Does a Q-Learning NetLogo Extension Simplify the Development of Agent-based Simulations?

Eloísa Bazzanella,[1] **Fernando Santos**[1]

[1]Departamento de Engenharia de Software
Universidade do Estado de Santa Catarina (UDESC)
Ibirama – SC – Brazil

elobazzanella@gmail.com, fernando.santos@udesc.br

Abstract. *Agent-based modeling and simulation is a simulation paradigm that allows focusing on individuals, their interactions, and the complex behavior that emerges from them. Agent-based simulations are typically developed in simulation platforms that provide features related to agents. One such platform is NetLogo, to which a reinforcement learning extension was made available recently. The extension provides commands for using the Q-Learning algorithm, but no evaluation on whether it simplifies the development of simulations is available. This paper presents a quantitative evaluation on using the extension in two simulations: the classic cliff walking problem; and a real-world, adaptive traffic signal control (ATSC) simulation. Results show that the size of simulations source code developed using the extension is smaller than those developed without using it, giving evidence that the extension simplifies the development of simulations*

1. Introduction

Computer simulations have been used to study phenomena and support decision making. For example, traffic simulations assist in the design of transport infrastructure, and evacuation simulations assist the design of stadiums that can be quickly evacuated in emergency situations. It is not trivial to develop analytical models that simulate the behavior of these complex systems. Producing them is a task that has been extensively investigated in the context of agent-based modeling and simulation (ABMS), a simulation paradigm that makes use of agents to reproduce and study a phenomenon under investigation [6].

The ABMS paradigm allows focusing on the individuals (agents) and the implications resulting from their behavior and from the interaction among them. Artificial intelligence techniques can be incorporated into agents to enable them to adapt to changes in themselves or in the environment [6]. One of such techniques is reinforcement learning, that enable agents to learn through experience. A well-known reinforcement learning method is Q-Learning [16].

Although agent-based simulations can be developed with general-purpose languages such as Java, there are languages and platforms specifically tailored for developing agent-based simulations. These platforms provide agent and simulation features that simplify the development of agent-based simulations. A popular agent-based simulation platform is NetLogo [17]. Statistics show that, among all the simulations available in the CoMSES [5] repository in 2020, 33.6% were developed in NetLogo. It provides its own programming language, with high-level commands for operations frequently required in

agent-based simulations. There are modules (called *extensions*) that provide additional commands and features for developing simulations. Recently, an extension that provides commands for using the Q-Learning method in simulations was made available.[1] However, no evaluation regarding whether this extension simplifies the development of agent-based simulations is available.

In this paper we present a quantitative evaluation conducted to investigate whether the existing NetLogo Q-Learning extension simplifies the development of simulations. The evaluation considered two simulations: the classic cliff walking problem; and an adaptive traffic signal control (ATSC) simulation. Both simulations were implemented using the Q-Learning extension, and their source code were compared to existing Q-Learning implementations without the extension. For the comparison we used the *lines of code* software size metric. Results show that the size of simulations source code developed using the extension are 13.72% (cliff walking) and 47.06% (ATSC) smaller than source codes developed without using it, giving evidence that the existing NetLogo Q-Learning extension simplifies the development of simulations.

The remaining of this paper is organized as follows. Section 2 presents the required background on ABMS, reinforcement learning, and the existing NetLogo Q-Learning extension. In Section 3 we describe the conducted quantitative evaluation, which includes: the adaptive traffic signal control simulation and the reasons why it was selected for our study; the procedure and metric adopted in the evaluation; and the obtained results. Finally, Section 4 presents concluding remarks and future work.

2. Background

This section presents concepts related to our quantitative evaluation. We first introduce the ABMS paradigm. Then, we describe reinforcement learning, in particular the Q-Learning algorithm. Finally, we present the existing Q-Learning NetLogo extension and detail how it is used to incorporate the Q-Learning method into agent-based simulations.

