Development and Implementation of a Case-Based Reasoning Approach to Speed-Up Deep Reinforcement Learning through Case-Injection for AI Gameplay

Marcel Heinz[1], Jakob M. Schoenborn[1,2] and Klaus-Dieter Althoff[1,2]

[1] University of Hildesheim
Universitätsplatz 1, 31141 Hildesheim, Germany
{heinzm, schoenb}@uni-hildesheim.de
[2] German Research Center for Artificial Intelligence (DFKI)
Trippstadter Str. 122, 67663 Kaiserslautern, Germany
kalthoff@dfki.uni-kl.de

Abstract. Game environments offer properties that are useful for researching challenges in artificial intelligence (AI). Gaming enables testing, evaluation, and preparation of new methods for real-world scenarios. Reinforcement learning (RL) has undergone enormous further development in the recent years. The usage of artificial neural networks makes it possible to use reinforcement learning algorithms in complex environments. To learn feasible solutions, RL agents have to interact with the environment and learn based on their experience. Many scenarios require long training times and a vast amount of training data. Reusing previously experience knowledge can be the key to shortened training cycles and improved performance. Case-based reasoning (CBR) is another methodology of artificial intelligence using experiences from previous situations for solving new situations by adapting known solutions. Therefore, CBR appears to be particularly suitable for knowledge transfer in the area of reinforcement learning and is applied to improve the learning process of RL agents within video games. First, this work develops a theoretical approach in order to show in a second step the practical feasibility with the help of a prototypical implementation. The evaluation of the proposed method confirms reduced training time and improved performance.

Keywords: Case-Based Reasoning, Case-Injection, Reinforcement Learning, Transfer Learning, Gaming

1 Introduction

With the development of autonomous vehicles [8], automated diagnoses of diseases [7], and rapidly evolving language assistants, the subject of AI continues to appear in everyday life. For companies, research institutions or countries, AI has become a panacea for many areas and promises solutions to urgent questions of the future. To solve these problems, a variety of different approaches have been steadily researched, tested, and evaluated with techniques such as RL, (Deep) Neural Networks (DNN) and CBR among a multitude of other approaches as well. We developed and implemented intelligent agents that are able to reuse knowledge in a proficient way to overcome some of the obstacles of RL agents, such as accelerating the learning phase. These hurdles are the limitation of the learned task and the costly process of learning. RL agents still apply their knowledge in a rather small operating area. In order to work in other games, it has to be retrained, which is cost-intensive and time-consuming. A combination of CBR, deep learning (DL) and RL together with the concepts of multi-agent systems (MAS) and transfer learning (TL) can lead to a sophisticated solution that overcomes the outlined difficulties.

The core of this work is the development of a general procedure based on the CBR methodology for knowledge transfer within a DRL environment. The underlying work examines the structure of a systematic transfer learning approach, in which, as far as possible, all process steps run automatically. In order to build the bridge from theory to practice, the proposed algorithm is prototypically implemented[3] and then evaluated.

2 Related Work

This section reviews research that is related to our new approach. Since the topics CBR and DRL are in any case of great importance in AI research, the main focus of this section is on the area of games.

Bianchi et al. examined the knowledge transfer through CBR components within RL environments [3]. The main focus of the work is a case-based policy inference algorithm. This accelerates the learning process through an intelligent selection of similar, already learned cases within the case base. The publication transfers the existing knowledge within a Q-learning environment. In contrast, this work applies knowledge transfer to a DRL agent. The proposed algorithm shows success in terms of temporal performance and more robust learning behavior. The more similar cases found, the lower ϵ can be chosen and thus influences the exploration-exploitation behavior [3]. The work uses a fixed value for ϵ. A grid-world environment evaluates the proposed algorithm, in which an agent starts at any randomly chosen point and has to navigate to a target point. Based on this situation, the similarity measure consists of the Euclidean distance from the start to the destination. In contrast, the suggested architecture of this work calculates the similarity of visual and numerical inputs. In the experiment phase, initial cases are generated, which are then reused in the later transfer of knowledge. During the learning process, the cases from the case base are evaluated and the best case is used. Bianchi et al. showed that their approach achieves

[3] Code available: https://github.com/marcel-heinz/peng

the same amount of reward as conventional Q-learning approaches in a shorter time.

