Explaining the Behavior of Reinforcement Learning Agents using Association Rules

Zahra Parham[1], Vi Tching de Lille[2], and Quentin Cappart[1]

[1] Ecole Polytechnique de Montréal, Montreal, Canada
{zahra.parham,quentin.cappart}@polymtl.ca
[2] StockholmSyndrome.ai, Montreal, Canada
vitching@stockholmsyndrome.ai

Abstract. Deep reinforcement learning algorithms are increasingly used to drive decision-making systems. However, there exists a known tension between the efficiency of a machine learning algorithm and its level of explainability. Generally speaking, increased efficiency comes with the cost of decisions that are harder to explain. This concern is related to *explainable artificial intelligence*, which is a hot topic in the research community. In this paper, we propose to explain the behaviour of a deep reinforcement learning algorithm thanks to standard data mining tools, i.e. association rules. We apply this idea to the design of *playing bots*, which is ubiquitous in the video game industry. To do so, we designed three agents trained with a deep Q-learning algorithm for the game *Street FighterTurbo II*. Each agent has a specific playing style. Our experiments show that association rules can provide interesting insights on the behavior of each agent, and reflect their specific playing style. We believe that this work is a next step towards the explanation of complex models in deep reinforcement learning.

Keywords: Association Rules · Explainable Reinforcement Learning

1 Introduction

Multiplayer video games refer to video games that involve more than one person playing together at the same time, either as a team (cooperative game) or as an opponents (competitive game). In such games, the interactions between players are of the utmost importance and must be carefully designed in order to make the game enjoyable. However, ensuring and maintaining proper interactions all throughout the playing session is a hard goal to achieve in practice. First, enough people must be available to play the game, and second, people must remain active until the end of the game. Besides, the more people are involved in the game, the more difficult it is to ensure these goals. For instance, each session of the massively-played game *Leagues of Legends* is roughly about 30 minutes and involves 10 players, split into 2 teams. Having only one player leaving the game deteriorates general satisfaction, especially for the impacted team.

A natural solution to this issue is to integrate artificial agents, dedicated to mimic the behaviour of human players. Such agents are commonly referred to as

bots. When a person is missing or leaves the game before the end, the bot will replace the player. Despite the simplicity of this idea, building believable and fun-to-play bots is a non-trivial task: they must be developed specifically for each game and their behaviour in the game must remain realistic for the other players. An additional asset is to be able to replace the player with a bot having a similar playing style as the replaced player in order to smooth the transition. Albeit possible and already used by big video game companies, building efficient and human-like bots are generally beyond the range of independent studios with limited resources. It is why an innovative way to program bots should be designed. The requirements are as follows: the approach should be (1) *generic*, meaning that it must be possible to use the approach for different games, (2) *credible* as it should mimic a human behaviour, and (3) *transparent*, in the sense that a developer must be able to understand the rationale behind the actions of the bot and to re-calibrate it if required.

In another context, *reinforcement learning* [23] has been successfully applied to various kinds of video games in combination with *deep learning* [13], *imitation learning* [18], and *league-style training*, such as for Dota 2 [3], Minecraft [8], or Doom [11]. The idea is to let the bot plays the game, reward it when appropriate actions are performed, and use this reward as feedback to train the agent. Once trained, the agent can then be used for new sessions of the game. Provided that the learning was successful, good performances from the bot are expected. There exist in the literature a plethora of learning algorithms that can be used in this context. Notable examples are DQN [17], PPO [22], DDPG [15], or SAC [9]. From an industrial point of view, the main benefit of this approach is that the training algorithm is generic, only the definition of the environment changes from one game to another. This directly ensures the first requirement about the genericity of the approach. However, the requirements on the credibility and transparency remain unaddressed. Broadly speaking, these concerns are related to *explainable artificial intelligence* [5]. It must be possible to understand and trace why specific predictions are performed by a model. By doing so, the confidence and that we can have in the model is increased. This aspect is critical for many real applications, such as in healthcare [19]. Although widely studied for supervising learning approaches [4], there are fewer methods dedicated to explainability in reinforcement learning [20], and even less that are applied on the video game industry [16].

