Producer Consumer

Programming Assignment   5
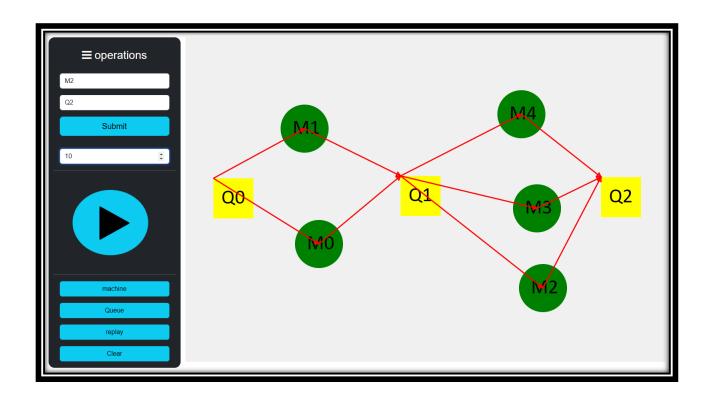
P r e p a r e d   B y

Ahmed Hesham   **Abdelrahman Wael**

Abdelrahman Mohamed   **Mohamed Amin**

Overview

A program to simulate the line of production as a queueing network

<u>**Application functionalities:**</u>

- **Add machine and Queues**
- **Replay the process**
- **Specify number of tasks**
- **Clear the tasks**

Design Patterns

1) Observer:

The machine has an array of observables which are the queues. When the machine is done with a task it notifies the queue in order for it to send the next product if exist.

2) Producer-Consumer:

The production and consumption processes are asynchronous where the queue stores the products temporarily until the machine is ready. When the queue is empty the machine stays in the ready state.

3) Snapshot:

Snapshot for storing the diagram in the caretaker

4) Singelton:

A singleton class for the diagram as we can only run one instance of it. Contains array of machines, queues and threads.
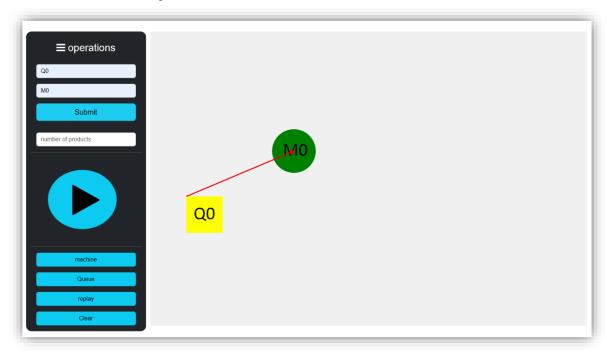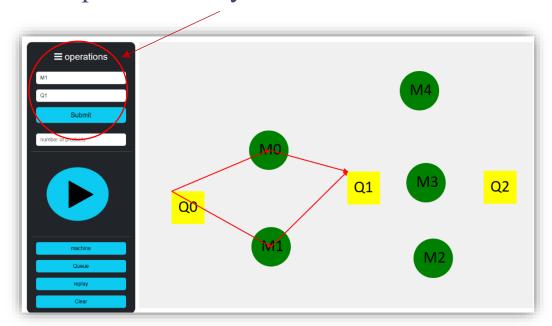
1) Start adding machines and queues by pressing their buttons in the operations menu
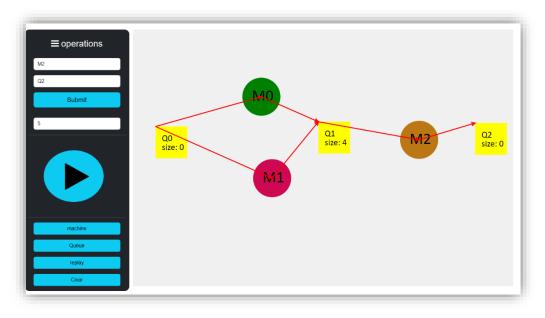
2) After they appear drag and drop them to design the network you want
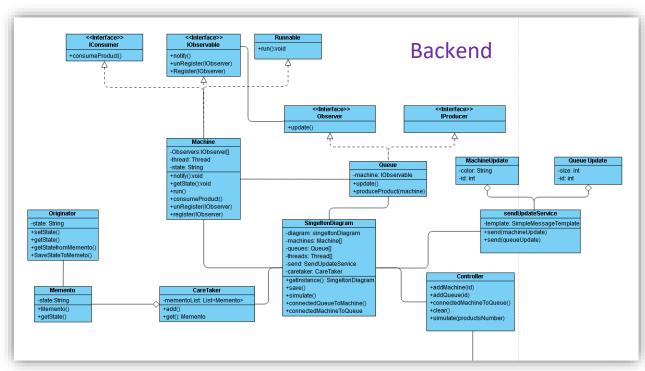
3) Connect them by entering the start and end positions of the arrow. Then enter the number of tasks and press the Play button to start.
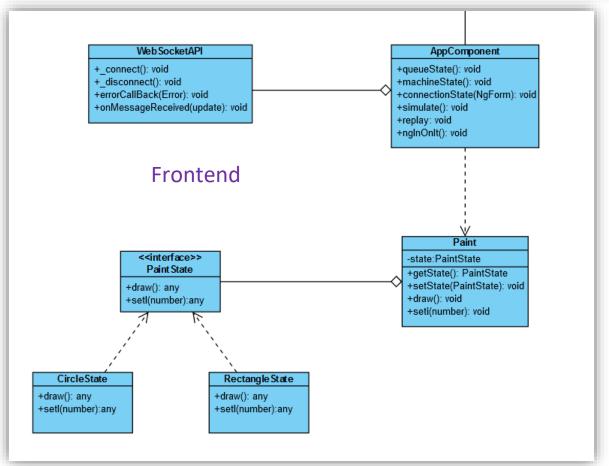
4) Tasks will start to be processed through the network. Machines change colors randomly to indicate the current product and the queues display the number of products they hold