In the work of J. Hillman [4], RL improves the learning process with the help of CBR and TL in a first-person shooter scenario. Hillman points at a practical benefit for the use in real-world scenarios in which the state space grows enormously. RL is used within the source domain to generate various cases for the later target domain. The similarity measure for the retrieval is calculated from the distance of the agent to the objects within its environment. In addition, the algorithm accesses the case base within each episode after each step to find a suitable case. If the retrieval is not set optimally, this can lead to performance losses in terms of required time. The algorithm presented here leads to a significant improvement within the realized domain. Kolbe et al. introduce an approach that combines RL and CBR to improve the performance of an first person shooter (FPS) agent [5]. For this purpose, Kolbe et al. build on the knowledge generated by Hillmann [4] and develop a MAS inhabiting three interacting agents in a Unity 3D game environment. To ensure that case retrieval delivers useful cases, a RL agent was also implemented, which delivers the appropriate actions from the case base on the basis of stored sequences. The variable reward function, which is useful for this promising combination, adapts to the saved sequences and the overall win chance. In our approach, the retrieval process is supported by a RL component, which increased the performance in the conducted tests.

3 Past Experience Network for Gameplay (PENG)

This section lays the theoretical basis for the subsequent development of the procedure. Besides, this section presents the Past Experience Network for Gameplay (PENG) process model, which reflects the basic features of an intelligent agent architecture that reuses knowledge in the Atari2600 gaming environment. Atari is well-known for famous arcade games such as Pacman and Breakout. The PENG system builds on the basic principles of the CBR methodology and fulfills the following characteristics: Manual input for game setup, flexible application options with different domains, dealing with visual game environments, MAS approach and evaluation of trained agents to identify a suitable agent. Figure 1 shows the proposed CBR cycle of the PENG architecture.

3.1 Nomenclature

As part of this work, a model repository $\mathcal{R}$ equals the case base. Also, in the PENG process model, a case τ is divided into two parts, problem and solution. Equation 1 formalizes the model repository and equation 2 describes an individual case.

$$\mathcal{R} := \{\tau_0, \tau_1, \ldots, \tau_n\} \tag{1}$$

$$\tau := (\Lambda_\tau, \mu_\tau) \tag{2}$$

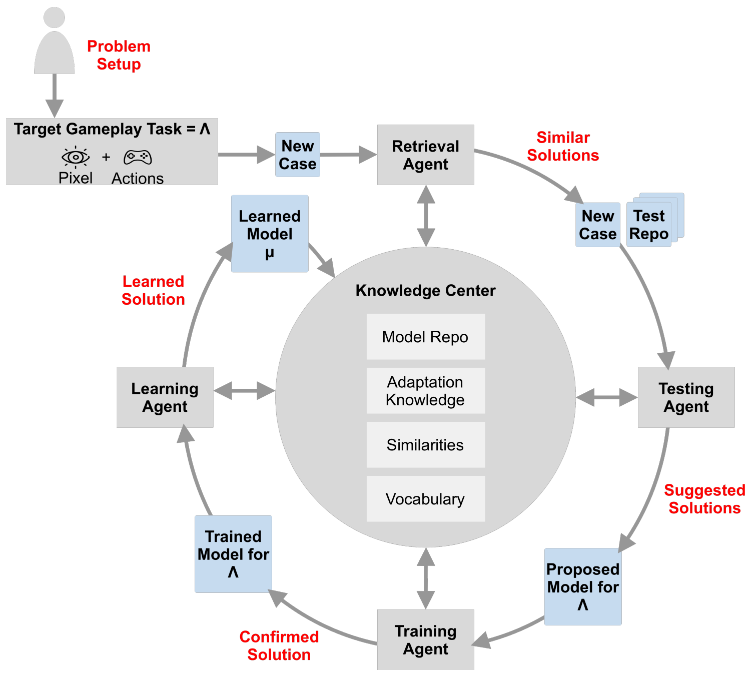

Fig. 1. Proposed CBR cycle of PENG, based on [1]

