Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

REINFORCEMENT LEARNING IN ANYLOGIC SIMULATION MODELS: A GUIDING EXAMPLE USING PATHMIND

Mohammed Farhan

Industrial, Manufacturing, & Systems Engineering
University of Texas at Arlington
701 South Nedderman Drive
Arlington, TX 76019, USA

Brett Göhre
Edward Junprung

Pathmind.Inc
1328 Mission Street
San Francisco, CA 94103, USA

ABSTRACT

Reinforcement Learning has recently gained a lot of exposure in the simulation industry. In this paper, we demonstrate the use of reinforcement learning in AnyLogic software models using Pathmind. A coffee shop simulation is built to train a barista to make correct operational decisions and improve efficiency that directly affects customer service time. The trained policy outperforms rule-based functions in terms of customer service time and throughput.

1 INTRODUCTION

Reinforcement Learning (RL) as a simulation optimization technique has shown substantial results in the fields of game playing and robotics (Kaelbling et al. 1996). Lately RL applications have spread across domains like supply chain, social, environmental and health sciences. In Paternina-Arboleda et al. (2005), a stochastic lot-scheduling problem (SELSP) was optimized using a dynamic RL policy to outperform various cyclic policies to meet production constraints and keep setup, inventory, and backorder costs low. In Probert et al. (2019), RL developed context-dependent dynamic response policies to minimize infectious disease outbreaks outperforming human generated static policies. In Olsen and Fraczkowski (2015), RL was used to study the coevolution of a predator-prey environment using an agent-based model to provide population dynamics and evolutionary insights in species. A summary of different applications of RL can be found in Li (2017).

Coffee shop operations have been previously studied with a focus on optimizing employee availability, customer table placement, and reducing customer service time. In Kadioglu (2017), a coffee shop simulation model concluded that increasing more baristas reduced the average service time. However, the cost of adding more baristas increased the overall operational cost of the coffee shop. In this paper, we study the operations of an imaginary coffee shop with a focus on the barista's actions and show how the sequence of actions affects the overall performance of the coffee shop by using RL. This model acts as a guiding example that shows the ease of applying RL in AnyLogic models using the Pathmind Library. This method can be easily mimicked to represent similar service industry models with similar decision points.

2 REINFORCEMENT LEARNING

Reinforcement learning is a core field of machine learning that deals with sequential decision-making (François-Lavet et al. 2018). In Sutton and Barto (2018), RL is defined as a branch of machine learning which deals with mapping situations to actions with a goal to maximize a numerical reward. Reinforcement learning follows the principles of a Markov Decision Process (MDP) which comprises an agent, its environment, its states, the possible actions, and a reward value shown in Figure 1. The agent is the decision maker in the model and everything that can influence the agent's decision is its environment. At the start of any learning, an agent is in State (S_t). Each action (A_t) an agent takes moves the agent to the next state (S_{t+1}) and impacts the environment which provides a certain reward (R_t). The reward obtained can either be a positive or negative reward. This process repeats itself trying to maximize the overall reward value (Sutton and Barto 2018). The general principle is to model the immediate impact of actions taken to yield better long-term outcomes. The actions a learning agent takes may not have an immediate reward which is known as a delayed reward. The process of trial and error search and delayed reward become the most important feature of RL.

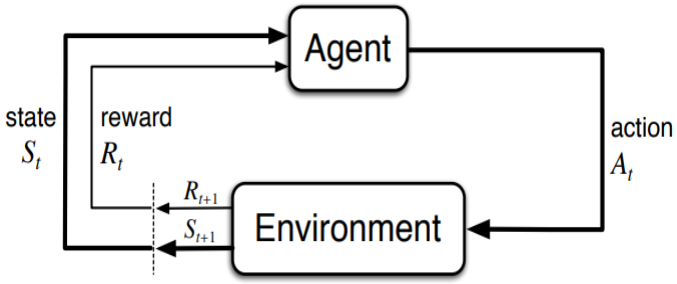

Figure 1: Agent-environment interaction in an MDP (derived from Sutton and Barto (2018)).

