Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2022)

Automated Play-Testing Through RL Based Human-Like Play-Styles Generation

Pierre Le Pelletier de Woillemont[1,2], **Rémi Labory**[2], **Vincent Corruble**[1]

[1] Sorbonne Université, CNRS, LIP6, F-75005
[2] Ubisoft La Forge, France
pierre.le-pelletier-de-woillemont@ubisoft.com, remi.labory@ubisoft.com, vincent.corruble@lip6.fr

Abstract

The increasing complexity of gameplay mechanisms in modern video games is leading to the emergence of a wider range of ways to play games. The variety of possible play-styles needs to be anticipated by designers, through automated tests. Reinforcement Learning is a promising answer to the need of automating video game testing. To that effect one needs to train an agent to play the game, while ensuring this agent will generate the same play-styles as the players in order to give meaningful feedback to the designers. We present *CARMI*: a Configurable Agent with Relative Metrics as Input. An agent able to emulate the players play-styles, even on previously unseen levels. Unlike current methods it does not rely on having full trajectories, but only *summary data*. Moreover it only requires little human data, thus compatible with the constraints of modern video game production. 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.

1 Introduction

In the video game realm, the main goal of Reinforcement Learning (RL) has usually been to achieve superhuman performances (Silver et al. 2017) (Vinyals et al. 2019). Another interesting application for RL is the pursuit of human behavior (Jacob et al. 2021), in order to perform automated game testing. These automated tests can have varying objectives: find the most resource consuming assets of the game or flag possible unintended exploits in the gameplay. In our case we wish to perform automated tests in order to help assess more accurately the difficulty of the game.

The results of these automated tests are most useful to the designers during the production phase of the game, during which there is very little human data available (only the occasional human play-tests). Moreover, this data is rarely in the form of full trajectories but rather in the form of *summary data*, i.e. in the form of a few key metrics, or *play-modes*, that are of most interest to the designers (e.g. number of shots fired during a game). Indeed, tracking and storing full players' paths (state-action tuples) is a big constraint to put on developers, especially in the early stages of the production, as it involves issues of bandwidth and computational resources. Whereas summary data is cheaper to acquire and to store. Furthermore, the pipeline for tracking and storing summary data is usually the same during the production of the game, as it is post launch. In short, logging summary data is cheaper, easier to implement and is most likely already being done by the developers. In this work, the available human data is assumed to be very limited and in a summarized format: no full trajectories, only a few key metrics, so as to fit with the constraints of game production.

To assess the difficulty of a game, distinguishing between performance (i.e. success) and play-style is important: a game can be perceived hard for a subset of the players because of their style of play, not because of the game itself. So the performances relative to the play-style should be reported to the designers, as well as the overall difficulty, so as to help them make informed decisions. A play-style is measured using key metrics which capture different aspects of the gameplay.

We present here a Configurable Agent with Relative Metrics as Input (*CARMI*) agent, which aims at generating a continuum of play-styles, fitting with the players distribution, making the sampling of human-like play-styles straightforward, even on levels previously unseen by the agent and the players. This agent is configurable with respect to the metrics used to define the play-styles and is obtained through a single RL training loop.

The definition and measure of a play-style is done on summary data (e.g. the number of shots fired and the number of stabs made with a knife during a game). However, here it is normalized level by level, using the players distribution. Therefore the play-styles are defined not as absolute but as relative to the players population, making them independent from the levels and their designs. Take for example a play-style characterized by "a high use of a riffle over a knife". Instead of measuring "high use" as "X times during a game" it would be measured "X times more than the other players", making the definition of this play-style applicable no matter the level.

The agent learns a policy conditioned to matching a desired play-style. This is done by giving the desired relative summary data z as input to the agent, and building a reward function that orients the agent towards matching this z. It is the main contribution of this work: training an agent

conditioned by a play-style which is defined in relationship to human play-styles distribution. This means that the agent learns the distribution of the players in the play-style space solely through the reward function. Proceeding this way produces two results. First, it ensures that the agent is able to cover at least the same space as the human players in the space of the play-styles. Second, the independence between the play-styles and the level design allows the agent to easily generalize the play-styles to newly unseen levels. This is due to the fact that the agent has learnt to associate any value of z to a behavior, no matter the level, which allows better generalization. The new direction presented by this paper results from the capacity to generalize human summary data to new levels using few human data, to better fit with the constraint of the industry.