2.1. Agent-based Modeling and Simulation

Agent-based modeling and simulation (ABMS) is a simulation paradigm that makes use of agents to reproduce and study a phenomenon under investigation [6]. In the ABMS paradigm, a simulation is composed of agents that can interact with each other and are situated on an environment they can perceive and modify through their actions. Agents have two fundamental properties: they are autonomous in their decision making towards their objectives; and they are able to interact with each other. Agents are able to decide based on logical deductions, just like human reasoning, or via deduction combined with a decision-making mechanism. Additionally, agents can have different skills and architectures, and they can perform distinct roles [18].

ABMS has been used to model and run simulations in many application domains, such as traffic, ecology, economics and epidemiology [9]. In these cases, ABMS has been chosen as the simulation paradigm because it is able to incorporate the inherent complexity of individual behavior and interactions in real-world scenarios [9]. Artificial intelligence techniques can be incorporated into agents to improve their behavior. Bazzan and

[1] `https://github.com/agentbasedsimulations/qlearning-netlogo-extension`

Klügl [3], for example, enumerate studies on applying agents endowed with intelligent abilities to improve traffic and transportation systems.

Agent-based simulations can be developed with general-purpose programming languages such as Java and Python. However, there are agent-based simulation platforms which provide agent and simulation related features that ease the development of simulations. One of such platforms is NetLogo [17]. NetLogo is a programmable modeling environment that allows simulating natural and social phenomena. It provides a programming language, with high-level commands for features frequently required in agent-based simulations, such as for setting up the environment and moving agents through it. Additional commands and features are provided via modules (called *extensions*). One of such extensions, which is detailed next in Section 2.3, provides commands for using the Q-Learning reinforcement learning method.

2.2. Reinforcement Learning

Humans usually learn through interactions with the environment. Throughout life, these interactions are an important source of knowledge about the environment and about themselves. Learning through interaction is a fundamental idea of almost all theories of learning and intelligence [15].

There are learning methods that can be incorporated into agents to enable them to learn over time. Reinforcement learning (RL) is one of such methods, in which an agent learn through experience. According to Monteiro and Ribeiro [11], RL is "a computational learning paradigm in which a learning agent seeks to maximize a performance measure based on the reinforcement (reward or punishment) it receives when acting on an unknown environment". RL methods often adopt a state-based representation of the environment. Periodically, the agent must select which action to execute. When an action is executed, the agent receives a reward signal based on the outcomes of previous states and actions. By observing the reward received when executing different actions on different states for a period of time, an agent running a RL method is able to learn an optimal control policy (one that maximizes the expected reward).

Q-Learning [16] is one of the available RL methods. Q-Learning works by estimating optimal state-action values, named *Q-values*. Each *Q-value* is a numerical estimator of quality for a given pair of state and action. Therefore, a *Q-value* $Q(s, a)$ represents the maximum discounted sum of future rewards the agent can expect to receive if it starts in state s, choose action a, and then continues to follow an optimal policy.

Q-Learning maintains a data structure called *Q-table*, which stores a *Q-value* for every pair (s, a). As the agent acts on the environment, $Q(s, a)$ values are updated to consider the reward signal r received when the action a is executed in state s. In addition to a set S of states and A of actions, Q-Learning has two parameters: a learning rate α and a discount factor γ. The first specifies how much of the agent expertise is replaced by the outcomes of recently experienced actions. The last describes the agent preference for immediate over future rewards.