2.1 What Simulation Models are Fit for Reinforcement Learning?

There are primarily three types of system simulation methods: Discrete-Event Simulation (DES), Agent-Based Modeling (ABM), and System Dynamics (SD) (Owen 2008). Any simulation model that follows the Markov decision process principles can be catered to use RL. In simulation models, RL acts as an optimizer by itself, so models that are only for demonstration purposes or that use different input parameters to observe the resulting change in the output metrics cannot benefit from the sequential decision making aspect of RL. Due to the stochastic nature of real-world environments such as a coffee shop and the actions of a barista following an MDP principle, this mechanism is suitable for our use case.

2.2 Why Use RL and How is it Better Than Other Optimization Techniques?

The focus of RL optimization is to maximize or minimize a certain reward which is then reflected in the desired outcome. Machine learning models are heavily data centric (Schumann 2018). The actions performed by the RL agent are dynamic in nature and are only dependent on observations from the environment (simulation model or real world deployment). A simulation model is only required for training the model-free policy, but it can work as a stand-alone model-free "Oracle" for decision making in real deployment. Optimization techniques like heuristics or stochastic optimization are static in nature and are optimal for specific scenarios (Amaran et al. 2016). The user creating the heuristics must have strong domain knowledge of the problem. The process of creating and testing various policies is time consuming and could generate sub-optimal results. Any change in parameters of the simulation model would require another non RL optimization experiment run which is also a tedious task. A good example would be that of a traffic light optimization experiment. The non-RL optimization experiment would generate an optimal

static duration of time for a traffic phase. Any change in the traffic schedule would make the solution non-optimal. An RL policy can control the traffic phases dynamically based on the data provided and would not require multiple experiments to obtain optimal results.

3 HYBRID SIMULATION MODEL

The simulation model was created using AnyLogic 8 Professional 8.5 software. The model time unit is seconds and the simulation run time is 12 hours starting at 7 am and ending at 7 pm. The coffee shop model has two agents – Customer and Server. The customer agents are modelled as a population of agents following a discrete event modelling process while the server agent is the barista which is modelled with the agent-based modelling principles. The combination of the two methods constitute the hybrid simulation model of the coffee shop described in this paper. Rondini et al. (2017) highlights the advantages of using a hybrid simulation model (DES and ABM) over a pure DES model in customer-oriented product service system models.

3.1 Discrete-Event Model

The customer agents are modelled using a pedestrian library available in the AnyLogic software. The pedestrian library works almost the same as the process flow library with the exception that the pedestrians move according to the physical rules provided in the simulation environment and can also make decisions based on the situation in the environment. The customers follow a discrete first in, first out flow. Customers arrive at the coffee shop following an hourly rate schedule shown in Table 1. Multiple arrival schedules showing a range of variability like Table 1 are used for training. The arrival time is recorded for each customer in a variable called timestamp inside the customer agent.

Table 1: Customer hourly arrival rate.

Time period	Customer arrival rate
07:00 am – 08:00 am	21
08:00 am – 09:00 am	28
09:00 am – 10:00 am	33
10:00 am – 11:00 am	30
11:00 am – 12:00 pm	28
12:00 pm – 01:00 pm	25
01:00 pm – 02:00 pm	23
02:00 pm – 03:00 pm	31
03:00 pm – 04:00 pm	24
04:00 pm – 05:00 pm	26
05:00 pm – 06:00 pm	29
06:00 pm – 07:00 pm	30

The customers go through three service blocks while in the system - Place Order, Collect Order and Pay Bill before they exit the shop. Each service block requires the presence of the server. The service times for each block is listed below in Table 2. The customer flow is shown in Figure 2. A kitchen cleanliness variable with an initial value of 1.0 being the highest is added to the model. Each time a customer collects his order, the kitchen cleanliness variable value is reduced by 0.01. A function called collectOrderDelay() is created to calculate the delay time for collect order service block. With a kitchen cleanliness level of 1.0 the function uses PERT distribution with (minimum=30, maximum=90, mode=60) seconds as delay time. With every 0.01 decrease in kitchen cleanliness, an additional 1 second delay is added to the total delay

time for the collect order service block. If the server does not attend to the customer at the place order block within a balk time limit, the customer would leave the coffee shop without placing the order. Statistics for queue length, balked customer count, successful customer count, and wait times for each service block are recorded.