Based on this context, we propose to use data mining tools, such as *association rules* [2], in order to provide meaningful information that can be used to infer explanations about decisions carried out by reinforcement learning agents. It is done as follows: when the agent is deployed on a game, it generates samples consisting of a set of observations and of the decision that it has carried out. The proposed idea is to use association rule mining tools in order to detect which components of the observations are highly correlated with specific decisions. Then, we can deduce that these observations are features that often trigger the decision. We evaluate this idea on the 2-players competitive game *Street Fighter Turbo II*. We trained three agents for this game. Each of them

(a) Action: jumping (Player 1 - red). (b) Action: punching (Player 2 - yellow).

Fig. 1: Illustrations of Street Fighter Turbo II game.

is characterized by a specific playing style (aggressive, defensive, and balanced) and has been trained accordingly. The mined rules that we computed from millions of samples obtained from each agent show that we can discriminate each agent by its playing style and thus explain their behavior.

The structure of the paper is centered on this case study. The next section presents the nature of the game and the preprocessing steps that have been done to build the environment. Then, Section 3 formally describes this environment in order to be leveraged in the subsequent section by the reinforcement learning agent. The rule extraction methodology is then explained in Section 5. Finally, Section 6 presents the rules that we obtained for each agent.

2 Case study: Street Fighter Turbo II

Street Fighter II Turbo is a competitive fighting game released by Capcom for arcades in 1992. Briefly, the game features two opponents. The goal of each is to deplete the health of the opponent before the timer expires. The winner is the surviving player or, in case of timeout, the player having the most remaining health. To do so, each player can perform a variety of actions, such as moving forward, moving backward, jumping, crouching, kicking the enemy, etc. Illustrations of the game interface with three actions are proposed in Fig. 1.

From the point of view of a human player, actions are performed thanks to a predefined combination of keys on a keyboard. By limiting the combination to at most 2 keys as proposed in [6], we consider 21 different actions. They are summarized in Table 1. The raw environment of any video game is the visual frames displayed to the player, i.e., a grid of pixels. Although such an input can be successfully leveraged by deep learning architectures, i.e., thanks to a convolutional neural network [14], it does not give an input that is understandable by humans. For such a reason, the first step is to pre-process the visual frames and to translate them into a set of high-level features. To do so, we used *BizHawk*

emulator[3] to obtain low-level information located in the RAM and related to a specific state of the game. Among the many features in the RAM, we have extracted the following features, with the possible values they can take, summarized in Table 2. As we can see, we have 15 different features, 6 of them are related to a specific player, and 3 of them relates to both players.

Table 1: List of available actions.

Action name	**Description**
movingForward	The agent walks in the forward direction
movingBackward	The agent walks in the backward direction
jumping	The agent jumps straight up
jumpingForward	The agent jumps in the forward direction
jumpingBackward	The agent jumps in the backward direction
jumpingWithKicking	The agent jumps in the forward while kicking the opponent
neutralJumpingStrong	The agent punches medium while jumping
farStandingRoundhouse	The agent kicks hard while standing far
farStandingFierce	The agent punches hard while standing far
farStandingJab	The agent punches light while standing far
farStandingShort	The agent kicks light while standing far
farStandingForward	The agent kicks high side while standing
crouchingShort	The agent kicks low while crouching
crouchingForward	The agent kicks with good reach while crouching
crouchingStrong	The agent punches medium crouching
crouchingJab	The agent punches light while crouching
crouchingFierce	The agent punches hard while crouching
crouchingRoundhouse	The agent kicks hard while crouching
sitDown	The agent sits down in place
sitBackward	The agent walks in the backward direction while siting down
idling	The agent stays in place without doing any action

3 Definition of the Environment

The first step in reinforcement learning is to define an *environment* as a *Markov Decision Process* (MDP). Briefly, let $\langle S, A, T, R \rangle$ be a tuple representing a deterministic and fully observable environment, where S is the set of states, A the set of actions that an agent can perform inside the environment, $T : S \times A \to S$ is the transition function leading the agent to another state, and $R \times S \times A \to \mathbb{R}$ is a function rewarding (or penalizing) the realization of an action $a \in A$. The behaviour of an agent is defined by a policy $\pi : S \to A$, indicating the action to be performed on a specific state. The goal of an agent is to learn a policy maximizing the accumulated reward during its lifetime, defined as a sequence of

[3] https://github.com/TASEmulators/BizHawk

Table 2: Summary of the observations used to create the environment.

