**Producer/Consumer Simulation Program**

**Names and ids:**

Name1: البراء مصطفى محمد

ID1: 22010636

Name2: ايمن ابراهيم محمد قطب

ID2: 22010656

Name3: عبد الرحمن السيد صديق

ID3: 22010870

Name4: اسلام هاني حسين عبدالمنعم

ID4: 19015425

**How to run:**

*Backend Setup:*

1. install [JDK Builds from Oracle](https://jdk.java.net/) and an IDE to get running with java.
2. Install [Spring Initializr](https://start.spring.io/) and choose (spring web/ WebSocket from dependencies)
3. Navigate to the backend folder.
4. Import the project into the IDE and run the spring app or run with Maven on terminal (mvn spring-boot: run)
5. Ensure the server is running on <http://localhost:8080>.

*Frontend Setup*

1. Install [Node.js — Run JavaScript Everywhere](https://nodejs.org/en)
2. Navigate to the frontend folder.
3. Install dependencies: npm install
4. Install dependencies: npm install react-use-websocket
5. Start the frontend server: npm run dev

*Get started!*

1. Visit the URL that shows up in your terminal after running **npm run dev** to view the paint app in your browser (it should be something like <http://localhost:XXXX/>).
2. Now the simulation app should be running on your browser.

**Applied Design Patterns:**

1. **Controller-Service-Repository Pattern (Layered Architecture)**:

* The application follows the layered architecture where controllers (REST controllers) handle HTTP requests, services encapsulate business logic, and repositories interact with the database.

1. **DTO (Data Transfer Object) Pattern**:

* Adding the required DTO to map the data taken from frontend to the classes

1. **Dependency Injection**:

* Classes such as services and repositories are injected into controllers using @Autowired, promoting loose coupling.

1. **Singleton Pattern:**

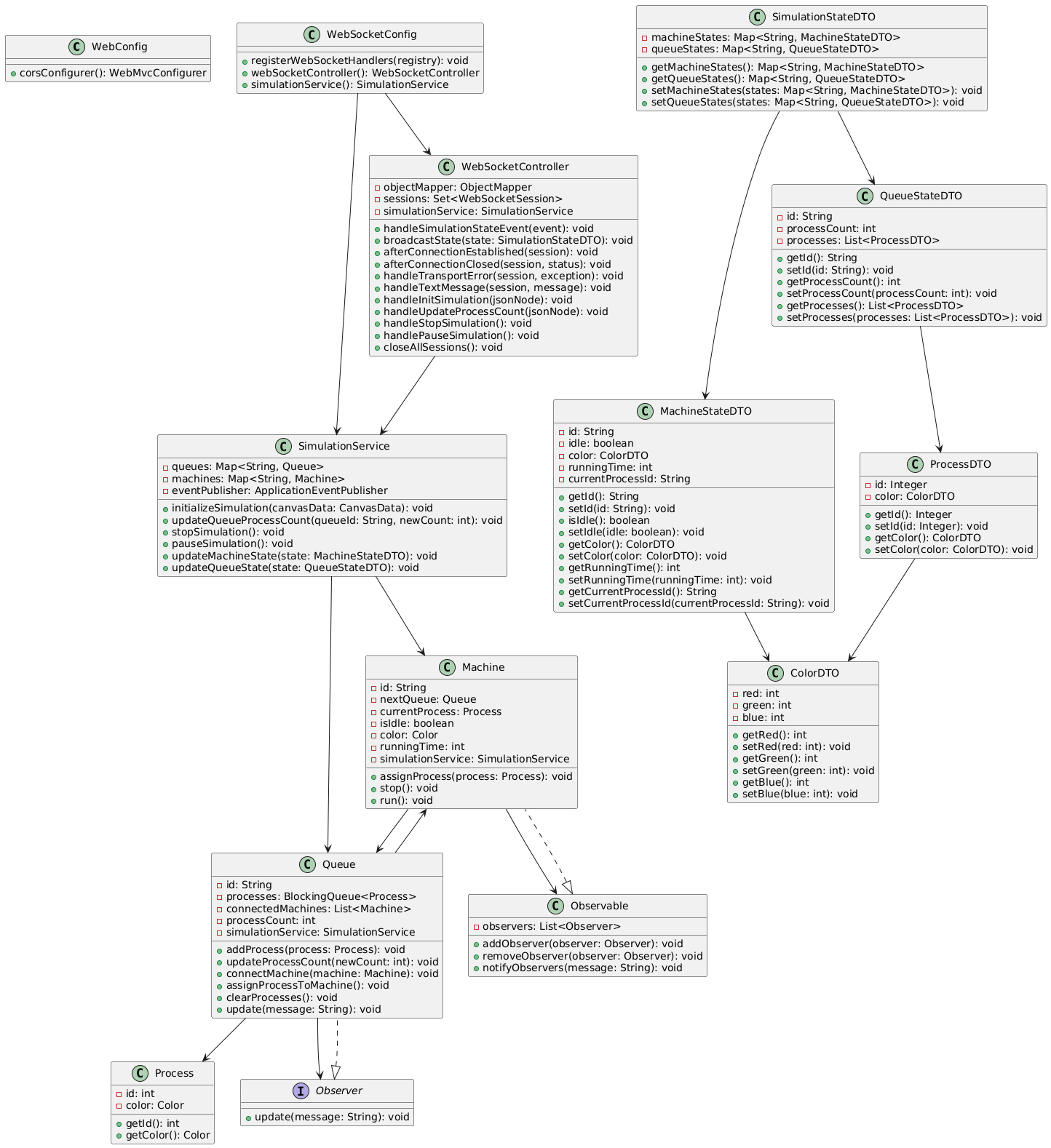
* Spring by default uses Singleton scope for beans.