With the target gameplay task Λ and the model μ as in Equation 3 and 4

$$\Lambda_{\mathcal{T}} := (\text{Pixel Values, Number of Actions}) \tag{3}$$

$$\mu_{\mathcal{T}} := (\text{NN Architecture, Policy}) \tag{4}$$

The standard CBR methodology retains relevant knowledge within the knowledge base. Within this knowledge base are the containers for general knowledge such as adaptation knowledge or similarity measures. The PENG method stores all available knowledge inside the knowledge center (KC). Figure 2 briefly describes the general process for the PENG architecture. The entire process is described in detail in the following.

3.2 Similarity Measure

The similarity measure is an integral part of the overall PENG architecture. As described above, the PENG component accesses the gameplay and nb_action attributes. The local similarity measures should be designed in such a way that they cover the particular characteristics of the individual attributes. Therefore, the local similarity measure at the attribute level should have the following properties:

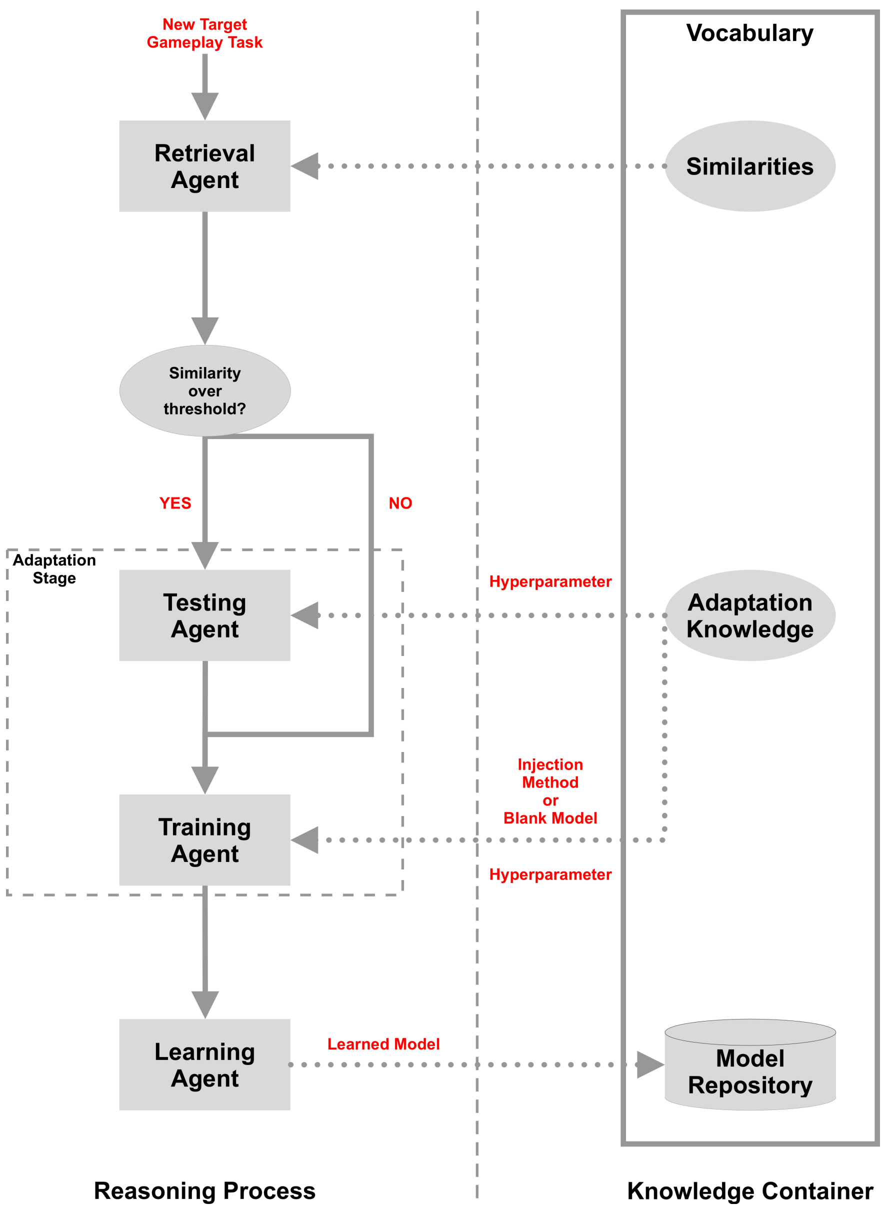

Fig. 2. PENG Process Model based on [2]

– A reliable similarity measure for integer values
– A reliable similarity measure for image data

We employ the Euclidean distance to measure the distance between the nb_action attribute. Since the proposed PENG system also works with image data, the retrieval step requires a similarity measure that handles pixel values and interprets the stored data. The image classifier applies two DNNs, which use different prediction methods. On the one hand, the prediction layer uses the softmax function. On the other hand, the last layer uses the sigmoid function. The model with the softmax activation function is stronger in classifying already known data, while the model with the sigmoid activation function has an advantage when showing the network environments that it has not seen before. The sigmoid function shows nonetheless the (estimated) probability that a new environment belongs to a trained class. Equation 5 presents the gameplay similarity, where SIP refers to 'sigmoid prediction' and SOP to 'softmax prediction'.

$$Sim_{gameplay} = w_{sigmoid}SIP + w_{softmax}SOP \tag{5}$$

Equation 6 shows the amalgamation function for the global similarity. When calculating the global similarity, the attributes receive an individualized weight w to bias the retrieval.

$$Sim_{global} = w_{nb\_action}Sim_{nb\_action} + w_{gameplay}Sim_{gameplay} \tag{6}$$

3.3 Retrieval Agent

