Modular Reinforcement Learning Framework for Learners and Educators

Rachael Versaw
The Behrend College, The Pennsylvania State University
Erie, Pennsylvania, USA
rlv5088@psu.edu

Samantha Schultz
The Behrend College, The Pennsylvania State University
Erie, Pennsylvania, USA
sls6571@psu.edu

Kevin Lu
The Behrend College, The Pennsylvania State University
Erie, Pennsylvania, USA
kkl5245@psu.edu

Richard Zhao
University of Calgary
Calgary, Alberta, Canada
richard.zhao1@ucalgary.ca

ABSTRACT

Reinforcement learning algorithms have been applied to many research areas in gaming and non-gaming applications. However, in gaming artificial intelligence (AI), existing reinforcement learning tools are aimed at experienced developers and not readily accessible to learners. The unpredictable nature of online learning is also a barrier for casual users. This paper proposes the EasyGameRL framework, a novel approach to the education of reinforcement learning in games using modular visual design patterns. The EasyGameRL framework and its software implementation in Unreal Engine are modular, reusable, and applicable to multiple game scenarios. The pattern-based approach allows users to effectively utilize reinforcement learning in their games and visualize the components of the process. This would be helpful to AI learners, educators, designers and casual users alike.

CCS CONCEPTS

• **Applied computing** → **Computer games**; **Interactive learning environments**; • **Computing methodologies** → **Reinforcement learning**; • **Human-centered computing** → *Visualization systems and tools*.

KEYWORDS

reinforcement learning, education, computer game, visual design, design pattern

ACM Reference Format:
Rachael Versaw, Samantha Schultz, Kevin Lu, and Richard Zhao. 2021. Modular Reinforcement Learning Framework for Learners and Educators. In *The 16th International Conference on the Foundations of Digital Games (FDG) 2021 (FDG'21), August 3–6, 2021, Montreal, QC, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3472538.3472583

1 INTRODUCTION

With better hardware and the proliferation of game development software such as game engines, more people are interested in creating video games. Researchers and professionals have developed lots of tools to support game development in different ways. However, support tools for teaching the use of reinforcement learning in games are lacking. Reinforcement learning allows an agent in game to learn what to do without being explicitly scripted [4]. Deploying reinforcement learning in games can have many advantages. However, creating learning algorithms from scratch for specific uses every time is not easy, especially for non-programmers, learners and casual users. Learners and designers may not have the necessary technical knowledge to effectively implement the algorithm that is best suited for their situation [25]. Therefore, the invention of a module-based visual reinforcement learning tool allows learners and game designers to have more options for the deployment of AI. It allows casual users without in-depth AI programming knowledge to learn and utilize reinforcement learning in their games.

Our objective is to create a framework for the universal abstraction of reinforcement learning algorithms and its implemented tool, which we call EasyGameRL, that can work for various genres of video games. This would allow a user to link our tool to their game, which would add reinforcement learning functionality to the game. The popularity of the Unreal engine is aided by its support of both C++ and Blueprints visual scripting system. Visual scripting provides a short learning curve for hobbyists to quickly get on board. We demonstrate in this paper that by using EasyGameRL, it becomes easy to add reinforcement learning into a game, have actions outputted to game so that the actions can be performed by any agent in game. The learning happens seamlessly in the background until the desired result. The tool is based on visual design patterns. It is modular and expandable, allowing a programmer to add new algorithms and increase the pattern library. Furthermore, the visual representation of the tool allows for better understanding of the AI algorithms.

2 BACKGROUND

2.1 Related Work

As game development becomes more accessible and popular among hobbyists who are not trained as computer programmers, researchers have been investigating methods to better deploy AI in games for this audience. Researchers have for a long time examined design patterns in various aspects of game design such as levels [8, 14] or AI [24]. Each design pattern describes a family of known solutions to a common problem [13]. Utilizing common design patterns, researchers have created tools such as ScriptEase [3] that allows a game designer to script complex game scenes by choosing and adapting design patterns. These tools would automatically generate the programming code for the games. Using a similar idea, Van Rozen [26] presented a tool that used a graphical interface to aid with the game mechanics design process. In the education of game design and development, researchers have long noted the effectiveness of visual elements in scripting [18].