Observation name	Domain	Description
$\mathsf{isMoving}(p)$	$\{0,1\}$	Indicate if the player p is currently moving
$\mathsf{isCrouching}(p)$	$\{0,1\}$	Indicate if the player p is currently crouching
$\mathsf{isJumping}(p)$	$\{0,1\}$	Indicate if the player p is currently jumping
$\mathsf{horizontalCoord}(p)$	$[0,498]$	The horizontal coordinate of player p
$\mathsf{verticalCoord}(p)$	$[0,204]$	The vertical coordinate of player p
$\mathsf{horizontalDelta}(p_1,p_2)$	$[0,189]$	The horizontal distance between player p_1 and p_2
$\mathsf{verticalDelta}(p_1,p_2)$	$[0,158]$	The vertical distance between player p_1 and p_2
$\mathsf{health}(p)$	$[0,176]$	The remaining health of player p
$\mathsf{remainingTime}$	$[0,99]$	The time until the end of the game (99 seconds in total)

states $s_t \in S$ with $t \in \{1,\ldots,\theta\}$. This is commonly referred to as an *episode*. The final state s_θ is referred to as the *terminal state* and is commonly reached when a halting condition is reached. This formalization is common in any task related to reinforcement learning [23]. The model we have designed for *Street Fighter II Turbo* is as follows:

State A state $s \in S$ is defined as a sequence $\langle x_1,\ldots,x_{15}\rangle$ of 15 features. It corresponds to the observations summarized in Table 2. A state is terminal when one of these three conditions is fulfilled: (1) the health of the first player is depleted, i.e., $\mathsf{health}(p_1) = 0$, (2) the health of the second player is depleted, i.e., $\mathsf{health}(p_2) = 0$, or (3) when the timer is exceeded, i.e., $\mathsf{remainingTime} = 0$.

Action An action $a \in A$ simply corresponds to an available action proposed in Table 1. There are then 21 possible actions that the agent can perform inside the environment.

Transition The transition function updates the current state s_t according to the action performed at the time t. The definition of the transition directly relies on the game mechanisms as executed by the emulator. For instance, assuming that $\mathsf{farStandingJab}$ is an action performed by the first player and that deals 40 damages to the opponent, the state information s_{t+1} related to $\mathsf{health}(p_2)$ get the value $\mathsf{health}(p_2) - 40$. Internally, the transition between two states correspond to 7 visual frames.

Reward The goal of the reward is to encourage the agent to perform actions that will lead it to win the game. A simple way to define the reward is to only give a positive value when the agent wins the game, and a negative value when it loses it. This reward signal is defined in Equation (1). It indicates that an action a perform at a state s is positively rewarded if the next state is a terminal state corresponding to a victory for the first player. On the other hand, it is negatively rewarded (i.e. punished) in case of defeat.

$$R^{\mathsf{final}}(s_t,a) = \begin{cases} 1 & \text{if } \mathsf{health}(p_1) > \mathsf{health}(p_2) \wedge \mathsf{isTerminal}(s_{t+1}) \\ -3 & \text{else if } \mathsf{health}(p_1) < \mathsf{health}(p_2) \wedge \mathsf{isTerminal}(s_{t+1}) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The values of 1 and -3 have been calibrated manually. The drawback is that non-zero rewards are collected only at the end of an episode. This yields the *sparse reward* issue, which is known to complicate the training process [21]. We tackle this issue by introducing an intermediate reward, that can be collected in the middle of an episode. This is also known as a *reward shaping* method. The idea is to evaluate the impact of an action on the remaining health of both players. Intuitively, each health point depleted from the opponent will be rewarded, and each health point that is inflicted will be punished. Let $\Delta_t^{\texttt{health}}(p) = \mathsf{health}_{t+1}(p) - \mathsf{health}_t(p)$ be the difference in the health for the player p between state s_{t+1} and s_t, the intermediate reward is defined in Equation (2).