Figure 1 shows the Q-Learning algorithm. Q-Learning lets the agent run for a number of episodes. In each episode the agent starts from an initial state and goes through countless states until it reaches a terminal state. At each step, the agent selects an action a for the current state s using a selection policy. Greedy action selection policies exploit

Input: S, A, $\alpha \in (0,1)$ $\gamma \in (0,1)$
foreach $episode$ **do**
  $s \leftarrow initial\_state$
  repeat
    choose an action a for state s using a selection policy (e.g., ϵ-greedy)
    perform the action a
    observe the new state s' and the reward r received
    update *Q-table*:
      $Q(a,s) \leftarrow Q(a,s) + \alpha(r + \gamma\, max_a\, Q(a',s') - Q(a,s))$
    $s \leftarrow s'$
  until $s \neq terminal\_state$;
end

Figure 1. Q-Learning algorithm. Adapted from Russel and Norvig [13]

the current agent expertise and increase immediate reward. However, with such greedy policies the agent may not explore some actions and end up learning sub-optimal control policies. An alternative selection policy commonly used with Q-Learning to avoid sub-optimal policies is the ϵ-greedy [15]. With ϵ-greedy the agent behaves greedily most of the time, but with probability ϵ it selects an action at random. The ϵ value is decreased by an ϵ-decay rate to let the agent exploit the optimal policy after a learning period.

Following the Q-Learning algorithm, the chosen action is performed by the agent, which observes the new state s' and the reward r received. The $Q(a,s)$ value is then updated in the *Q-table* according the Q-Learning update rule, which considers the learning rate α, the reward r, the discount factor γ, and the expected future reward provided by following the optimal policy for the new state s' onward (given by the action a' that maximizes $Q(a',s')$). Finally, the current state s is updated and the learning process is repeated until the agent reaches a terminal state.

2.3. Q-Learning NetLogo Extension

An *extension* is a NetLogo module that extends its programming language with additional commands and features. Recently, a NetLogo extension with a ready-to-use implementation of the Q-Learning algorithm was made available [7]. The extension provides additional NetLogo commands to specify the following Q-Learning elements: states, actions, the reward, the action selection policy, the end of episode clause, the episode reset procedure, learning rate, and discount factor. In addition, the extension also provides commands to execute the Q-Learning algorithm when the agent is expected to act and learn. The extension, called *Q-Learning Extension*, can be installed via the NetLogo Extension Manager and its documentation is available online.[1]

We describe how the Q-Learning extension works by means of the classic *cliff walking* problem [15] shown in Figure 2. In this problem the environment is a grid, in which a subset of cells represents a cliff. The goal of a *Walker* agent is to learn how to go from the starting cell S to the goal cell G without falling off the cliff (gray cells). The agent can move up, down, right, and left. A learning episode ends when the agent reaches the goal cell or falls off the cliff. If the agent falls off, its reward is -100; otherwise, its reward is -1 for each cell it has visited.

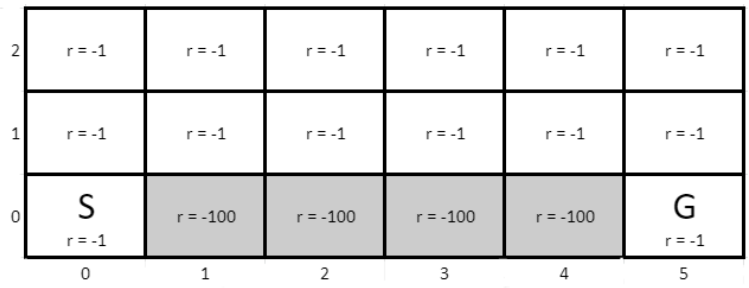


Figure 2. Cliff walking problem. Adapted from Sutton and Barto [15]

The NetLogo source code developed with the Q-Learning extension for the cliff walking problem is partially shown in Figure 3.[2] The `setup` procedure (lines 1–12) is where the simulation is set up. The `ask` block in lines 3–11 requests that all `Walker` agents execute a few commands provided by the Q-Learning extension in order to set up the Q-Learning algorithm. These commands are described next.

```
1  to setup
2      clear-all
3      ask Walkers [
4          qlearningextension:state-def ["xcor" "ycor"]
5          (qlearningextension:actions [goUp] [goDown] [goLeft] [goRight])
6          qlearningextension:reward [rewardFunc]
7          qlearningextension:learning-rate 0.4
8          qlearningextension:discount-factor 0.2
9          qlearningextension:action-selection "e-greedy" [0.8 0.99]
10     ]
11 end
12
13 to go
14     ask Walkers [
15         qlearningextension:act
16         qlearningextension:learn
17     ]
18 end
```