Among AI techniques, reinforcement learning has become a huge topic of recent gaming research [30]. AlphaGo and its successors [20] have achieved worldwide fame. Glavin and Madden [7] used the SARSA(λ) algorithm in a first-person shooter, aiming to improve bots' performance compared to fixed-strategy opponent bots. Pinto and Coutinho [17] designed and evaluated a fighting game combining reinforcement learning with Monte Carlo tree search. Florensa et al. [5] used reinforcement learning to tackle the circumstance where there was need for an agent to perform multiple tasks. In their paradigm, tasks were generated automatically, and the reinforcement learning agent automatically learned and performed those tasks. Yeh et al. [31] proposed and proved that a multistage temporal difference learning method could effectively improve the performance for 2048-like games. Adaptive learning rates had been explored by researchers [1]. Emigh et al. [4] showed that using nearest neighbor states to bootstrap value function updates could speed up convergence. Deep reinforcement learning combines reinforcement learning with deep neural networks, utilizing the strength of deep neural network to address issues such as large state spaces. Torrado et al. [23] tested the performance of several deep reinforcement learning algorithms on General Video Game AI games, by connecting the framework to the OpenAI Gym environment. Drastic performance differences were found among the learning algorithms between games. StarCraft and its sequel are popular platforms for deep reinforcement learning. Lee et al. [10] created competitive modular agents by implementing a novel modular architecture. Each module in the architecture had its own responsibility and could be optimized independently or jointly. Xu et al. [29] developed a competitive StarCraft bot by proposing a new deep reinforcement learning framework to improve the macro action selection performance. Frazier and Riedl [6] tested if interactive machine learning could help deep Q networks reduce perceptual aliasing in Minecraft by leveraging the knowledge of a human teacher. Results showed that their Learning from Advice agent could converge on a robust policy even in an environment with aliasing. Bontrager et al. [2] studied pitfalls in deep reinforcement learning.

While the video game research community has embraced the use of AI, the use of reinforcement learning in the game development community, especially among learners and casual users, is sparse. The ML-Agents toolkit [9] for the Unity game engine provides game programmers with the opportunity to develop their own AI using machine learning. While it has been used extensively by game researchers (Stephens and Exton [21], among many others), its primary usage still requires extensive programming and technical background. The research in this paper aims to address this challenge by providing a framework for reinforcement learning AI that is accessible to learners and casual users, and friendly to AI educators.

In a survey of current practice and teaching of AI [28], researchers and educators found that machine learning is one of the most desired and recommended topic, and games kept students' interest. Machine learning topics are complex and AI educators have used creative methods to help students learn, such as using robots [12]. Vlist et al. [25] found that it was important to provide design students with a tangible tool to interact with a learning system. MacCormick and Zaman [11] also tested the use of parallel ideas using visual programming. Algorithm visualization is a powerful aid in learning [16]. In this research, we provide a visual tool built using our proposed framework so that learners and AI educators can directly visualize the AI algorithmic process.

2.2 Reinforcement Learning Preliminaries

Reinforcement learning is an area of machine learning where the agent is not told what to do and has to discover the appropriate actions to maximize a notion of a reward. In a reinforcement learning environment, there are:

- a set of world states s ∈ S
- a set of actions: a ∈ A
- an unknown reward function R(s,a) → r that outputs a reward
- an unknown state transition function T(s,a) → s' that takes a state into the next state

The goal of reinforcement learning is to discover a policy, which is a mapping π from states to actions that maximizes expected future reward: π(s) → a.

A commonly used reinforcement learning algorithm is Q-Learning [27]. It is an off-policy reinforcement learning algorithm where the agent keeps state-action values Q(s,a) and uses these values to choose the best action to take in each state. The Q(s,a) values are updated through a trial-and-error process using an update formula. Often the state space is large and explicitly storing Q(s,a) in tabular form is not feasible. Therefore, approximation methods, ranging from linear combinations to deep convolutional neural networks have been used to approximate Q(s,a).

3 MODULE-BASED FRAMEWORK

Design patterns have been used for the purpose of video game scripting. By adapting design patterns for a specific situation, they can be used to effectively script complex game encounters, such as the quests commonly found in role-playing video games.