1. **Concurrency Pattern:**

* to enable multiple threads for machines

1. **Observer Pattern:**

* To keep track of the machine flow (observable) to inform the connected queue (observer) that it’s ready for processing a product.

**UML**

**design decisions:**

1. **Use of WebSocket for Real-Time Communication**

* **Decision**: WebSocket was chosen for real-time, bidirectional communication between the frontend and backend.
* **Reason**: WebSocket allows the backend to push updates (e.g., machine/queue states) to the frontend without the need for polling, ensuring low latency and efficient communication.

1. **Observer Pattern for Queue-Machine Interaction**

* **Decision:** The Observer pattern was used to manage interactions between Queue and Machine.
* **Reason:** This pattern decouples the Queue and Machine classes, allowing queues to notify machines when processes are available and machines to notify queues when they become idle. This promotes flexibility and scalability.

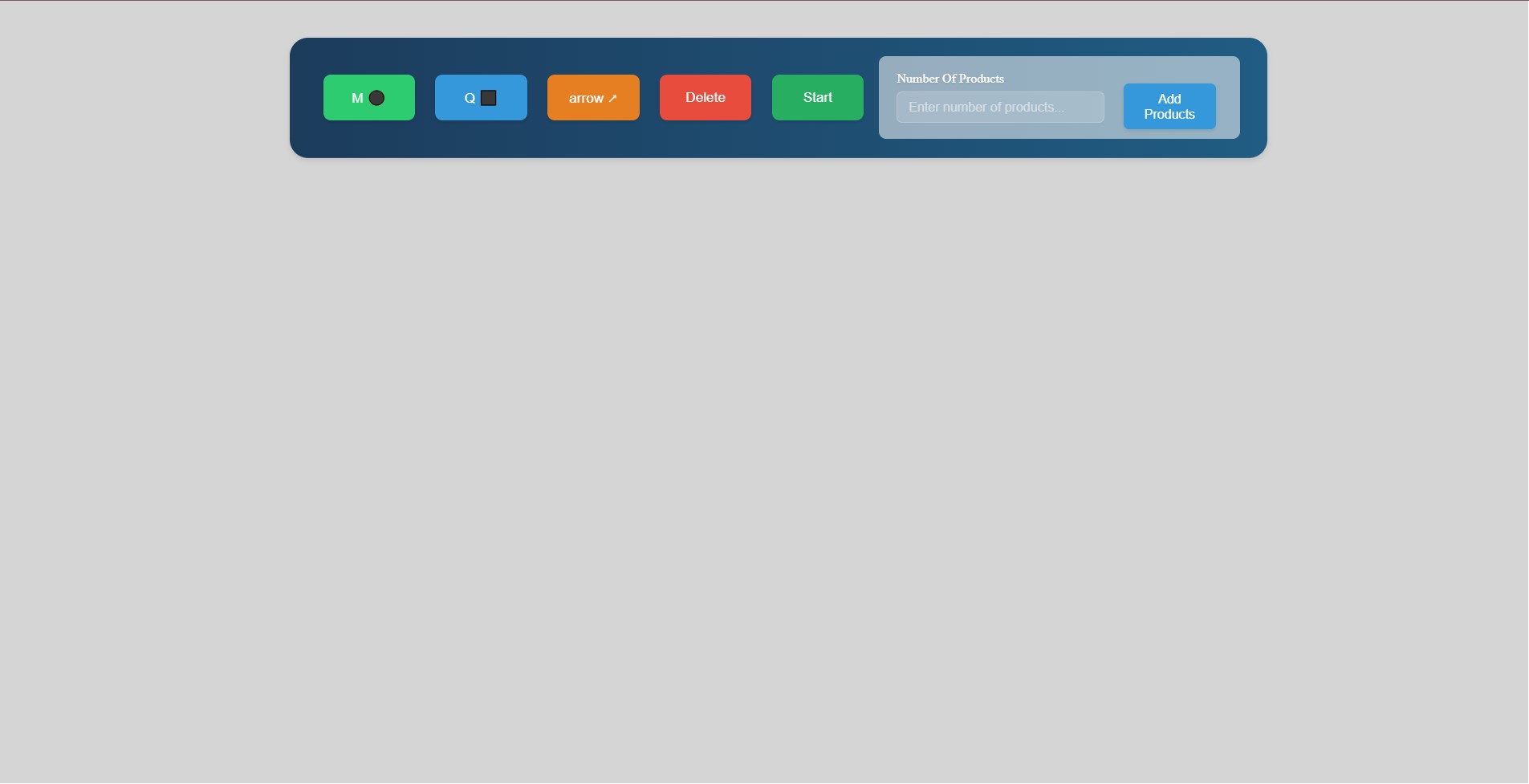
1. **Event-Driven Architecture for State Updates**

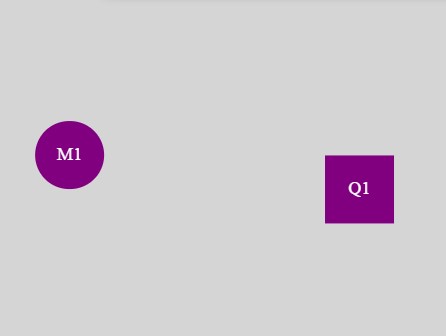
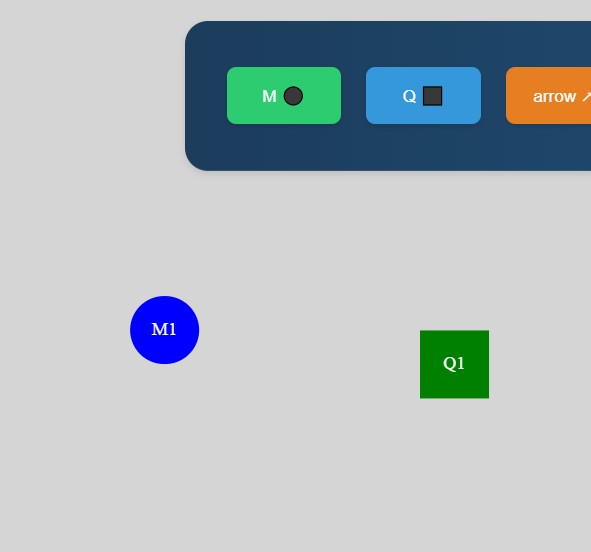
* **Decision:** An event-driven architecture was implemented using ApplicationEventPublisher to broadcast state updates.
* **Reason:** This approach ensures that state changes (e.g., machine or queue updates) are propagated to all connected clients in real-time, without tightly coupling the SimulationService and WebSocketController.