Figure 3. Cliff Walking Simulation Implemented with the Q-Learning Extension

The `qlearningextension:state-def` command (line 4) specifies the state representation. This command takes as argument a list of agent variables (attributes) whose values characterize a state. In the cliff walking, the state is characterized by the x and y coordinates of the agent position (stored by the `xcor` and `ycor` agent attributes).

To specify the actions considered by the Q-Learning algorithm, the extension provides the `qlearningextension:actions` command (line 5). This command takes as argument a list of NetLogo procedures, each of them corresponds to an action the agent can execute. In the cliff walking simulation, `goUp`, `goDown`, `goLeft`, and `goRight` procedures are implemented by the developer to move the agent towards these directions.

[2] We refer the reader to Kons and Santos [8] for the complete source code of the *cliff walking* simulation.

To specify the reward received by the agent whenever it acts, the extension provides the `qlearningextension:reward` command (line 6). The developer only needs to inform the procedure that computes and returns the reward value. To specify the learning rate and the discount factor, the extension provides the commands `qlearningextension:learning-rate` (line 7) and `qlearningextension:discount-factor` (line 8), respectively.

The extension provides two action selection policies: *random-normal* and ϵ-*greedy*. Both select an action at random with a given probability. However, in the ϵ-*greedy* policy such a probability is periodically reduced by a factor, as described earlier in Section 2.2. To specify the action selection policy, the extension provides the command `qlearningextension:action-selection`, that takes as argument the name of the policy and a list with its parameters (line 9).

The `go` procedure (lines 13–18) is where the behavior of the *Walker* agents is implemented. The extension provides two commands to activate the Q-Learning algorithm. The `qlearningextension:act` command (line 15) makes the agent choose and run an action according to the selection policy. The `qlearningextension:learn` command (line 16) makes the agent observe the new state and reward received, and update the *Q-table*, as described previously in Section 2.2.

As it can be seen, by using the commands provided by extension a developer does not need to implement the Q-Learning algorithm from scratch. However, it is worth to mention that there is no evaluation of whether this extension simplifies the development of agent-based simulations. We conducted such evaluation in the present paper.

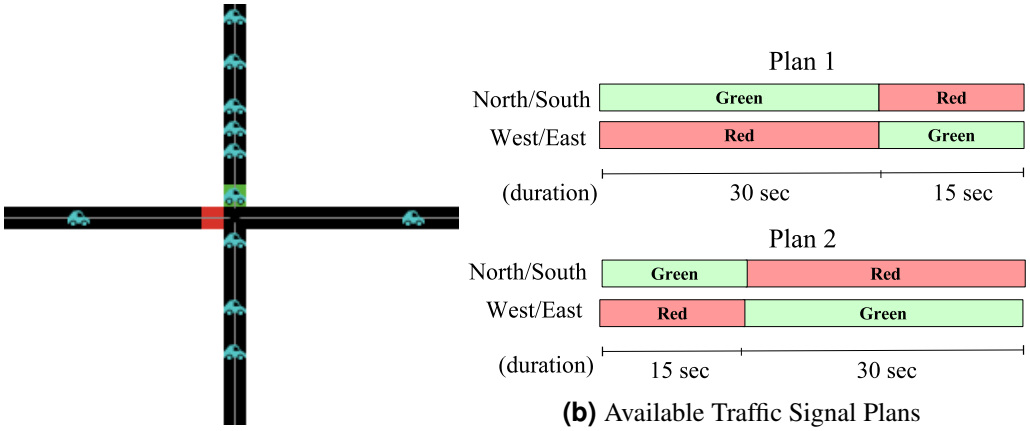
3. Quantitative Evaluation

The goal of the evaluation is to investigate whether the existing Q-Learning NetLogo extension simplifies the development of simulations. The evaluation considered two simulations: the *cliff walking* simulation described in Section 2.3, and an adaptive traffic signal control (ATSC) simulation. The latter was chosen because ATSC is a real-world problem in which successful applications of intelligent agents have been reported [2, 3]. This section describes the ATSC simulation, the evaluation procedure, and obtained results.