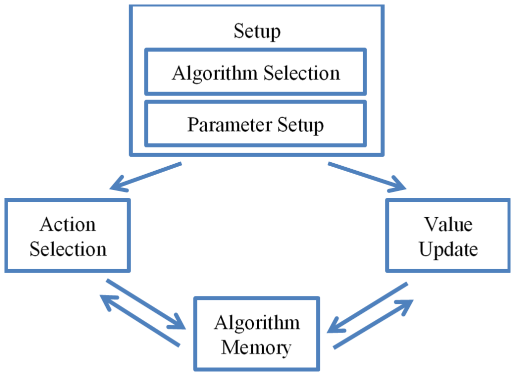
We propose a visual design pattern-based approach specifically to apply reinforcement learning in video games. In this framework, we treat each component of a common reinforcement learning algorithm as a modular pattern and provide a library of such modules for users. Current reinforcement learning approaches range from Q-Learning with linear function approximation to parallel deep Q network methods such as A3C [15]. We divide a reinforcement learning algorithm into three components: Setup, Action Selection, and Value Update. Each component consists of modules from a library that a user can choose and combine. Figure 1(a) shows the architecture of the EasyGameRL framework. In the Setup component, algorithm selection and hyperparameter setup are done. Depending on the algorithm selected, hyperparameters, such as the learning rate, may be available as modules where the user can specify their initial values and behaviors. Any hyperparameter not initialized by the user will be assigned default values. The selections done in Setup directly affects how Action Selection is done, and how Value Update is done. These two components are automatically adjusted by the tool after the user creates the desired setup. Action Selection produces an action according to the chosen algorithm and sends the action to the game to be performed. Value Update processes the changes in Q(s,a) values or the corresponding values used for approximation, depending on the algorithm. Action Selection and Value Update never directly interact with each other. They always pass information through Algorithm Memory, which is the part that is hidden from users but used by the framework to store information, both online and offline.

Figure 1(b) uses Q-Learning as an example and shows how the three components form the entire algorithm. Three hyperparameters are commonly used: learning rate α, discount factor γ, and exploration rate ϵ which is used by policy π. By dividing a reinforcement learning algorithm into modular components, each part can be generalized as a design pattern, and they can be adapted to different circumstances. Users can put together an algorithm by picking and choosing from different modules as they wish.

4 IMPLEMENTATION AND DISCUSSIONS

The implementation of the EasyGameRL framework is in Unreal Engine 4 using visual design patterns. Unreal offers a complete suite of tools and assets with the ability for programmers to create their own features through plugins. This tool is an extension to Blueprints scripting of the Unreal engine. By targeting the Unreal engine platform, our tool is readily available to a huge existing user base, and that supports our goal of helping game learners and educators around the world. The algorithms were implemented in C++ to ensure performance, but the user interface was integrated with Unreal Blueprints visual scripting. A learner can use the tool and visualize the structure of an AI in their game without writing any C++ code, and the tool automatically calls or uses C++ libraries in the backend.

In this visual implementation of EasyGameRL, modules are presented as a library of visual nodes. We use Q-Learning as an example. For the Setup component, a user needs to specify the number of actions available to the hostile creature. Let us say there are four actions. The user specifies this by selecting the "Set Number of Actions" node (Figure 2) in the library. Similarly, the user can specify that linear function approximation should be used with three features. The user can stop here and select the "Initialize Q-Learning" node, which would set the algorithm to Q-Learning with linear function approximation, and set all hyperparameter values (learning rate, exploration rate, and discount factor, etc.) to their defaults. However, the user can manually pick and choose their nodes, set these values, and specify their behavior such as how fast they should decay. Other algorithms, such as SARSA [19], or functionalities such as eligibility traces [22], can be chosen similarly in the Setup component. These nodes would insert code into the



(a)

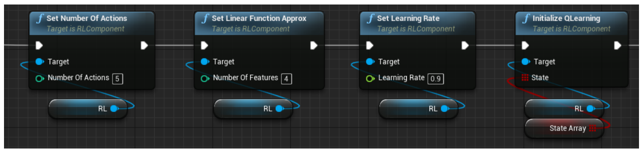
```
// **Setup**
For each s, a, initialize Q(s,a)
Observe current state s

Repeat (until convergence):
    // **Action Selection**
    Select action a based on policy $\pi$
    Perform action a
    // **Value Update**
    Receive reward r
    Observe new state s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha \ ( r + \gamma \max_{a'} Q(s', a') - Q(s,a) )$
    $s \leftarrow s'$
```

(b)

Figure 1: (a) The architecture of the EasyGameRL framework. (b) The Q-Learning algorithm represented as the three components.

existing code of the game on the AI agent that would be utilizing this reinforcement learning algorithm.


