Decision Making from Confidence Measurement on the Reward Growth using Supervised Learning
A Study Intended for Large-scale Video Games

D. Taralla, Z. Qiu, A. Sutera, R. Fonteneau and D. Ernst
Department of Electrical Engineering and Computer Science, University of Liège, Liège, Belgium

Keywords: Artificial Intelligence, Decision Making, Video Games, Hearthstone, Supervised Learning, ExtraTrees.

Abstract: Video games have become more and more complex over the past decades. Today, players wander in visually- and option- rich environments, and each choice they make, at any given time, can have a combinatorial number of consequences. However, modern artificial intelligence is still usually hard-coded, and as the game environments become increasingly complex, this hard-coding becomes exponentially difficult. Recent research works started to let video game autonomous agents learn instead of being taught, which makes them more intelligent. This contribution falls under this very perspective, as it aims to develop a framework for the generic design of autonomous agents for large-scale video games. We consider a class of games for which expert knowledge is available to define a state quality function that gives how close an agent is from its objective. The decision making policy is based on a confidence measurement on the growth of the state quality function, computed by a supervised learning classification model. Additionally, no stratagems aiming to reduce the action space are used. As a proof of concept, we tested this simple approach on the collectible card game *Hearthstone* and obtained encouraging results.

1 INTRODUCTION

The vast majority of computer games feature a mode where the player can play against artificial intelligence (AI). Back in the eighties with *Pac-Man* for instance, the avatar was already chased by little ghosts. Building an AI for them was not very complicated given their restrained environment and goal. However, many papers (see e.g. (Safadi et al., 2015)) have underlined the fact that as games continue to feature richer and more complex environments, designing robust AI – an AI capable to adapt its behavior to any possible game situation – becomes increasingly difficult.

Traditionally, game agents are built in a scripted way, meaning that the developers hard-code the way these agents recognize and react to different situations. This old-fashioned approach is no longer relevant for modern games as it is often too complicated to be applied to them in relation to the time and resources developers are given for developing a game. Unfortunately, many video game companies continue to stick to this approach. As a result, the games' agents have started to fall behind compared to the complexity of the games' environments, and the players are left empty-handed when it comes to measuring themselves against a challenging autonomous agent.

Indeed, the common way to solve the problem of the scripted AI – too overwhelming to be applied – is to reduce the number of strategies an autonomous agent can exhibit. Moreover, as every game state cannot be considered in advance, numerous variables specific to the game have to be discarded for the quantity of predefined situations a scripted agent can recognize to remain tractable. Altogether, these shortcuts make the designed agent redundant where decision making is concerned. It means that the average player is quickly able to recognize the agent's strategies when it plays, removing all the fun and challenges the player could experience by being surprised by the moves of the AI.

Fortunately, with the advent of the machine learning era, we could be at the dawn of finding techniques where AI in games is no longer coded by humans, but rather where the AI has *learned* to play the game in the same way any human player would, through experience. This modern approach, nevertheless, does not come without its challenges, as for example large-scale video games feature complex relations between

Taralla, D., Qiu, Z., Sutera, A., Fonteneau, R. and Ernst, D.
Decision Making from Confidence Measurement on the Reward Growth using Supervised Learning - A Study Intended for Large-scale Video Games.
DOI: 10.5220/0005666202640271
In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 2*, pages 264-271
ISBN: 978-989-758-172-4

the entities they manage, leading to difficulties for simulating side-trajectories in the course of a game. The number of variables necessary to fully describe the game state is another example of such challenges.

In this context, this paper proposes a few advances for the generic design of autonomous agents in large-scale video games. This approach is based on learning classification with extremely randomized trees (or *ExtraTrees*). In our setting, the quality of the next state is incomputable because the system dynamics is unavailable (see Section 3.1). However, we are given a dataset composed of noised realizations of this dynamics. An ExtraTrees model is trained on this data to predict the evolution of the quality of the next state based solely on the current state and an available action. Finally, the agent's policy is greedily defined in the confidence the ExtraTrees model has in predicting a positive quality growth for the available actions given the current state.

We study the case of *Hearthstone* because it exhibits the large-scale challenges we mentioned earlier, namely the intractability of simulating side-trajectories – the complexity of dependencies between the game entities makes this an awkward task – and the difficulty of working with game states featuring numerous complex variables.

