Assignment #5: Producer-Consumer

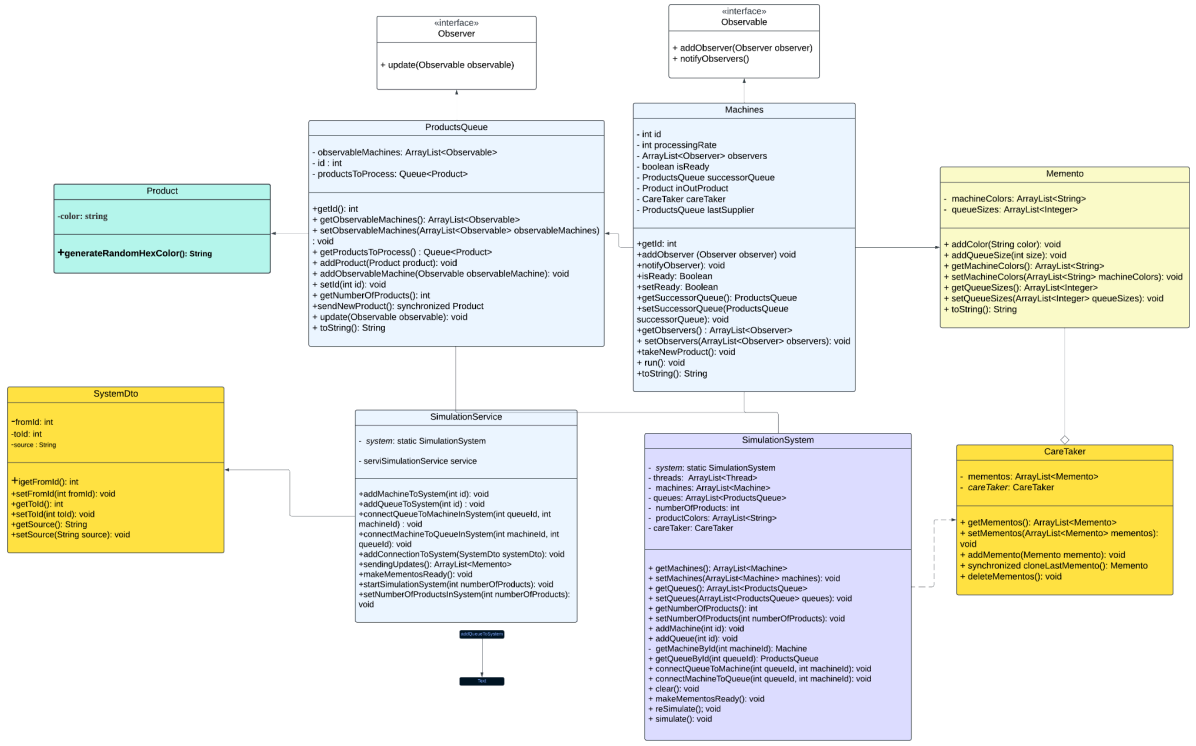
Team Members:

1. **Rowan Gamal Ahmed Elkhouly** - 21010539
2. **Yomna Yasser Sobhy** - 21011566
3. **Yousef Mahmoud Mohamed Ahmed** - 21011630
4. **Mustafa Mohamed Elkaranshawy** - 21011375

Steps of running the code:

- The backend and the frontend files should both be installed with all the included dependencies and packages
- The backend file should run first as we made it run on a default port of 8080
- The frontend file should be run afterward, and the local host is opened in the browser

UML diagrams:



For Higher Quality:
https://lucid.app/lucidchart/506df145-c24f-4126-a278-8a9723a28bb7/edit?view_items=vFi8li.ps7wM&invitationId=inv_7d9b55f5-4f49-45d8-a2ec-ff99d9244f71

Features:

- Simulation program that represents the sequence of product processing
- Adding any number of stages “Queues” and any number of processes “Machines”.

- The ability to restart the last simulation without need to create the system again.
- The ability to clear the current system and execute another one.

Extra Features:

- GUI animations that helps the user to identify products at different stages of the process.