3.1. The ATSC Simulation with Q-Learning

In the area of traffic signal control, the goal is to develop traffic control systems that (i) maximize the overall capacity of the traffic network; (ii) maximize capacity of critical routes and intersection that represent bottlenecks; (iii) minimize negative impacts of traffic on the environment and energy consumption; (iv) minimize travel times; and (v) increase traffic safety [2]. In such systems, traffic signal devices (e.g., traffic lights) are used to control the traffic flow. In scenarios with complex traffic demands, traffic control systems should be able to *adapt* their policies to the current traffic conditions. Agent-based systems is an alternative that has been considered for creating these ATSC systems. By being endowed with learning capabilities, agents can learn traffic control policies in real time and thus optimize the overall traffic flow considering the existing infrastructure [10].

The environment of an ATSC simulation is a traffic network, which is composed of links and nodes that represent road lanes and intersections, respectively. In this paper, we adopted the model proposed by Oliveira and Bazzan [12], which specifies a traffic signal



(a) Traffic Network

(b) Available Traffic Signal Plans

Figure 4. Elements of the ATSC Simulation

controller (TSC) agent in charge of managing traffic light indicators. TSC agents are created at intersection and perceive the queue length on incoming lanes. The design of a TSC agent involves a set of concepts from the traffic control domain, which are described next. A *stage* describes a particular set of allowed traffic movements for vehicles in the lanes of the intersection. A *phase* is a period in which the indicators of the corresponding stage are green, allowing the traffic flow. Finally, a *plan* is a set of phases assigned to stages plus the sequence in which they are activated.

Figure 4a shows the traffic network considered in the evaluation. The network is composed of two one-way lanes. Vehicles on the vertical lane move from north to south, while on the horizontal lane they move from west to east. Each lane is 32 units long, which means that there can be at most 16 vehicles stopped on each direction at the intersection, waiting for the green light to cross it. The simplicity of this network allows focusing on the use of the Q-Learning algorithm, which is the goal of our evaluation. Figure 4b shows the two plans adopted in our simulation. Each plan has a total duration of 45 seconds. Plan 1 allows the traffic to flow north/south in the first 30 seconds, and then west/east in the last 15 seconds. In turn, plan 2 allows the traffic to flow north/south in the first 15 seconds, and then west/east in the last 30 seconds. Therefore, plan 1 gives priority to the north/south flow, and plan 2 to the west/east flow. In our simulation, a new vehicle is inserted on the vertical lane at every 4 seconds, while in the horizontal lane a new vehicle is inserted at every 15 seconds. Consequently, the traffic demand on the vertical lane is about 3.75 times higher than on the horizontal lane.

The Q-Learning specification for this ATSC simulation follows from Oliveira and Bazzan [12] and Santos et al. [14]. The goal of using Q-Learning in this simulation is to allow the TSC agent to learn which plan should be executed at every 45 seconds so as to minimizes the traffic jam (stopped vehicles) at intersections. The *state* definition is based on the queue length of the incoming lanes, and it is represented by a pair of values that corresponds to the number of stopped vehicles on each lane. Therefore, the state s at a particular instant is given by *(stopped north, stopped west)*. Each traffic signal plan is considered as an *action*, and therefore $A = \{plan1, plan2\}$. The *reward* is given by *0 - (stopped north + stopped west)*, which means that the higher the number of stopped vehicles at both lanes, the lower the reward received by the agent.