The paper is structured as follows: Section 2 briefly reviews related work on artificial intelligence for video games. Section 3 states the learning problem and Section 4 describes the proposed solution, along with one application to the game *Hearthstone*. Section 5 presents and discusses experimental results. Finally, Section 6 concludes and gives paths to follow for future work.

2 RELATED WORK

2.1 Research on Video Games

For several years now, the video game field has been the center of many research works. This research is motivated by the development of new technologies to increase entertainment values, but also by the fact that video games present themselves as alternate, low-cost yet rich environments to assess the performance of machine learning algorithms.

As mentioned by (Gemine et al., 2012), one or two principal objectives are usually pursued in video game AI research. The first is to create agents that exhibit properties making them more amusing to play with, where the second is about patching complex games for which challenging agents do not yet exist. For those complicated games, the goal of course is to increase the performance of the AI. In both cases, the general idea can be summarized as obtaining an AI whose performance is similar to that of humans.

Human-like behavior in video games has already been approached in several studies, often under the name *Imitative Learning*. Such studies include, for example, improvements in more natural movement behavior and handling weapon switching, aiming and firing in FPS (First Person Shooter) games (Bauckhage et al., 2003; Gorman and Humphrys, 2007).

2.2 Machine Learning and Video Games

In recent years, MCTS has been seen as the method of choice for creating AI agents for games (van den Herik, 2010). In particular, it was successfully applied to Go (Lee et al., 2009; Rimmel et al., 2010) and a wide variety of other game environments (Browne et al., 2012). Attempts to apply MCTS and other simulation-based techniques to games that are difficult to simulate also exist, for example in the RTS genre (Soemers, 2014; Sailer et al., 2007) or *Magic: The Gathering* (Ward and Cowling, 2009; Cowling et al., 2012), a card game similar to *Hearthstone*. These approaches however usually solve subproblems of the game they are applied to, and not the game itself, through the abstraction of many variables.

Alternatively, by using neural network-based classifiers, Bayesian networks and action trees assisted by quality threshold clustering, some researchers concerned about AI performance issues in large-scale video games were able to successfully predict enemy strategies in StarCraft (Frandsen et al., 2010).

Machine learning techniques are, however, not restricted to game agent application. With the emergence of massively multiplayer games for instance, game editors have begun to integrate those techniques into their matchmaking and player toxicity detection algorithms (see e.g. *League of Legends* (Riot Games, 2006)).

3 PROBLEM STATEMENT

3.1 Hypothesis

Games usually feature numerous objectives, like winning the game, minimizing the damage taken by a certain character, kill another agent, etc. For some of these objectives, state quality functions can be defined

in every state from expert knowledge, such that they measure how close an agent is from the corresponding objective. These objectives constitute the problems to which this approach is applicable.

Additionally, modern games have large and diversified action spaces, and it is difficult to simulate side-trajectories in the course of a game. Thus, given any game, we place ourselves in the context where (1) on-line action simulations are not possible to establish what consequences these have on the game (i.e. the system dynamics are unavailable) and (2) no stratagems aiming to reduce the action space are used.

3.2 State Vector

We define by $\mathcal{S}$ the set of all state vectors. A state vector $s \in \mathcal{S}$ contains data describing the state of the game at a given time.

The game state is modified when an action is executed. This action might be taken explicitly by an agent, or triggered by an event. The next section describes how we represent such actions.

3.3 Action Vector

The set of actions that could occur in the targeted game is considered to be unknown. In this way, the framework we develop does not depend on the kind of actions at hand by any means, nor in fact on a class of game in particular.

Formally, we define by $\mathcal{A}$ the set of actions that can be taken in a game. Note that, in some games, not all actions can be taken in a given state $s \in \mathcal{S}$. We will thus further define

$$\mathcal{A}_s := \{a \in \mathcal{A} \mid a \text{ can be taken in state } s\}.$$

3.4 State Quality Function

We assume that we know a bounded function

$$\rho : \mathcal{S} \to \mathbb{R}$$

associating to a given state $s \in \mathcal{S}$ a score representing the quality of s with respect to the agent's objective in this state, whatever this objective is. For *Hearthstone* for example, some expert players helped us design a ρ function indicating how much good a position an agent is in to win given current state information.

It should be noted that the value of $\rho(s)$ could additionally depend on information memorized in previous states.

