ELECTRICAL TEAM TRAINING

TASK 8

Group Task

TASK 8.1 Autobots Restaurant

Preface

- Creating and using a Github Repository is **Mandatory,** start your project by initializing a repo for you and your teammates

  Guidelines on using github:

  - Make sure that your repo is **Private**, and add in your teammates and your supervisor
  - **Do not push directly to the main branch**. For each feature, create a new branch with a descriptive name (e.g., feature/menu-class). Once the feature is fully implemented, either merge it into **main** or, if you need to test it alongside another feature, create a separate integration branch (e.g., integration/menu-restaurant) before merging to **main**.
  - **Include a README,** it should explain what the program does and how to use it, any **UML** made should also be inserted into the README file in GITHUB
  - **Remember to always git commit**
- This task is flexible ,there isn't a single fixed answer to the requirement. You're free to customize and adapt it as you see fit.
- The task is meant to be run with multiple different computers, each teammate roleplaying as a customer and only one teammate roleplaying as a restaurant
- Try to record a video with your teammates showcasing the task working from different computers connected to the same ROS network

About

The Autobots open a futuristic restaurant on Earth, blending Cybertronian technology with human cuisine. Their mission: *serve good food, protect customers from Decepticons, and make sure no one leaves hungry.*

Requirements

You are required to build a program where the user can choose to act as either a **Restaurant** or a **Customer**, with both roles communicating through **ROS Services and actions.**. The implementation must follow **Object-Oriented Programming (OOP) principles** to ensure modularity and reusability. You have to use **Python .** A video showcasing the program should be made

- **Restaurant Role:**
  - Accepts incoming service requests from customers, which represent food orders.
  - Processes the orders and sends back a Ros service **response**

containing the details (price, preparation time, confirmation, etc.).
    - Simulates the order processing by sending out **feedback updates** (e.g., "Order is being prepared", "Out for delivery", "Delivered") to the customer Using ROS actions.
- **Customer Role:**
    - Browses the menu and selects food items.
    - Applies customizations (e.g., size, toppings, extras).
    - Confirms the order and sends it to the restaurant as a service **request**.
    - Receives the service **response** (order confirmation, total cost, estimated time)
    - Also receives from the restaurant **ROS action feedback** about the order if it is confirmed and is being prepared/delivered

The communication flow should feel like a real ordering system, with ROS services handling the **order request/response cycle**, and ROS actions handling the order updates, providing **progress feedback** to the customer.

The interface can be inside the terminal, **With the exception of the menu** being a very simple GUI

Notes

- Maximize the use of OOP principles as possible (Abstraction, Inheritance, Polymorphism and Encapsulation).
- Do not write your code immediately, try to make a UML of the general design of each part of the program first on a piece of paper or on a software like app diagram. And **add**

this UML to the repo's README as it can be used as a reminder during discussion and as documentation

Details

1. Food Items

Objective:

Define abstract classes that inherit from each other so that you can create customizable and modular objects anywhere inside of this task

Details:

- Each food item must include:
  - Name (e.g., *Energon Pizza*)
  - Price (base cost)
  - Description (short detail of the item)
  - Cooking Time (base preparation time in minutes)
  - Size Options (Small, Medium, Large)
- Each item must be grouped into one of the following **subcategories** :
  - Main course(*Autobot Mains)* e.g. Pizzas, pastas, hamburgers and steaks
  - Snacks (*Decepti-Snacks*)e.g. Chips, onion rings and Nachos
  - Drinks (*Energon Drinks*) e.g. Sodas and Milkshakes
  - Desert (*Unicron Treats*) e.g. Chocolates and Biscuits
- Customers can **customize their orders** by choosing size, toppings, or extras.
  - **Extras** (e.g., extra cheese, spicy oil, dipping sauces, extra chocolate) must:

- Increase the **price** of the item.
- Add to the **cooking time** of the item.

- Example:
  "Energon Pizza, Large, with extra cheese and spicy oil"
  → Large size increases base price and time.
  → Each extra adds additional cost and preparation time.

- **Requirements:**
  - It is required to create a modular abstract classes of each subcategory using the above details with the flexibility of customizing the order
    - Meaning that if the customer ordered a small cheese burger with an extra patty, the program should create an object named cheese burger, with its price and preparation being that of a small cheese burger + a patty

2. Customer and Menu System

- **Objective:**
  Build an **interactive Transformers themed menu** with a **GUI** where customers can place their orders. The menu is a **colorful, interactive GUI** which allows the user to **Choose** a food item, **Customize** the food, place an order with the ability to choose between eating inside the restaurant or delivering the food to a location

- **Details:**
  - The menu will contain food items. Food items are objects created from the abstract classes of the food items from the Food items section
  - Each food item in the menu must include **The name, price, Description** of the item
  - The menu must be divided into multiple categories according to the item's category and type, e.g. Main course category which contains all the main courses food categories like the pizzas category which contains multiple types of pizza E.g. Energon pizza, Optimus pizza, Cheese pizza etc
  - Customers can **customize** their orders, meaning they can decide the size, the extras or the toppings (e.g., "Energon Pizza, Large, extra cheese and spicy oil").
  - When the customer finish customizing their order and confirm it,, the option of dining in or delivery should appear, and the Order then should be sent as a ros request
  - The customer should first receive a confirmation response from the restaurant via a ROS service, indicating whether the order is accepted or rejected. Once the order is confirmed, the customer then acts as a ROS action client to track the progress of the order until it is completed.
    - **Bonus:** The customer should have the ability to cancel an order in the middle of preparation or delivery, and the customer should pay cancellation fees. ROS actions are preemptible. meaning they can be cancelled mid way through.