The retrieval agent is the entry point to the PENG system, whenever the system receives a new target gameplay task Λ. The main goal of this agent is to search within the model repository for saved game environments that are similar to the query case in order to train a more powerful DRL agent. Consequently, the system utilizes problems and solutions from the past. Ideally, this agent should output a significantly reduced section of the model repositories in the form of $\mathcal{R}_{test}$. The retrieval agent's function is to select appropriate models in $\mathcal{R}_{test}$, that are as similar as possible to the current task Λ. Whenever the retrieval agent recovers a similar model from $\mathcal{R}$, the DRL agent has the opportunity to learn from past experience. Moreover, already trained and tested models are reused or recycled, which contributes to the challenges task limitation and the costly learning process. The main steps are:
1. Initialize the new environment, 2. Load the model repository and set the query case, 3. Calculate the similarity between the number of actions, 4. Calculate the similarity for gameplay images, 5. Calculate the global similarity.

3.4 Testing Agent

The testing agent receives the most similar case from the retrieval agent within the $\mathcal{R}$. The main goal of this agent is to apply the chosen model inside $\mathcal{R}_{test}$

with the corresponding architecture and policy to the query environment and to select the best performing model. In other words, the agent uses $\mathcal{R}_{test}$ to identify a model that is already performing well for the target gameplay task Λ. However, with the preselected models in $\mathcal{R}_{test}$, it is not guaranteed that the stored models μ are also usable for the current application Λ. The agents output is a suggested model for the current task Λ, that the testing agent passes on. The main steps are:
1. Define the most similar model with architecture and policies, 2. Check if the last layer of the DRL agent has to be changed, 3. Build the DRL agent, 4. Test query environment with every policy from the most similar environment, 5. Select the best performing policy.

3.5 Training Agent

The central part of the training agent is to provide a reasonable solution for the query environment. This objective can be achieved by either train the agent with the Q-injection method from scratch, when no similar game environment was found. The second way is to train the new game with the model-injection procedure, which applies a similar solution from the model repository.

