Discrete Event Simulation for Facility Layout Design of Precise Molecular Diagnostic Centers

Hong-Mo Yang[1], Dong-Hyung Lee[2], Moonsoo Shin[3]

[1,2]Department of Smart Production and Management Engineering, Hanbat National University, Daejeon, Republic of Korea, [1]hongmo0217@naver.com
[3]Department of Industrial and Management Engineering, Hanbat National University, Daejeon, Republic of Korea

As the prevalence of high-risk infectious diseases increases, interest in automated disease diagnostic facilities has been growing. However, while conventional large diagnostic centers employ expensive fully automated diagnostic equipment, small and medium-sized diagnostic centers rely on manual diagnosis or individually separated small-scale automatic diagnosis for each diagnostic process due to limited operating space. To enhance productivity of the small and medium-sized centers, a precision molecular diagnostic system that integrates small-scale automated equipment for each diagnostic process has been developed. Nonetheless, determining the optimal number of equipment and required manpower for each diagnostic process remains a significant problem that needs to be addressed. In this study, to tackle this issue, we employ a discrete event simulation technique to simulate and analyze deployment of diagnostic equipment in a molecular diagnostic center. In particular, we explore various types of facility layouts to propose an optimal deployment strategy for diagnostic equipment in a precision molecular diagnostic facility, considering the size and volume of inspections performed in the diagnostic system.

Keywords: Diagnostic facility, Molecular diagnostic automation system, Point-of-care molecular diagnostics, Facility layout analysis.

1. Introduction

Recently, as COVID-19 broke out and spread globally, creating a pandemic, many efforts have been made to optimize the efficiency of diagnosis and treatment. In particular, these high-risk infectious diseases pose a serious threat to public health due to their spread, fatality rate, and difficulty in treatment. High-risk infectious diseases are new or recurrent diseases, and there are many cases for which there is no or insufficient appropriate vaccine or treatment. High-risk infectious diseases have characteristics such as high transmissibility, high fatality rate, difficulty in treatment and prevention, negative social/economic impact, and global threat.

There is currently a growing interest in rapid and highly accurate automated diagnostic facilities due to the recent spread of the high-risk infectious diseases. On-site immunodiagnostic methods are primarily utilized for screening purposes; however, for more precise diagnostics, molecular diagnostics, which is a genetic amplification technology, is necessary. Nevertheless, fully automated molecular diagnostic equipment is mainly found in huge-sized diagnostic centers due to its substantial size and cost.

In the case of small to medium-sized diagnostic centers, limitations on a budget and operational space bring about manual handling of processes which range from preparing test samples to pre-processing genetic extraction. This results in complex tasks, a need for a significant number of specialized personnel, and increased risk of infection among healthcare workers (Koo et al., 2018).

To address these challenges, precision molecular diagnostic automation systems have emerged (Koo et al., 2018), which integrate automated equipment for each stage of the molecular diagnostic process, including genetic extraction, reagent dispensing, genetic amplification, and data analysis. However, determining the number of required equipment units and personnel for each diagnostic process with consideration on the size and type of the diagnostic center remains a challenging issue.

In this study, we make a simulation model of a molecular diagnostic process for various types of diagnostic centers from large sized ones to small sized ones. In particular, we employ a Python-based discrete event simulation framework, which is named as SimPy. Based on extensive analysis, we select the best processing model and conduct simulations with respect to various facility layouts. This allows us to propose the optimal facility layout of an integrated automation system for precision molecular diagnostics, considering size and test volume of the diagnostic center.

2. LITERATURE REVIEW

2.1. Integrated Automation System for Precise Molecular Diagnostics

The integrated automation system for precision molecular diagnostics is designed for the purpose of determining disease infection status. It makes rapid and accurate extraction, amplification, and testing of genetic material from infectious agents or causative substances, such as bacteria or viruses, found in blood, urine, saliva, and other bodily fluids. On the other hand, traditionally, the molecular diagnostic process of 'extraction - dispensing - genetic amplification (PCR) - analysis' was carried out manually as depicted in Figure 1. The integration of this process into a compact automated system represents a significant advancement.

The automation system results in enhanced service quality from the perspective of reduced retesting rates, improved productivity, and risk reduction of inter-infection among healthcare professionals, due to the standardized test process. However, the challenge lies in determining the quantity of required equipment and man-power for each specific molecular diagnostic process, with respect to the size and type of diagnostic center.