- **Requirements:**
  - It is required to create a menu which is categorized to 4 sub categories (Autobots mains, Decepti-Snacks, Energon Drinks and Unicorn Treats), with the Autobots Mains category having sub categories such as pizzas and hamburgers. It should be noted that each category should have at least 1 or 2 custom made food item (E.g. Energon Pizza, which is a pizza made out of Energon)

3-Restaurant System

- **Objective:**
  Build a robust Restaurant system that is able to take in any food type and process the order, it should be able to send back responses according to the food item specifications, eg how long will it take, and should also send back feedback to the customer of the order status (e.g., "Order is being prepared", "Out for delivery", "Delivered") through the use of ros actions

- **Requirements**
  - After the customer sends his order request through ROS service request, The restaurant class should send a service response to the customer mentioning the **customer's name, the order, and confirmation of the order**
    - Confirmation of the order can be

decided by the teammate roleplaying as the restaurant

- The restaurant class should have a method that process order, the processing goes through 2 stages:
  - First stage is cooking the meal, the customer class should act as a ROS action client and the restaurant should send **continuous** feedback to the customer about the time remaining before the meal is ready, and once it's ready, it should notify the customer about it.
  - Second stage occurs if the customer chose that the meal should be delivered to him, in this case another ROS action should be activated in which the delivery guy sends feedback according to the time remaining before he delivers the food

Individual Tasks

TASK 8.2: Optimus prime vs Megatron

The metal skies of Cybertron are ablaze. Entire districts of Iacon lie in ruin, shattered by decades of war. Smoke rises into the heavens as the remaining Autobots and Decepticons retreat, leaving two towering figures standing alone amidst the wreckage.

Across the battlefield, their optics lock.

- **Optimus Prime**, battered but unyielding, carries the weight of Cybertron's freedom on his shoulders.

- **Megatron**, fueled by Dark Energon and blind ambition, radiates sheer power and malice.

The war for Cybertron has come down to this.

Requirements

- Design a C++ program in a github repo to simulate a fight between Optimus prime and megatron using OOP principles
- The fight alternates between Optimus prime and megatron, with each character taking turns to attack.

- Each weapon has a **chance** to miss
- After each attack, print the following:
  - The character performing the attack.
  - The weapon used, and whether or not it hit
  - The remaining health of the opponent.
- Ending the Fight: The fight ends when either Optimus prime or megatron's health drops to zero or below.
- Print a message declaring the winner and the final state of the fight.
- **You must create a state machine diagram of the project**

Notes

- Maximize the use of OOP principles as possible (Abstraction, Inheritance, Polymorphism and Encapsulation).

Attribute	About
Health	It represents the character's health and it's affected by the

	enemy's weapon according to the weapon damage (some weapons have special functions). Initial: 100 for each character

- **Optimus prime weapons**

Weapon	Damage	Accuracy	Description
Ion rifle	6	100%	Optimus Prime's signature energy rifle, a reliable and powerful ranged weapon known for accuracy and stopping power
Energon swords	12	80%	Twin forearm-mounted energy blades, used for fast, close-quarters combat with precision and speed.
Shoulder canon	45	25%	High-powered plasma launcher mounted on Optimus's shoulder, providing devastating long-range firepower.

- **Megatron weapons**

Weapon	Damage	Accuracy	Description

Fusion Cannon	9	90%	Megatron’s signature arm-mounted weapon, unleashing concentrated energy blasts with devastating force.
Fusion Sword	18	70%	A massive Cybertronian blade used in brutal close combat, capable of cutting through even Autobot armor.
Tank Mode	60	15%	His alternate form transforms into a heavily armored tank, providing unmatched firepower and mobility.

TASK 8.3: Docker image

In their fierce duel, Megatron used a clever decoy to escape the battlefield. Knowing his rival would strike again, Optimus recorded the strategy he used during the fight inside a **Docker image**, ensuring the Autobots could study, replay, and refine their tactics for the battles to come.

Requirements

- Create a dockerfile that creates an image of Task 8.2 and then build that docker file
- Verify that your image works correctly by running a container
- Upload the image to Dockerhub

Submission

- For Group Task 8.1, add in your Supervisor and submit the github

repo link as well as the video link

- For task 8.2, submit a github repo containing the code
- For Task 8.3, Submit the link for the image in dockerhub and for a github repo containing the Dockerfile
- Submission Link: https://forms.gle/vnnYDcRXTKvRE1vW9
- Task Deadline Monday at 25th August at 11:59 PM