Table 2: Delay time for service blocks. * triangular (min, max, mode) **exponential (min, max, shift, stretch) ***uniform (min, max).

Service block	**Delay time distribution (in seconds)**
Place order process	Triangular (8, 25, 12) *
Prepare and deliver order process	collectOrderDelay ()
Bill customer process	Exponential (10, 45, 10, 2) **
Customer balk time	Uniform (8, 12) ***

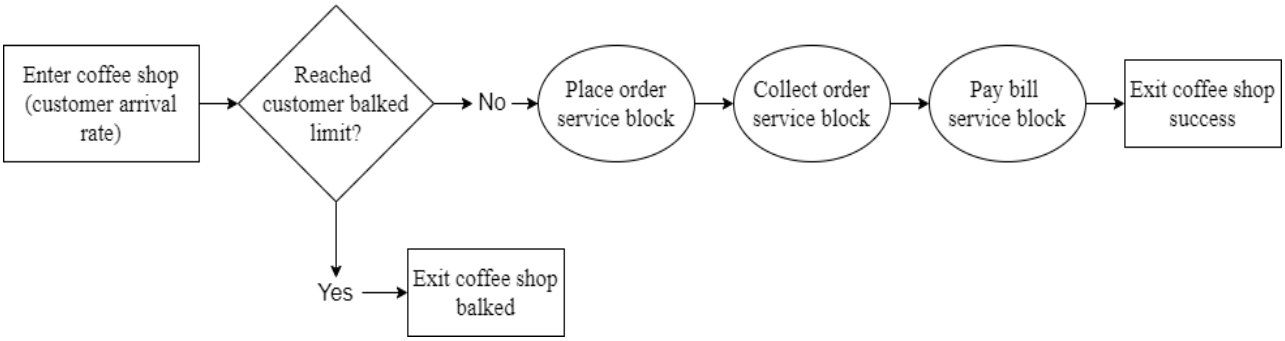

Figure 2: Customer discrete-event flow.

3.2 Agent-Based Model

The Barista in the model is the decision maker and can take up tasks when idle such as – taking order, preparing and delivering order, bill customers and clean the kitchen. The state chart library of AnyLogic is used to model the barista's actions. The state chart for the server is shown in Figure 3.

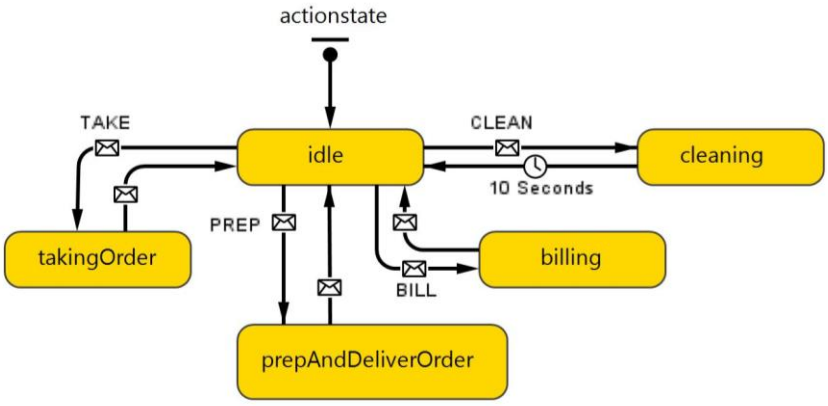

Figure 3: Barista's state-chart.

The tasks are performed based on sending and receiving messages between the customers and the barista. The actions of the barista are user controlled and can be controlled using a rule-based function to send messages to the barista to trigger a specific action. The tasks can only be triggered when the barista is in the idle state. Once the customer passes the defined delay time shown in Table 2 for their respective service blocks, they send a message back to the server releasing the server back to idle state to perform more tasks. The barista spends 10 seconds in cleaning state before returning to idle state. Each cleaning state increases the kitchen cleanliness variable by 0.1.