3.5 Problem Formalization

We can see games as discrete-time, stochastic systems whose dynamics can be formalized as follows:

$$\tau : (s,a,s') \mapsto P(s' \mid s,a), \quad t = 0,1,...$$

with $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}_s$.

As highlighted in Section 3.1, we make the assumption in this paper that the system dynamics are not available. Instead, we are given a dataset $\mathcal{D}$ of noised realizations of these dynamics.

From $\mathcal{D}$, a two-class classification model is trained to predict, for a given (*state*, *action*) pair (s,a), whether the expected difference

$$\mathop{\mathbb{E}}_{s'}\left[\rho(s') - \rho(s)\right] = \sum_{s'}\left[\tau(s,a,s')\rho(s')\right] - \rho(s)$$

is positive or not. Moreover, this classification model comes with a confidence function on the affiliation of an object (s,a) to the positive class, written $C_{\rho,\mathcal{D}}(s,a)$. From this confidence function, we define our decision making strategy as

$$h(s) = \mathop{\mathrm{argmax}}_{a \in \mathcal{A}_s} C_{\rho,\mathcal{D}}(s,a).$$

The policy h defined hereinabove is thus *greedy in* $C_{\rho,\mathcal{D}}$.

4 LEARNING ARCHITECTURE

4.1 Obtaining a Dataset

The process of generating the database can be as simple as letting random players play the game for some time, and making them explore the space of available state-action combinations. Indeed, when a random player chooses to take action a in state s, it can save the value of $\rho(s)$, execute a to arrive in s' and compute $\rho(s')$. The tuple $(s,a,\mathrm{sgn}[\rho(s') - \rho(s)])$ can then be inserted into $\mathcal{D}$. How to deal with samples whose $\rho(s') - \rho(s)$ evaluates to zero is specific to the targeted application – They can either be included in one of the two classes, or simply not included in $\mathcal{D}$ at all.

4.2 ExtraTrees Classification

Considering the targeted application, which is predicting the soundness of taking a certain action based on a large number of features, there was a requirement to use a robust learning algorithm able to deal with the possible idiosyncrasies of ρ while ensuring it handled the numerous features correctly. To this end,

the decision to use random tree-based ensemble methods was taken. Decision trees are inherently suited for ensemble methods (Sutera, 2013), and in particular ExtraTrees are quite efficient compared to other tree methods (Geurts et al., 2006).

Nevertheless, the main caveat of using this kind of classifier is the memory it requires. Indeed, because the games' complexity is not bounded in our theory, it may not be assumed that the learned models will always fit in the computer memory as the trees might become arbitrarily complex. One should therefore take care to limit the tree growing by tuning the algorithm parameters correctly and to include a tree-pruning pass.

4.3 Assessing the Classifier Performance

When it comes to classifiers, it is well-known that the accuracy (i.e., the fraction of successfully classified samples in a given test set) is not a good measure to assess a model generalization performance (Provost et al., 1998). It is recommended when evaluating binary decision problems to use the Receiver Operating Characteristic (ROC) curve, describing how the fraction of correctly classified positive samples varies with the fraction of incorrectly classified negative samples. Nonetheless, ROC curve analysis might overestimate the performance of a binary classifier if there is a large skew in the class distribution (Davis and Goadrich, 2006). Because no constraints were put on the given dataset $\mathcal{D}$, one cannot rely on the fact that it provides a balanced number of positive and negative samples.

This is why along ROC curves analysis we validate our binary classifier with the help of Precision-Recall (PR) curves. PR curves have been mentioned as an alternative to ROC curves for tasks with a large skew in the class distribution (Craven, 2005; Bunescu et al., 2005; Goadrich et al., 2004). Indeed, when the proportion of negative samples is much greater than that of the positive ones, a large change in the fraction of false positives can lead to a minimal change in the false positive rate of the ROC analysis, because they are underrepresented in the test set. Precision analysis on the other hand relates false positives to true positives rather than true negatives, and therefore does not present the same flaw as ROC analysis in the case where negative samples are overrepresented in the test set. Therefore, analyzing both graphs to assess the quality of the model is necessary.

