Configurable Agent With Reward As Input: A Play-Style Continuum Generation

Pierre Le Pelletier de Woillemont
Ubisoft, Player Analytics France
Sorbonne Université, CNRS, LIP6, F-75005
Paris, France
pierre.le-pelletier-de-woillemont@ubisoft.com

Rémi Labory
Ubisoft, Player Analytics France
Montreuil, France
remi.labory@ubisoft.com

Vincent Corruble
Sorbonne Université
CNRS, LIP6, F-75005
Paris, France
vincent.corruble@lip6.fr

***Abstract*—Modern video games are becoming richer and more complex in terms of game mechanics. This complexity allows for the emergence of a wide variety of ways to play the game across the players. From the point of view of the game designer, this means that one needs to anticipate a lot of different ways the game could be played. Machine Learning (ML) could help address this issue. More precisely, Reinforcement Learning is a promising answer to the need of automating video game testing. In this paper we present a video game environment which lets us define multiple play-styles. We then introduce *CARI*: a Configurable Agent with Reward as Input. An agent able to simulate a wide continuum range of play-styles. It is not constrained to extreme archetypal behaviors like current methods using reward shaping. In addition it achieves this through a single training loop, instead of the usual one loop per play-style. We compare this novel training approach with the more classic reward shaping approach and conclude that CARI can also outperform the baseline on archetypes generation. This novel agent could be used to investigate behaviors and balancing during the production of a video game with a realistic amount of training time.**

***Index Terms*—Reinforcement Learning, Reward Shaping, Play-style, Video Game, Multi-objectives agent**

I. Introduction

In the board and video game realm, the main goal of Reinforcement Learning (RL) has been to achieve superhuman performances, on board games [1], simple video games [2] or even highly complex ones [3]. However, another interesting application for RL is the pursuit of human decision modeling. In addition to learning how to win, this approach tries to solve the problem of playing like a human player. Playing like a human is not only an issue of performance level but it adds a new dimension to the problem: the play-style. With modern video games being more complex, the number of different play-styles can be significant. The simulation of various play-style, or play personas [4], has gained interest lately, partly due to their usefulness for the video game designers.

Having such an agent, that takes into account a play-style, can be of great value to the game designers, as it can be used to test the game in an automated way and give them precious feedback about whether their intentions are actually translated into the game. It can for instance be useful to check if the difficulty of certain parts of the game are as intended. Even for game difficulty assessment it is useful to distinguish between performance (i.e. success) and style of play. For example a game can be perceived as very hard for a subset of the players because of their style of play, not because of the game itself. This kind of feedback can be of immense value to the designers.

This is not the first time RL is being used for game balancing : [5] and [6] used RL trained agents to dynamically adjust the difficulty of an opponent. In order to provide the game designer with such an agent for game balancing purposes, one must incorporate the different play-styles in this model. The idea of using trained agent to automate play-testing has been explored in depth by Holmgård et al. through procedural personas generation [7], [8], [9]. In their work they mostly focus on archetypal agents and their alignment with players' decisions. In addition they also showcase how those stereotypical play personas can be used to improve content generation. The novelty of our approach is that through a single training phase, instead of one per play-persona, we produce an agent that is configurable with respect to the reward function in a very granular and interpretable way. It is done by giving the agent the reward function coefficients as input. We call this approach the Configurable Agent with Reward as Input (*CARI*). This allows in turn not only to generate caricatural behaviors, but to query any play-style possible and thus create a wide continuum of play-styles to easily be sampled from. It is granular because of the level of details with which a given play-styles can be simulated. It is also interpretable because of the fact that for any play-style, its definition is based on the reward coefficient values, each having very interpretable definition.

In this paper we give first a quick overview of the existing approaches that aim to solve the problem of learning various personas and those aimed at modeling human decision. We then present two approaches to inject information about play-styles into the model without having human data at hand. Both relying on reward shaping, the first relies on training with a fixed reward function, and serves as a baseline. The other approach is based on a dynamic reward function which changes throughout the training. This agent, called CARI, is configurable with regard to the play-style and the performance objectives, not constrained to solely archetypal personas and is obtained through a single training phase.

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

II. Background

A. Play Personas – Play-Styles

Canossa and Drachen [4] adapt the "persona" framework introduced by Alan Cooper [10] in the field of Human Computer Interaction. They defined personas within a video game as the expression of the persona but within the limited space of a specific video game and called it *play personas*. Tychsen and Canossa [11] make the distinction between play mode, play-style and play-personas. It is a distinction based on the level of data aggregation. Play-mode "refers to the behavior of a player with respect to one or a few discrete metrics, within the same overall group or type of metrics". From there play-style is defined as "a set of composite play-mode". And finally play-personas represent the "larger-order patterns that can be defined when a player uses one or more play-styles consistently". Our work here focuses on play-styles.

