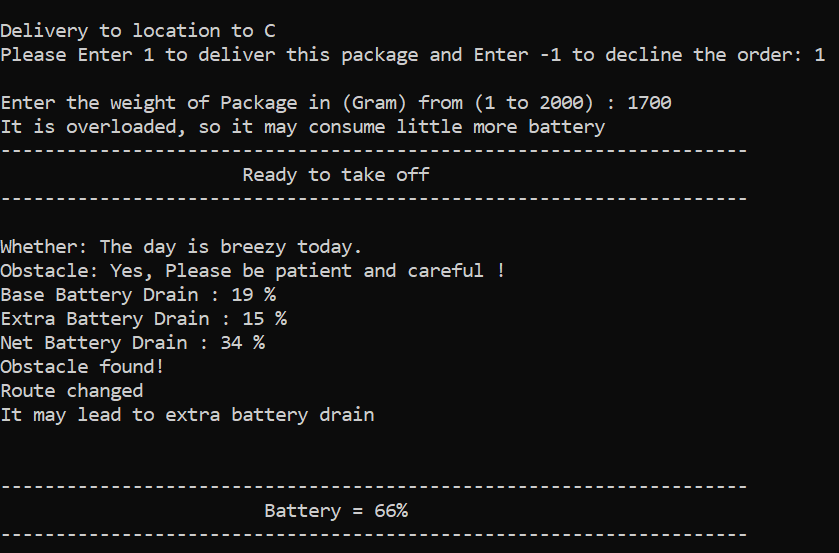


Test Case 1:



Test Case 2:

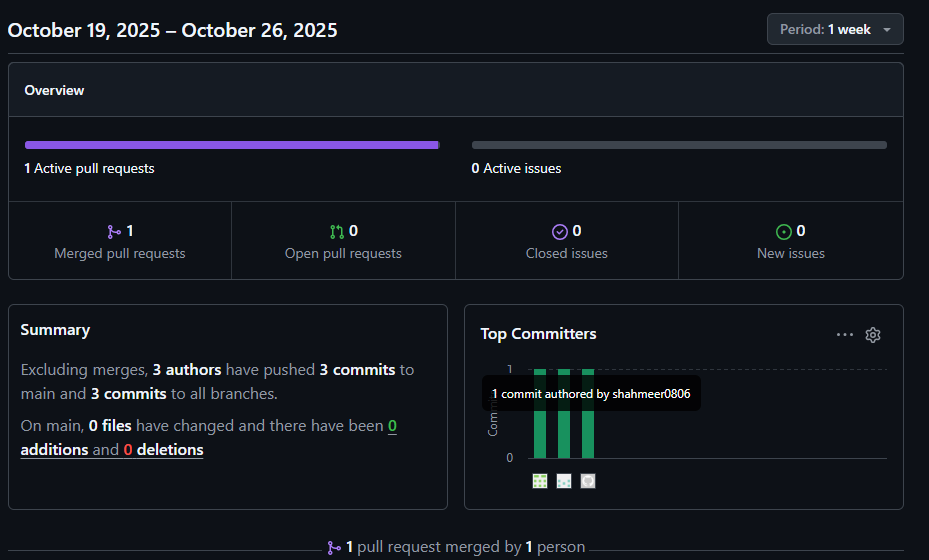




**Team Collaboration Summary:**

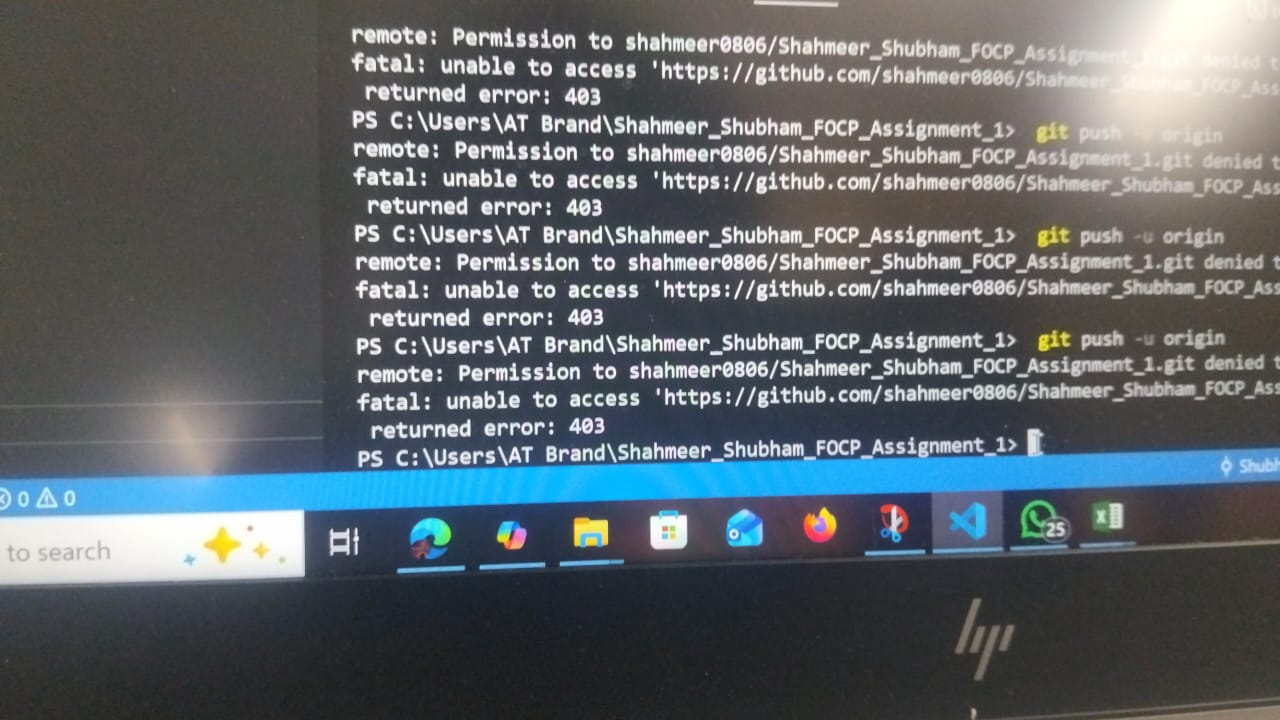
The simplest way to put it is that Muhammad Shahmeer Shaheedi is the Logic Designer and the Code Tester/ Documenter. He started the project by tackling basic logical problems, such as the if-else relationships between the environmental/delivery conditions, and the choices that the software within the device. He had posted the original basic logic code on our original GitHub, but the repo faced issues so Shubham posted it on a new repo for him. The screenshotted evidence is:





Moreover, he is the main code tester and documenter of this project. He ran the code multiple times to see if it ran smoothly and proposed additional improvements. He played the major role in documenting the project and took over the tasks relating to Readme and organized the deliverables.

On the other hand, Shubham Kumar is the main Programmer and Git Manager. He built the major code on top of Shahmeer’s basic logic-based decision determination code, ensuring that the code was well readable and didn’t drop a huge mass of information at once when it was run. He also created the group’s GitHub repository after the initial repository was facing issues with uploads, as can be seen in the images below. We had created a repo a week prior, but faced problems when Shubham was trying to push his file onto it.



After making the new repository, though, Shubham put up his edited version of Shahmeer’s Logic-based code (file named “Logic v1.cpp”). Then, he pushed his main code onto the repository, but there was still room for improvement. This readme sample is based on said main code (titled “tempCodeRunnerFile.cpp”). After completing this document sample, Shahmeer will push the file on GitHub to allow a cross-check with Shubham so the latter could give his insight on the information conveyed. Currently, before the committing of this document, the repository looks like this:

**AI Tool Reflection (if any):**

During the process of refining the code/adding extra features to the code, we did refer to GitHub Copilot for minor aid at times. To be more specific:

* We turned to AI to help us make a small string of code that allows the user to simulate the start of the day by pushing the “Enter” button on his keyboard.
* Additionally, we referred to AI while trying to come up with a definitive equation for the performance score. It must be noted that we didn’t use the AI tool to come up with the code for said equation, we just required a mathematical representation of what the equation was supposed to look like.
* Moreover, we were slightly confused about how we were supposed to go about the 10% system malfunction event, which is why we turned to AI to help clear our concerns/confusion. We got to know that we supposed to merely put in a *rand* function to execute the random system error.
* Along with this, to understand how to encounter the possibility of negative battery percentage, we referred to AI. It must be noted once again that we didn’t copy the code.
* We took help from AI tools to learn how to bring a delay in the running of the program so that the conditions could be simulated over some time and the output is easier to follow.

**Future Improvements:**

Of course the project we’ve made is merely a primitive model, and a lot of improvements can be made in the future to make sure that the device is well optimized and travels with higher battery efficiency. We do have a few proposals on how this could be improved:

* We could have the drone account for extra battery drainage due to issues like battery overheating. The battery drain because of the said issue could be displayed on the program.
* We could also design the drone in a way such that it carries multiple packages at once. Now of course, carrying more weight would cause a greater battery drain during certain trips when the combined weight is greatest. After every trip, the total weight of the drone decreases, causing less weight-induced drain. This would better highlight real world conditions.