—----------

Applied Design Patterns:

1. Single Threaded design pattern
2. Producer Consumer Design Pattern
3. Singleton Design Pattern
4. Observer Design Pattern
5. Memento / Snapshot Design Pattern

How We Applied Required Design Patterns:

From concurrency patterns:

1. **Single Threaded design pattern:** we have applied it to prevent concurrent access to critical data (such as queues) throughout the program by the use of synchronised keyword
2. **Producer-Consumer design pattern:** we have applied it To coordinate the asynchronous production and consumption of products by machines so that no two machines take the same product by mistake or no queue that supplies a non-existing product to a machine

Other design patterns:

1. **Singleton design pattern:** we applied it to prevent the existence of multiple objects in memory of objects that must only be created once in the system such as the SimulationSystem class and the CareTaker class and also the SimulationService class

2. **Observer design pattern:** we applied it through both the observer interface and its "update" method and the observable interface and its "addObserver, notifyObserver" methods so that when each machine finishes its processing on the product it notifies its observer (which is the queue supplying it with products) and after that the queues supply their observables (machines) with new products through the update method

3. **SnapShot design pattern:** we applied it to save the the state of the system so that we can replay it again, we applied it by the implementation of the CareTaker class and the memento class where the memento represents the

state saved and the CareTaker is the class that saves this memento represents (the originator class is our SimulationSystem class)

Design Decisions:

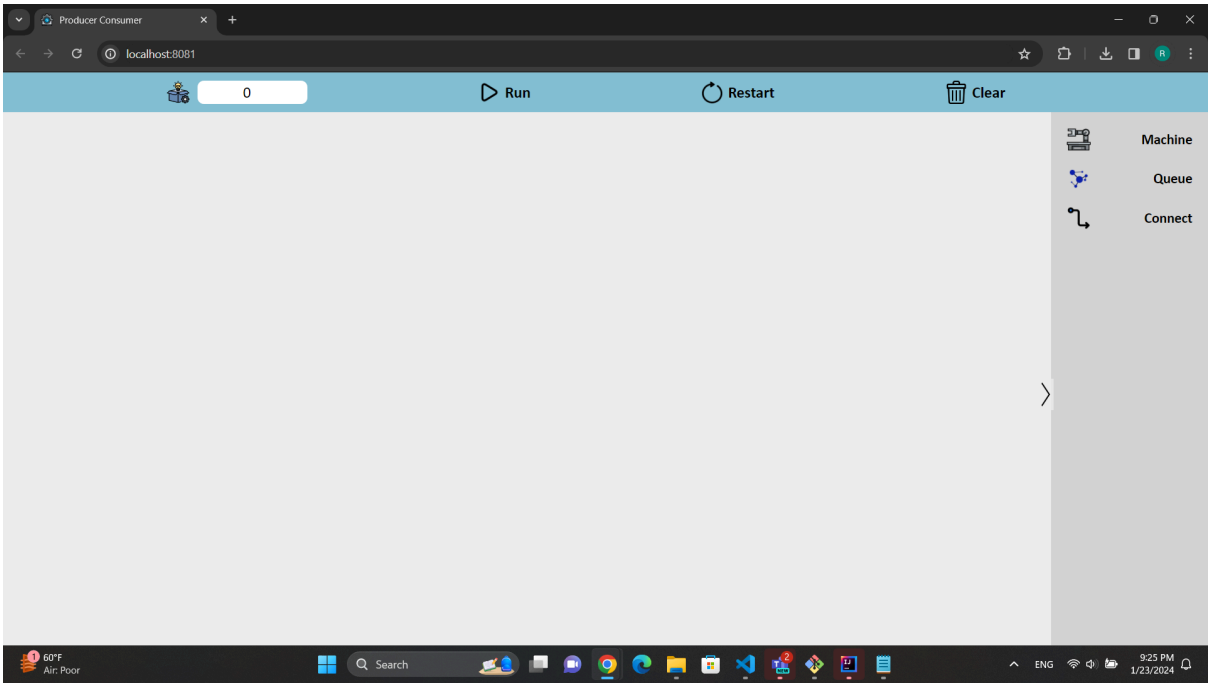
- We divided tools/widgets into **the navigation bar and sidebar**
- The **Navigation Bar** contains the options:
    - Specification of the number of products
    - Run Button
    - Replay Button
    - Clear/Reset Button
- The **Sidebar** is concerned with the diagram itself, and has options:
    - Machine
    - Queue
    - Connection/Link
- All options are given icons for easier identification.
- All buttons and tools are interactive and transitioning when hovering over them for **better interactivity**.
- When running the simulation, the color of the machine changes, and its octa-shape rotates to show it's currently processing the product