Not all play-styles perform at the same level. In the case of a game of infiltration for example, a player who likes to rush in and charge head-on will most likely not perform well. Those observed play personas result from the analysis of quantitative players' data gathered via telemetrics [11], for instance via clustering algorithms, and indicate the ways the game is played. Most games today, especially the ones coming from major studios, have built-in tracking mechanisms allowing some form of clustering on the players to be done, based on their behavior within the game. It can be done on key features [12] or even on sequence-based inputs [13]. This combination of tracking and clustering is done today on many of the big budget games [14]. It allows designers to know not only *who* plays their game, but also *how* it is being played. In this paper, we suppose we do not have any human data available, and focus solely on generating varying and rich play-styles. We do so to fit with the assumption that the usefulness of such agent is greater during the production phase than after. And because of this assumption we have to assume that we can not rely on human data.

B. Human Decision Modeling

To approximate human decision making, Inverse Reinforcement Learning (IRL) [15] could potentially be used. The 2 main drawbacks of IRL methods in our case, are the absence of available data and the fact that these algorithms assume homogeneity among the trajectories. This homogeneity assumption goes against the idea of generating varying, and therefore heterogeneous, play-styles.

There have been many attempts to tackle this idea of play-styles. To that effect Holmgård et al. use RL [7], evolutionary [8] and MCTS [9] agents. They demonstrated that reward, fitness or utility function shaping is a great way to achieve variety in the style of play of the agents. Moreover they demonstrated that stereotypical play-styles can be a good low-cost, low-fidelity approach to automated play-test [8]. They generate 4 play-styles and show that they bear acceptable resemblances with the players on a simple game. They then generate the same 4 play-styles in a more complex environment [9], showing the coherence and scalability of their approach. To better fit with the language used in the video game industry we use the term *archetypes* instead of stereotypical play-styles. A few issues arise with their approach. The first is the limitation to only 4 archetypes. On more modern and complex games there are more than 4 play-styles and most players do not lie at the edge of the behavioural space but rather are a mix of different aspect of multiple archetypal play-styles, and will likely lie on a continuum between those archetypes. This is simply due to the fact that the more complex the game mechanics, the more numerous the possible *play-modes* and therefore the more numerous the number of play-styles defined from those play-modes. So, fitting all play-styles into 4 archetypes can become unrealistic as the game mechanics become more complex. Even in [7] they note that on one of the play-trace the agent closest to the player is the random one, this could either mean that 4 archetypes is not enough or that relying solely on archetypes is not sufficient. The second limitations of training separate agents is the reward definition for each of them: how to define the play-style part of the reward function and to efficiently balance it with the performance part. Our approach solves both problems by training once and allowing iteration afterward. This allows the practitioner to find the correct balance between play-style and performance to produce meaningful archetypes more easily. It can also be used to generate any behaviors as mix of these archetypes on all dimensions explored in a very granular way.

Holmgård et al. also compared their approach to 'clones' [16], and solve this issue of play-styles variety by fitting one clone per level and per play-style. This solution, while providing good results, is however very impractical in the video game industry where training an agent can take hours or days at a time. Finally, [17] have shown that more classical behavior cloning is very limited in its performance results and that, even though it requires a simulation and a reward function, RL should provide better performance.

C. Simulation Environment

There exist already video games that have been used to train RL algorithms, but none that could really be of use for our particular problem of varying play-styles generation. The most famous one is the Atari suite [18] which, while being very useful due to the vast variety of games, is limited notably by its simplicity: it is not the type of video game where would emerge significantly varied and numerous play-styles. For this to emerge, one needs a more complex game. A good example is StarCraft II [3] where the vast complexity of the game offers many different approaches. Unfortunately the main drawback of this game is that it is actually too complex, requiring a vast amount of computation, and very long training time. We needed something complex enough to be able to have different play-styles and some of the challenges that come with training in a complex modern video game, but simple enough to have moderate computation requirements.

This is the reason why we developed our own video game environment of a discrete, turn based, shooter-strategic video

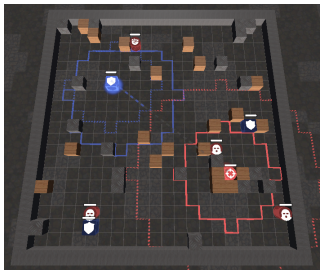


Fig. 1: Screenshot of the environment

game, see Fig. 1. In this video game 2 teams fight to the death on a 2D square based board with covers laid randomly. One team, composed of 3 heroes, is controlled by the agent trying to learn, and the other team is controlled by hand-crafted decision functions, designed by us like any Non Player Character (NPC) would be in most video games.

III. Proposition

Here we compare two approaches to create agents that can play the game well – meaning achieve a certain number of wins, but can also play it in a specific fashion. The first, very straightforward one is the simple shaping of the reward so as to orient the agent toward a single given playstyle, it serves as our RL baseline. CARI, the novel one, on the other hand is based on a reward changing episode to episode. Allowing for a single model that is configurable with respect to the plays-style in a very granular way. It relies on having the coefficients of the reward function given as input, in the state description. The two approaches are then tested on their ability to generate 7 archetypal play-styles. The word *archetypes* is used to define these extreme stereotypical behaviors, so that an archetype plays with a caricatural style. The archetypal play-styles focused on can be each summarized as follows:

- Sniper (**Sn**): prefers the rifle to the knife
- Contact (**C**): prefers the knife to the rifle
- Grouped (**G**): keeps its team tightly grouped together
- Scattered (**Sc**): spread its team as much as possible
- Safe (**Sa**): very adverse to damage taken
- DPS (**D**): focuses on damage inflicted
- Win Only (**W**): cares solely about winning

Notice the Win only archetype. While not being a style of play, per say, it is there to serve as a comparative model, to be aware of what is the maximum performance achievable, with a given RL algorithm and given resources at hand. We focus mainly on 6 archetypes. These 6 archetypes were selected because they are the most commonly observed in such games.

