Conceptualization and Implementation of a Reinforcement Learning Approach Using a Case-Based Reasoning Agent in a FPS Scenario

Marcel Kolbe[1], Pascal Reuss[1,2], Jakob Michael Schoenborn[1,2], and Klaus-Dieter Althoff[1,2]

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

Abstract. This paper describes an approach that combines case-based reasoning (CBR) and reinforcement learning (RL) in the context of a first person shooter (FPS) game in the game mode deathmatch. Based on an engine written in C#, Unity and a simple rule-based agent, we propose a FPS agent who is using a combination of case-based reasoning and reinforcement learning to improve the overall performance. The reward function is based on learned sequences of performed small plans and considers the current win chance in a given situation. We describe the implementation of the reinforcement algorithm and the performed evaluation using different starting case bases.

Keywords: Case-Based Reasoning · Reinforcement Learning · First Person Shooter · Multi-Agent System · Planning

1 Introduction and motivation

To prove the functionality of an artificial intelligence, multi-agent systems became a popular application area. We take a look at the first person shooter (FPS) domain, a sub-genre of action video games. In a typical FPS game, there are two teams of typically human players trying to overcome the opposing team by either eliminating each member of the opposing team or by successfully complete another objective, such as planting a bomb at a certain place or by preventing the opposing team to do so. While both teams are actively playing at the same time, most FPS are limited by a round-time of approximately five minutes and a limited map size. Each individual FPS game differentiates itself from other games by having certain unique characteristics, for example, the extent of reality. However, a FPS game can be generalized to the following: The human player takes over the sight of an agent, as if he or she would be placed inside of the game. To emphasize this, environmental sounds, for instance, the sound of a step when moving through the terrain, are implemented to increase the immersion into the game. An agent starts with a certain level of health points (HP, usually 100) and with a basic pistol weapon equipped, having a limited amount of ammunition. The goal is to find and eliminate the opposing agent. Each round, credits are earned based on the outcome of the last round to buy better weapons or supply like armor and health packs to increase the current health points. This domain is not limited to one versus one fights but rather is usually applied to five versus five combat. This increases the complexity a lot since factors like communication and planning between the agents have to be considered: while the game is running, the current state changes each millisecond by usually not knowing the enemies position and by creating predictions of the enemies next possible steps. Since it is a very advantageous situation to see the enemy before the enemy sees you, one of the most common tactics to gain this advantage is to "camp", that is, waiting and hiding for an indefinite amount of time[3] at a point of interest until the enemy passes along. However, especially in the currently most played FPS games like Fortnite or Players Unknown Battleground, the so-called mode "Battle Royale" mode gained an increasing popularity. This mode is usually a free-for-all mode, meaning everyone fights for himself against any other agent[4]. Using this mode, after a certain amount of time, the size of the level shrinks until no place is left to hide and the last two surviving agents will have to engage each other. The longer one can avoid fights and look for equipment, the better. Thus, an AI agent would need to plan accordingly when a rather defensive behavior has to be switched to a comparably aggressive behavior. Since the spawn locations and spawn timings of these equipment differ each round, each game the player is able to experience a new situation. This is where our research adds on.

While CBR and RL were and are used in different game domains (see section 2), the use of CBR and RL in a FPS game in the game mode of deathmatch, either one-on-one or team-based is new and has different strategies, tactics and challenges than game modes like conquest or capture the flag. Another goal of or research is to apply the CBR and RL approach to FPS in the game mode deathmatch and evaluate the progress of our agents during the games.

As we will later see more in detail in section 3.1, one might think of one round of a FPS game as one case which can be stored and retrieved. We went on a more detailed level and used the current perception of the agent as situation and mapped a corresponding action as a solution to the current situation. In our first version, we used 17 attributes, such as currentAmmunition, distanceToEnemy, among many others and 15 initial cases on a FPS game designed in Unity [9]. However, the initial problem space was not large enough, leading to a unique

[3] If not limited by a certain objective, e.g., planting a bomb during a limited time frame.

[4] However, most of the games also incorporate a variant where teaming is allowed.

overall dominant tactic by always looking to pick up the weapon upgrade and therefore nullifying the benefits of CBR. While we increased the complexity of the level itself, we ask the research question whether we can improve the retrieval of the best cases by using reinforcement learning to prevent the agent from being stuck in a corner, not changing his situation and thus not retrieving another case to get himself out of the corner.