To classify a sample (s,a), the model yields the confidence $C_{\rho,\mathcal{D}}(s,a)$ it has on classifying the sample in the positive class. This confidence is computed as the mean predicted *positive* class probability of the trees in the forest. The predicted *positive* class probability of an input sample in a single tree is the fraction of *positive* learning sample cases in the leaf the input sample falls in. This way of classifying samples means that the performance of the model will depend on the probability threshold used to predict whether a sample is positive or not. Each point on an ROC or PR curve is the performance of the model for a given threshold; thus, according to the needs of the targeted application, one can select the confidence threshold c for which is considered the best compromise between the true positive rate, false positive rate and precision. The best trade-off is defined by the targeted game and goal as, for instance, missing an opportunity (predict a false negative) is not necessarily worse than making a mistake (predict a false positive) in all games.

4.4 Selecting a Good Action

Figure 1 is an activity diagram describing the action selection process. From now on, when an agent in state $s \in \mathcal{S}$ is presented the set of available actions $\mathcal{A}_s = \{a_1, ..., a_n\}$, it simply has to use its classification model to evaluate $C_{\rho,\mathcal{D}}(s,a_i)$, $\forall i$ s.t. $a_i \in \mathcal{A}_s$. It then extracts from $\mathcal{A}_s$ the actions a_i such that $C_{\rho,\mathcal{D}}(s,a_i) \geq c$, which produce a set of valuable actions to take. The agent can then choose the one its classifier finds the most probable to trigger a growth of ρ. In the case where the extracted action set is empty, the agent should decide to do nothing.

On a side note, for some objectives it might be more relevant to obtain the order of the available actions and to execute them all in this order instead of discriminating whether they are bad or good ones. In this case, the value of parameter c can be set to zero. However, the study of ROC and PR curves should still be conducted to assess the quality of the classifier.

4.5 Application

4.5.1 Why *Hearthstone*

The framework developed above was chosen to be applied to the turn by turn, 2-player collectible card game *Hearthstone: Heroes of WarCraft*, (Blizzard Entertainment, 2014). The objective of the agent is to win the game by applying a well-known strategy of the field (namely *board control*).

Hearthstone exhibits challenges that are commonly encountered in large-scale video games, such as the impracticality of simulating side-trajectories on-line – the complexity of dependencies between the

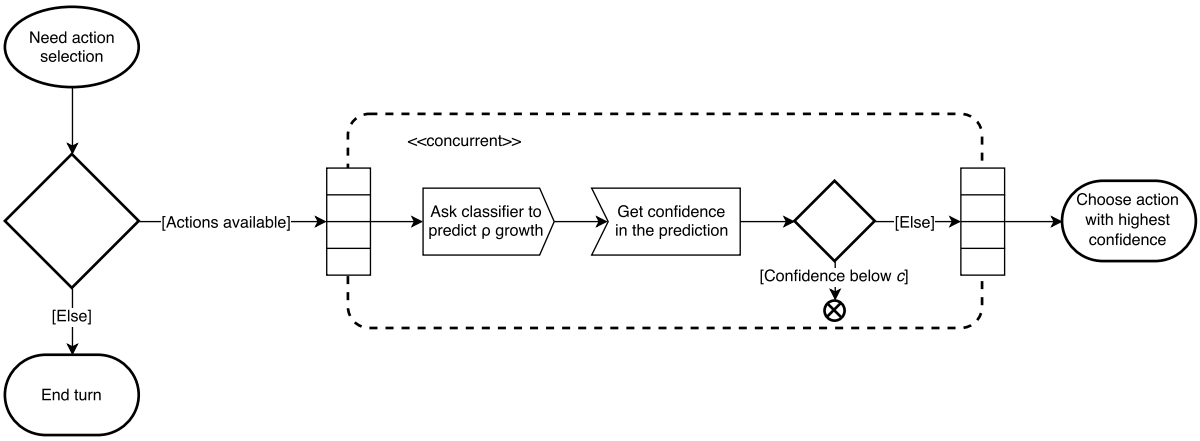

Figure 1: The action selection process.

game entities makes this a hard task – and the difficulty of working with game states featuring numerous, complex variables – card identifiers, characteristics of the cards on the board,... This particular game is thus a fair choice to assess the viability of the presented approach.