A. Notation

A RL problem is stated as follows. An agent evolves in an environment. Based on the current state at time t: s_t, it chooses an action $a_t = \pi(s_t)$. This action a_t is then applied in said environment, yielding a new state s_{t+1} and a reward signal. This reward signal is usually expressed as a function of the state s: $R_\theta(s) = \sum_{i=1}^{n} r_i * \theta_i(s)$. Where θ denotes the

TABLE I: Reward coefficients and events definitions

Notation (r)	Associated With (θ)
r_{Stab}	Stabbing an enemy
$r_{CvrShooting}$	Shooting on a cover trying to shoot at an enemy
$r_{HeroShoot}$	Shooting at an enemy and actually hitting it
$r_{UsefulShld}$	An enemy tries to hit a hero but hits its shield instead
$r_{NmyDamage}$	An enemy actually deals damage to a hero
$r_{HeroDistance}$	Average distance between heroes at the end of every turn
r_{Win}	Wining a game

TABLE II: Reward coefficients and their respective ranges

r	Min ($r_{.,min}$)	Max ($r_{.,max}$)
r_{Stab}	-1.0	3.0
$r_{CvrShooting}$	-2.0	1.0
$r_{HeroShot}$	-1.0	2.5
$r_{UsefulShld}$	-1.0	2.5
$r_{NmyDamage}$	-3.5	1.0
$r_{HeroDistance}$	-3.5	3.5
r_{Win}	0.0	20.0

n-events which are rewarded and r the n-rewards associated with said events. The most common case in zero sum games is to have the agent being rewarded on 2 occasions: +1 in case of a victory and -1 in case of a defeat. Which translates into: $n = 2$, $\theta = \{win, lose\}$ and $r = \{+1, -1\}$. In this case we go a step further and add r as a parameter of the reward function R. The reward function is then expressed as follow: $R_\theta(r, s) = \sum_{i=1}^{n} r_i * \theta_i(s)$, with again r the reward coefficients and θ the events to associate a reward signal with. The issue of how to define what are these particular events $\{\theta_i\}_{i\in[1:n]}$ rests with the practitioner.

We introduce the theoretical space Φ which represents all the possible play-styles that can be defined using the event defined by θ. Meaning that for any play-style $\phi \in \Phi$, $r^{(\phi)}$ are the reward coefficients associated with this play-style. We also introduce $\Phi_{Arch.} = \{Sn, C, G, Sc, Sa, D, W\} \subset \Phi$ which are the archetypal play-styles defined earlier. So, for example, to simulate a *Sniper* play-style one would select the reward coefficients $r^{(Sn)}$ and use the reward function $R_\theta(r^{(Sn)}, .)$ to train a *Sniper* agent.

The events, rewards and intervals used here are available in Table I and Table II. The min and max values for each reward coefficients will serve later on to define the different archetypes. For simplification purpose we consider these intervals as being discrete, each with a step-size of 0.1, and 1 for r_{Win}. So that for example: $r_{Win} \in \{0, 1, 2, \ldots, 18, 19, 20\}$ and $r_{Stab} \in \{-1.0, -0.9, \ldots, 2.9, 3.0\}$.

It is important to note, at this point, that one needs to select the bounds for each interval based on one's expert knowledge and past experiences. It is a constraint of the approach and should be the focus of future work: develop a methodology to choose these intervals. We do believe this process of interval selection could be automated in the future, using human data. It is also important to note that choosing the correct reward coefficients $r^{(\phi)}$ to induce a given play-style ϕ is a work that needs to be done by the practitioner at some point, for all RL applications, especially to video games. Taking such large

TABLE III: Reward coefficients values for Archetypes play-styles. Note the bold cell, which represent the axes along which a particular archetype is defined.

	$r^{(Sn)}$	$r^{(C)}$	$r^{(Sa)}$	$r^{(D)}$	$r^{(G)}$	$r^{(Sc)}$
r_{Stab}	**min**	**max**	$\frac{1}{4}$max	$\frac{3}{4}$**max**	$\frac{1}{2}$max	$\frac{1}{2}$max
$r_{HeroShot}$	**max**	**min**	$\frac{1}{4}$max	$\frac{3}{4}$**max**	$\frac{1}{2}$max	$\frac{1}{2}$max
$r_{CvrShooting}$	**min**	max	max	max	max	max
$r_{NmyDamage}$	max	max	**min**	max	max	max
$r_{UsefulShld}$	min	min	**max**	min	min	min
$r_{HeroDistance}$	0	0	0	0	**min**	**max**
r_{Win}	$\frac{1}{2}$max	$\frac{1}{2}$max	$\frac{1}{2}$max	$\frac{1}{2}$max	$\frac{1}{2}$max	$\frac{1}{2}$max

The *min* and *max* values refer to the bounds defined in Table II.

intervals as done here, allows the practitioner to select the correct coefficients after the training is done, and not have to retrain every time they want to change the coefficients. This is amongst the main contributions of this work: train once and iterate afterwards on a already trained and configurable model.