3.2. Procedure and Metrics

We adopted the following procedure to conduct the evaluation: (i) we implemented both simulations (cliff walking and ATSC) using the Q-Learning extension; and (ii) their source code was compared to other source codes implemented without the extension. The metric we selected for the comparison is the size of the simulation source code. More specifically, we use the number of lines of code (LoC), which is often used as a software size metric in cost estimation methods, such as Function Points [1] and COCOMO [4].

To assert that the implementations using the Q-Learning extension were consistent, we ran both simulations and inspected the policies learned by the agents. Regarding the Q-Learning parameters, we adopted $\alpha = 0.1$; $\gamma = 0.3$ for both simulations. For the cliff walking simulation, we adopted $random\text{-}normal = 0.7$. And for the ATSC simulation, we adopted $\epsilon = 0.7$ and $\epsilon\text{-}decay = 0.995$. The ATSC simulation results are based on 5 runs of 86400 ticks each.[3]. The cliff walking results are based on 5 runs of 350 ticks.

Table 1 shows the agent *Q-table* with the *Q-value* average for each pair *state-action* after running the cliff walking simulation. The maximum *Q-value* for each *state*, which correspond to the control policy learned by the agent, is emphasized with bold. As we can see, the *max* values are in the *state-action* pairs that form the optimal policy. For example, in the initial state *(0,0)*, the policy indicates to execute the *Up* action, so as to move towards the target without falling off the cliff. Then, the policy indicates to execute the *Right* action, to keep going towards the goal. Finally, the policy indicates to execute the *Down* action when the agent is located above the goal state.

Table 1. Q-Table of the Cliff Walking Simulation

States	**Actions**			
(x,y)	*Right*	*Down*	*Left*	*Up*
(0, 0)	-99,9999	-1,4281	-1,4282	**-1,4277**
(0, 1)	**-1,4259**	-1,4280	-1,4272	-1,4267
(0, 2)	**-1,4248**	-1,4255	-1,4254	-1,4252
(1, 1)	**-1,4207**	-99,8333	-1,4253	-1,4235
(1, 2)	**-1,4209**	-1,4217	-1,4237	-1,4214
(2, 1)	**-1,4075**	-99,5073	-1,4138	-1,4109
(2, 2)	**-1,4123**	-1,4134	-1,4174	-1,4146
(3, 1)	**-1,3727**	-96,9192	-1,3873	-1,3869
(3, 2)	**-1,3931**	-1,3941	-1,4004	-1,3991
(4, 1)	**-1,2776**	-91,2390	-1,3335	-1,3335
(4, 2)	**-1,3505**	-1,3535	-1,3697	-1,3548
(5, 1)	-1,1439	**-0,9911**	-1,1997	-1,2276
(5, 2)	-1,3006	**-1,2678**	-1,3006	-1,2976

Table 2 shows the *Q-table* for the ATSC agent. Only the states explored by the agent are shown. In this simulation, the agent learned to always choose the *Plan 1* action, given that the traffic demand in the vertical lane is higher than the horizontal lane, as previously detailed in Section 3.1.

[3]We assume 1 tick = 1 second, thus we simulate a 24h learning period for the TSC agent.

Table 2. Q-Table of the ATSC Simulation

States *stopped vehicles (north, west)*	Actions *Plan 1*	*Plan 2*	States *stopped vehicles (north, west)*	Actions *Plan 1*	*Plan 2*
(4, 2)	**-9,8587**	-13,9570	(10, 2)	**-9,6641**	-16,1368
(5, 2)	**-9,5479**	-13,5261	(11, 1)	**-2,5743**	-4,7753
(6, 2)	**-8,6826**	-10,6698	(11, 2)	**-9,8594**	-18,6116
(7, 2)	**-1,1338**	-1,7662	(12, 1)	**-2,1176**	-5,0304
(8, 2)	**-4,2193**	-7,3978	(13, 1)	**-7,9200**	-11,5943
(9, 1)	**-8,8922**	-18,0331	(14, 1)	**-13,7738**	-21,0598
(9, 2)	**-9,4045**	-15,6132	(15, 1)	**-14,2968**	-19,0880
(10, 1)	**-9,8652**	-19,4982	(16, 1)	**-15,6289**	-21,1684