4.5.2 *Hearthstone* Basics

The game features a battle between two players, each owning a deck of 30 cards. Each player has 30 health points, and their goal is to reduce the enemy's health to zero. Every turn, a player draws a card from his deck and can play some from his hand, which is hidden to his enemy. Playing a card reveals it to the opponent and costs some *crystals*, available in small quantities each turn. We distinguish two kinds of cards: on the one hand there are *spells*, with special effects, and on the other there are creatures (or *minions*). A minion brought into play is dropped on its owner's side. Minions can attack once per turn and remain in play as long as they are alive. Minions can feature special effects as well.

About the game configuration used in this work, we restrained ourselves to a subset of cards from the original game.

4.5.3 Three Binary Classifiers

Because of the diversity of the possible actions (or *decisions* to take) in *Hearthstone*, action representation is another challenge. Indeed, some need features that are irrelevant for others, so unless dealing with missing values in feature vectors representing those decisions, they cannot all be represented with the same structure.

Consequently, we could not directly apply the framework developed. A *divide and conquer* approach was taken, breaking this problem into subproblems for which the theory would be applicable. The division would be made on the action space: we separate it in disjoint subspaces where each subspace contains actions of the same type. The key aspect is that all actions belonging to the same subspace would use the same vector representation.

There are three kinds of decisions an agent in *Hearthstone* might have to make at any time.

Play a Card. This decision needs features describing the card to play.

Select a Target. When playing a card, typically a spell, it is sometimes necessary to select a target. This decision requires features describing the target character *and* the effect to be applied to it.

Attack with a Minion. This decision requires features describing the minion that performs the attack *and* the target character of this attack.

This divide and conquer approach means that not one but three binary classifiers will be required for the supervised learning architecture.

4.5.4 Practical Application

Algorithms. We used the Scikit-Learn library (Pedregosa et al., 2011) for the supervised learning algorithms. We also used this library to get the ROC curves of our models. A C++ library was developed to simulate *Hearthstone*, with a companion program responsible of testing the agent against random, scripted and human players.

Another multi-threaded program dynamically linked to the *Hearthstone Simulator* library is able to simulate a large number of games simultaneously and uses the logging capability of the library to generate training and test sets.

We made the codes of these programs and library publicly available.

Datasets. We generated vast datasets by simulating 640,000 games with random players playing with random decks. The card set however was fixed and contained 56 different cards, among which 40 minions and 16 spells.

The sample sets used to train the classifiers were built with exactly 400,000 positive samples and the same amount of negative ones. Samples for which $\rho(s') - \rho(s)$ would evaluate to zero were excluded from the training sets.

ROC/PR Analysis To assess the quality of the trained models, an ROC/PR study was conducted. These curves can be obtained from a trained classifier by using a representative test set. The test set we used was distributed exactly as the training set, with the same size, but with different random seeds to the ones used for generating the training set. With a ROC/PR analysis, the objective is to maximize the *Area Under the Curve* (AUC) for both the PR and ROC curves (Hanley and McNeil, 1982). Because the training and test sets are distributed perfectly evenly between positive and negative samples, the PR curve provides the same information as the ROC curve (the PR study generally highlights that the accuracy is overestimated because of an unbalanced test set (Davis and Goadrich, 2006)); this is why we only conducted an ROC analysis.

The objective has been already well achieved: the AUCs of the ROCs obtained for the *play*, *target* and *attack* classifiers were 94.48%, 99.07% and 94.15% respectively, which is an excellent result for a prototype. Indeed, let us underline that these curves were obtained with a minimal amount of work done on the vector representation of states and actions and on the state quality function design. The same holds for the extremely randomized trees classifiers' parameters, which were simply set to the default values recommended by (Geurts et al., 2006). Only a hundred trees were used for each classifier.

Thresholding. Remember that the classifier outputs *class affiliation probabilities*, or in other words its confidence in classifying a sample in the positive class. Thresholding can be used to discriminate between what is considered positive and what is not.

At first, one could use the ROC curves to determine the best confidence threshold such that most of the judgment errors can be avoided. However, when using thresholding, results were mediocre. Nevertheless, the lower the threshold the higher the performance of the agent, up to the maximum when the threshold was set to zero. Indeed, usually in *Hearthstone*, it is better to miss an opportunity than to mistakenly do something. This explains why the win rate grows with the reduction of the threshold: with a smaller threshold, the precision (fraction of true positives that are not missed) is bigger at the expense of a higher false positive rate. The number of missed opportunities thus decreases.