B. Baseline – One model per play-style

This approach is the most straightforward. The purpose is to obtain an extremely caricatural behavior, it serves as a RL baseline. Once the archetypes $\Phi_{Arch.}$ are defined, one needs to map those definitions to corresponding reward coefficients $\{r^{(\phi)}\}_{\phi\in\Phi_{Arch.}}$, or more accurately define for each archetype where each of its reward coefficients should sit in the intervals defined earlier (see Table II). Those values, for each of the archetypes, are detailed in Table III.

Once the seven (6 play-styles archetypes and 1 win only) $\{r^{(\phi)}\}_{\phi\in\Phi_{Arch.}}$ are defined, one model is trained per $\phi \in \Phi_{Arch.}$. Thus yielding seven agents: six archetypes and one *over-achiever*. We call this approach the **archetype training** approach.

Note that with this approach one has to carry out one training for each archetype that is targeted. Since *only* 7 archetypes were targeted here, 7 training phases needed to be conducted. It is acceptable in this case, but it is less and less so as the number of archetypes grows, for obvious time and resources constraints. Hence the need to find a way to create similar results but with only one learned model, whatever the number of play-styles. Moreover, if for some reason the practitioner needs to retrain the model (e.g. due to major changes in the environment) then they need to do so six times, this can become very time and resources consuming. Also if there is, and there usually is, an iteration process to find the correct $r^{(\phi)}$ to produce the desired play-style ϕ, this process of re-training the agents can become very tedious.

C. CARI – One Model for all play-styles

An overview of this approach is available in Algorithm 1. Instead of performing multiple trainings, each with fixed reward coefficients $r^{(\phi)}, \phi \in \Phi_{Arch.}$, one single training is carried out with the reward coefficients changing randomly, within their respective intervals (see Table II), each episode. At the beginning of each episode a new value is drawn for each of the reward coefficients (line 5. of Algorithm 1), and

Algorithm 1: CARI training

Result: A single configurable agent (*CARI*) that can adapt its play-style to $r^{(\phi)}, \forall\phi \in \Phi$

1 Define the particular events: $\{\theta_i\}_{i\in[1:n]}$ (see Table I);
2 Define the reward coefficients bounds: $\left\{[r_{i,min}, r_{i,max}]\right\}_{i\in[1:n]}$ (see Table II);
3 Define L the number of episodes to train on;
4 **for** $l = 1$ *to* L **do**
5 $\quad r^{(l)} \sim U(\left\{[r_{i,min}, r_{i,max}]\right\}_{i\in[1:n]})$;
6 $\quad$ Define the following reward function $R_\theta(r^{(l)}, s) = \sum_{i=1}^{n} r_i^{(l)} * \theta_i(s)$;
7 $\quad$ Reset environment;
8 $\quad$ **while** *Episode is not done* **do**
9 $\quad\quad s_t = env_t(state)$;
10 $\quad\quad s_t = [s_t, r^{(l)}]$;
11 $\quad\quad a_t = \pi(s_t)$;
12 $\quad\quad s_{t+1}, R_\theta(r^{(l)}, s_{t+1}) = env_t(a_t)$;
13 $\quad$ **end**
14 **end**

a training episode is run with these rewards coefficients. The 5th line of Algorithm 1 is the equivalent of sampling a random $\phi \in \Phi$ (not just $\Phi_{Arch.}$). One needs to add this change to the information available to the agent. To this effect, we simply append these reward coefficients values $r^{(\phi)}$, as is, to the state of the model Now, the state that the agent takes as input to decide which action to choose is no longer limited to what is going on on the board, but contains also what kind of reward to expect for certain actions, i.e. what the desired behavior is.

By proceeding this way, even though the reward function changes from one episode to the next, it does not actually change with respect to the newly formed state, because the coefficients of this reward function are included in it. By doing so, all the requirements for the algorithm to converge are preserved as the system remains inside a Markov Decision Process (MDP). It is now an MDP in which the objective varies every episode and a model "aware" of this change. It means that solving this newly defined MDP amounts to solving the original MDP for all play-styles $\phi \in \Phi$.

This approach learns, through only one training, to behave as any $\phi \in \Phi$, including the 7 archetypes $\Phi_{Arch.}$. It is capable to access a large continuum repertoire of behaviors. Maybe some players do play like the *Contact* archetype, but it is certain that a lot more players are in between a *Contact* and a *Sniper*, with some degrees of variation. It can also be that some players do play as a combination of a *Contact* and a *Scattered* approach. This method of learning allows to simulate all these cases: the case were the play-style is an archetypal one, the case were the play-style is not an archetypal one and the case in which the play-style is a combination of multiple archetypes. Thus we name this agent the Configurable Agent with Reward as Input: CARI. It is configurable because the values of the reward coefficient, unlike the rest of the state

given to the agent, is given by the practitioner, or even the game designer. By picking a any given $r^{(\phi)}$ one can therefore simulate ϕ. This is the main contribution of this work: generate a continuum of play-styles Φ to choose from, the model is no longer constrained by the extreme behavior. The practitioner is also no longer constrained by having to run multiple training to create multiple play-styles, and multiple training for each play-style ϕ to find the correct reward function $r^{(\phi)}$.