Q-Injection The Q-injection method introduces an additional probability ϕ, besides the parameter ϵ that corresponds to the exploration-exploitation strategy. ϕ determines if the agent injects Q-values during the training phase. From the beginning, the Q-values are not definite, therefore random numbers between [0,1] are drawn as a first heuristic in order to inject values into the DRL agent. The range of numbers refers to the minimum and maximum values that are possible for the Q-values in this setting. This procedure takes advantage of the initial instantiated Q-values that are zero. Moreover, Q-values turn into values between [0,1] after training. Based on this, we assume that the injection of Q-values > 0 has a positive influence on learning behavior.

Model-Injection In this case, the DRL agent uses the complete model of the previously trained agent, including architecture and policy, to solve Λ. The proposed model serves as a blueprint for the new target gameplay task. Thereupon the training agent injects the experienced model, including the policy into the new agent. Once this transfer process is complete, the training phase starts. The main steps are:
1. Initialize the DRL agent, 2. Check for transfer mode (Q-injection vs. model-injection), 3. Check if the last layer of the DRL agent has to be changed, 4. Build the DRL agent, 5. Config and compile the DRL agent, 6. Train the DRL agent.

3.6 Learning Agent

The central task of the learning agent is to save the newly generated information and knowledge, that the PENG method can reuse for succeeding tasks.

The learning agent consists of two main components, first a learning component and second a maintenance part. CBR systems store and process experience knowledge, ordinarily with different approaches. The agent can build up knowledge or reject the currently learned model τ_A. Consequently, the agent decides whether she retains a model or not. The most obvious way is to compare the recently learned model with the models within the model repository. Only if the newly learned model is significantly different from the previous models within the model repository, the agents saves it. For the implementation stage of the learning agent, we used this comparison method, which learns models that are fundamentally different from the previous ones. The main steps are:
1. Clean the model repository and learn new models, 2. Collect image data of the game for NN, 3. Image augmentation, 4. Train the NN for image classification.

4 Experiment and results

The evaluation of the PENG system is essential to assess the theoretical elaboration and practical implementation more precisely. Therefore, this section describes various experiments that have been carried out in order to determine the performance of the system. The proposed framework consists of several parts, based on the CBR methodology. Nevertheless, the main focus is on the behavior achieved by the DRL agent. For this reason, the core of the experiments will rely on the results of the agent performance, supported by the PENG methodology.

4.1 Experiment Setup

The defined specifications prior to the first run:

- PENG starts with an empty model repository
- the retrieval agent starts after the system trained three games in advance
- similarity threshold of 0.4 as first-fit heuristic
- weights for the amalgamation function are set equal to 0.5 for nb_actions and 0.5 for gameplay
- parameters of the DRL agent correspond to the work of Minh et al. [6]
- a deep Q network (DQN) is used as the baseline

Table 1 shows the used hyperparameters within the DQN and the PENG experiments. The PENG system trained each game for 2,000,000 steps, and the environment was freely selected from the Atari2600 repository.

4.2 Metrics

This subsection implements some metrics that provide sufficient information about the success or failure of the proposed PENG method.

	DQN	**PENG**
Training Steps	2.000.000	2.000.000
Learning Rate	0.00025	0.00025
Gamma	0.99	0.99
Target Model Update	10.000	10.000
Warmup Steps	50.000	50.000
Epsilon Start	1.0	0.5
Epsilon End	0.1	0.1
Annealing Steps	1.000.000	1.000.000
Replay Limit	1.000.000	1.000.000
Reward	[-1.0, +1.0]	[-1.0, +1.0]

Table 1. Hyperparameters: DQN vs. PENG

Performance Increase/Decrease Algorithms that take less effort to achieve a defined goal are successful in practical implementation. Therefore, it is desirable to investigate how many steps were needed to reach the same reward level as the baseline algorithm after 2,000,000 steps and vice versa. Equation 7 and equation 8 formalizes this metric.