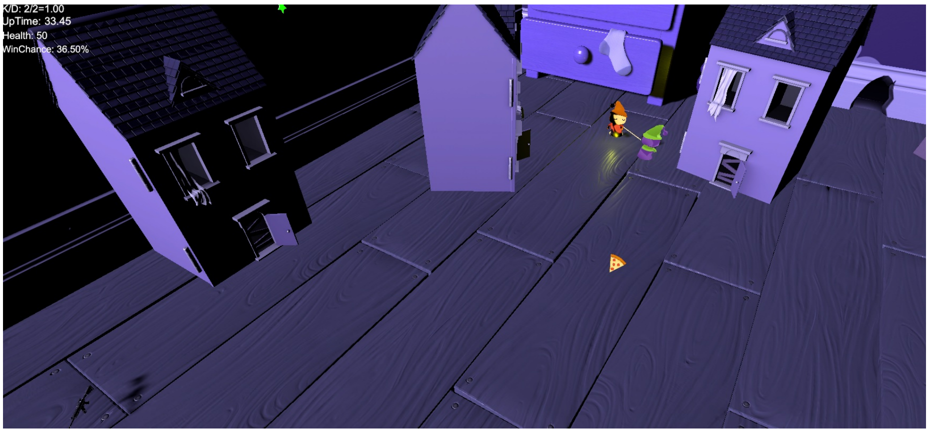


Fig. 1. CBR agent fights a rule-based AI while a piece of pizza and a better weapon (at the bottom left corner) are available. When the CBR agent considered himself as running low on hit points, he moved to collect the piece of pizza and thus restores his hit points.

For our own testing purposes, we used a FPS game developed by Jannis Hillmann as part of his master's thesis using a combination of Unity, C# and JAVA [5]. Fig. 1 shows an example of a CBR agent (purple) fighting a rule-based AI (orange). This platform in general is intended to support other domains, for example, real-time strategy games and economical simulations to name a few.

To enable another learning component besides case-based reasoning for our agent, we propose to use reinforcement learning (RL). Using the definition from Sutton, RL enables to derive actions from situations. Situations are mapped to an aim such that actions are connected with certain rewards or punishments (negative rewards) [10]. During each situation, the agent has to perceive the current state of the environment he is situated in. One of the advantages of reinforcement learning is that not only the current situation but also near-future situations can be considered. Combining the lessons learned from failed experiences and the positive experiences from properly rewarded situations leads to the most important aspect of RL and consequently to the creation of an intelligent plan during a reasonable time frame.

2 Related Work

RL has been used in gaming AI before mainly in the context of real time strategy (RTS) games. One approach was developed by Wender and Watson in StarCraft II [14]. StarCraft II as a RTS game has almost the same requirements and conditions, e. g., real-time decision making with incomplete information, as a FPS game but with more focus on macro- and micromanagement. Thus, the domains are to some extent interchangeable and approaches can be transferred by applying the necessary adjustments since the focus of the two genres are different. Using reinforcement learning in combination of case-based reasoning, we want to prove that a learning agent can always defeat a rather static rule-based agent after enough experiences have been collected and thus lead to a beneficial approach. According to Wender and Watson, the use of different forms of CBR and RL in the context of AI research is one of the most common ways of working, regardless of whether the methods work together or act separately. The two approaches support each other's respective problem areas. [14]

There are many other approaches that combine CBR and RL in RTS games. The approach of [13] combines goal-driven autonomy with RL to coordinate the learning efforts of multiple agents in a RTS games by using a shared reward function. In [11] a generalized reward function is used to apply several RL algorithms to a RTS game, while [12] propose two RL algorithms bases on neural networks to optimize the build order in StarCraft II. In the last years a new approach called Deep RL was developed and used in several research work like [1], [3], [4] and [7]. All these approaches uses CBR and RL to improve the micro and/or macro management in RTS games, but RL can also be used in turn-based games [15] and in FPS games. The work of Auslander and his colleagues [2] deals with the use of a RL logic in the context of a first person shooter Unreal Tournament, developed with the UNREAL Engine. Their work is based on the previous successes of various knowledge-based learning approaches that have relied on the mechanics and dynamics of games, e. g. Star Craft or Backgammon. Instead of learning "one-on-one" confrontation, the work describes group-oriented behaviors and strategies based on the combination of RL and the CBR approach for the game Unreal Tournament. This is a challenge for Unreal Tournament as different game modes with different goals are supported. On the one hand it can be a team-based deathmatch and on the other hand it can be a game type called conquest, in which the killing of opponents is secondary. The ultimate goal is to defend a specific point on the map. According to [2], the game mode conquest is particularly interesting for the development of cooperative strategies in the context of a team-oriented approach. Furthermore, it offers a good opportunity to observe the learning behaviour of the team.