We can therefore use this agent to play-test new levels, using play-styles defined from human data collected on previous levels. And because the agent is conditioned using a very interpretable input z, it could also be used for automated play-tests using designers' hand-made play-styles, since each dimension of z corresponds to a specific metric.

In this paper we first give a more formal definition of play-styles as well as the existing methods that aim at learning various personas and those aimed at modeling human behavior. We then introduce the environment: a turn based strategic shooter. And finally compare our results with the existing method CARI (Le Pelletier de Woillemont, Labory, and Corruble 2021).

2 Background

2.1 Play-Styles As A Combination Of Play-Modes

Canossa and Drachen adapt the "persona" framework introduced by (Cooper 2004) in the field of Human Computer Interaction. Tychsen and Canossa make the distinction between play-mode, play-style and play-persona. It is a distinction based on the level of data aggregation. A play-mode is 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-modes". And finally play-personas represent the "larger-order patterns that can be defined when a player uses one or more play-styles consistently". The focus of this paper is on the simulation of human-like play-styles through the generation of behaviors yielding play-modes similar to players'.

2.2 Automated Play-Testing

Recent work as been done to develop automated play-tests using machine learning (ML) approaches, more specifically using agents trained through (Deep) RL. There are two main categories of test: technical tests (e.g. frame rate, bugs) and gameplay tests (e.g. difficulty assessment, game consumption analysis). Our approach focuses on difficulty assessment for the experiments while taking into account the diversity of approaches and resulting play-styles.

RL can be use to improve game testing (Bergdahl et al. 2021) in order to find unintended exploits in the video game. It can also be used to measure what is achievable in the game. For example, (Sestini et al. 2022) train an agent to uncover what is possible in the environment, but should not be, in order to flag bugs and glitches for the designers to fix. Alonso et al. trained an agent to navigate complex 3D environments, which can be used for Non Playable Character (NPC) development with complex navigational skills, or in order to measure the feasibility of procedurally generated goals.

However, in this work we are interested in training an agent that can inform us about the difficulty of the game. Therefore, this agent must produce diverse human-like play-style in order to provide meaningful feedback.

2.3 Diverse Human-Like Behavior

To approximate human behavior, Inverse Reinforcement Learning (IRL) (Ng, Russell et al. 2000) can potentially be used. The 2 main drawbacks of IRL methods are the homogeneity assumption in the trajectories and the quantity of the data necessary for such approaches to yield convincing results. In our case, neither full trajectories nor large dataset are available. Kangasrääsiö and Kaski developed an approach to perform IRL using solely summary data to alleviate this problem. However they assume homogeneity in the data, in the sense that all the data comes from the same expert. This homogeneity assumption goes against the idea of generating varying, and therefore heterogeneous, play-styles.

There are a few possibilities to insert heterogeneous play-styles into an agent. To that effect, Holmgård et al. have demonstrated that shaping the reward (2014), fitness (2014) or utility function (2019) produces variety in the style of play of the agents. Moreover they showcased that archetypes (i.e. stereotypical play-styles) can be a good low-cost, low-fidelity approach to automated play-test. Because each archetype requires its own training and its own reward function, they limit themselves to only 4 archetypes.

Le Pelletier de Woillemont, Labory, and Corruble (2021) solved this issue by training a *CARI* agent where the coefficients of the reward function are given to the RL agent as input. Concretely, they train a policy $\pi(a|s, w)$ where w are the coefficients of the reward function: $r = w \cdot \theta$, where θ represents all the events which induce a reward signal. Additionally one must define W, the intervals within which the w will be sampled during the training. Thus creating a continuum of available play-styles to sample from at inference time (simply using varied w). This approach allows to train one single model rather than multiple ones. It moves the problem of selecting the proper w to generate the desired play-style after the training rather than before, it does not however remove it.