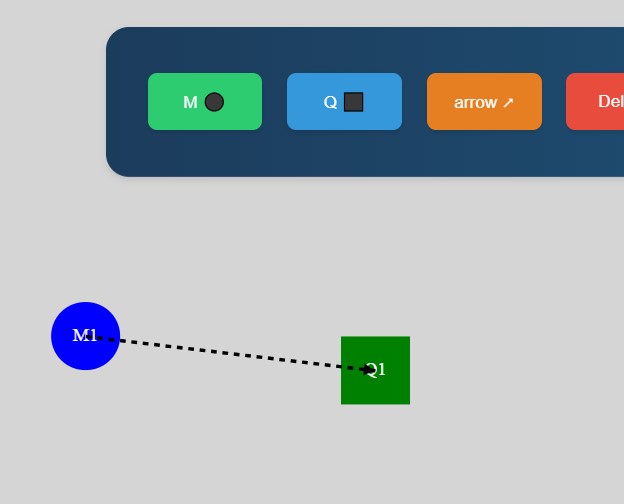
1. **Separation of Concerns in Simulation Logic**

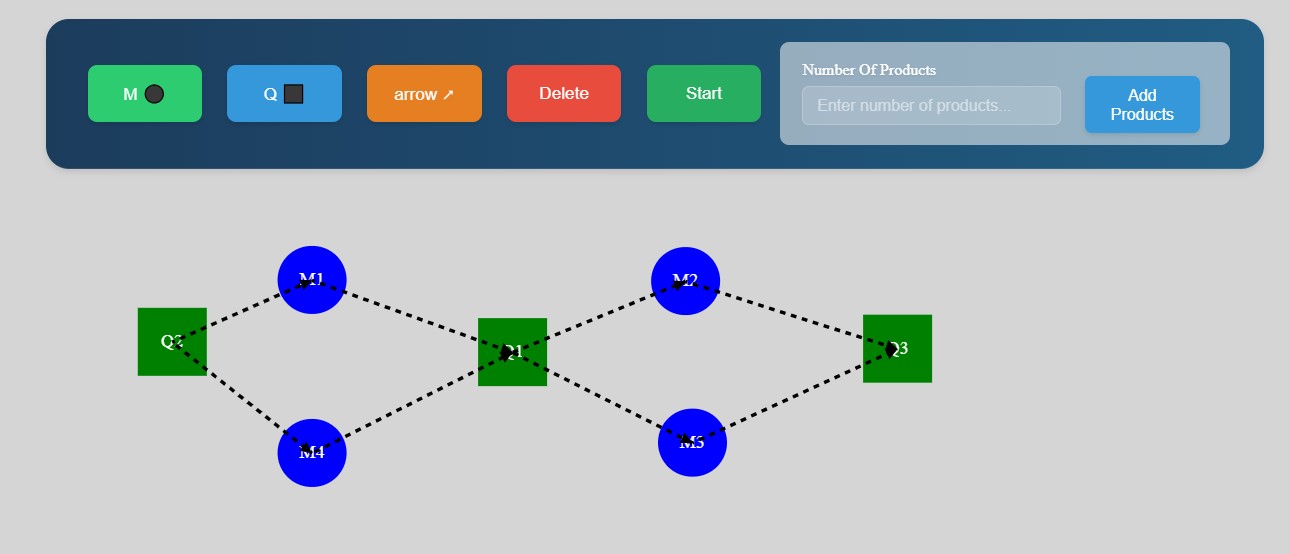
* **Decision**: The SimulationService handles all simulation logic, while the WebSocketController manages WebSocket communication.
* **Reason**: Improves maintainability by keeping the simulation logic independent of the communication layer.