IV. Experimental Setup

A. Game Environment

In this section we describe the game environment in deeper details. As stated before it is a discrete, turn-based, strategic shooter. A team of 3 heroes, controlled by an agent, face off a team of 5 enemies on a 2D square based board, for a maximum of 10 turns. Each hero has the capacity to move, shoot at an enemy, stab an enemy (i.e. melee attack) and apply a shield on itself. The shield will absorb one attack before disappearing, and then it won't be available for 2 more turns. Every character has a few basic defining statistics. The most important ones are its health, its range of movement and its range of fire. So for example during one turn, the agent (controlling the whole hero team) can move around and stab an enemy with hero number 1, then shoot an enemy with hero number 3 and finally put a shield on hero number 2 before skipping the rest of its turn, effectively starting the enemy's turn.

Regarding the enemies, they have available all the actions that the heroes have, apart from the shield which they do not possess. There are 3 types of enemies, distinct based on their *stats*: health, range of move, range of shot and damage. They can be summed up as a high health, low range of movement and high damage (close up fighter), a low health and high range of shot (distance type) and an in-between fighter. The board is a 20x20 square in which 40 covers are spawn randomly at the beginning of each turn. The spawn position of the heroes and the enemies are also random, allowing for a wide range of possibilities to be encountered during training.

The state returned by the environment is two fold: an image-like segmentation map (called "board") of size 20x20 indicating on each cell what object is inside it (which hero, which enemy, a cover or nothing at all) and an array comprising the rest of the information needed (called "general info"): the number of turns left, the current stats of each hero and each enemy (health, range of movement, range of shot, damage and so on). All in all after some preprocessing operations (in which we will not dive into due to space constraints) this state is of size 3804. The actions available for the agent are, for each hero, moving in any direction (diagonal included), shooting and stabbing plus the shield action. The agent also has the "end of turn" action, it sums up to an action space of size 61.

B. ACER

To test our agents, we chose to use the ACER [19] algorithm. There are a few reasons why ACER was chosen. First of, it is a discrete action algorithms which suits the problem well. It is an on and off policy algorithm, allowing for both fast convergence and better use of the data generated. The off-policy part is coupled with a replay buffer, which is prioritized following [20]. Another major reason for choosing ACER is the possibility to run multiple environments in parallel in an asynchronous fashion. This is quite useful for training agents with an environment that is not perfectly stable and could crash. By "running multiple environments in parallel" we do mean using multiple environments to train the same model, each sharing the same weights and populating the same replay buffer.

The actor-critic aspect of ACER is also an advantage. It will allow, in the future, to be able to use supervised learning for pre-training. Having those 2 components could also help in the interpretation and explainability of the model [21]. Say one wishes to use the model to inform game developer about the difficulty of the game. Reporting on the difficulty of the game is one thing, being able to explain *why* is even better. Having an actor-critic architecture allows a wider range of approaches when it comes to explainability.

The neural network architecture used is very straightforward. First, the board is passed through convolutional layers, then it is flatten and concatenated with the general info array and passed through dense layers and finally split in two to output both the policy and Q-value needed. For CARI, the reward function coefficients are added to the state, by simply incorporating them into the general info array as is. The simplicity of this method is also to be noted: it is something that can be applied to any RL problem without much work on the practitioner part.

Note that all rewards are normalized to be bounded by 1. For each training phase done, three environments in parallel (each participating in the training of the same model) were used, each running 20,000 training steps, each composed of a transition of 50 time-steps, training the same model. An episode last, on average, around 220 steps, meaning in total, a full model training lasts around $20000 * 50/220 \sim 4,500$ episodes. The only reason we run for 20,000 training steps is that it is equivalent to 15 hours given our computational setup, which is a reasonable constraint to aim for in future real-life use-case. We wanted to compare models for which the same number of resources (time and computational power) was allocated. The results shown later are to be read as "given a fixed amount of resources ...". Our setup is a single computer with a 12 core CPU and a NVIDIA GeForce GTX 1070 GPU.

V. Evaluation and Results

A. Evaluation Process

For the 1st approach models, i.e. the baseline archetypes agents, once trained, these models are used to run 500 randomly generated games and track a few key metrics. For the 2nd approach, i.e. the CARI agent, after training, we run the same 500 games and track the same key metrics, with the only difference that for this model we do this tracking once with each of the archetypes reward coefficients $\{r^{(\phi)}\}_{\phi \in \Phi_{Arch.}}$ as input. This paper aims at comparing the baseline and the CARI agent. It also aims at comparing them with the

TABLE IV: Mean of the key-metrics for our 3 approaches, averaged over 500 games for each archetype. The cells in bold represent the axis along which the play-style of each archetype is defined.