Another work in the context of a FPS game is presented by [8] for the MineCraft game. The goal of this game is not to fight against each other like in Unreal Tournament or our approach, but to harvest resources and build structures. The approach combines hierarchical task networks and RL algorithms to adapt the plans for agents acting in the dynamic world. A deep RL approach for the FPS game Doom was developed by [6].

3 Architecture and structural dependencies

Every agent and the environment is built in Unity 3D and thus implemented as a C# project. We use three different agents: Player Agent, Planning Agent, Communication Agent. The *Player Agent* uses an inherited method called `update()` which updates the agent perception, i.e., the agent received information throughout its sensors or proposed plans of the planning agent. As the update()-method triggers multiple times per second, this value has to be evaluated considering the fairness towards a human enemy who can only evaluate a limited amount of perceptions. With each update()-cycle, the agents sends a request to the *Communication Agent*. This agent is connected with the myCBR component which uses JAVA as a programming language. Thus, a corresponding interface using the communication via TCP/IP has been established. Each time the Communication Agent receives a request, the agent formulates a request to the CBR system to retrieve the most similar case to the current situation. Once the most similar case has been retrieved, the proposed solution (which usually results in a proposed action) will be sent to the *Planning Agent*. This agent evaluates the proposed action from the Communication Agent and forms a plan, which will be sent to the Player Agent and will be followed until a new plan will be proposed. As a simple example, the Player Agent perceives that he is low on health (e.g., $< 20\,\%$ hit points (HP)), so that perception will be transferred to the Communication Agent. This agent retrieves as the most similar case, that picking up a piece of pizza leads to gain back most of the lost hit points. Thus, the planning agent formulates a plan to find and to pick up a piece of pizza. This plan will be followed, until it has been picked up, until the agent dies, or until another plan gains a higher priority (e.g., self-defence or perceiving another, better, healing item). The structure builds up as described can be seen in Fig. 2:

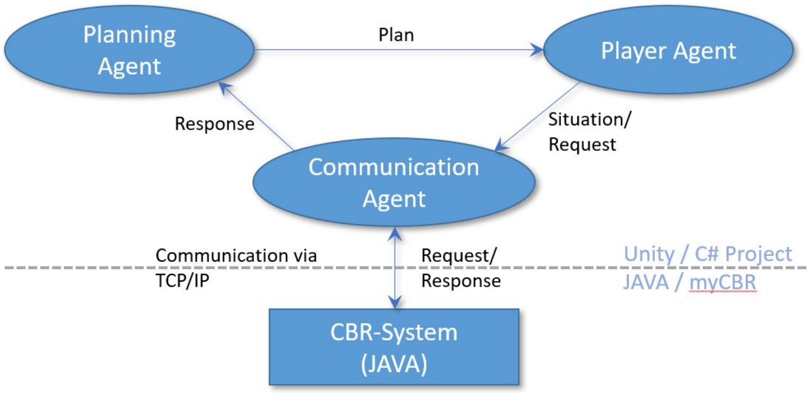


Fig. 2. FPS-Agent architecture using a Unity project written in C# and myCBR written in JAVA as the CBR system.

3.1 Using reinforcement learning in addition to case-based reasoning

Whenever the first prototype of our CBR agent was able to defeat the enemy, the agent tried to identify the next action, e.g., collection items. However, based on the retrieval of a flat hierarchical case base - and thus, searching through the whole case base, the agent got frequently stuck if invalid cases, e.g., “MoveTo;Shoot” while no enemy is alive or “CollectItem<ammunition>” while the weapon is reloaded and full of ammunition. While the agent was elaborating over these invalid actions, the enemy re-spawned and thus killing the enemy took the most relevance back again. Resulting of this behavior, we added a reinforcement learning component to evaluate positive and negative agent behaviors. As a side note, it might also be feasible to evaluate the retrieval process and to take a deeper look at the cases themselves. To prune the considered case base, we take a deeper look at the most valuable attributes when deciding on which action to take next:

1. isEnemyVisible & isEnemyAlive
   Boolean. Whenever the enemy is visible, cases with combat actions should be preferred over planning cases. If it is known that the enemy is not alive, the procurement of items takes way higher precedence.
2. distanceToEnemy
   Symbol with attributes: “near, middle, far, unknown”. The distance to the enemy is unknown, if the enemy is invisible (despite one might be able to extrapolate the estimated distance based on the last time the enemy has been seen). Since accuracy has not yet been modeled in the prototype, this attribute only takes increasing relevance if the enemy and the agent both want to pick up the same item at the same time.
3. ownHealth
   Symbol with attributes: “critical, few, middle, much, full”. This is the most important attribute. The hit points of an agent are perceived as an integer, but we rather introduce categories of HP to simplify the agents behavior reasoning. Every plan should evaluate this value when considered to be executed.

Example: After killing the enemy, the agent is left with 50 % HP. A health container is 7 yards away, while a weapon upgrade is 9 yards away (in the same direction). The agent plans to pick up the health container and re-evaluates the new situation first, e.g., checking the visibility of the enemy, before picking up the weapon upgrade. To evaluate the current situation, we define the following probability to win $win_{sit} \in [0, 1]$ based on the current situation:

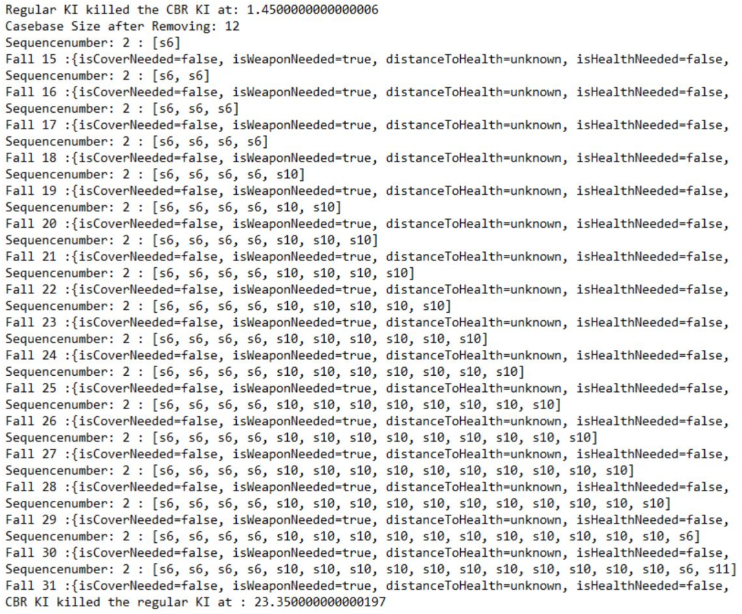
$$\begin{aligned} win_{sit} = &(Health_{CBR} - Health_{AI}) \cdot w_{Health} \\ &+(Weapon_{CBR} - Weapon_{AI}) \cdot w_{Weapon} \\ &+(Ammunition_{CBR} - Ammunition_{AI}) \cdot w_{Ammunition} \end{aligned}$$

The corresponding weights w_i with $\sum_{w_i} = 1$ are for testing purposes implemented in a static way, but in future work, the parameters should ultimately be learned and optimized during the revise step of the CBR cycle (e.g., it might be acceptable to follow an aggressive behavior despite having low HP when having a very good weapon). We categorize the result of the calculation into three areas, whereas the size of the areas shall be learned during the games but will be initialized with the mentioned values:

- Rejection area: $[0, 0.4]$
  The agent perceives that he is in a disadvantageous situation (e.g., because of low health or a lesser effective weapon). Usually, cases with passive behavior like searching and collecting health container will be selected.
- Risk area: $(0.4, 0.6)$
  This is the most flexible area of calculation. Should certain behaviors fail for multiple times, the corresponding thresholds for dynamic case manipulation have to be adjusted in accordance to the perceived result (reinforcement learning).
- Acceptance area: $[0.6, 1]$
  The agent perceives that he is in an advantageous situation and thus performs aggressive behavior, e.g., actively searching for the enemy.