$$R^{\mathsf{intermediate}}(s_t, a) = \alpha^{\mathsf{win}} \Delta_t^{\mathsf{health}}(p_2) - \alpha^{\mathsf{lose}} \Delta_t^{\mathsf{health}}(p_1) \tag{2}$$

On this equation, α^{win} is a positive coefficient giving incentive to the agent to deplete the health of the opponent, and α^{lose} a second coefficient giving it incentive to not lose health. Based on both equations, the reward function used in our model is as follows.

$$R(s_t, a) = R^{\mathsf{intermediate}}(s_t, a) + R^{\mathsf{final}}(s_t, a) \tag{3}$$

4 Learning algorithm

The learning algorithm relies on a deep Q-learning approach [17]. Briefly, the idea is to estimate the quality of taking an action a from a state s. This estimation is referred to as a *Q-value* and is obtained thanks to a trained deep neural network. In our case, we used a fully-connected neural network of two hidden layers of 64 neurons each together with a `ReLU` activation [7]. The output is a real value for each action, corresponding to the *Q-value*. Once estimated, the agent policy consists in always selecting the action that has the best Q-value.

Three agents have been designed in this work. Each of them has been trained between 2,000,000 and 3,000,000 time steps. This corresponds to around 10,000 game sessions and 15 hours of training time on a Intel(R) Xeon(R) CPU @ 2.30GHz CPU and a Tesla P100-PCIE-16GB GPU using Adam optimizer [12]. Additionally, the game consists of three episodes (i.e. three rounds) but we only consider the first one in each game session. Finally, we would like to point out that building the most efficient agent was not the goal of this project. In contrast, our goal was to build only a decent agent and to show that its behaviour can be successfully explained thanks to association rules.

5 Explanation with Association Rules

This section describes the methodology we used to extract information explaining the behaviours of the trained agents. We do it by means of association rules [1]. Briefly, the idea is to extract relevant correlations from a large database

$\mathcal{T}$ of transactions. In our context, each transaction is defined as a set $\{I_1, \ldots, I_n\}$ of n items together with a specific item Y. An association rule is defined as an implication of the form $I \rightarrow Y$, where $I = \{I_j, \ldots, I_k\}$ is a subset of existing items. The goal is to find association rules that are the most frequent in $\mathcal{T}$. Provided with this information, we can then infer that the item Y is often obtained when the items I are also present. Three metrics are commonly used for determining how relevant a rule is. There are as follows:

1. The *support* indicates how frequent a rule is. It is computed as the ratio between the transactions containing both I and Y with the total number of transactions in $\mathcal{T}$. It is formalized in Equation (4).

$$\mathsf{Support}(I \rightarrow Y) = \frac{\#\left\{t \in \mathcal{T} \;\middle|\; I \in t \wedge Y \in t\right\}}{\#\left\{t \in \mathcal{T}\right\}} \tag{4}$$

2. The *confidence* is the ratio between the transactions containing both I and Y, with the transactions containing only I. It is formalized in Equation (5).

$$\mathsf{Confidence}(I \rightarrow Y) = \frac{\#\left\{t \in \mathcal{T} \;\middle|\; I \in t \wedge Y \in t\right\}}{\#\left\{t \in \mathcal{T} \;\middle|\; I \in t\right\}} \tag{5}$$

3. The *lift* measures the performance at predicting the presence of both Y and I in a transaction against a random prediction. A lift of 1 indicates that the probability of occurrence of both I and Y are independent. In such a case, no relevant rule can be drawn involving those items. This measure is formalized in Equation (6). Intuitively, a lift of 2 shows that the Y of the corresponding rule is twice more likely to be present compared to the average.