	Contact			**Sniper**		**Grouped**		**Scattered**		**Safe**		**DPS**		**Win Only**	
Agent	Heuristic	B.	CARI	B.	CARI	B.	CARI	B.	CARI	B.	CARI	B.	CARI	B.	CARI
Stabbings	**8,9**	**13,8**	**13,7**	**1,3**	**0,4**	8,5	8,1	6,1	7,1	6,4	8,5	**9,9**	**7,4**	7,0	7,2
Shots	**6,3**	**6,6**	**5,8**	**12,2**	**12,4**	8,6	10,9	8,1	9,0	11,1	10,3	**8,6**	**10,7**	10,7	10,2
Ratio in Covers	5%	19%	13%	**2%**	**2%**	16%	18%	14%	11%	18%	15%	10%	11%	8%	4%
Heroes Dist.	5,6	7,3	5,6	9,3	8,7	**3,0**	**4,8**	**17,9**	**16,6**	9,0	6,6	9,3	7,4	9,8	7,0
Shields	2,6	3,3	1,8	2,5	0,8	2,5	2,0	2,8	2,1	**3,7**	**3,1**	1,9	0,9	2,6	1,3
Lost HP Heroes	81%	91%	92%	74%	86%	72%	84%	81%	82%	**63%**	**50%**	84%	83%	66%	64%
Lost HP Enemies	75%	90%	88%	95%	91%	84%	94%	75%	86%	90%	96%	**93%**	**97%**	97%	98%
Win	21%	53%	45%	77%	67%	44%	72%	26%	41%	61%	88%	75%	82%	**89%**	**92%**
Lost	47%	31%	34%	15%	29%	19%	15%	27%	20%	15%	9%	20%	15%	9%	6%
Draw	32%	17%	21%	9%	3%	38%	14%	48%	39%	25%	3%	6%	2%	3%	2%
Turns	8,2	9,2	9,1	7,6	7,5	9,8	9,5	9,7	9,5	8,3	6,6	8,2	7,9	7,2	7,4
Steps	173	123	155	112	99	151	98	138	169	215	242	96	97	103	150

Heuristic refers to the heuristic developed earlier, *B.* refers to the baseline archetype agents and *CARI* refers to our method.

commonly used way to automate game testing in the video game industry nowadays. To that effect we introduce a simple agent: a *Contact* heuristic. The heuristic agent is here to know what can be achieved in terms of archetype with something that is rule-based, i.e. something that game designers would typically be doing in the industry to test the game in an automated way. This heuristic is a simple behavior tree where the heroes try to get as close as possible to the closest enemy, try to stab it and maybe shoot it if it is close enough. It does not take into account the fact that it plays a team of 3 heroes, rather it plays 3 heroes separately. It also does not take into account the previous actions taken, and so there is no real long-term strategy. Note that the behavior tree controlling the enemies can not be used to control the heroes because the game-play does not allow it: there are more enemies than heroes and only the heroes have access to a shield. We wish to see if using RL can help get a better archetype without all the constraints and limitations of a hard-coded heuristics. But the key results of this work is the following: using the already trained CARI we use it to play 20,000 episodes (i.e. 20,000 games). And for each game random reward coefficients were selected. We display the results as a series of graphs with the horizontal axis being the value of a given reward coefficient and the vertical axis being the key metric associated with said reward coefficient. We will dive deeper into the generation of these graphs later on. These graphs highlight our main contribution: demonstrating our continuously configurable model and allowing the generation and simulation of a very wide variety of play-styles to chose from.

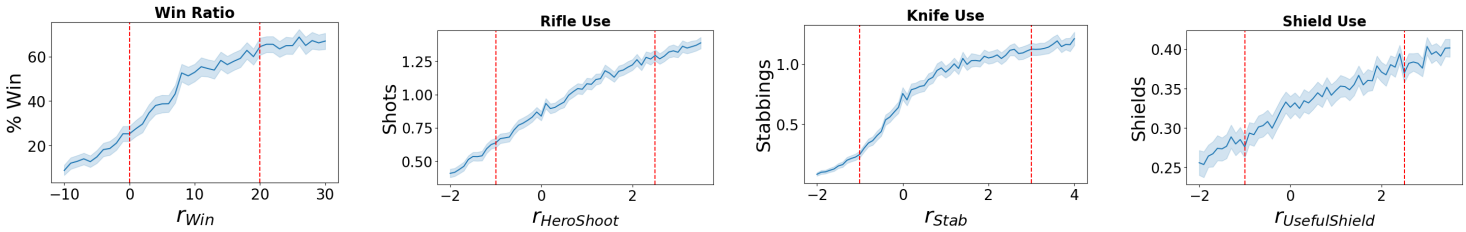
To summarize our evaluation, 3 approaches are compared. The heuristic as a baseline to what is usually used in video games productions. The archetype training agents as a baseline to what is the usual way to generating play-styles through reward shaping. The CARI training which is our contribution and which allows with a single training loop, to be able to behave in many different ways that could be used to investigate behaviors and balancing during the production of a game with a realistic amount of computing time.