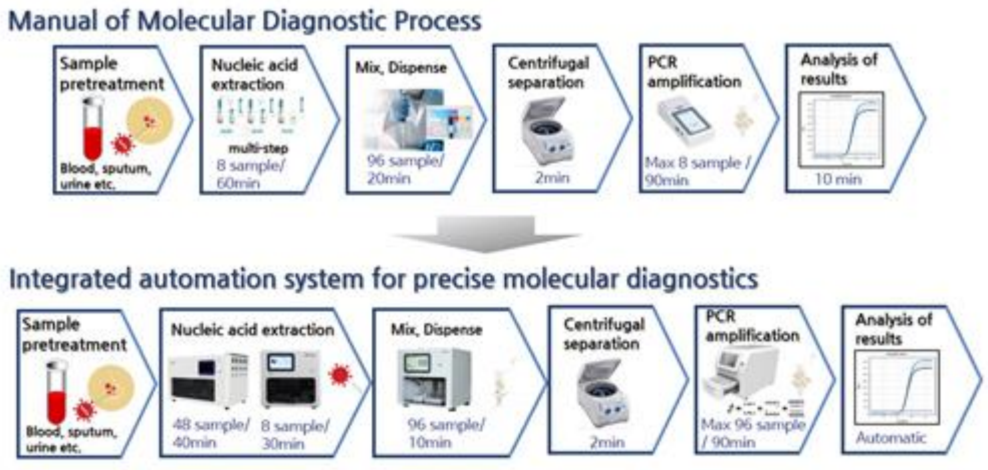


Fig. 1: Integrated automation system for precise molecular diagnostics

2.2. Discrete Event Simulation Module

Discrete event simulation (DES) is a development approach widely used for decision optimization in production systems by employing reinforcement learning (George, 2013). Typically, DES is utilized in situations where high complexity and limited resources are involved, offering the advantage of swiftly assessing system behavior (Caro & Moller, 2016). Depending on distribution policy, it falls into the categories of commercial off-the-shelf (COTS) and open-source software. Using DES, various scenarios related to production system operation can be simulated to make optimal decisions, ultimately leading to increased productivity and reduced risks. Table 1 provides an example of reinforcement learning-based DES simulation research.

Table 1: Previous research papers on using a DES tool for reinforcement learning

DES tool	Author (year)	Target problem (interface)	Algorithm	Category
AnyLogic	Thomas et al. (2018)	Scheduling in Manufacturing system(RL4J)	Deep Q-learning (DQN)	COTS
	Jang et al. (2018)	Traffic signal control (RL4J)	Deep Q-learning (DQN)	COTS
	Farhan, Göhre, and Junprung (2020)	Coffee shop operation(Pathmind)	Proximal policy optimization (PPO)	COTS
	Pinciroli et al. (2020)	Energy system operation(Pathmind)	Proximal policy optimization (PPO)	COTS

Simio	Greasley (2020)	Operation in Manufacturing system(RL function)	Q-learning	COTS
Plant Simulation	Rabe et al. (2017)	Logistic operation(MySQL)	Deep Q-learning (DQN)	COTS
	Shiue, Lee, and Su (2018)	Scheduling in Manufacturing system©	Deep Q-learning (DQN)	COTS
	Mayer, Classen, and Endisch(2021)	Operation in Manufacturing system(TCP/IP)	Proximal policy optimization (PPO)	COTS
Flexsim	Pires et al. (2021)	Operation in Manufacturing system(SQL)	Deep Q-learning (DQN)	COTS
	Preston (2017)	Material handling plan(Excel)	No algorithm (study on RL interface)	COTS
SimPy	Stricker et al. (2018)	Scheduling in Manufacturing system (Python)	Deep Q-learning (DQN)	Open Source
	Menda et al. (2018)	Bus and aircraft control(Python)	Trust region policy optimization (TRPO)	Open Source
	Woo et al. (2021)	Scheduling in Manufacturing system(Python)	Deep Q-learning (DQN)	Open Source

COTS DES tools provide various modeling features, visualization and analysis functions. In particular, AnyLogic, Simio, FlexSim, and ExtendSim offer optimized environments for reinforcement learning development. However, COTS DES tools are somewhat expensive, and thus it is challenging for individuals or small companies to use. Additionally, they typically lack open-source code and require annual subscription fees for improvements and development, which can pose sustainability issues. In contrast, open-source DES tools like SimPy offer cost-effectiveness and the advantage of continuous improvement driven by various user community, enhancing reliability despite slightly reduced convenience.