Class Diagram

Backend

<<Interface>> IConsumer
+consumeProduct()

<<Interface>> IObservable
+notify()
+unRegister(IObserver)
+Register(IObserver)

Runnable
+run():void

<<Interface>> Observer
+update()

<<Interface>> IProducer

Machine
-Observers:IObserver[]
-thread: Thread
-state: String
+notify():void
+getState():void
+run()
+consumeProduct()
+unRegister(IObserver)
+register(IObserver)

Queue
-machine: IObservable
+update()
+produceProduct(machine)

MachineUpdate
-color: String
-id: int

Queue Update
-size: int
-id: int

Originator
-state: String
+setState()
+getState()
+getStatefromMemento()
+SaveStateToMemeto()

SingeltonDiagram
-diagram: singeltonDiagram
-machines: Machine[]
-queues: Queue[]
-threads: Thread[]
-send: SendUpdateService
-caretaker: CareTaker
+getInstance(): SingeltonDiagram
+save()
+simulate()
+connectedQueueToMachine()
+connectedMachineToQueue

sendUpdateService
-template: SimpleMessageTemplate
+send(machineUpdate)
+send(queueUpdate)

Controller
+addMachine(id)
+addQueue(id)
+connectedMachineToQueue()
+clear()
+simulate(productsNumber)

Memento
-state:String
+Memento()
+getState()

CareTaker
-mementoList: List<Memento>
+add()
+get(): Memento

Frontend

Web SocketAPI
+_connect(): void
+_disconnect(): void
+errorCallBack(Error): void
+onMessageReceived(update): void

AppComponent
+queueState(): void
+machineState(): void
+connectionState(NgForm): void
+simulate(): void
+replay: void
+ngInOnIt(): void

<<interface>> Paint State
+draw(): any
+setl(number):any

Paint
-state:PaintState
+getState(): PaintState
+setState(PaintState): void
+draw(): void
+seti(number): void

CircleState
+draw(): any
+setl(number):any

Rectangle State
+draw(): any
+setl(number):any

Video Link

https://drive.google.com/open?id=16H8eGbOfhrdNV-0e42JsTZs_FM4PPj1I&authuser=ahmehesham1652001%40gmail.com&usp=drive_fs