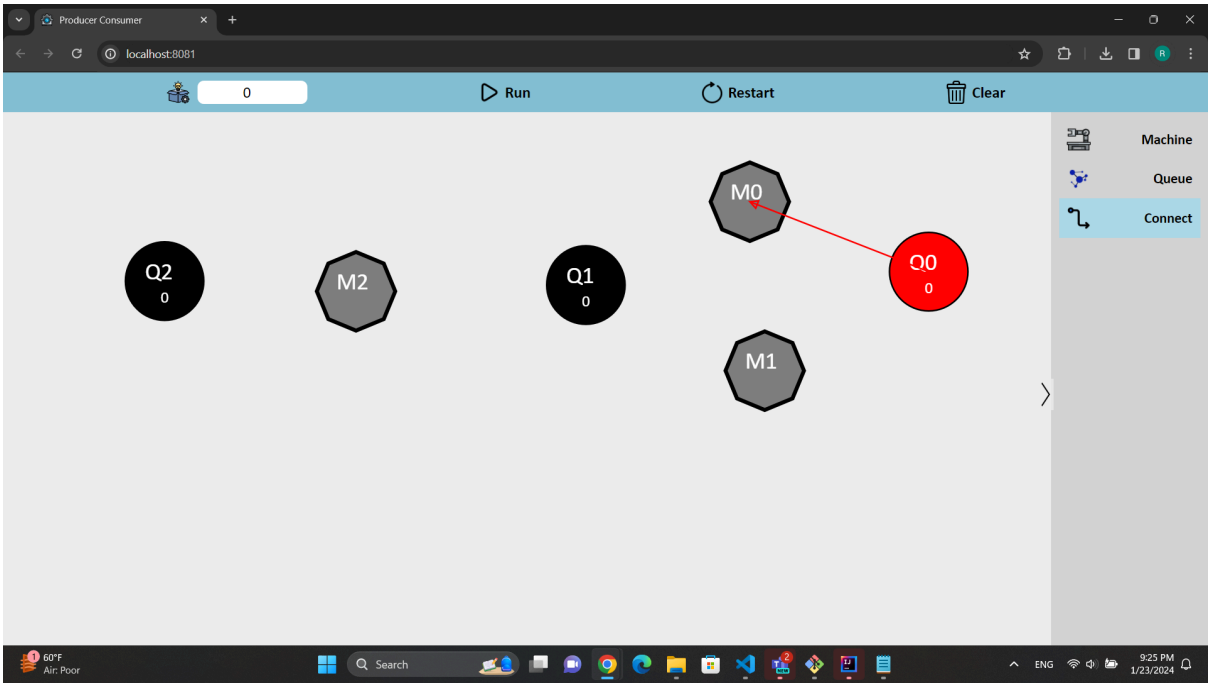
—----------

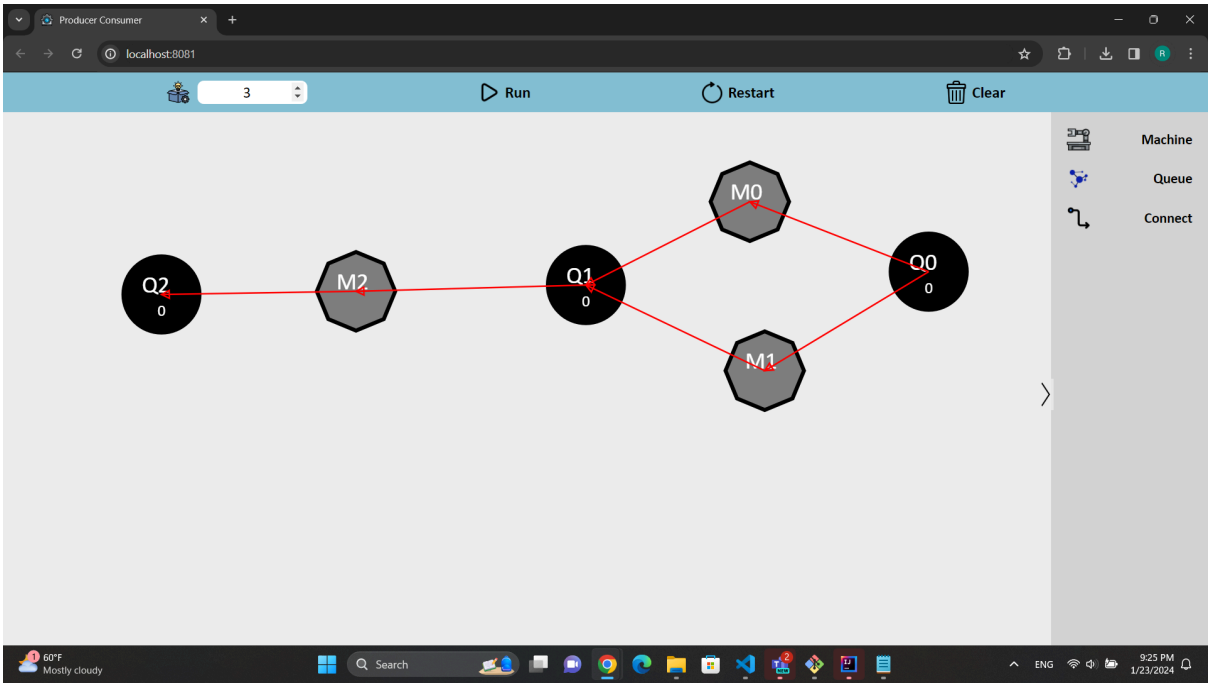
User Guide:

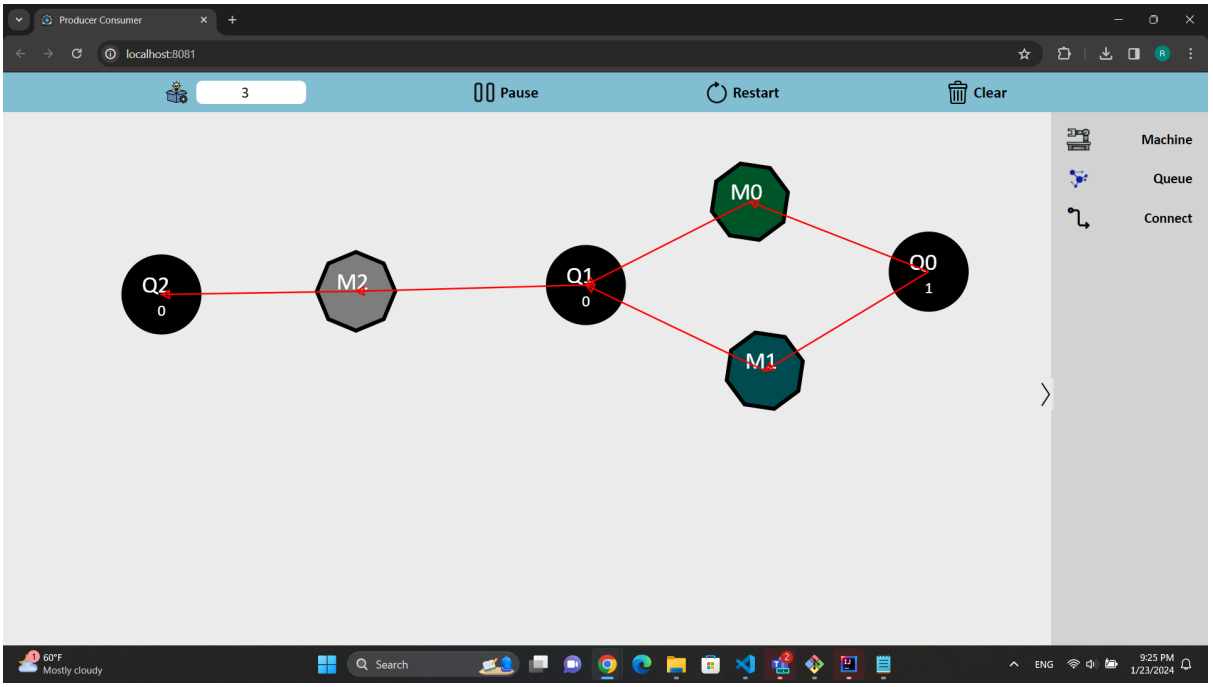
- Upon opening the application the drawing board is fully white and empty.
- To create a machine or queue, click on the item in the sidebar then click at any place on the whiteboard.
- To add a connection click on the connect icon in the sidebar, then choose the first machine/queue as a source, then it will be colored in red, then choose the second queue/machine.
- You can choose the number of products from the toolbar.
- To run the program press the **Run** button then the Queues and Machines will change their state according to the flow of the program.
- You can replay the last operation using the Restart button.
- You can reset the program by pressing the Reset button then you can add and run another operation.