$$\mathsf{Lift}(I \rightarrow Y) = \frac{\#\left\{t \in \mathcal{T} \;\middle|\; I \in t \wedge Y \in t\right\}}{\#\left\{t \in \mathcal{T} \;\middle|\; I \in t\right\} \times \#\left\{t \in \mathcal{T} \;\middle|\; Y \in t\right\}} \tag{6}$$

In our case, we opted to find rules maximizing the lift, while ensuring a minimum support threshold of 0.01 and a minimum confidence threshold of 0.01 as well. There exist many algorithms in the literature for finding association rules, such as *Apriori* [2] or its variants [25, 24]. In this work, we propose to use association rules to explain the behaviour or our agent. To do so, we mapped the state of the reinforcement learning environment (i.e., information from Table 1) with the $\{I_1, \ldots, I_n\}$ items, and the actions that are taken (i.e., information from Table 2) with the Y item. We collected such information by letting the trained agents play the games 10,000 times. It roughly gives a database $\mathcal{T}$ of 100,000 transactions for each agent. One difficulty that arose is that standard association rule mining algorithms assume that the items have categorical values. It is not the case of some observations of the environment, such as the remaining health of a player ($\mathsf{health}(p)$). Although alternative algorithms exist in the literature for such

Table 3: Summary of the discretization performed on the observations.

Modified observation	Domain	Categorical domain
$\mathsf{horizontalCoord}(p_1)$ $\mathsf{horizontalCoord}(p_2)$	$[0, 498]$	$\mathsf{rightOfP2} : \mathsf{hCoord}(p_1) > \mathsf{hCoord}(p_2)$ $\mathsf{leftOfP2} : \mathsf{hCoord}(p_1) < \mathsf{hCoord}(p_2)$
$\mathsf{horizontalDelta}(p_1, p_2)$	$[0, 189]$	$\mathsf{close} : [0, 63]$, $\mathsf{middle} : [64, 126]$, $\mathsf{far} : [127, 189]$
$\mathsf{verticalCoord}(p)$	$[0, 204]$	$\mathsf{jumping} : [0, 191]$, $\mathsf{standing} : [192, 204]$
$\mathsf{health}(p)$	$[0, 176]$	$\mathsf{low} : [0, 58]$, $\mathsf{medium} : [59, 117]$, $\mathsf{high} : [118, 176]$
$\mathsf{verticalDelta}(p_1, p_2)$	$[0, 158]$	Not used (redundant with $\mathsf{verticalCoord}$)
$\mathsf{remainingTime}$	$[0, 99]$	Not used (not player-dependant)

a situation [10], we selected the option to discretize each numerical observation into a set of meaningful categories. The main reason is that categorical data are easier to interpret. A summary of this discretization is proposed in Table 3. For instance, the first discretization shows that we have a category $\mathsf{rightOf}$ if the horizontal coordinate of player p_1 is higher than the horizontal coordinate of player p_2. The rule extraction has been carried out in `R` using `arules` package[4] and was executed in less than 30 seconds for each agent.

6 Analysis of the Rules Obtained

This section presents the best rules that we have been able to extract from the agents we trained. Three agents are presented: a *balanced*, a *defensive*, and an *aggressive* one. Their differences relates to the reward function that has been used to define the environment. More details specific to each agent are proposed in the next subsections.

Rules for a Balanced Agent

The first agent we trained uses the reward function defined in Equation (3) with $\alpha^{\mathsf{win}} = \alpha^{\mathsf{lose}} = 1$. Intuitively, this agent has an equal incentive to protect its health and to deplete the health of the opponent, hence its qualification of being balanced. The progression of the reward during the training phase is illustrated in Fig. 2. Then, Table 4 shows the top-5 rules obtained, sorted by their lift, and with the minimum threshold of 0.01 for the support and the confidence. Interestingly, we can see that the agents perform both aggressive ($\mathsf{jumpingWithKicking}$) and defensive ($\mathsf{jumpingForward}$) actions with a relatively similar lift value. For instance, an interpretation of the first rule is that the agent is likely to jump toward the enemy when it is not too far ($\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{middle}$) and on the left side of the opponent ($\mathsf{horizontalCoord}(p_1) = \mathsf{leftOfP2}$). The result is to land behind the opponent, which is a common tactical move for this game.