3.3 Baseline Methods

A barista today is trained to perform their job based on a rulebook or experience from former baristas (Monk and Ryding 2007). The barista's actions in the model can be manually controlled by using user defined buttons for each action or functions that get called if a condition is met. A user can choose to select an action if the barista is in the idle state by pressing a button during the simulation run or an event can be created to trigger these actions. To test our RL policy results, rule-based functions were created that automate the action sequence a barista can perform. For any action selection to be valid, two primary conditions are always checked irrespective of any method:

- The actions can be triggered only when the barista is in idle state.
- The queue length at the service blocks should be greater than zero if an action is chosen for that respective service block.

3.3.1 Rule-Based Function 1

Customers in the place order queue are given priority over other tasks, followed by preparing order, and then billing customers. Once the customer queues are empty, the kitchen cleanliness variable is checked. If the kitchen cleanliness variable is lower than 0.9, cleaning action is taken. The code snippet for this function is shown in Figure 4.

```
if(server.actionstate.getActiveSimpleState() == server.idle){
    if (placeOrder.queueSize(placeQueue) > 0) {
    send (TAKE, server);
    } else if (collectOrder.queueSize(waitToCollectArea) > 0) {
    send (PREP, server);
    } else if (payBill.queueSize(waitToPayArea) > 0) {
    send (BILL, server);
    } else if (kitchenCleanliness < 0.9) {
    send (CLEAN, server);
    }
}
```

Figure 4: Code for rule-based function 1.

3.3.2 Rule-Based Function 2

Barista's actions follow a first in, first out (FIFO) flow for customers. The barista prioritizes the billing queue first, followed by the preparing and delivering order queue and then the place order queue. Once the customer queues are empty, the kitchen cleanliness variable is checked. If the kitchen cleanliness variable is lower than 0.9, cleaning action is taken.

3.3.3 Rule-Based Function 3

The kitchen cleanliness directly affects the delay time in preparing and delivering customer orders. So, the priority is given to clean the kitchen first if kitchen cleanliness level is below 0.9. The customers in the place order queue are given second preference followed by the collect order queue and then the billing queue.

Average service time, successful customers, balked customers and average kitchen cleanliness are the four output metrics for this study. After running Monte Carlo replications with 95% confidence using the rule-based heuristics, the mean value and confidence intervals (CI) of the output metrics are shown in Table 3. The desired result would be to have lower service times and balked customers, and higher successful customers and kitchen cleanliness. From the results, Rule-Based Function 2 outperformed other heuristics in almost all output metrics setting a baseline for the trained policy.

Table 3: Baseline results.

Heuristic	Statistic	Average service time of customer (in minutes)	Number of successful customers	Number of balked customers	Average kitchen cleanliness level
Rule-Based Function 1	mean	17.9	153.2	26.5	0.62
	95% CI	16.4 , 19.3	145.3 , 161.2	22.6, 30.4	0.6, 0.64
Rule-Based Function 2	mean	4.8	259.4	5.8	0.91
	95% CI	4.7 , 4.9	258.2 , 260.6	5.1 , 6.5	0.906, 0.914
Rule-Based Function 3	mean	19.9	222	25.14	0.95
	95% CI	17.7, 22.1	215.9, 228.2	20.6, 29.6	0.949 , 0.950

4 REINFORCEMENT LEARNING IMPLEMENTATION USING PATHMIND

There are 5 important elements when implementing RL in Anylogic simulation models using Pathmind Library – Observation Function, Reward Variables, Action Function, Action Trigger, and Reward Function. Every element plays an important role in making sure the RL agent learns and behaves effectively. The RL elements are included in an AnyLogic software library called PathmindHelper from Pathmind.Inc (Pathmind 2020). All RL related functions are added in PathmindHelper before exporting the model for training on the Pathmind web application.

4.1 Observation Function

All the actions an RL agent takes are based on the values that the observation function outputs. This makes this function the eyes of the RL agent for the learning process. The observations provide the necessary information needed about the simulation environment and impact the actions to be taken. The queue lengths at each service block, the kitchen cleanliness level, and the current simulation time are sufficient to train this model. More information can be provided based on the outcome of the training to fine tune the results or speed up the learning process. The code snippet for observation function is shown in Figure 5.

```
double[] obs = new double[]{placeOrder.size(),collectOrder.size(),payBill.size(),
kitchenCleanliness,time()};
return obs;
```

Figure 5: Code for observation function.