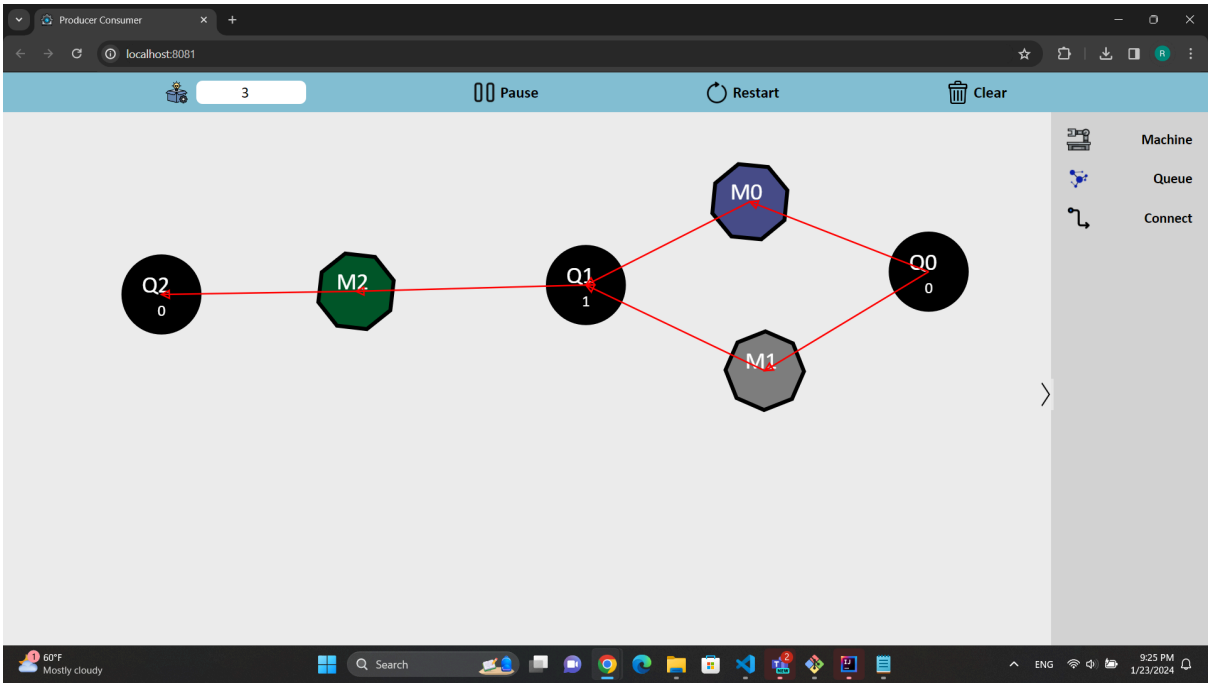
User Interface:

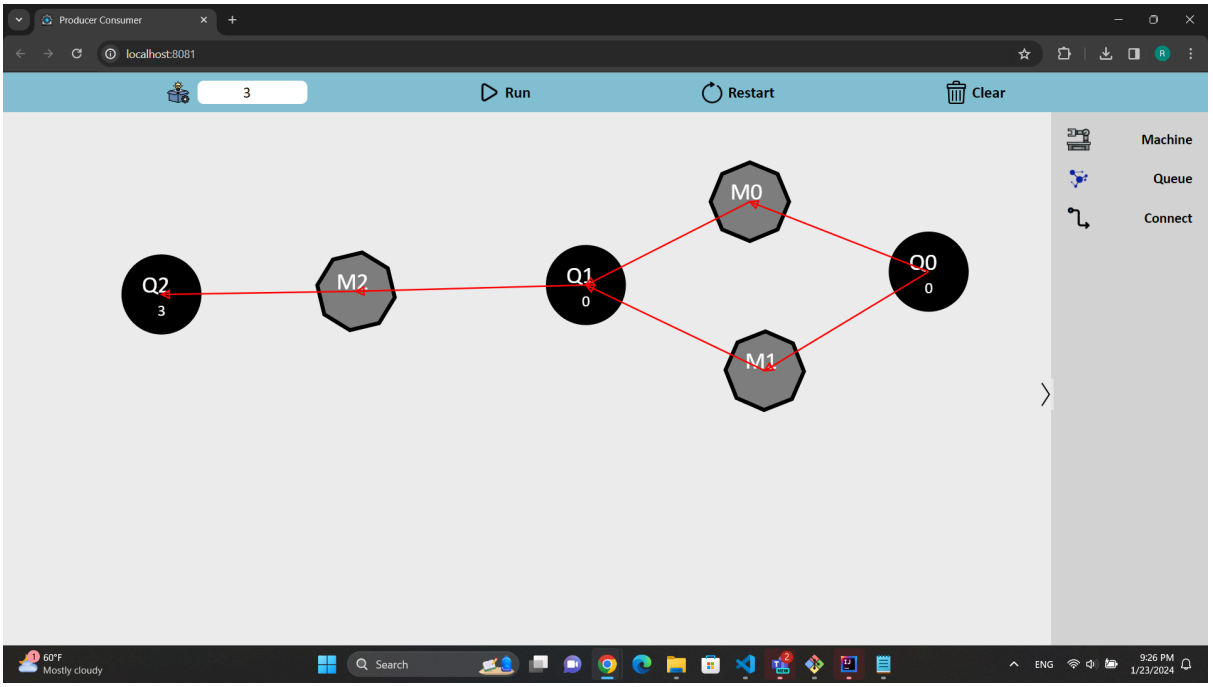

Producer Consumer
localhost:8081
0
Run
Restart
Clear
Machine
Queue
Connect
60°F
Air: Poor
Search
ENG
9:25 PM
1/23/2024


Producer Consumer
localhost:8081
0
Run
Restart
Clear
Machine
Queue
Connect
M0
Q0
0
Q2
0
M2
Q1
0
M1
60°F
Air: Poor
Search
ENG
9:25 PM
1/23/2024


Producer Consumer
localhost:8081
3
Run
Restart
Clear
Machine
Queue
Connect
M0
Q2
0
M2
Q1
0
Q0
0
M1
60°F
Mostly cloudy
Search
ENG
9:25 PM
1/23/2024


Producer Consumer
localhost:8081
3
Pause
Restart
Clear
Machine
Queue
Connect
M0
Q2
0
M2
Q1
0
Q0
1
M1
60°F
Mostly cloudy
Search
ENG
9:25 PM
1/23/2024


Producer Consumer
localhost:8081
3
Pause
Restart
Clear
Machine
Queue
Connect
M0
M1
M2
Q0
0
Q1
1
Q2
0
60°F
Mostly cloudy
Search
ENG
9:25 PM
1/23/2024


Producer Consumer
localhost:8081
3
Run
Restart
Clear
Machine
Queue
Connect
M0
M1
M2
Q0
0
Q1
0
Q2
3
60°F
Mostly cloudy
Search
ENG
9:26 PM
1/23/2024