Set Number Of Actions
Target is RLComponent
Target
Number Of Actions 5
RL

Set Linear Function Approx
Target is RLComponent
Target
Number Of Features 4
RL

Set Learning Rate
Target is RLComponent
Target
Learning Rate 0.9
RL

Initialize QLearning
Target is RLComponent
Target
State
RL
State Array

Figure 2: An example Setup component in the Unreal EasyGameRL implementation, showing a visual flowchart of hyperparameter setup and algorithm selection steps.

In the Action Selection component, the user needs to add the node "RL Select Action," and for each possible return value, map it to an actual action in game for the AI agent to perform. Assuming the game has been developed to allow these actions, mapping an integer to each action is straightforward (Figure 3).

Finally, for Value Update, EasyGameRL automatically sets the correct algorithm when the user chooses the "RL Update" node (Figure 4). Here the Reward is a variable defined by the user. The

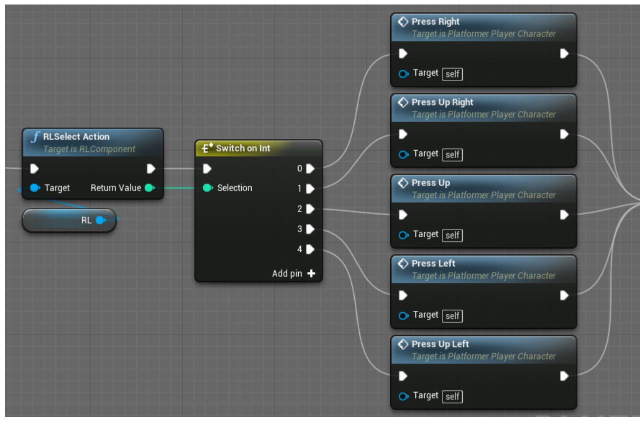


Figure 3: An example Action Selection component.

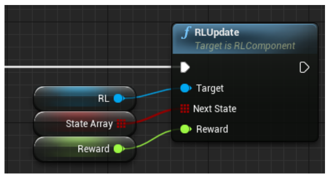


Figure 4: An example Value Update component.

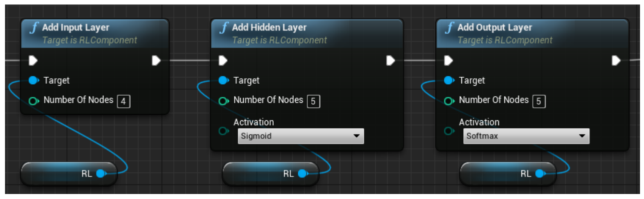


Figure 5: An example showing layers and their hyperparameters in a DQN created in Setup.

State Array is an array of features also defined by the user for linear function approximation. Disabling this single node stops further learning but keeps the learned values and the game logic intact. This is useful when using or testing pre-trained parameters.

Linear function approximations on Q(s,a) generally works well on small state spaces. When state space gets large, such as when raw pixels of the screen are used to form states, Deep Q Network (DQN) is often deployed. To use a DQN, a user can use the "Add Hidden Layer" node to add a hidden layer to the neural network with the specified number of hidden nodes and activation function. Figure 5 shows an example with input, hidden, and output layers. The "Initialize Q-Learning" node is replaced by "Initialize DQN" and the other components are used in the same way. The visual representation provides a clear explanation of the structure of a deep neural network and the flow of the algorithms.

The system saves learned values to a file and can use these the next time a game starts. If a user is concerned about the unpredictability of applying reinforcement learning online, pre-trained parameters can be used.

EasyGameRL also provides the functionality to help learners visualize the effects of hyperparameters for tuning purposes. The system can track the performance of different values of a hyperparameter and let the learner visualize how the hyperparameter affected learning.

