Modular Reinforcement Learning Framework for Learners and Educators

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

Kevin Lu  
The Behrend College, The Pennsylvania State University  
Erie, Pennsylvania, USA  
kk5245@psu.edu

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

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 \cite{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 \cite{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 SARS(A) 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 no 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 topics, 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. McCormick 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 \( s \in S \)
- a set of actions: \( a \in A \)
- an unknown reward function \( R(s,a) \rightarrow r \) that outputs a reward
- an unknown state transition function \( T(s,a) \rightarrow s' \) that takes a state into the next state

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

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 $\alpha$, discount factor $\gamma$, and exploration rate $\epsilon$ which is used by policy $\pi$. 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 existing code of the game on the AI agent that would be utilizing this reinforcement learning algorithm.

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
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 $\alpha$-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 $\alpha$ value led to fluctuating results that had not converged while the smaller $\alpha$ 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.

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