$$\text{PI} = 100 \cdot \frac{\text{Total Steps}_{DQN}}{Steps_{PENG|Reward_{PENG}=\text{Total Reward}_{DQN}}} \tag{7}$$

$$\text{PD} = 100 \cdot \frac{Steps_{DQN|Reward_{DQN}=\text{Total Reward}_{PENG}}}{\text{Total Steps}_{PENG}} \tag{8}$$

Episode Reward The achieved reward in the particular episode provides an insight into the learning behavior of the agent over the entire period. Equation 9 defines the Episode Reward.

$$\text{Episode Reward} = \sum_{step=0}^{\text{Episode End}} Reward_{step} \tag{9}$$

Sum Reward RL agents are conventionally measured by how beneficial a proposed policy is or how much reward an agent gets in its environment. One way to demonstrate the performance of an RL agent is to show the total of all rewards received over the entire period. Equation 10 defines the sum reward.

$$\text{Sum Reward} = \sum_{i=0}^{\text{Total Episodes}} Reward_{Episode_i} \tag{10}$$

4.3 Results

This section provides the test results of the experiments based on the implemented method. We present the experiment results of the model-injection procedure, and the results of the Q-injection method. The abbreviations from the data tables refer to the following Atari2600 games:

– B - Breakout
– MP - MsPacman
– S - Seaquest
– A - Alien

Model-Injection The results in Table 2 show that the presented method has achieved a definite increase in performance if the PENG system found a similar case within the model repository. The technique achieved the most significant performance increase of 121.54%. In this case, the target gameplay task was 'Breakout', and the retrieval agent recovered the most similar case 'Breakout' from the model repository.

Approach	**B**	**MP**	**S**	**A**
DQN Steps	2000000	2000000	2000000	2000000
PENG Steps	1645490	1924129	1941309	1821558
PI/PD	121.54%	103.94%	103.02%	109.80%

Table 2. Experiment Results for model-injection - PI/PD

Table 3 shows the measured values for the cumulative reward over the entire time. The average reward and maximum reward at the end of training are significantly higher in all gaming environments. The obtained results indicate that the search for a similar case leads to a not inconsiderable increase in the overall reward.

Approach			**B**	**MP**	**S**	**A**
		Mean	9161.17	54012.14	8011.16	33211.40
DQN	SR	Max	41640.00	132169.00	21640.00	81105.00
		Std	10959.72	39447.67	6413.29	23882.56
		Mean	17887.57	56945.64	9889.01	42653.81
PENG	SR	Max	51909.00	136346.00	22445.00	89691.00
		Std	15658.34	40482.38	6784.25	26854.50
		Mean	195.25%	105.43%	123.44%	128.43%
100% $\frac{\textbf{PENG}}{\textbf{DQN}}$	SR	Max	124.66%	103.16%	103.72%	110.57%
		Std	142.87%	102.62%	105.78%	112.44%

Table 3. Experiment Results for model-injection - Sum Reward(SR)

Q-Injection Table 4 describes whether the method is attributable to a performance increase. The data shows a comparison between the baseline method with the experiment of the Q-injection method. Since all experiments were stopped after 2,000,000 steps, the games Breakout, MsPacman, and Alien achieved fewer rewards in total within the PENG learning cycle. Based on the 2,000,000 steps, there is only a performance increase of 102.99% in the game Seaquest.

Table 5 shows the recorded data points of the accumulated rewards over the entire training period. The data shows that only the game Seaquest achieved

Approach		**B**	**MP**	**S**	**A**
DQN	Steps	1928880	1958882	2000000	1948749
PENG	Steps	2000000	2000000	1941861	2000000
PI/PD		96.44%	97.94%	102.99%	97.44%

Table 4. Experiment Results for Q-injection - PI/PD

more rewards throughout the 2,000,000 steps, increasing by 104.78%. The lowest value achieved the Breakout environment with 95.34%. Interestingly, the average for the sum of the rewards is higher in the game Alien, but the maximum result is below the baseline method.