4.2 Reward Variables

The reward variable function provides information about the objective of the training. The user specifies what variables are impacted from the actions a learning agent takes and can incentivize or penalize the actions in the reward function. Kitchen Cleanliness, throughput, balked customers and average service time are the output metrics for the study. The code snippet for reward variables is shown in Figure 6.

```
new double[]{kitchenCleanliness, payBill.out.count(), custFailExit.countPeds(),
serviceTime_min.mean()}
```

Figure 6: Code for reward variable function.

4.3 Action Trigger

The action trigger informs the learning algorithm when to trigger the next action. Actions can be triggered by a cyclic timed event say – every second or every 5 minutes. The trigger can also be due to a condition being satisfied such as the server being in a certain state. For this model, we will trigger the action as a combination of time and condition. The action is triggered every second, but can only occur when the server is in the idle state.

4.4 Action Function

In this function, all possible actions a learning agent can take are mentioned. The function needs to pass an argument of type *int*. Each action chosen by the policy would correspond to a specific task a learning agent can do like in the rule-based functions. In this model, the tasks are - taking an order, preparing and delivering an order, billing a customer, and cleaning the kitchen. The action array can have 4 possible actions - 0, 1, 2 and 3. The code snippet for the action function is shown in Figure 7.

```
if(server.actionstate.getActiveSimpleState() == server.idle) {
if (action==0 && placeOrder.queueSize(placeQueue) > 0 ) {
send (TAKE, server);
} else if (action==1 && collectOrder.queueSize(waitToCollectArea) > 0) {
send (PREP, server);
} else if (action==2 && payBill.queueSize(waitToPayArea) > 0){
send (BILL, server);
} else {
send (CLEAN, server);
}}
```

Figure 7: Code for action function.

4.5 Reward Function

A reward function is used to incentivize or penalize the actions a learning agent takes. The objective of the learning agent would be to maximize its reward. The reward function is written for the reward variables specified in Section 4.2. When an action trigger occurs, the learning agent saves the values of the reward variable before the action is performed and after the action is performed. Two values called the “before” and “after” are used to create the reward function. To maximize a reward variable, a positive reward can be given if the “after” variable is higher than the “before” variable. In this experiment, maximizing the kitchen cleanliness and number of successful customers is rewarded and minimizing the number of balked customers and service time is penalized. Two reward function experiments are evaluated.

4.5.1 Reward Function 1

The objective of this experiment is to train the policy for a desired outcome of having higher throughput, lower balked customers and lower average service time. A value of 5 is multiplied to the average service time to enable equal weightage of all variables since the difference in average service time between two time steps is in decimals whereas the other two reward variables are integers. The code for reward function 1 is shown in Figure 8.

```
reward += after[1] - before[1];     // Maximize successful exits
reward -= after[2] - before[2];     // Minimize balked customers
reward -= (after[3] - before[3])*5; // Minimize average service time
```

Figure 8: Code for reward function 1.

4.5.2 Reward Function 2

The objective of this experiment is to include an incentive for kitchen cleanliness since it directly affects the overall service time. A value of 10 is multiplied to the difference of kitchen cleanliness levels to enable equal weightage on all reward variables. The code for reward function 2 is shown in Figure 9.

```
Reward = (after[0] - before[0])*10; // Maximize Kitchen Cleanliness level
reward += after[1] - before[1];     // Maximize successful exits
reward -= after[2] - before[2];     // Minimize balked customers
reward -= (after[3] - before[3])*5; // Minimize average service time
```

Figure 9: Code for reward function 2.

In one of the trial experiments, the reward function for the training only incentivized reducing the overall service time. The policy obtained from this training decided to not take any orders from the customers, hence keeping the service time to the lowest possible value of zero while the customers balked in the system. It managed to achieve its highest reward score, but that did not match the actual objective of the experiment. Hence, reward function shaping becomes an important piece for training since RL optimization is data driven.