There are 2 types of key metrics used: the global ones and the play-style oriented ones. The global ones are: percent of games won, lost or draw (reached the limit of 10 turns), average number of turns, average number of steps. The play-style oriented ones are: average number of stabbings, average number of shots, percent of shots that ended in a cover, the number of shields used, the average distance between heroes, the percent of health lost per the heroes' team and per the enemies' team at the end of the game. The first type of metrics is used to measure pure completion performances, whilst the second type of metrics is used to measure play-style compliance.

B. Results

All the results of the 3 different approaches introduced earlier are available in Table IV.

1) Heuristic: The first approach is the most common baseline in video games production teams today: the heuristic one. The heuristic agent respects the archetype asked from it : it uses the knife more than the rifle. The main drawback is the win ratio (21%), this is due to the fact that if one wishes to hard-code a heuristic it is very challenging to develop one that can deal with multiple objectives – the two objectives being: playing with a given play-style and also performing well. It is up to the practitioner to decide how much of one objective to sacrifice in favor of the other, thus entering into a tedious iterative loop of trial and error. There is always the risk that, with any big change in the environment, the loop is to be done all over again. It ends up being very time consuming, which is what is observed in today's video game productions. So the heuristic agent, while respecting key components of the target archetype, does not seem able to actually play the game. It does not display any real strategy in the sense that it plays each hero independently from the others, leading to a somewhat unrealistic way of playing the game. It even loses its advantage of easy implementation when one starts looking at other, more complex play-styles, or if one needs to iterate to find the right balance in the actions to achieve the desired results.



We only display those 4 graphs due to space constraints but we observe similar results on the other reward coefficients

Fig. 2: Key-Metrics observed with respect to corresponding reward coefficients using CARI, with their 95% confidence interval in light blue. The 2 vertical doted red lines represent the bounds of the intervals used during training.

2) Baseline Vs CARI: The first thing to notice is that behaviors generated either by the baseline archetypes or by CARI yield the expected results, both in terms of absolute statistics but also in terms of their respective arrangement with the other behaviors:

- The *Sniper* uses the rifle more than anyone else (around 12 Shots per game) while having the lowest number of shots in end in covers (2%)
- The *Contact* uses the knife more than any other (almost 14 stabbings per game)
- The *Grouped* and the *Scattered* are respectively the most tightly grouped (around 4 cells of distance between heroes) and most spread out (around 17) teams
- The *Safe* is loosing the least amount of HP during games (55% *Lost HP Heroes* per game)
- The *DPS* inflicts the most damages (95% *Lost HP Enemies* per game)
- The *Win Only* wins more than any other (90% *Wins*)

Some of the baseline archetypes have some very low win rates (e.g. 26% for the *Scattered*). As to why is that, it is quite clear that it is caused mainly by the number of games ending up in draws. This means that the negative reward perceived when getting a timed out (or a loss for that matter) does not outweigh the play style rewards perceived during the game. In other word, the archetype focus on play-style at the expense of the victory. The best examples of that behavior are the Grouped and Scattered archetypes. These 2 perceive a reward at the end of each turn based on the average distance between all heroes, and it seems that they would rather finish on a draw and gather those end-turn reward rather than win and renounce gathering those additional end-turn reward. Waiting for the last turn to win the game is not something they were able to learn consistently. So, having better win ratio would require tweaking the reward function and thus entering in the tedious iterative loop mentioned earlier. The other drawback of this method is that 7 different models had to be trained, which translates into $7 * 15 = 105$ hours of training.

It seems that CARI solves this issue. Let us take the example of the *Grouped*. It goes from 44% win rate to 72%. But the average number of turns is only reduced a little: from 9.8 to 9.5. That means that the agent is taking advantage of both the end-turn reward and the end-game reward. It is able to learn such things because, and this is intrinsic to this approach: all possible play-styles are trained simultaneously. Meaning that the agent is well aware of what the end-turn and end-game rewards are, simply because it was encouraged to perceive one and/or the other depending on the episode. There still remains some imperfections; if we look at its counterpart: the *Scattered*. It goes from 26% to 41% win rates. While still being an improvement it remains quite low. This could be explained by the fact that such a high *Average Heroes Dist.* of around 17 is just intrinsically detrimental to the winning abilities of the agent. Spreading one's team that much does not allow for any meaningful winning strategies. Lowering the $r_{HeroDistance}$ or increasing the r_{Win} might solve this issue. CARI allows for such tweak without needing any retraining of any kind, making this iteration loop to find the proper $r^{(Sc)}$ much faster.

3) CARI's Behavioral Continuum: The other main motivation, beyond archetypal behaviors, is to be able to access the play-styles in-between those archetypes. While extreme reward coefficients $\{r^{(\phi)}\}_{\phi \in \Phi_{Arch.}}$ were used to produce Table IV, smaller rewards coefficients could be used and so create varying degrees of these play-styles. This is usually the case with human players: there are a few extreme archetypes and a lot more people in between. With this sort of model we can get both of those groups. To that effect we report additional results. To obtain those additional results, the already trained CARI agent was used to play 20,000 different randomly generated games. Each of these levels were played using a different randomly generated reward function, by sampling as in line 5 of Algorithm 1, and keeping it constant during the level. The reason random reward functions are drawn instead of looping over all possible combinations is because, our reward coefficients intervals make up for a total of $4.32 * 10^{15}$ possible combinations. It is important to note that there is no training of any kind going on here. The results generated through this process are showed in Figure 2. The metrics represented here are normalized by the number of turns. For example, it is not the number of shots in a game, but in fact the average number of shots per turn.