Figure 6 shows an example using a 2D side-scroller game with platforms, enemies, and traps. The goal is for the player character to get to the end of the level on the right end. Reward is given based on how far to the right the character has moved and for staying alive. We use EasyGameRL to let the AI take control of the character to learn how to play this game. A user can set up the SARSA algorithm using EasyGameRL by dragging in and connecting 12 nodes (4 in Setup, 7 in Action Selection, and 1 in Value Update). By comparison, when a programmer coded the SARSA algorithm, the same result required approximately 400 lines of C++ code, a much larger overhead.

Figure 7 shows an example of hyperparameter visualization using the same game. It shows the performances of the AI based on cumulative reward using two values of the hyperparameter, $\alpha = 0.9$ and $\alpha = 0.5$. No α-decay was set. The traces were averaged over ten episodes of 500 discrete steps each. From this graph we can see that the large constant α value led to fluctuating results that had not converged while the smaller α value led to convergence and more consistent rewards after 300 steps. This serves an educational value because a learner can explore and experiment with the hyperparameters.

Figure 6: Example 2D side-scroller game showing platforms, enemies, and a trap. Avoiding enemies and traps is essential.

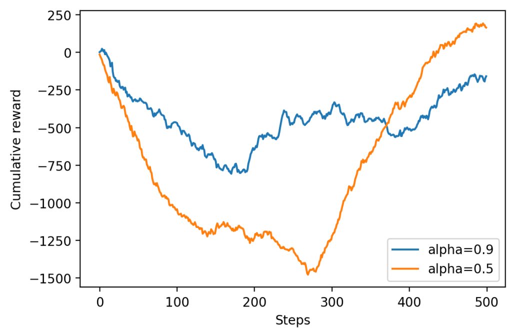


Figure 7: A cumulative reward visualization showing the results of learning based on two different learning rates.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented a framework and its implementation for efficiently adding a reinforcement learning algorithm to an existing video game. It is aimed at AI learners, educators, designers and casual users. We presented examples to show how the framework handled different reinforcement learning components. There are a lot of potential for the future expansion of EasyGameRL. The framework and tool are designed to be modular. Adding a new reinforcement learning algorithm to the tool takes incremental and relatively small effort on the part of a C++ programmer, and the resulting functionality can be used by any learners and designers who may not have or need a deep understanding of a particular reinforcement learning algorithm. It allows for reinforcement learning to be introduced in game development curriculums. By choosing and swapping modules, AI educators can quickly create different reinforcement learning prototypes and demonstrate to students the effects of each small change on the result of the AI process. For learners, this allows for hands-on experimentation of reinforcement learning in their learning process.

For future work, we will conduct user studies on the tool to assess its usability among different user groups. Qualitative analysis will be performed based on user surveys and quantitative analysis will be performed to analyze the effectiveness of learning and the efficiency of use.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Patrick Byrnes for his suggestions and the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Samineh Bagheri, Markus Thill, Patrick Koch, and Wolfgang Konen. 2014. Online adaptable learning rates for the game Connect-4. *IEEE Transactions on Computational Intelligence and AI in Games* 8, 1 (2014), 33–42.

[2] Philip Bontrager, Ahmed Khalifa, Damien Anderson, Matthew Stephenson, Christoph Salge, and Julian Togelius. 2019. " Superstition" in the Network: Deep Reinforcement Learning Plays Deceptive Games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 15. 10–16.

[3] Maria Cutumisu, Duane Szafron, Jonathan Schaeffer, Matthew McNaughton, Thomas Roy, Curtis Onuczko, and Mike Carbonaro. 2006. Generating ambient behaviors in computer role-playing games. *IEEE Intelligent Systems* 21, 5 (2006), 19–27.

[4] Matthew S Emigh, Evan G Kriminger, Austin J Brockmeier, José C Príncipe, and Panos M Pardalos. 2014. Reinforcement learning in video games using nearest neighbor interpolation and metric learning. *IEEE Transactions on Computational Intelligence and AI in Games* 8, 1 (2014), 56–66.

[5] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. 2018. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*. PMLR, 1515–1528.

[6] Spencer Frazier and Mark Riedl. 2019. Improving deep reinforcement learning in minecraft with action advice. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 15. 146–152.

[7] Frank G Glavin and Michael G Madden. 2014. Adaptive shooting for bots in first person shooter games using reinforcement learning. *IEEE Transactions on Computational Intelligence and AI in Games* 7, 2 (2014), 180–192.