Memory Usage. Because extremely randomized trees classifiers' parameters were the default ones, those classifiers were built unnecessarily large – approximately 14 gigabytes of RAM are required to hold the three classifiers in memory. However, this flaw should be considered in light of the fact that this application is just a proof of concept; a proper tuning of the algorithms should increase the performance and resource usage.

5 EXPERIMENTAL RESULTS

5.1 Methodology

The resulting agent was up against:

1. a random player which plays cards and makes its minions attack randomly, choosing to end its turn only when no other actions are available;
2. a scripted player which implements a medium-level strategy.

The agent and its opponents shared the same deck composition to prevent decks differences from biasing the results. For each opponent, 10,000 games were simulated. For the sake of comparison, a random player also faced the scripted player using the same protocol. We chose to have a simulation of 10,000 games because we empirically showed that the win rate had converged past this value. Figures 2 and 3 confirm this behavior, for the agent against both the random and scripted players respectively.

Additionally, the ExtraTree classifiers depend on a random seed given at the beginning of their training. To check the influence of the random seed on the quality of the results, we trained four more classifier sets on the same database with distinct random seeds. These four classifier sets were then subjected to the same experiment as the original agent and displayed differences that were not significant (less than 1%, absolute error).

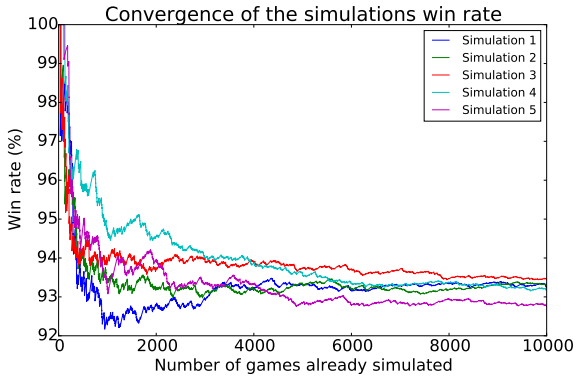

Figure 2: Convergence of the simulation win rates of the agent against the random player. Each curve represents the win rate at the different moments of the simulation. Sufficient convergence is attained near 10,000 games.

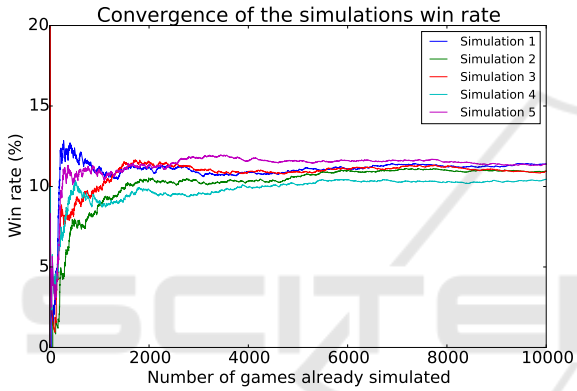

Figure 3: Convergence of the simulation win rates of the agent against the scripted player. Each curve represents the win rate at the different moments of the simulation. Sufficient convergence is attained near 10,000 games.

5.2 Win Rate

The details of the win and loss rates are presented in Table 1. As a comparison, this table also shows the win and loss rates of a random player playing against the scripted player: the agent has 92.8%-win rate against the random player and 10.5% against the scripted one, whereas the random player only obtained a 0.934%-win rate against this same scripted player. This last result proves that the agent learned to develop some reasoning the random player did not. In particular, we measured the participation of the classifier responsible for target selection by replacing it with random target selection. Notably, the agent win rate against the random player dropped to about 60%. This last result highlights what was already indicated above: the agent learned to make valuable decisions.

Table 1: Average win rates observed over five simulations with distinct random seeds, on 10,000 games each. "A" stands for the agent, "R" for the random player and "S" for the scripted one. The remainder of each line is the average tie rate.