It is quite clear that CARI has learned to adapt its behavior, its style of play, depending on which reward coefficient it is subject to. The most interesting one is the win ratio. Even on something very abstract such as winning or loosing, it has learned how to adapt to what is asked of it. The other thing to note here is the 2 red doted lines on each graph representing

the intervals seen during training by the model. As we can see the model learns to generalize quite well on any dimension to previously unseen reward coefficients values and to adapt its behavior accordingly. These results are very encouraging as they show that one can train a model to learn many different ways to play the game at once. All this in just one training. It also means that one could possibly increase the number of play-styles one wishes to obtain without increasing the amount of training time needed. This is a tremendous gain both in terms of time and computation power. It also requires no human data, making it feasible to apply to any video game in production, even during the early stages.

It does however raise a few challenges. The two main ones are: how to choose the interval of the reward coefficients and how to then choose in these intervals which values to select as to generate a human-like agent. Many different avenues might be explored. For example it could be that training a model in a supervised fashion to predict which reward coefficients are used based on the metrics observed might yield interesting results, and allow no human data in the training loop. Another avenue worth exploring are Quality-Diversity evolution algorithms [22], which could be used with a CARI agent to generate high-performing archetypes easily. This model, allowing to query behaviour from an already trained model, can help speed up many algorithm of the sort by moving the training of the RL algorithm outside the main iteration loop, for example it could help achieves the same results as in [23] without having to train a deep RL algorithm at every iterations.

VI. Conclusion

We have shown the benefits of using RL over heuristics to create stereotypical agents, and of using varying reward functions across episodes rather than across training phases to create a continuum of play-style through one single training phase. This configurable agent allows better sampling in the space of possible play-styles and does not require any human data, which makes this approach realistic with the constraints of the video game industry. Moreover, our approach is agnostic to the RL algorithm and to the environment used meaning it can easily be applied on many policy optimization problems.

While effective, the approach presented here still requires expert knowledge about the environment, as many RL applications. Using players data could alleviate this problem in the future. Using players data could also help understand whether players' play-styles are more of a continuum, between different archetypes, or more distinct clusters. With the CARI training approach, whether it is the former or the latter, the model is equipped to simulate both since it can simulate the continuum and it can easily be constrained to simulate given clusters.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki *et al.*, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[4] A. Canossa and A. Drachen, "Patterns of play: Play-personas in user-centred game development," in *DiGRA Conference*, 2009.

[5] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Extending Reinforcement Learning to Provide Dynamic Game Balancing," Edinburgh, United Kingdom, pp. 7–12, Jul. 2005.

[6] G. Andrade, G. Ramalho, and V. Corruble, "Automatic computer game balancing: a reinforcement learning approach," Utrecht, Netherlands, pp. 1111–1112, Jul. 2005.

[7] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Generative agents for player decision modeling in games," in *Proceedings of the Ninth International Conference on the Foundations of Digital Games*. Society for the Advancement of the Science of Digital Games, 2014.

[8] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving personas for player decision modeling," in *2014 IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–8.

[9] C. Holmgård, M. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through mcts with evolved heuristics," *IEEE Transactions on Games*, vol. 11, no. 4, pp. 352–362, Dec. 2019.

[10] A. Cooper, *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity (2nd Edition)*. Pearson Higher Education, 2004.

[11] A. Tychsen and A. Canossa, "Defining personas in games using metrics," in *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, 2008, p. 73–80.

[12] Y. Norouzzadeh Ravari, S. Bakkes, and P. Spronck, "Playing styles in starcraft," 2018, european GAME-ON Conference on Simulation and AI in Computer Games.

[13] A. Canossa, S. Makarovych, J. Togelius, and A. Drachen, "Like a dna string: Sequence-based player profiling in tom clancy's the division," ser. Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE, 2018.

[14] K. Werder, S. Seidel, J. Recker, J. Kundert-Gibbs, N. Abboud, and Y. Benzeghadi, "Data-driven, data-informed, data-augmented: How ubisoft's ghost recon wildlands live unit uses data for continuous product innovation," pp. 86–102, 04 2020.

[15] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[16] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Personas versus clones for player decision modeling," in *Entertainment Computing – ICEC 2014*, Y. Pisan, N. M. Sgouros, and T. Marsh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 159–166.

[17] A. Kanervisto, J. Pussinen, and V. Hautamäki, "Benchmarking end-to-end behavioural cloning on video games," *2020 IEEE Conference on Games (CoG)*, pp. 558–565, 2020.

[18] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, 07 2012.

[19] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," 2017.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[21] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding Atari agents," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 10–15 Jul 2018, pp. 1792–1801.

[22] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[23] R. Shen, Y. Zheng, J. Hao *et al.*, "Generating behavior-diverse game ais with evolutionary multi-objective deep reinforcement learning," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 7 2020, pp. 3371–3377.