In this study, we utilize the open-source Python-based SimPy for modeling the batch simulation of molecular diagnostic equipment processes. SimPy, as shown in Figure 2, is capable of performing discrete event simulations and offers a wide range of functions. It can work in conjunction with data processing packages such as Pandas and Numpy, enabling various data simulations (Agostino et al., 2012).

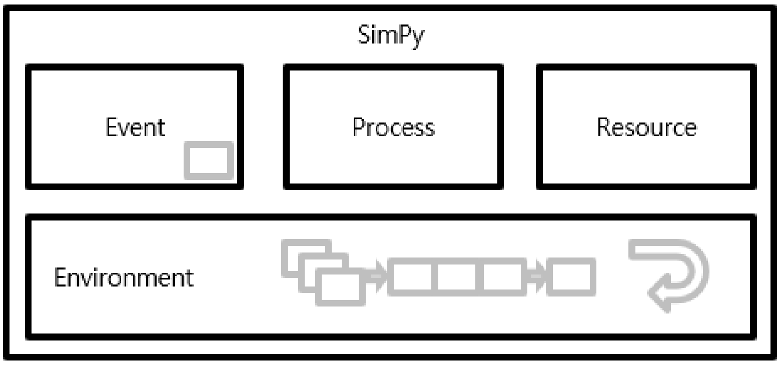


Fig. 2: Simulation-based environment (Nam et al., 2022)

Notably, SimPy.Environment included in the the SimPy package manages time, defines events, and progresses them (Oh et al., 2022). SimPy.Event is responsible for modeling time progression during the simulation process, handling resource calls provided by SimPy, and modeling generators. Lastly, SimPy.Resource models the resources required for processes. It consists of Resource for modeling resources with similar characteristics, Store for storing discrete resources, and Container for storing continuous resources (Nam et al., 2022).

3. RESEARCH METHODOLOGY

3.1. Facility Layout Based on Diagnostic Center Size

In Republic of Korea, diagnostic centers vary in size, including 28 tertiary comprehensive hospitals, 328 general hospitals, 1,398 clinics, and 244 healthcare institutions (Healthcare bigdata hub, 2022). Except for tertiary comprehensive hospitals such as university hospitals, the majority of diagnostic centers lack the space to operate large-scale diagnostic equipment. Consequently, they resort to manual diagnostics or separate the diagnostic equipment for different processes.

Therefore, we aim to propose an optimal facility layout with respect to the size of the diagnostic center, which is suitable for the equipment of extraction, dispensing, and genetic amplification that can be used even in diagnostic centers that cannot operate large-scale diagnostic equipment.

Depending on the size and type of diagnostic center, the approach varies as shown in Table 2:

(1) Comprehensive hospitals, with a daily testing volume of over 1,000 cases, have deployed four large-capacity extraction equipment units, allowing simultaneous testing of different specimens (blood, sputum, tissues, etc.), and four PCR devices for diagnosing up to four different infections concurrently.

(2) Blood test centers, with a daily testing volume of over 500 cases, utilize two large-capacity extraction equipment units solely for blood testing.

(3) On-site centers, like emergency rooms, with a daily testing volume of under 500 cases, emphasize rapid testing. Therefore, four small-capacity extraction equipment units

with fast processing times are deployed along with two PCR devices to enable concurrent diagnosis of multiple infections.

(4) In the case of local hospitals that conduct fewer than 500 individual tests per day, they deploy one large-capacity extraction equipment and one PCR device.

(5) For small-scale hospitals, clinics, and public health centers with limited daily testing volumes, economical molecular diagnostics are implemented in small testing rooms. This involves the deployment of either one large extraction equipment unit or two smaller ones, along with a single PCR device.

Please note that the above configurations are designed to suit the testing capacities and requirements of each type and size of diagnostic center.

