Decision Making from Confidence Measurement on the Reward Growth using Supervised Learning

David Taralla, Zixiao Qiu, Antonio Sutera, Raphael Fonteneau and Damien Ernst

dtaralla@ulg.ac.be

Abstract

Video games have become more and more complex over the past decades. Nevertheless, artificial intelligence (AI) in these games is usually still hard-coded and this is a difficult task given the problem complexity. In order to ease this process, recent research works started to develop learning agents. We investigated this track by developing a framework for the generic design of autonomous agents for large-scale video games based on a state quality function. 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. No stratagems aiming to reduce the action space are used. As a proof of concept, we tested this approach on the collectible card game *Hearthstone* and got encouraging results.

Problem Statement

Hypothesis & Constraints

- **State quality function available.** For some game objectives like winning, killing another agent, etc., 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.
- **The system dynamics are unavailable.** Modern games have large and diversified action spaces that have a prohibitive amount of consequences. The cost of simulating side-trajectories in the course of a game being too high, we assume that the dynamics are unavailable.
- **Original action space.** It is difficult to aggregate actions by nature or function in a general way as modern video games have large and diversified action spaces. The action space of a game is thus left untouched.

State Quality Function

A *state quality function* is 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.
NB: $\rho(s)$ could additionally depend on information memorized in previous states.

Problem Formalization

Games can be seen as discrete-time, stochastic systems whose dynamics are formalized as:

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

with $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}_s$ (the set of available actions in state s).

We make the assumption 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

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

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

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

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

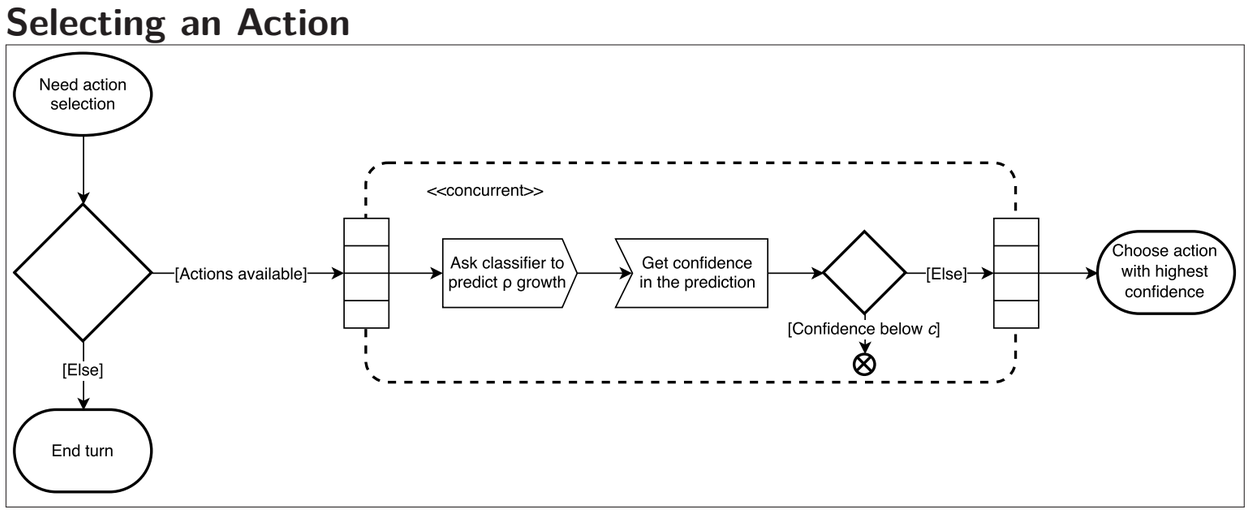
Learning Architecture

Obtaining a Dataset

Let random agents explore the space of available state-action combinations. Indeed, an agent in state s can

1. compute and save the value of $\rho(s)$,
2. take a random action $a \in \mathcal{A}_s$ which leads it to state s',
3. compute $\rho(s')$,
4. insert the tuple $(s, a, \text{sgn}[\rho(s') - \rho(s)])$ in $\mathcal{D}$.

Selecting an Action



Application to *Hearthstone*

Game 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 put 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.

Three Binary Classifiers

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 needed to select a target. This decision requires features describing the target character *and* the effect to be applied on it.
- **Attack with a minion.** This decision requires features describing the minion that performs the attack *and* the target character of this attack.

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

Experiments

The resulting agent was faced 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.

	Win rate	Loss rate
Agent vs. Random	92.8%	7.034%
Agent vs. Scripted	10.5%	89.36%
Random vs. Scripted	0.934%	99.0%

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 has also faced the scripted player using the same protocol.

The details of the win and loss rates are presented in the table. 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 0.934% against this same scripted player. **This last result proves that the agent learned to develop some reasonings the random player did not**. In particular, we measured the participation of the classifier responsible of 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 said above: the agent learned to take valuable decisions.

Conclusion

We presented a generic approach to design intelligent agents for complex video games. This approach has been applied to *Hearthstone*, a popular two-player and partially observable card game. The results clearly showed that the agent designed naively following our framework had learned to apply some strategies a random player did not. Indeed, the agent **wins approximately 93% of the games against a random player, and 10% of the time against a medium-level scripted player**. The latter result might seem poor, but it **has to be compared** to what a random player obtains against the same opponent, which is **less than 1%**. This confirms that the trained agent succeeded in extracting at least a few valuable moves in a large variety of states, from a data set composed of thousands of random moves. From a data mining perspective, this result is by itself encouraging.

Acknowledgements: R. Fonteneau is a postdoctoral fellow of the F.R.S.-FNRS (Belgium). A. Sutera is a PhD fellow of the FRIA (Belgium). This work is supported by the IUAP DYSCO, initiated by the Belgian State, Science Policy Office.