1 INTRODUCTION

This paper describes a program that simulates a production line in automotive industry. This system manages different stages in production process by separating various events class by class. This simulation code is written in java language. As a result, this simulation simulates the processes and disruptions in the production line and calculates delays. In this way, factory managers are aware of the problem first and can start optimization processes early.

2 METODOLOGY

2.1 Event Management

In this section, we are performing a simulation that allows us to examine automotive assembly processes. First of all, we need to check whether the machine is usable. It must be calculated the maintenance period in case of machine failure. Another thing that is important here is labor time. It should be calculated during labor time. In some cases, the workers shift may not start. In such cases, the time elapsed until the employee's shift starts is included in the calculation. The assembly continues to work while all these operations are taking place.

2.2 Machine Failures and Repairs

- **MachineBreakdownEvent:** This term is a phrase used to represent the time when the machine gives an error or breaks down. In such cases, the condition of the machine is updated and a repair plan related to this process is created.
- **MachineRepairEvent:** This expression explains that the existing problem of a machine has been solved and the machine has been made reusable.
- A solution is implemented that allows the necessary backup of the machine to be taken and operated.

Worker.java | ProcessingEvent.java | MultiProduct.java | MachineBreakdownEvent.java

```java
package com.automotive.simulation;

public class MachineBreakdownEvent extends Event {
    private Machine machine;

    public MachineBreakdownEvent(double time, Machine machine) {
        super(time);
        this.machine = machine;
    }

    @Override
    public void process() {
        if (Simulation.getInstance().shouldContinueSimulation(getTime())) {// check if
            machine.setBroken(true);
            Simulation.getInstance().scheduleEvent(new MachineRepairEvent(getTime() + m
        }
    }
}
```

MachineRepairEvent.java

```java
package com.automotive.simulation;

public class MachineRepairEvent extends Event {
    private Machine machine;

    public MachineRepairEvent(double time, Machine machine) {
        super(time);
        this.machine = machine;
    }

    @Override
    public void process() {
        if (Simulation.getInstance().shouldContinueSimulation(getTime())) {// check if
            machine.setBroken(false);
        }
    }
}
```

Figure 2.2 BreakDown and Repair Event State

2.3 Production Processes

- **ProcessingEvent:** It is an event in which the production of products takes place. In this case, the working time of the worker and the condition of the machine are checked. After all these are examined, production starts.
- **PackagingEvent:** It is an event related to the packaging process of products whose production has ended. As in the previous incident, the worker's working time is calculated and the packaging process is started after checking the condition of the machine.
- **QualityControlEvent:** It is a process in which the quality of the products produced is controlled. It checks the machine status and worker's working hours, then starts the quality control process.
- **RawMaterialArrivalEvent:** It represents the arrival of the raw material and the process of processing it for production. It is an event that follows a process related to the processing of the raw material after its arrival has been calculated.

2.4 Worker and Shift Management

In this section, the information of the workers and the shift processes in which the workers work are examined. The working hours of workers affect and determine the timing of other events.

- **Shift:** It contains information about the working shifts of the workers. The shift in which the workers work determines the time and it is the situation that allows other events to take place in these time zones.

```
public void process() {
    Simulation simulation = Simulation.getInstance();// get simulation instance

    if (!worker.getShift().isWithinShift(getTime())) {
        // if worker is not within shift, wait until start of shift
        double nextEventTime = worker.getShift().getStartTime();
        if (simulation.shouldContinueSimulation(nextEventTime)) {
            simulation.scheduleEvent(new ProcessingEvent(nextEventTime, product, machine, worker));// schedule processing event
        }
        return;
    }
```

Figure 2.4 Workers' Shift

2.5 Simulation Management

- **Simulation:** Provides general management of the simulation. It includes functions such as planning and processing of events, tracking machine and worker status. It controls whether the simulation will continue within a certain time interval.

Performance Analysis:

Product	Raw Material Count	Processing Count	Assembly Count	Q. Control Count	Packaging Count
Product A	969	11	6	0	396
Product B	977	5	2	0	730
Product C	968	7	4	0	632

Identifying Bottlenecks:
Bottleneck Stage: Raw Material
Max Queue Size: 977

Optimizing Resource Allocation:

Stage	Current Machines	Current Workers	Optimal Machines	Optimal Workers	Difference
Processing	3	6	4	7	1 machines, 1 workers
Assembly	2	4	3	5	1 machines, 1 workers
Quality Control	2	3	3	4	1 machines, 1 workers
Packaging	3	5	3	5	0 machines, 0 workers

Figure 2.5 Results

2.6 Product Number

Here the simulation is basically calculated separately based on the number of 2 types of products. If there is only one product, it is tested for single product, if there is more than one product, it is tested for multiple products.

Worker.java | MachineBreakdownEvent.java | ProcessingEvent.java | MultiProduct.java

```java
package com.automotive.simulation.test;

import com.automotive.simulation.FinishedProduct;

public class MultiProduct {
    public static void main(String[] args) {
        // create a simulation
        Simulation simulation = Simulation.getInstance(3, 6, 2, 4, 2, 3, 3, 5); // mach

        // create a list of finished products
        List<FinishedProduct> products = Arrays.asList(
                new FinishedProduct("Product A"),
                new FinishedProduct("Product B"),
                new FinishedProduct("Product C")
        );

        // create raw materials initially and schedule raw material arrival events at r
        double initialArrivalInterval = 5.0; // per 5 units of time, add new raw materi
        for (FinishedProduct product : products) {
            for (int i = 0; i < 1000; i++) { // add 1000 raw materials initially
                simulation.addRawMaterial(product.getName());
            }
            simulation.scheduleEvent(new RawMaterialArrivalEvent(0, product, initialArr
        }

        // start  simulation
        simulation.run();
    }
```

SingleProduct.java

```java
package com.automotive.simulation.test;

import com.automotive.simulation.FinishedProduct;

public class SingleProduct {
    public static void main(String[] args) {
        // create a simulation
        Simulation simulation = Simulation.getInstance(3, 6, 2, 4, 2, 3, 3, 5);
 // machine and worker numbers

        // create raw materials initially and schedule raw material arrival events at r
        double initialArrivalInterval = 5.0; // per 5 units of time, add new raw materi
        FinishedProduct product = new FinishedProduct("Single Product");
        for (int i = 0; i < 10; i++) { // add 10 raw materials initially
            simulation.addRawMaterial(product.getName());
        }
        simulation.scheduleEvent(new RawMaterialArrivalEvent(0, product, initialArrival

        // start the simulation
        simulation.run();
    }
}
```

Figure 2.6 Single-Multiple Product

CONCLUSION

As a result, in this simulation, different numbers of people, different working hours, different numbers of machines and different efficiency ratios were analyzed. Improvements can be made in certain areas according to the results of the simulation. This way, resource utilization is made more efficient.

Barış Çetin 20161701047
Murat Berkay Akosman 20161701028