Table 2: Requirements of each type and size of diagnostic center

Type	Daily inspection volume	Number of simultaneous samples	Number of concurrent diagnoses	Extraction equipment Required quantity		PCR equipment Required quantity
				48 sample/ 40min	8 sample/ 30min	96 sample/ 90min
Comprehensive hospitals	<1000	<4	<4	4	-	4
Blood test centers	<500	<1	<1	2	-	2
Emergency room, On-site centers	>500	<4	<1	-	4	2
Local hospital	>500	1	1	1	-	1
Small-scale hospitals, Public health centers	>300	>2	1	-	2	1

3.2. Discrete Event Simulation Modeling

We configure six steps for molecular diagnostic process as follows: Step1) decapping, Step2) extraction, Step3) dispenser, Step4) PCR setup, Step5) PCR running, and Step6) analysis. For each step, we set parameters including testing capacity, processing time, and equipment range as shown in Figure 3. We then conduct simulations using the SimPy package, assuming a total testing period of 9 hours (540 minutes) per day and five different cases considering the various total number of tests and the workers' various working hours.

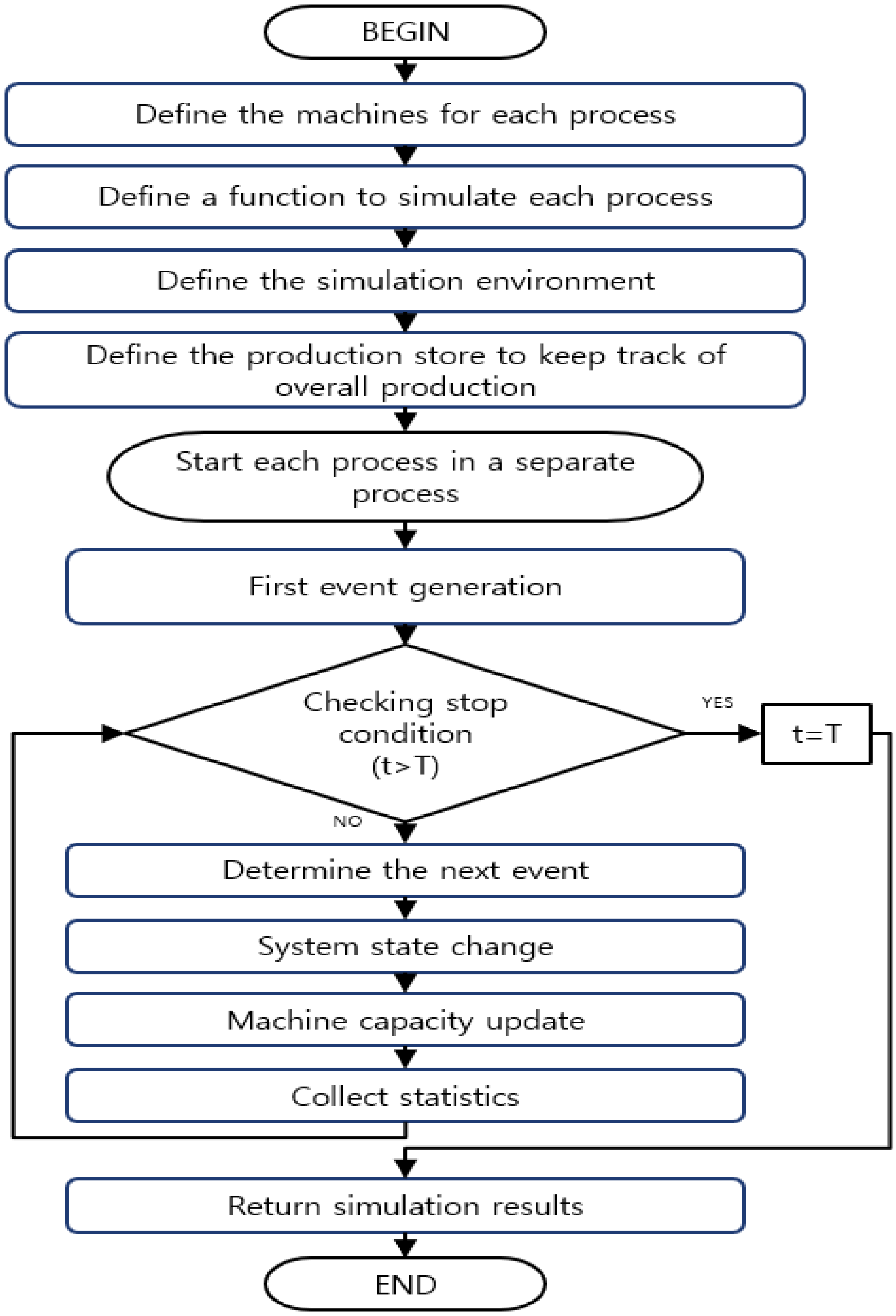
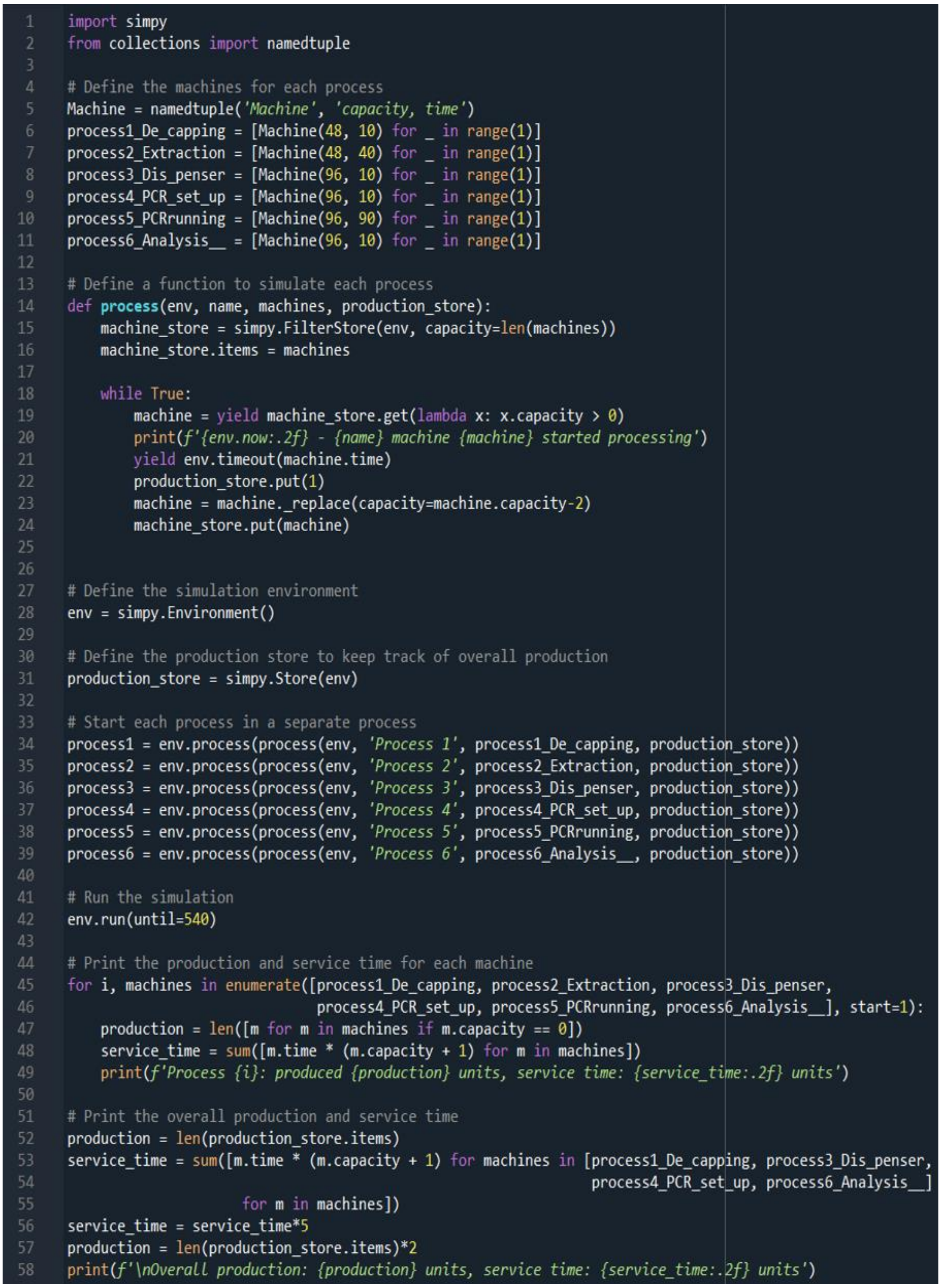

BEGIN
Define the machines for each process
Define a function to simulate each process
Define the simulation environment
Define the production store to keep track of overall production
Start each process in a separate process
First event generation
Checking stop condition (t>T)
YES
t=T
NO
Determine the next event
System state change
Machine capacity update
Collect statistics
Return simulation results
END