To evaluate the risk area using reinforcement learning, cases need to be adjustable during run-time. To implement this, we are using sequences. A sequence saves temporarily every case retrieved from the CBR component during the lifespan of an agent. Whenever the agent dies, the corresponding sequence will be terminated. Fig. 3 depicts one exemplary sequence where a case has been added during each step. The sequence itself then contains the corresponding case. These sequences can be simplified by combining reoccurring cases, but are kept this way for illustrative purposes.

These sequences are an important part in the next topic: Logging. According to reinforcement learning, we want to move through a sequence (which, again, represents the whole lifespan of an agent) and evaluate the experience that have been made. We want to save positive experiences in our case base by rather setting a higher weight on the case initializing cases than on the later redundant “moveTo;Shoot” cases which do not necessarily contain a lot of valuable information. The same approach is used for deleting cases. To prevent an overeager deletion of cases, we only delete cases if the size of the case base is larger than 50 and the win rate of the current situation has dropped below 10 %.



```
Regular KI killed the CBR KI at: 1.4500000000000006
Casebase Size after Removing: 12
Sequencenumber: 2 : [s6]
Fall 15 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6]
Fall 16 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6]
Fall 17 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6]
Fall 18 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10]
Fall 19 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10]
Fall 20 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10]
Fall 21 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10]
Fall 22 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10]
Fall 23 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10]
Fall 24 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10, s10]
Fall 25 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10, s10, s10]
Fall 26 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10, s10, s10, s10]
Fall 27 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10, s10, s10, s10, s10]
Fall 28 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10, s10, s10, s10, s10, s10]
Fall 29 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10, s10, s10, s10, s10, s10, s6]
Fall 30 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
Sequencenumber: 2 : [s6, s6, s6, s6, s10, s10, s10, s10, s10, s10, s10, s10, s10, s10, s10, s6, s11]
Fall 31 :{isCoverNeeded=false, isWeaponNeeded=true, distanceToHealth=unknown, isHealthNeeded=false,
CBR KI killed the regular KI at : 23.350000000000197
```

Fig. 3. Case base log of the sequence of an agent.

3.2 Evaluation

As stated before, the initial situation was the prototype developed by Hillmann who implemented a simple rule-based and a case-based agent [5][9]. The evaluation of the prototype was rather disappointing towards the CBR agent: The agent lost heavily in comparison to the simple rule-based agent. There were five tests with each 15 minutes, calculating the kill-death ratio by grating +1 score whenever one killed the opposing agent and reducing the score by 1 whenever one has been killed by the opposing agent. The CBR agent ended on average with a kill-death ratio of -16, with decreasing tendency. Observations of these matches have shown, that the rule-based agent had more possessing time of the machine gun which leads to kill-streaks of five to six kills, while the CBR agent was rather hesitant to prioritize picking up the machine gun due to false similarity assessments. In addition, the limited complexity of the modeled domain seems also to be unfavorable towards the CBR agent. Nevertheless, we wanted to show the results of our observations by adding reinforcement learning to the CBR agent:

As Auslander et al. stated, RL approaches need time to gain a significant influence at the environment [2]. This led us to the decision to increase the testing time by 45 minutes and to test the reuse of the case base. Since our proposed reinforcement learning approach bases heavily on sequences (and thus, on the underlying case bases), we used four different initial case bases and compared them:

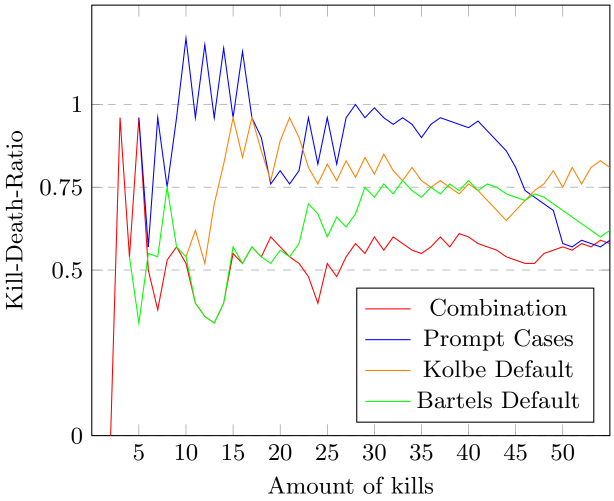


Fig. 4. Evaluation results of the first run (extract).