Approach			**B**	**MP**	**S**	**A**
DQN	SR	Mean	9161.17	54012.14	8011.16	33211.40
		Max	41640.00	132169.00	21640.00	81105.00
		Std	10959.72	39446.67	6413.29	23882.56
PENG	SR	Mean	8583.87	51706.76	7430.91	35012.35
		Max	39701.00	128620.00	22675.00	79180.00
		Std	10568.42	36998.50	6351.68	23743.15
$\mathbf{100\%\frac{PENG}{DQN}}$	SR	Mean	93.70%	95.73%	92.76%	105.4%
		Max	95.34%	97.31%	104.78%	97.63%
		Std	96.43%	93.79%	99.04%	99.42%

Table 5. Experiment Results for Q-Injection - Sum Reward(SR)

4.4 Discussion

The results of this paper are two-fold concerning the DRL learning process. We showed that the model-injection method provides a distinct advantage when the model repository stores a similar case. On the opposite, the Q-injection method yields to an equal or even more miserable result, compared to the baseline method. The applied t-Test verified the findings for both methods with a high significance (p=0.01). However, our paper examined a limited representation of Atari2600 games that could affect the validity of the data. In future work, we advise conducting the experiments on a high-performance computer system, in order to obtain more results in parallel and scrutinize more games or other game environments. On a side note, the Q-injection method was instantiated with random values between [0,1]. Therefore, a better-constructed initialization technique for the corresponding Q-values may produce better results. Furthermore, we measured the achievement of the DRL agent to reason about the acceleration of the learning process. Further analyses should also focus on the required time for the complete PENG cycle since improvements regarding the learning period also correlate with the overall process. For the evaluation of the results, the algorithm operated for 2,000,000 steps. Future investigations should test a different stop criterion for assessing the procedure, such as the number of episodes or the reward to be achieved. Also, we implemented the

PENG framework prototypically with a first-fit heuristic. Consequently, we suggest that especially the testing agent and training agent evolves, in order to inject a more beneficial model with probably more robust results. Despite the mentioned obstacles, the outcomes indicate that PENG can support the DRL learning behavior within the Atari2600 gaming environment and can serve as a benchmark for further developments linked to our procedure.

5 Conclusion

We provided and evaluated the proposed PENG methodology and processed the measurement results from the experiments. The defined metrics were applied to the model-injection and Q-injection methods. The validation showed that the model-injection procedure has a clear advantage and we found indications for a lower performance of the Q-injection method. The statistical analysis shows that the model-injection methodology yields a significant improvement in training behavior, whereas the results from the Q-injection procedure can be put into perspective, since the result unveils no significant difference between DQN as baseline and PENG, except for the Breakout game. The obtained knowledge inside the KC can be further tested for other games, especially in terms of testing the process of adapting knowledge between different games. Furthermore, the parameters for both, the DRL agent and any agent involved in the PENG methodology can be further investigated.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Commun. **7**(1), 39–59 (1994)
2. Amailef, K., Lu, J.: Ontology-supported case-based reasoning approach for intelligent m-government emergency response services. Decision Support Systems **55**(1), pp. 79–97 (2013)
3. Bianchi, R.A.C., et al.: Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning. Journal of Intelligent & Robotic Systems **91**(2), pp. 301–312 (2018)
4. Hillmann, J.: Conception and development of a prototype for a multi-agent-system with learning capabilities using case-based reasoning in the first-person perspective scenario. (2017), master thesis at University of Hildesheim
5. Marcel Kolbe, Pascal Reuss, J.M.S., Althoff, K.D.: Conceptualization and implementation of a reinforcement learning approach using a case-based reasoning agent in a FPS scenario. CEUR Workshop Proceedings **2454** (2019)
6. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), pp. 529–533 (2015)
7. Newman, T.: Could artificial intelligence be the future of cancer diagnosis? (2019), https://www.medicalnewstoday.com/articles/325750.php, last validation: 02/17/2020
8. Saleem, F.: The future of self-driving cars: New generation of transportation (2018), https://innov8tiv.com/the-future-of-self-driving-cars-new-generation-of-transportation/, last validation: 02/17/2020