5 TRAINING ON PATHMIND

The simulation model is exported as a standalone Java application from AnyLogic after inputting all RL functions. The exported folder is uploaded on the Pathmind web application to run the RL training on cloud. The cloud platform allows users to run multiple experiments with different reward functions at faster computational speeds to get the desired results. Two elements determine the accuracy and speed of the training - RL training algorithm and RL hyperparameter tuning methodology. Pathmind uses the proximal policy optimization (PPO) algorithm because of its ease of implementation, tuning, and state-of-the-art performance on discrete and continuous action spaces (Schulman et al. 2017). A method called Population-Based Training (PBT) pioneered by Google's DeepMind was applied to train this model. The objective of PBT is to automatically discover the best set of hyperparameters that encourage the learning agent to find the best performing policy as quickly as possible (Jaderberg et al. 2017). PBT was found to be much more time efficient in unearthing the best performing policy compared to other techniques (e.g. grid search). Pathmind automates this process to simplify a user's experience on running RL experiments.

Users create experiments on the Pathmind web application and frame the reward function mentioned in Section 4.5 before beginning the training process. The web application shows the learning process curve

by displaying the mean reward score per iteration (4,000 learning steps) as in Figure 10. A complete simulation run is called an episode and each episode can have varying amounts of learning step sizes due to the stochastic nature of the model. Each line in the graph represents a set of hyperparameter settings for the RL. An upward converging trend is expected to confirm that the policy is learning with increasing iterations. If the mean reward score does not change for a certain period of iterations, training is complete and the RL policy will be ready to be downloaded. The policy can be imported back into AnyLogic to test the results of the training.

6 RESULTS AND FINDINGS

A Monte Carlo experiment with 95% confidence is run using the trained policy and the output metrics are shown in Table 4. To compare the different outputs obtained we make use of weighted scores. These weights can vary based on the objective of a coffee shop owner. One owner might want to emphasize on throughput while the other might give importance to kitchen cleanliness. In this paper, we will try four different weight scores as shown in Table 5.

The comparison between the best baseline method and RL policies is shown in Table 6. Even though kitchen cleanliness was not a motivating factor in Reward Function 1, it managed to score the highest in all weighted scores. Reward Function 2 was denser as it incentivized higher kitchen cleanliness along with the other reward variables. It managed to increase the average kitchen cleanliness factor by 0.01, but at the cost of increasing the average service time. Since it spent action steps to maximize kitchen cleanliness the other output metrics were affected. This implies that the optimal solution lies between the output values from these two reward functions.