Fig. 3: Discrete-event Simulation model



```python
import simpy
from collections import namedtuple

# Define the machines for each process
Machine = namedtuple('Machine', 'capacity, time')
process1_De_capping = [Machine(48, 10) for _ in range(1)]
process2_Extraction = [Machine(48, 40) for _ in range(1)]
process3_Dis_penser = [Machine(96, 10) for _ in range(1)]
process4_PCR_set_up = [Machine(96, 10) for _ in range(1)]
process5_PCRrunning = [Machine(96, 90) for _ in range(1)]
process6_Analysis__ = [Machine(96, 10) for _ in range(1)]

# Define a function to simulate each process
def process(env, name, machines, production_store):
    machine_store = simpy.FilterStore(env, capacity=len(machines))
    machine_store.items = machines

    while True:
        machine = yield machine_store.get(lambda x: x.capacity > 0)
        print(f'{env.now:.2f} - {name} machine {machine} started processing')
        yield env.timeout(machine.time)
        production_store.put(1)
        machine = machine._replace(capacity=machine.capacity-2)
        machine_store.put(machine)


# Define the simulation environment
env = simpy.Environment()

# Define the production store to keep track of overall production
production_store = simpy.Store(env)

# Start each process in a separate process
process1 = env.process(process(env, 'Process 1', process1_De_capping, production_store))
process2 = env.process(process(env, 'Process 2', process2_Extraction, production_store))
process3 = env.process(process(env, 'Process 3', process3_Dis_penser, production_store))
process4 = env.process(process(env, 'Process 4', process4_PCR_set_up, production_store))
process5 = env.process(process(env, 'Process 5', process5_PCRrunning, production_store))
process6 = env.process(process(env, 'Process 6', process6_Analysis__, production_store))

# Run the simulation
env.run(until=540)

# Print the production and service time for each machine
for i, machines in enumerate([process1_De_capping, process2_Extraction, process3_Dis_penser,
                              process4_PCR_set_up, process5_PCRrunning, process6_Analysis__], start=1):
    production = len([m for m in machines if m.capacity == 0])
    service_time = sum([m.time * (m.capacity + 1) for m in machines])
    print(f'Process {i}: produced {production} units, service time: {service_time:.2f} units')

# Print the overall production and service time
production = len(production_store.items)
service_time = sum([m.time * (m.capacity + 1) for machines in [process1_De_capping, process3_Dis_penser,
                                                                process4_PCR_set_up, process6_Analysis__]
                    for m in machines])
service_time = service_time*5
production = len(production_store.items)*2
print(f'\nOverall production: {production} units, service time: {service_time:.2f} units')
```

Fig. 4: Discrete-event Simulation Python code

Figure 4 shows a source code to implement the overall simulation procedure. The code simulates different stages of a manufacturing process, monitors the operation and production of each machine, and evaluates the performance of the manufacturing process.

4. RESULT AND DISCUSSION

To validate the simulation model, we compare the result from the simulation with manual process scheduling analysis, as shown in Table 3. Based on a 9-hour workday, daily testing volume resulting from simulation shows an average accuracy of 96% compared to the process scheduling analysis. Additionally, from the perspective of the total working time per person excluding equipment operation time, the simulation result shows an average accuracy of 95%.

Table 3: Validation the Python SimPy Simulation Model

Case	A. Process Scheduling for Model	B. simulation-Python SimPy Model	Model Accuracy	C. Process Scheduling for Model	D. simulation-Python SimPy Model	Model Accuracy
	Daily inspection volume		[B/A*100] (%)	Total hours worked one person (min)		[D/C*100] (%)
Comprehensive hospitals	1440	1392	96.67	365	330	90.41
Blood test centers	768	792	96.97	195	210	92.86
Emergency room, On-site centers	352	366	96.17	310	297	95.81
Local hospital	384	372	96.88	200	200	100.00
Small-scale hospitals, Public health centers	176	188	93.62	310	300	96.77

5. CONCLUSION AND FUTURE WORK

In this paper, we present a discrete-event simulation analysis using SimPy, based on standard Python. We examine daily testing capacity with respect to facility types and analyze facility setup costs and required workforce for the integrated precision molecular diagnostic automation system. The results of this study will be helpful in the decision-making of diagnostic centers interested in establishing an integrated precision molecular diagnostic system.

For further research, it is necessary to improve facility layout in order to reduce congestion time by considering logistics flow according to the configuration of diagnostic center.