[4] https://github.com/mhahsler/arules/

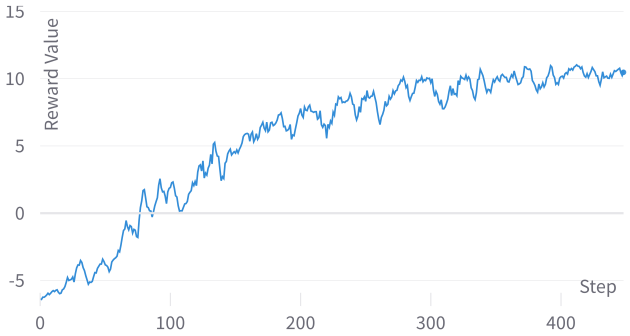

Fig. 2: Evolution of the reward during the training for the balanced agent.

Table 4: Top-5 rules obtained for the balanced agent (lift measure).

Rule antecedent (I)	Rule consequent (Y)	Lift
$\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{middle}$, $\mathsf{isMoving}(p_2) = 1$, $\mathsf{horizontalCoord}(p_1) = \mathsf{leftOfP2}$	jumpingForward	9.34
$\mathsf{isCrouching}(p_2) = 0$, $\mathsf{isMoving}(p_2) = 1$, $\mathsf{horizontalCoord}(p_1) = \mathsf{rightOfP2}$	jumpingBackward	7.34
$\mathsf{verticalCoord}(p_1) = \mathsf{jumping}$, $\mathsf{isMoving}(p_1) = 0$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{far}$	jumpingWithKicking	7.34
$\mathsf{isCrouching}(p_2) = 1$, $\mathsf{isMoving}(p_2) = 1$, $\mathsf{health}(p_1) = \mathsf{high}$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$	movingBackward	5.97
$\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$, $\mathsf{isJumping}(p_1) = 0$, $\mathsf{isCrouching}(p_2) = 0$, $\mathsf{horizontalCoord}(p_1) = \mathsf{leftOfP2}$	crouchingStrong	3.02

Rules for a Defensive Agent

As our goal is to assess whether association rules can find relevant rules explaining the behaviour of a reinforcement learning agent, we trained another agent, which has incentive to protect its health. This is done by setting $\alpha^{\mathsf{win}} = 0$ and $\alpha^{\mathsf{lose}} = 1$. Intuitively, the agent does not receive reward anymore if it hits the opponent, but it still gets punished if it loses health. Provided that our hypothesis is correct, the top rules should be related to more defensive actions. They are summarized in Table 5. A first observation is that the rules obtained are highly different than the ones related to the balanced agent. The first top-rule (crouchingFierce) is an in-between aggressive/defensive action and is obtained with a high lift value. The second top-rule (movingBackward) is a purely defensive action, which corroborates the fact that the agent is trained to have a more defensive game-play than the balanced one.

Table 5: Top-5 rules obtained for the defensive agent (lift measure).