Our evaluation is focused on the use of the Q-Learning algorithm. Therefore, only the LoC related to the agent's implementation were considered to compare the size of simulations source code. Blank lines, code comments, and code block delimiters were ignored. To avoid biases related to the source code indentation, we followed the conventions adopted in the simulations available in the NetLogo model library, in which each line contains a single code statement. The simulations source code is available online.[4]

3.3. Results and Discussions

Obtained results are shown in Table 3. To clarify the effects of using the Q-Learning NetLogo extension on the source code, we grouped the number of LoC by the following aspects regarding the simulation: *types and variables* (*T&V*), *setup*, and *execution*. The *T&V* aspect groups LoC for importing extensions and libraries, as well the definition of breeds and their variables. The *setup* aspect groups LoC related to the initialization of the simulation. In the ATSC simulation, such initialization consists of creating the traffic network, vehicles, and the TSC agent. In the cliff walking simulation, it consists of creating the cliff landscape and the Walker agent. Finally, the *execution* aspect groups LoC related to the behavior of agents.

Table 3. LoC of the Evaluated Simulations

Q-Learning Implementation	T&V	Setup	Execution	Total
Cliff Walking				
Original, without the extension	15	30	51	**96**
Using the extension	9	30	44	**83**
Adaptive Traffic Signal Control (ATSC)				
Original, without the extension [14]	27	23	119	**170**
Using the extension	16	18	63	**97**

[4] https://github.com/agentbasedsimulations/2021-wesaac-qlearning-netlogo-ext-evaluation-extras

With respect to *T&V*, by using the Q-Learning extension there is a reduction of 6 LoC in the cliff walking simulation (40%). In turn, the ATSC simulation using the extension required 11 (40.74%) few LoC. Without using the extension, both ATSC and cliff walking require extra LoC to specify additional agent variables and data structures to store all Q-Learning elements. These additional variables and data structures are encapsulated by the extension, so the developer does not need to declare them.

For the *setup* aspect, the size of the cliff walking source code is the same in both simulations. Regarding the ATSC simulation, the number of LoC was reduced by 21.74% (5 LoC). Such a reduction is because without the extension, additional LoC are required to initialize the Q-Learning algorithm, more specifically the *Q-table* data structure.

Concerning the *execution* aspect, the use of the extension produced source codes with fewer LoC in both simulations: 47.06% fewer in the ATSC simulation (73 LoC), and 13,72% fewer in the cliff walking (7 LoC). These reductions are due to the fact that by using the extension, the developer does not need to implement the Q-Learning algorithm previously described in Section 2.2. Instead, the developer only needs to invoke the *learn* and *act* commands, or the *learning* command, provided by the extension.

Overall, the agents source code developed using the extension was 13,54% (cliff walking) and 42.94% (ATSC) smaller than those developed without using it, which shows that the existing Q-Learning extension simplifies the development of simulations. Although our evaluation considered only these two simulations, which may raise questions regarding the generalization of the results for other domains, it is important to notice that the reduction in the number of LoC is due to the use of the commands provided by the extension. These Q-learning related commands are domain independent, which suggests that other simulations would also benefit from using the extension.

4. Conclusion

The agent-based modeling and simulation paradigm has been used to model and run simulations focused on the behavior of individuals and on the complexity that emerge from them. Agent-based simulations are usually developed in agent-based simulation platforms, as they provide features inherent to agents. NetLogo is a popular agent-based simulation platform, to which a reinforcement learning extension was recently made available. The extension provides commands for using the Q-Learning reinforcement learning algorithm. However, no evaluation on whether such extension simplifies the development of agent-based simulations was available yet.