- **Prompt Cases:** 12 cases which have been reduced to their core attributes to support a more efficient retrieval. This should lead to a faster creation of unique cases.
- **Kolbe Default:** 10 cases handpicked by M. Kolbe which focuses on the attributes mentioned in Section 3.1.
- **Bartels Default:** 15 cases which J.-J. Bartels chose during a similar researching topic. These cases have been adjusted to fit the attribute structure of M. Kolbe.
- **Combination:** 17 cases which have been combined from Kolbe and Bartels. However, only suitable cases have been selected so that there are not for example four initial, redundant "MoveTo;Shoot" cases.

Figure 4 presents the results of the first run. During the first minutes (i.e., the first kills), the performance of the case bases is similar. This result is to be expected since during the game, both agents start with a simple pistol gun and no items are available, thus, they do kill one after each other due to the respective health advantages. Once items do spawn, the results begin to differ - with the Prompt Cases leading with a 1.25 Kill/Death Ratio. However, after approximately 40 kills, each case base seems to commute in a certain level.

This becomes more apparent after looking at the later stages of the run (Fig. 5). The Prompt Cases show a promising result during the begin of the stages, but cannot seem to hold the level and has been eliminated ten times in a row. However, this death streak could not be identified during later stages of the run since the agent began to learn picking up health packs instead of running straight into the enemy (which the agent learned with reinforcement learning).

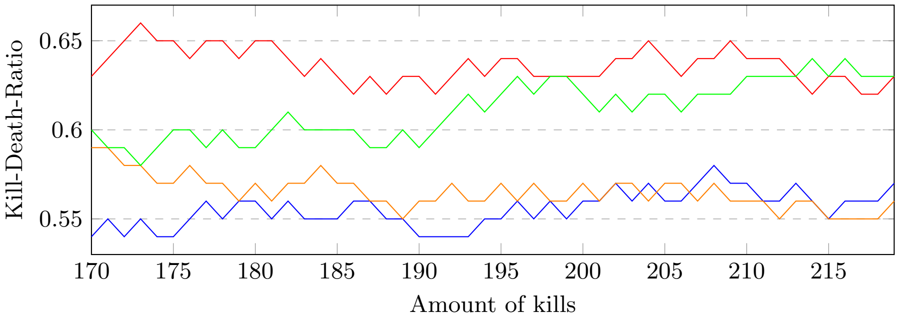


Fig. 5. Evaluation results of the first run during later stages of the run.

Since Bartels Default and the Combination show similar results to the Prompt Cases, we take a look at the resulting case bases after the test run and how they have evolved:

- **Prompt Cases:** 83 cases after 60 minutes
- **Kolbe Default:** 56 cases after 60 minutes
- **Bartels Default:** 81 cases after 60 minutes

We can see that Kolbe Default runs similar results to its contenders with an approximately 30 % smaller case base. This is due to deleting cases with negative outcome so that the case base still only contains relevant and positive experiences. The second test run - with using the outcome case bases from the first run - has been executed respectively, leading to the following results, ordered descending:

- **Prompt Cases:** K/D-Ratio of 0.55, 50 cases
- **Combination Default:** K/D-Ratio of 0.54, 120 cases
- **Bartels Default:** K/D-Ratio of 0.49, 50 cases

The results show again that the systematically created Prompt Cases using reinforcement learning show the best results. However, the overall result still needs improvement to gain a K/D-Ratio of at least 1 to actually win a game.

The results show that the integration of an RL approach into the system described in [9] leads to an improvement in the performance of the CBR agent. On the other hand it can be seen that the K/D-Ratio is still not good enough to win the game against the simple rule-based agent. Therefore, the brute-force play style of the rule-based agent still outperforms the planning play style of the CBR agent. The main reason for this seems to be the low complexity of the current level and game design. With increasing complexity of the level and the support of more strategies and tactics, like an ambush or keeping a certain distance during combat, the CBR approach should be able to outperform the simple rules.

4 Conclusion

In this work, we tried to show that the addition of reinforcement learning can be an overall improvement for a case-based reasoning agent in a relatively simple first person shooter scenario. While the here modeled domain does not hold too much complexity (which can be a disadvantage for CBR), the RL-CBR agent was still able to learn from the experiences he made. This can be proven by the results of our evaluation, as the agent made progress towards a better kill-death ratio. Nevertheless, there is still room for optimization. With the addition of RL, the learning progress of an agent could be shown and looks promising for further investigations, especially when the domain becomes more complex, for example, by using the Unreal Tournament engine which holds its own challenges.