	Win rate	Loss rate
A vs. R	92.8%	7.034%
A vs. S	10.5%	89.36%
R vs. S	0.934%	99.0%

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented a generic approach to design prototypes of intelligent agents for complex video games. This approach has been applied to *Hearthstone*, a popular two-player and partially observable card game. The results were quite encouraging considering the simplicity and naivety of the provided solution. They clearly showed that the agent designed following our framework had learned to apply some strategies a random player did not, even though the action and state space were huge. However, in absolute terms, the resulting agent performs pretty badly, so this agent would not be of any use in a real setting. Indeed, the agent wins approximately 93% of the games against a random player, but only 10% of the games played against a medium-level scripted player. Even though the latter result is poor in absolute terms, it has to be compared to the win rate a random player obtains against the same opponent, which is less than 1%. With this in mind, it is clear that the trained agent succeeded in extracting at least a few valuable moves in a wide variety of states from a dataset composed of thousands of random moves. From a data mining perspective, this result is by itself encouraging: it confirms that it is possible to learn strategies from complex datasets, even with simple and naive techniques like move classification. Furthermore, we draw the attention of the reader to the fact that for two players playing with the same decks, the reference point for assessing the performance of an agent is 50% and not 100%: the agent is stronger than its opponent above this value, weaker below, and of comparable skill when equal.

However, this proof of concept might not be easily applicable to all games, thus future work will attempt to make our approach of defining the representation of actions, states and the ρ state score function more generic regarding other large-scale video games. ExtraTrees for example, besides being efficient algorithms, also bring a lot of information about fea-

ture importance and thus feature selection (Breiman, 2001), that can be extremely valuable when designing the state and action vectors. It would also be of interest to assess the suitability of other algorithms such as deep neural networks in place of extremely randomized trees.

ACKNOWLEDGEMENTS

Raphael Fonteneau is a postdoctoral fellow of the F.R.S.-FNRS from which he acknowledges financial support. Antonio Sutera is a PhD fellow of the FRIA from which he acknowledges financial support.

REFERENCES

Bauckhage, C., Thurau, C., and Sagerer, G. (2003). Learning human-like opponent behavior for interactive computer games. In *Pattern Recognition*, pages 148–155. Springer.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43.

Bunescu, R., Ge, R., Kate, R. J., Marcotte, E. M., Mooney, R. J., Ramani, A. K., and Wong, Y. W. (2005). Comparative experiments on learning information extractors for proteins and their interactions. *Artificial intelligence in medicine*, 33(2):139–155.

Cowling, P. I., Ward, C. D., and Powley, E. J. (2012). Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(4):241–257.

Craven, J. B. M. (2005). Markov networks for detecting overlapping elements in sequence data. *Advances in Neural Information Processing Systems*, 17:193.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM.

Frandsen, F., Hansen, M., Sørensen, H., Sørensen, P., Nielsen, J. G., and Knudsen, J. S. (2010). *Predicting player strategies in real time strategy games*. PhD thesis, Masters thesis.

Gemine, Q., Safadi, F., Fonteneau, R., and Ernst, D. (2012). Imitative learning for real-time strategy games. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 424–429. IEEE.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63(1):3–42.

Goadrich, M., Oliphant, L., and Shavlik, J. (2004). Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction. In *Inductive logic programming*, pages 98–115. Springer.

Gorman, B. and Humphrys, M. (2007). Imitative learning of combat behaviours in first-person computer games. *Proceedings of CGAMES*.

Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36.

Lee, C.-S., Wang, M.-H., Chaslot, G., Hoock, J.-B., Rimmel, A., Teytaud, F., Tsai, S.-R., Hsu, S.-C., and Hong, T.-P. (2009). The computational intelligence of mogo revealed in taiwan's computer go tournaments. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(1):73–89.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Provost, F. J., Fawcett, T., and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In *ICML*, volume 98, pages 445–453.

Rimmel, A., Teytaud, F., Lee, C.-S., Yen, S.-J., Wang, M.-H., and Tsai, S.-R. (2010). Current frontiers in computer go. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):229–238.

Safadi, F., Fonteneau, R., and Ernst, D. (2015). Artificial intelligence in video games: Towards a unified framework. *International Journal of Computer Games Technology*, 2015.

Sailer, F., Buro, M., and Lanctot, M. (2007). Adversarial planning through strategy simulation. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 80–87. IEEE.

Soemers, D. (2014). Tactical planning using mcts in the game of starcraft1. Master's thesis, Maastricht University.

Sutera, A. (2013). Characterization of variable importance measures derived from decision trees. Master's thesis, University of Liège.

van den Herik, H. J. (2010). The drosophila revisited. *ICGA journal*, 33(2):65–66.

Ward, C. D. and Cowling, P. I. (2009). Monte carlo search applied to card selection in magic: The gathering. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 9–16. IEEE.