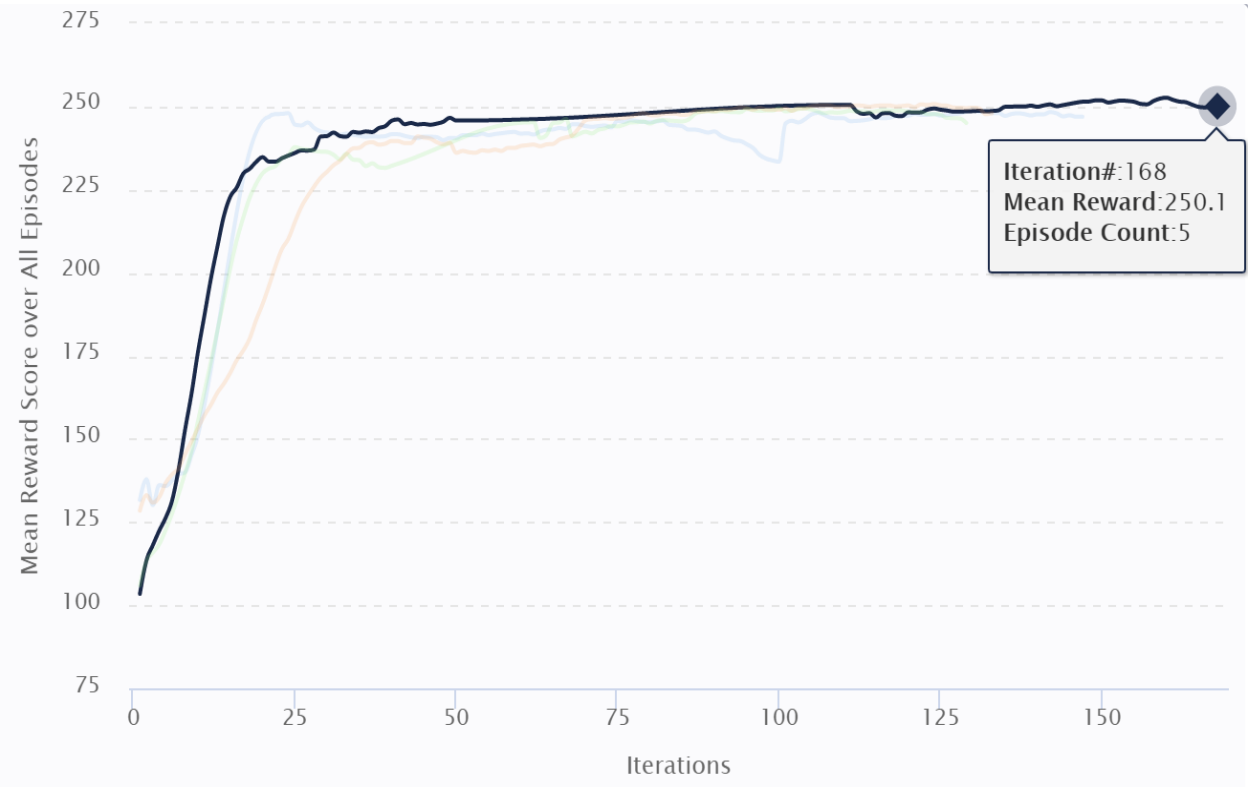

Figure 10: Reward function 1 training.

Table 4: Pathmind trained policy results.

Policy	Statistic	Average service time of customer	Number of successful customers	Number of balked customers	Average kitchen cleanliness level
Reward Function 1	mean	4.73	263.5	3.0	0.95
	95% CI	4.69, 4.77	262.1, 264.8	2.7, 3.3	0.949, 0.95
Reward Function 2	mean	5.45	261.62	2.66	0.96
	95% CI	5.36, 5.54	260.34, 262.9	2.29, 3.03	0.958, 0.962

Table 5: Weights for output metrics.

Weighted score	Average service time of customer	Number of successful customers	Number of balked customers	Average kitchen cleanliness level
Weighted Score 1	-0.33	0.33	-0.33	0
Weighted Score 2	-0.25	0.25	-0.25	0.25
Weighted Score 3	-0.50	0.50	0	0
Weighted Score 4	0	0.5	0	0.5

Table 6: Weighted scores.

Methods	Weighted score 1	Weighted score 2	Weighted score 3	Weighted score 4
Rule-based function 2	82.1	62.4	127.3	130.1
Reward Function 1	84.4	64.1	129.3	132.2
Reward Function 2	83.65	63.6	128.0	131.2

7 CONCLUSION & FUTURE SCOPE

This paper demonstrates the ease in applying Reinforcement Learning to AnyLogic simulation models using Pathmind Library so as to make better and faster decisions than rule-based heuristics. A hybrid simulation model (DES and ABM) of a conceptual coffee shop with a single barista as a learning agent was trained to perform actions using Reinforcement Learning. Two reward functions experiments were trained

using the Pathmind web application. The two policies obtained from these reward functions performed better than rule-based functions in all weighted scores. A user can create complex rule-based functions to set a higher baseline for the RL policy to beat. The user creating the heuristics has to be a domain expert to validate the results of the heuristics. RL relies only on the data provided by the user making it less biased compared to heuristics. The simplicity of using the Pathmind cloud-based platform is that it does not require prior knowledge of RL algorithms. This paper provides the basic concepts needed to apply RL in any AnyLogic simulation model that satisfies the model criteria mentioned in Section 2.1.

The future of RL implementation in the coffee shop service industry can be that of a digital teller that informs the barista of what its next actions should be. An artificial intelligent robot embedded with the policy can also perform the barista's tasks. The input in a real-world scenario can be from cameras or sensors installed in the coffee shop that provide the observations for the learning agent. The RL policy can be validated by testing it against a real-world coffee shop model. The RL policy could help coffee shop owners improve the efficiency of their operations and save time and cost on training new baristas.

ACKNOWLEDGMENTS

The RL training was facilitated by Pathmind. A special thanks to Tyler-Wolfe Adam from the AnyLogic software team for providing simulation modifications and insight.

REFERENCES

Amaran, S., N. V. Sahinidis, B. Sharda, and S. J. Bury. 2016. "Simulation Optimization: A Review of Algorithms and Applications". *Annals of Operations Research* 240(1):251-380.