In this paper, we conducted a quantitative evaluation to investigate whether the existing NetLogo Q-Learning extension simplifies the development of simulations. The evaluation considered two simulations: the classic cliff walking, and an adaptive traffic control simulation. Both simulation were implemented using the extension, and their source code were compared to existing implementations considering *lines of code* software size metric. The results showed that by using the extension the number of lines of code was reduced by 13.72% in the cliff walking simulation, and by 47.06% in the adaptive traffic signal control simulation. This gives evidence that the existing Q-Learning extension simplifies the development of simulations with NetLogo.

As future work, additional studies can be conducted with other simulations, to verify how our findings generalize to other agent-based simulation domains. Furthermore,

runtime might be a bottleneck in reinforcement learning applications, specifically when several agents learn simultaneously. Future studies should be conducted to investigate the scalability of the Q-Learning extension, given that execution time and memory consumption are out of the scope of this paper. Another possible work is to conduct an user study with humans, to evaluate if using the Q-Learning extension would also reduces the cognitive and development effort.

References

[1] Allan J Albrecht. Measuring application development productivity. In *Proceedings of the joint SHARE/GUIDE/IBM application development symposium*, volume 10, pages 83–92, 1979.

[2] Ana L. C. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multiagent Systems*, 18(3): 342–375, June 2009.

[3] Ana L. C. Bazzan and Franziska Klügl. A review on agent-based technology for traffic and transportation. *The Knowledge Engineering Review*, FirstView:1–29, 4 2013.

[4] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1(1):57–94, 1995.

[5] CoMSES. CoMSES Catalog, 2020. URL `https://catalog.comses.net/visualization/`. `https://catalog.comses.net/visualization/`, Acesso em: Jul/2020.

[6] Franziska Klügl and Ana L. C. Bazzan. Agent-based modeling and simulation. *AI Magazine*, 33(3):29–40, 2012.

[7] Kevin Kons. *Biblioteca Q-Learning para desenvolvimento de simulações com agentes na plataforma NetLogo*. Trabalho de conclusão de curso, Universidade do Estado de Santa Catarina (UDESC), 2019.

[8] Kevin Kons and Fernando Santos. Cliff walking with q-learning netlogo extension. CoMSES Computational Model Library, 2019. Retrieved from: `https://www.comses.net/codebases/b938a820-f209-4648-afc6-0946657c3484/releases/1.0.0/`.

[9] Charles Macal and Michael North. Introductory tutorial: Agent-based modeling and simulation. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 6–20, Piscataway, NJ, USA, 2014. IEEE Press.

[10] Patrick Mannion, Jim Duggan, and Enda Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In Leo Thomas McCluskey, Apostolos Kotsialos, P. Jörg Müller, Franziska Klügl, Omer Rana, and René Schumann, editors, *Autonomic Road Transport Support Systems*, pages 47–66. Springer, 2016.

[11] S. T. Monteiro and C. H. C. Ribeiro. Desempenho de algoritmos de aprendizagem por reforço sob condições de ambiguidade sensorial em robótica móvel. *Revista Controle & Automação*, 15(3):320–338, 2004.

[12] D. de. Oliveira and A. L. C. Bazzan. Multiagent learning on traffic lights control: effects of using shared information. *IGI Global*, pages 307–321, 2009.

[13] Stuart Russel and Peter Norvig. *Inteligência Artificial*. Rio de Janeiro: Campus, 2 edition, 2004.

[14] Fernando Santos, Ingrid Nunes, and Ana L. C. Bazzan. Model-driven agent-based simulation development: a modeling language and empirical evaluation in the adaptive traffic signal control domain. *Simulation Modelling Practice and Theory*, 83: 162–187, April 2018.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2 edition, 2018.

[16] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine learning*, 33 (3–4):279–292, 1992.

[17] Uri Wilensky. NetLogo, 1999. URL `http://ccl.northwestern.edu/netlogo/`. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[18] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.