Additionally, it may also be valuable to research on applying reinforcement learning models to optimize testing capacity at each time point.

References

1. Agostino, B., Francesco, L., Letizia, N., Eleonora, B., & Roberto, M. (2012). Simulation, analysis, and optimization of container terminals processes. Simulation, and Scientific Computing, 3(4), 1-20. https://doi.org/10.1142/S1793962312400065
2. Caro, J. J., & Moller, J. (2016). Advantages and disadvantages of discrete-event simulation for health economic analyses. Expert Review of Pharmacoeconomics & Outcomes Research, 16(3), 327-329. https://doi.org/10.1586/14737167.2016.1165608
3. Farhan, M. M., Gohre, B. E., & Edward, J. P. (2020). Reinforcement Learning in Anylogic Simulation Models: A Guiding Example Using Pathmind. Winter Simulation Conference (WSC). https://ieeexplore.ieee.org/document/9383916
4. George, S. F. (2013). Discrete-event simulation: modeling, programming, and Analysis. Springer Science & Business Media.
5. Greasley, A. D. (2020). Implementing reinforcement learning in simio discrete-event simulation software. Proceedings of the 2020 Summer Simulation Conference, 27, 1-11. https://dl.acm.org/doi/abs/10.5555/3427510.3427538
6. Healthcare big data hub. (2022). Health Insurance Review of Korea. https://opendata.hira.or.kr/
7. Jang, I. G., Kim, D. H., & Son, Y. S. (2018). An Agent-Based Simulation Modeling with Deep Reinforcement Learning for Smart Traffic Signal Control. Information and Communication Technology Convergence (ICTC), 1028-1030. https://ieeexplore.ieee.org/document/8539377
8. Koo, J. W., Kwon, O. W., Kim, G. M., & Lee, D. K. (2018). Automated Devices for Point-of-care Immunoassay and Molecular Diagnostic. Korean Society for Precision Engineering, (pp. 282-282).
9. Menda, Kunal, Chen, Y. C., Justin, G. N., James, W. B., Brendan, D. T., Mykel, J. K., & David, W. P. (2018). Deep reinforcement learning for event-driven multi-agent decision processes. IEEE Transactions on Intelligent Transportation Systems, 20(4), 1259-1268. https://doi.org/10.1109/TITS.2018.2848264
10. Nam, S. H., Yun, H. C., & Woo, J. H. (2022). Research for Discrete-event Simulation Framework based on Open-source. Proceedings of the Korean Operations and Management Science Society Conference, 2581-2588.
11. Oh, S. H., Cho, S. H., Nam, S. H., Cho, K. Y., Yun, H. C., & Woo, J. H. (2022). A Study on the Simulation Framework for the Application of Reinforcement Learning to Production Scheduling. Proceedings of the Korean Operations and Management Science Society Conference, 2541-2547.
12. Pinciroli, L. C., Baraldi, P. R., Compare, M. C., Esmaeilzadeh, S. H., Farhan, M. M., Gohre, B. T., Grugni, S. H., Manca, L. G., & Zio, E. C. (2020). Agent-based Modeling and Reinforcement Learning for Optimizing Energy Systems Operation and Maintenance: The Pathmind Solution. Proceedings of the 29th European Safety and Reliability Conference (ESREL). http://rpsonline.com.sg/proceedings/9789811485930/html/5863.xml
13. Stricker, N. C., Andreas, K. H., Roland, S. R., & Simon, F. E. (2018). Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Annals - Manufacturing Technology, 67(1), 511-514. https://doi.org/10.1016/j.cirp.2018.04.041
14. Thomas, K. K. (2017). Intelligent simulation: Integration of SIMIO and MATLAB to deploy decision support systems to simulation environment. Simulation Modelling Practice

and Theory, 71, 45-60. https://doi.org/10.1016/j.simpat.2016.08.007

15. Thomas, T. E., Koo, J. K., Somali, C., & Saurabh, B. (2018). MINERVA: A Reinforcement Learning-based Technique for Optimal Scheduling and Bottleneck Detection in Distributed Factory Operations. 2018 10th international conference on communication systems & networks (COMSNETS). https://ieeexplore.ieee.org/document/8328189
16. Woo, J. H., & Oh, D. (2018). Development of simulation framework for shipbuilding. International Journal of Computer Integrated Manufacturing, 31(2), 210-227. https://doi.org/10.1080/0951192X.2017.1407452