Rule antecedent (I)	Rule consequent (Y)	Lift
$\mathsf{isMoving}(p_1) = 1$, $\mathsf{health}(p_2) = \mathsf{medium}$, $\mathsf{isCrouching}(p_2) = 0$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$, $\mathsf{isJumping}(p_1) = 1$, $\mathsf{verticalCoord}(p_2) = \mathsf{standing}$	crouchingFierce	31.1
$\mathsf{isCrouching}(p_1) = 0$, $\mathsf{isMoving}(p_1) = 0$, $\mathsf{health}(p_2) = \mathsf{high}$, $\mathsf{isJumping}(p_1) = 0$, $\mathsf{health}(p_1) = \mathsf{high}$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{middle}$,$\mathsf{horizontalCoord}(p_1) = \mathsf{rightOfP2}$	movingBackward	19.27
$\mathsf{isMoving}(p_1) = 1$, $\mathsf{isJumping}(p_1) = 1$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$, $\mathsf{health}(p_1) = \mathsf{high}$ $\mathsf{verticalCoord}(p_2) = \mathsf{jumping}$	crouchingJab	16.0
$\mathsf{isCrouching}(p_1) = 1$, $\mathsf{isMoving}(p_1) = 0$, $\mathsf{health}(p_2) = \mathsf{low}$, $\mathsf{isCrouching}(p_2) = 0$, $\mathsf{isMoving}(p_2) = 1$	sitDown	7.03
$\mathsf{isCrouching}(p_1) = 0$, $\mathsf{health}(p_2) = \mathsf{medium}$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$, $\mathsf{health}(p_1) = \mathsf{high}$, $\mathsf{verticalCoord}(p_2) = \mathsf{jumping}$, $\mathsf{isMoving}(p_2) = 1$	jumpingWithKicking	5.73

Rules for an Aggressive Agent

Following the same idea, we performed the same analysis on an aggressive agent. It has been implemented by setting $\alpha^{\mathsf{win}} = 1$ and $\alpha^{\mathsf{lose}} = 0$. Intuitively, it still receives rewards when it hits the opponent, but it is not punished anymore when it loses health. Provided that our hypothesis is correct, the top rules should be related to more aggressive actions. They are summarized in Table 6. Compared to the defensive agents, roughly the same rules are obtained but within a different importance order. For instance, the second top-rule is now an attack instead of a defensive move. Another offensive move (jumpingWithKicking) also gains importance (lift of 12.3 instead of 5.73), showing that the agent is, as intended, more aggressive than the defensive one.

7 Conclusion and Future Work

Deep reinforcement learning is increasingly considered for driving decision-making systems. However, the trade-off between the efficiency of a model and its explainability level is a challenge, which is critical for numerous applications. In this paper, we proposed to use association rules in order to explain the decisions performed by an agent trained by a deep reinforcement learning algorithm. We proposed an application, on the *StreetFighter Turbo II* video game and trained three agents, each with a specific style of play. The results obtained show that the playing style of an agent has an impact on the rules obtained and on their rank. This directly corroborates the hypothesis that association rules can be a relevant tool to explain the behaviour of reinforcement learning algorithms. Although our application is for the video game industry, the approach proposed is generic and could be considered for other applications of reinforcement learning.

Table 6: Top-5 rules obtained for the aggressive agent (lift measure).

Rule antecedent (I)	Rule consequent (Y)	Lift
$\mathsf{isMoving}(p_1) = 1$, $\mathsf{health}(p_2) = \mathsf{medium}$, $\mathsf{isJumping}(p_1) = 1$, $\mathsf{isCrouching}(p_2) = 0$, $\mathsf{health}(p_1) = \mathsf{high}$, $\mathsf{verticalCoord}(p_2) = \mathsf{standing}$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$	crouchingFierce	39.67
$\mathsf{isMoving}(p_1) = 1$, $\mathsf{isJumping}(p_1) = 1$, $\mathsf{health}(p_1) = \mathsf{high}$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$, $\mathsf{verticalCoord}(p_2) = \mathsf{jumping}$, $\mathsf{health}(p_2) = \mathsf{medium}$, $\mathsf{horizontalCoord}(p_1) = \mathsf{leftOfP2}$	crouchingJab	20.29
$\mathsf{isCrouching}(p_1) = 0$, $\mathsf{isMoving}(p_1) = 0$, $\mathsf{health}(p_2) = \mathsf{high}$, $\mathsf{isJumping}(p_1) = 0$, $\mathsf{health}(p_1) = \mathsf{high}$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{middle}$, $\mathsf{horizontalCoord}(p) = \mathsf{rightOf}$	movingBackward	19.28
$\mathsf{isCrouching}(p_1) = 0$, $\mathsf{health}(p_2) = \mathsf{medium}$, $\mathsf{health}(p_1) = \mathsf{high}$, $\mathsf{horizontalDelta}(p_1, p_2) = \mathsf{close}$, $\mathsf{verticalCoord}(p_2) = \mathsf{jumping}$, $\mathsf{isMoving}(p_2) = 1$, $\mathsf{isJumping}(p_1) = 0$, $\mathsf{horizontalCoord}(p) = \mathsf{leftOf}$	jumpingWithKicking	12.3
$\mathsf{isCrouching}(p_1) = 1$, $\mathsf{isMoving}(p_1) = 0$, $\mathsf{isCrouching}(p_2) = 0$, $\mathsf{isMoving}(p_2) = 1$	sitDown	3.96