François-Lavet, V., P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. 2018. "An Introduction to Deep Reinforcement Learning". *Foundations and Trends in Machine Learning* 11(3-4):219-254.

Jaderberg, M., V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. 2017. *Population Based Training of Neural Networks*. http://arxiv.org/abs/1711.09846, accessed 14th July 2020.

Kadioglu, O. 2017. Applied Mathematics and Operations Research a Case: Starbucks Coffee Shop Simulation. Department of Mathematics, Bahçeşehir University. https://www.researchgate.net/publication/324748461_Applied_Mathematics_and_Operations_Research_A_Case_Starbucks_Coffee_Shop_Simulation, accessed 15th June 2020.

Kaelbling, L. P., M. L. Littman, and A. W. Moore. 1996. "Reinforcement Learning: A Survey". *Journal of Artificial Intelligence Research* 4:237–285.

Li, Y. 2017. *Deep Reinforcement Learning: An Overview*. http://arxiv.org/abs/1701.07274, accessed 8th July 2020.

Monk, D., and D. Ryding. 2007. "Service Quality and Training: A Pilot Study". *British Food Journal* 109(8):627–636.

Olsen, M. M., and R. Fraczkowski. 2015. "Co-evolution in Predator Prey Through Reinforcement Learning". *Journal of Computational Science* 9:118–124.

Owen, C., D. Love, and P. Albores. 2008. "Selection of Simulation Tools for Improving Supply Chain Performance". In *Proceedings of 2008 OR Society Simulation Workshop*, January 1st, Warwickshire, UK.

Paternina-Arboleda, C. D., and T. K. Das. 2005. "A Multi-Agent Reinforcement Learning Approach to Obtaining Dynamic Control Policies for Stochastic Lot Scheduling Problem". *Simulation Modelling Practice and Theory* 13(5):389–406.

Pathmind.Inc. 2020. http://pathmind.com, accessed 14th April.

Probert, W. J. M., S. Lakkur, C. J. Fonnesbeck, K. Shea, M. C. Runge, M. J. Tildesley and M. J. Ferrari. 2019. "Context Matters: Using Reinforcement Learning to Develop Human-Readable, State-Dependent Outbreak Response Policies". *Philosophical Transactions of the Royal Society Biological Sciences* 374(1776):20180277.

Rondini, A., F. Tornese, M. G. Gnoni, G. Pezzotta, and R. Pinto. 2017. "Hybrid Simulation Modelling as a Supporting Tool for Sustainable Product Service Systems: A Critical Analysis". *International Journal of Production Research* 55(23):6932–6945.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. *Proximal Policy Optimization Algorithms*. https://arxiv.org/abs/1707.06347, accessed 8th July 2020.

Schumann, B. 2018. Time to Marry Simulation Models and Machine Learning. https://www.benjamin-schumann.com/blog/2018/5/7/time-to-marry-simulation-models-and-machine-learning, accessed 14th June 2020.

Sutton, R. S., and A. G. Barto. 2018. *Adaptive Computation and Machine Learning. Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, Massachusetts: The MIT Press.

AUTHOR BIBLIOGRAPHY

MOHAMMED FARHAN is a PhD Candidate in the Department of Industrial, Manufacturing, & Systems Engineering at The University of Texas at Arlington. He is currently working as a graduate researcher with Pathmind.Inc with focus on combining simulation models and Reinforcement Learning. His research interests include hybrid simulation, human behavior modelling, artificial intelligence, and supply chain management. His expertise lies in modelling hybrid simulation models of socio-environmental systems using reinforcement learning capabilities. His email address is farhanm@exchange.uta.edu.

BRETT GÖHRE is a Lead Reinforcement Learning Scientist at Pathmind. His primary interest is building an adaptive decision-making AI into a generally accessible product. Brett holds degrees in Physics from the University of California, Berkeley and Santa Cruz. His email address is brett@pathmind.com.

EDWARD JUNPRUNG is a Product Manager at Pathmind. His primary interests include the practical applications of AI and Reinforcement Learning. Edward holds a B.S. in Management Science and Accounting from the University of California, San Diego. His email address is edward@pathmind.com.