References

1. Andersen, P., Goodwin, M. and Granmo, O.: "Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games". In: IEEE Conference on Computational Intelligence and Games (CIG), Maastricht, 2018, pp. 1-8.
2. Auslander, B., Lee-Urban, S., Hogg, C. and Muñoz.Avila, H.: "Recognizing the Enemy: Combining Reinforcement Learning with Strategy Selection using Case-Based Reasoning". In: *European Conference on Case-Based Reasoning*, Trier, Springer, 2008, pp. 59-73.
3. Dobrovsky, A., Borghoff, U. M., and Hofmann, M.: "Improving Adaptive Gameplay in Serious Games Through Interactive Deep Reinforcement Learning", In: *Cognitive Infocommunications, Theory and Applications*, Springer International Publishing, 2019, pp. 411-432.
4. Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P. and Whiteson, S.: "Stabilising Experience Replay for Deep Multi-agent Reinforcement Learning", In: *Proceedings of the 34th International Conference on Machine Learning*, Volume 70, 2017, JMLR.org, pp 1146-1155.
5. Hillmann, J.: "Konzeption und Entwicklung eines Prototypen fr ein lernfhiges Multi-Agenten-System mittels des fallbasierten Schlieen im Szenario einer First-Person Perspektive" (Conception and Development of a prototype for a multi-agent-system with learning capabilities using case-based reasoning in the first-person perspective szenario). Hildesheim, University of Hildesheim, 2017.
6. Lample, G. and Chaplot, D. S.: "Playing FPS games with deep reinforcement learning", In: *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
7. Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J.,Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, S., Lillicrap, T. P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R.: "StarCraft II: A New Challenge for Reinforcement Learning", In: *CoRR*, 2017.
8. Parashar, P., Sheneman, B. and Goel, A. K.: "Adaptive Agents in Minecraft: A Hybrid Paradigm for Combining Domain Knowledge with Reinforcement Learning", In: *Autonomous Agents and Multiagent Systems*, Springer International Publishing, 2017, pp. 86-100.

9. Reuss, P., Hillmann, J., Viefhaus, S., Althoff, K.-D.: “Case-Based Action Planning in a First Person Scenario Game”. In: Rainer Gemulla, Simone Ponzetto, Christian Bizer, Margret Keuper, Heiner Stuckenschmidt (Publ.). LWDA 2018 - Lernen, Wissen, Daten, Analysen - Workshop Proceedings. GI-Workshop-Tage “Lernen, Wissen, Daten, Analysen” (LWDA-2018) August 22-24 Mannheim Germany University of Mannheim 8/2018.
10. Sutton, R. S.: “Introduction: The Challenge of Reinforcement Learning”. In: *The International Series in Engineering and Computer Science (SECS, Volume 173) - Machine Learning*, Boston, Kluwer Academic O, 1992.
11. Sethy, H., Patel, A. and Padmanabhan, V.: “Real Time Strategy Games: A Reinforcement Learning Approach”. In: *Procedia Computer Science*, Volume 54, 2015.
12. Tang, Z., Zhao, D., Zhu, Y. and Guo, P.: “Reinforcement Learning for Build-Order Production in StarCraft II”. In: Eighth International Conference on Information Science and Technology (ICIST), Cordoba, 2018, pp. 153-158.
13. Ulit, Jaidee, Muñoz-Avila, Héctor and Aha, David W.: “Case-Based Goal-Driven Coordination of Multiple Learning Agents” In: *Case-Based Reasoning Research and Development*, Springer, Proceedings, 2013, pp. 164-178.
14. Wender, S., Watson, I.: “Combining Case-Based Reasoning and Reinforcement Learning for Unit Navigation in Real-Time Strategy Game AI”. In: *International Conference on Case-Based Reasoning (ICCBR)*, 2014, Cork, Ireland, Springer, Proceedings, 2014, pp. 511-525.
15. Wender, S., Watson, I.: “Using reinforcement learning for city site selection in the turn-based strategy game Civilization IV”. In: IEEE Symposium On Computational Intelligence and Games, Perth, WA, 2008, pp. 372-377.