However, an important limitation is that the input observations must be intrinsically explainable. It is not always the case, especially when the inputs are a grid of pixels. Targeting this limitation is an interesting line of future work.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD Rec. **22**(2), 207–216 (jun 1993). https://doi.org/10.1145/170036.170072, https://doi.org/10.1145/170036.170072
2. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB. vol. 1215, pp. 487–499. Citeseer (1994)
3. Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H.P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S.: Dota 2 with large scale deep reinforcement learning. CoRR **abs/1912.06680** (2019), http://arxiv.org/abs/1912.06680
4. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. Journal of Artificial Intelligence Research **70**, 245–317 (2021)
5. Došilović, F.K., Brčić, M., Hlupić, N.: Explainable artificial intelligence: A survey. In: 2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO). pp. 0210–0215. IEEE (2018)
6. Fletcher, A.: How we built an AI to play Street Fighter II — can you beat it? https://medium.com/gyroscopesoftware/how-we-built-an-ai-to-play-street-fighter-ii-can-you-beat-it-9542ba43f02b, accessed: 2022-11-18

7. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. pp. 315–323. JMLR Workshop and Conference Proceedings (2011)
8. Guss, W.H., Codel, C., Hofmann, K., Houghton, B., Kuno, N., Milani, S., Mohanty, S.P., Liebana, D.P., Salakhutdinov, R., Topin, N., Veloso, M., Wang, P.: The minerl competition on sample efficient reinforcement learning using human priors. CoRR **abs/1904.10079** (2019), http://arxiv.org/abs/1904.10079
9. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. pp. 1861–1870. PMLR (2018)
10. Hong, T.P., Kuo, C.S., Chi, S.C.: Mining association rules from quantitative data. Intelligent data analysis **3**(5), 363–376 (1999)
11. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaskowski, W.: Vizdoom: A doom-based AI research platform for visual reinforcement learning. CoRR **abs/1605.02097** (2016), http://arxiv.org/abs/1605.02097
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
13. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
14. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10), 1995 (1995)
15. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
16. Madumal, P., Miller, T., Sonenberg, L., Vetere, F.: Explainable reinforcement learning through a causal lens. In: Proceedings of the AAAI conference on artificial intelligence. vol. 34, pp. 2493–2500 (2020)
17. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
18. Osa, T., Pajarinen, J., Neumann, G., Bagnell, J.A., Abbeel, P., Peters, J., et al.: An algorithmic perspective on imitation learning. Foundations and Trends® in Robotics **7**(1-2), 1–179 (2018)
19. Pawar, U., O'Shea, D., Rea, S., O'Reilly, R.: Explainable ai in healthcare. In: 2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA). pp. 1–2. IEEE (2020)
20. Puiutta, E., Veith, E.M.: Explainable reinforcement learning: A survey (2020)
21. Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., Springenberg, J.T.: Learning by playing solving sparse reward tasks from scratch. In: International conference on machine learning. pp. 4344–4353. PMLR (2018)
22. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
23. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
24. Wu, H., Lu, Z., Pan, L., Xu, R., Jiang, W.: An improved apriori-based algorithm for association rules mining. In: 2009 sixth international conference on fuzzy systems and knowledge discovery. vol. 2, pp. 51–55. IEEE (2009)
25. Yuan, X.: An improved apriori algorithm for mining association rules. In: AIP conference proceedings. vol. 1820, p. 080005. AIP Publishing LLC (2017)