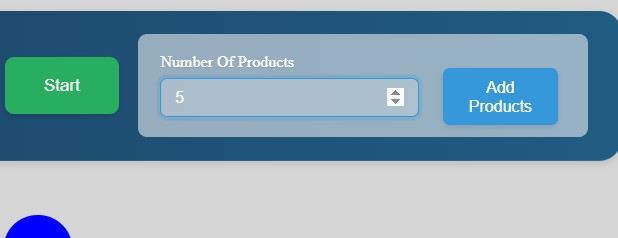
**user guide:**

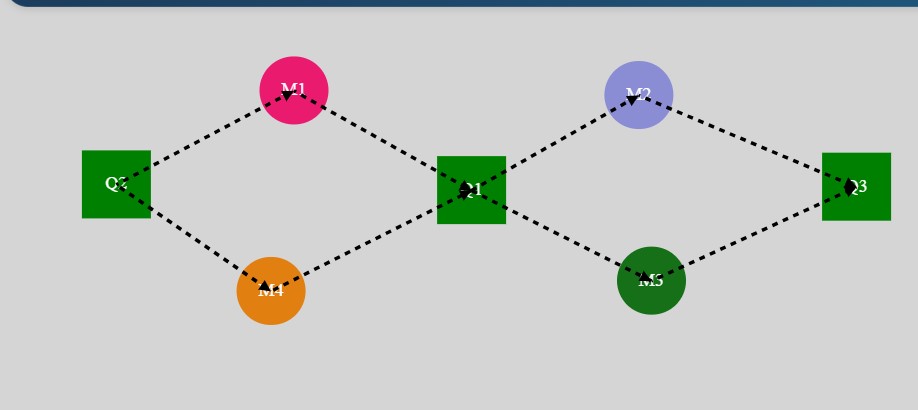
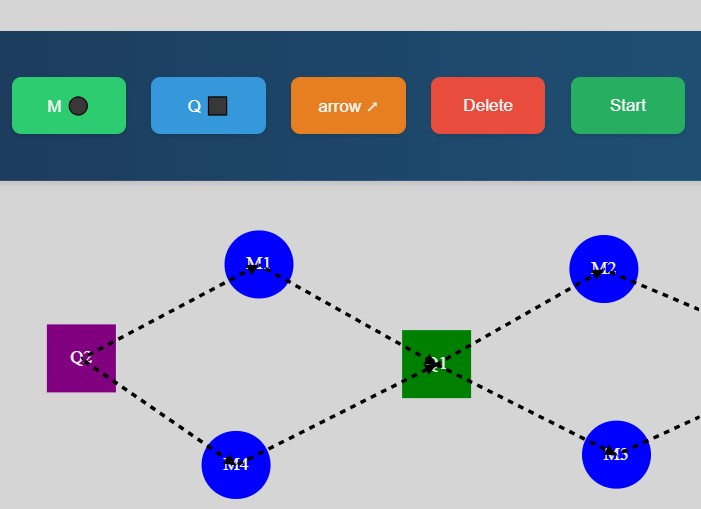
**UI:**

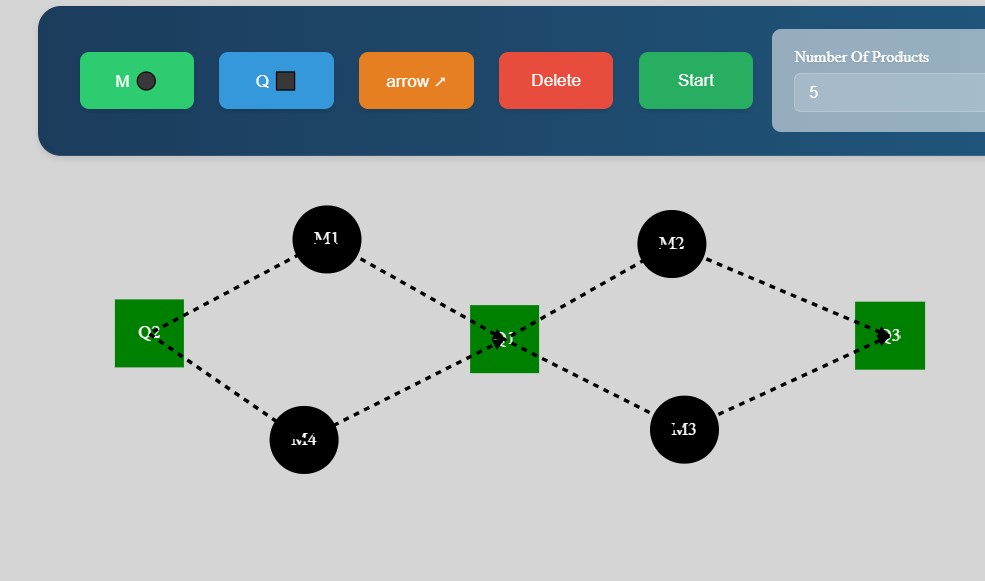
**Add machines/queues: select start and end of an arrow:**

**click on the arrow to connect them:**

**General UI:**

**Determine n of products:**

**Select the queue to start from then click start:  
during processing:**

**All done:**