[8] Kenneth Hullett and Jim Whitehead. 2010. Design patterns in FPS levels. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG)*. 78–85.

[9] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. 2018. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627* (2018).

[10] Dennis Lee, Haoran Tang, Jeffrey Zhang, Huazhe Xu, Trevor Darrell, and Pieter Abbeel. 2018. Modular architecture for starcraft ii with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 14.

[11] Daniel MacCormick and Loutfouz Zaman. 2019. SuBViS: the use of subjunctive visual programming environments for exploring alternatives in game development. In *Proceedings of the 14th International Conference on the Foundations of Digital Games (FDG)*. 1–11.

[12] Ángel Martínez-Tenor, Ana Cruz-Martín, and Juan-Antonio Fernández-Madrigal. 2019. Teaching machine learning in robotics interactively: the case of reinforcement learning with Lego® Mindstorms. *Interactive Learning Environments* 27, 3 (2019), 293–306.

[13] Matthew McNaughton, Maria Cutumisu, Duane Szafron, Jonathan Schaeffer, James Redford, and Dominique Parker. 2004. ScriptEase: Generative design patterns for computer role-playing games. In *Proceedings of the 19th International Conference on Automated Software Engineering, 2004*. IEEE, 88–99.

[14] David Milam and Magy Seif El Nasr. 2010. Design patterns to guide player movement in 3D games. In *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games*. 37–42.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[16] Thomas L Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, et al. 2002. Exploring the role of visualization and engagement in computer science education. In *Working group reports from ITiCSE on Innovation and technology in computer science education*. 131–152.

[17] Ivan Pereira Pinto and Luciano Reis Coutinho. 2018. Hierarchical reinforcement learning with monte carlo tree search in computer fighting game. *IEEE transactions on games* 11, 3 (2018), 290–295.

[18] Judy Robertson and Cathrin Howells. 2008. Computer game design: Opportunities for successful learning. *Computers & Education* 50, 2 (2008), 559–578.

[19] Gavin A Rummery and Mahesan Niranjan. 1994. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK.

[20] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[21] Conor Stephens and Chris Exton. 2020. Assessing Multiplayer Level Design Using Deep Learning Techniques. In *Proceedings of the International Conference on the Foundations of Digital Games (FDG)*. 1–3.

[22] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[23] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. 2018. Deep reinforcement learning for general video game ai. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 1–8.

[24] Mike Treanor, Alexander Zook, Mirjam P Eladhari, Julian Togelius, Gillian Smith, Michael Cook, Tommy Thompson, Brian Magerko, John Levine, and Adam Smith. 2015. AI-based game design patterns. In *Proceedings of the International Conference on the Foundations of Digital Games (FDG)*.

[25] Bram Van Der Vlist, Rick Van De Westelaken, Christoph Bartneck, Jun Hu, Rene Ahn, Emilia Barakova, Frank Delbressine, and Loe Feijs. 2008. Teaching machine learning to design students. In *International Conference on Technologies for E-Learning and Digital Entertainment*. Springer, 206–217.

[26] Riemer van Rozen. 2015. A Pattern-Based Game Mechanics Design Assistant. In *Proceedings of the International Conference on the Foundations of Digital Games (FDG)*.

[27] Christopher Watkins and Peter Dayan. 1992. Q-Learning. *Machine Learning* 8.3-4 (1992), 279–292. https://doi.org/10.1007/BF00992698

[28] Michael Wollowski, Robert Selkowitz, Laura Brown, Ashok Goel, George Luger, Jim Marshall, Andrew Neel, Todd Neller, and Peter Norvig. 2016. A survey of current practice and teaching of AI. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[29] Sijia Xu, Hongyu Kuang, Zhuang Zhi, Renjie Hu, Yang Liu, and Huyang Sun. 2019. Macro action selection with deep reinforcement learning in starcraft. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 15. 94–99.

[30] Georgios N Yannakakis and Julian Togelius. 2018. *Artificial intelligence and games*. Vol. 2. Springer.

[31] Kun-Hao Yeh, I-Chen Wu, Chu-Hsuan Hsueh, Chia-Chuan Chang, Chao-Chin Liang, and Han Chiang. 2016. Multistage temporal difference learning for 2048-like games. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 4 (2016), 369–380.