The CARI approach suffers from two main issues. The first is the absence of guarantee that the space of generated playstyles covers well enough the set of human play-styles. The second is that even if the CARI generated play-styles do include the players' play-styles, there is no straightforward way to know which reward coefficients w to select in order to simulate the human-like play-styles. This is due to the fact that the process still relies on finding a good combination of reward coefficients that will hopefully generate the desired play-modes. Both these issues are solved here by

directly giving the target values of the desired play-modes as the goal to the agent as part of its input state: replacing w by a z which encodes a desired play-style. These objectives are encoded and sampled using the distribution of the play-mode over the players population. This agent still allows access to a wide variety of play-styles but also ensure that the generated play-styles do in fact encompass the humans' ones.

However, there is no guarantee that the players' and the agent's full trajectories look the same, for that we would need full human trajectories to train on, which we assume to not have access to during the the production of the game. We are not claiming to generate human-like behaviors, but human-like play-styles, as they are measured : using summary data. The goal is to produce summary data that are player-like enough to take well informed decisions, on new levels.

2.4 Goal-Conditioned Reinforcement Learning

Goal-conditioned RL has been used in video game testing in different ways. For example (Roy et al. 2021), frame their problem as constrained RL, to create an agent with the desired behavior. Their approach relies on a main goal to achieve and several constraints to fulfill, and are automatically weighed throughout training. This approach does not allow the emergence of heterogeneous play-styles, necessary in our case, since the play-style is the goal to achieve, not a constraint to be met.

Goal-conditioned RL has been extensively studied (Kaelbling 1993) (Schaul et al. 2015a) (Chane-Sane, Schmid, and Laptev 2021). The framework is usually in the form of a policy which given a state s and a goal g, predicts actions which lead to the goal. The goal is usually a certain state of interest in the environment (e.g. a destination). In our case the goal is the encoded play-style z, relative to the players distribution. It is not a state in the environment, but a behavior to adopt. Moreover, this behavior is not absolute, but relative to the players, and should be matched on different levels. The only information about the distribution of the players given to then agent is thought the reward feedback.

3 Proposition

Our goal is to learn a play-style conditioned policy which generates trajectories yielding summary data as close as possible to a given objective. This policy is then used to emulate human-like play-styles, on previously unseen levels.

3.1 Notations

In our environment, let $\tau = (s_0, a_1, s_1, ..., a_T, s_T)$ denote the trajectory of a full episode. We introduce the summarizing function ψ which takes as input a trajectory and returns a set of M summary data (e.g. number of shots and stabs made): $\psi(\tau) \in \mathbf{R}^M$. In this work, we assume to have access to the summary data of N_P players over N_L levels of the game: $\{\psi(\tau_{l,p})\}_{l\in[1:N_L],p\in[1:N_P]}$. Additionally, μ_l and σ_l represent the mean and standard deviation of the players summary data on level l, such that $\psi(\tau_{l,.}) \sim \mathcal{N}(\mu_l, \sigma_l)$. We also note $\psi_l(\tau) := \frac{\psi(\tau)-\mu_l}{\sigma_l}$, the summary data on level l, normalized by the players distribution on that level (assuming a Gaussian distribution).

We introduce the policy $\pi_z(a|s) := \pi(a|s, z)$, with a the action, s the state and z some additional information (e.g. a play-style to emulate in our case or the reward coefficients in CARI) given to the agent, with $z \sim \mathcal{P}_z$, here we assume that $\mathcal{P}_z = \mathcal{N}(0, 1)$. In our case we wish for z to represent $\psi_l(\tau)$. Our goal is for π_z to generate τ_{π_z} yielding $\psi_l(\tau_{\pi_z})$ as close as possible to z. The reward function reflects this objective by being proportional to $-d_t := -\Delta(\psi_l(\tau_{\pi_z,t}); z)$, the current distance between the goal z and the agent. Where $\tau_{\pi_z,t}$ is the trajectory generated by π_z until time-step t. By formulating the problem this way, we can then use any RL algorithm to solve this MDP and generate an agent with a policy $\pi \in argmin_\pi\{\mathbf{E}_l[\mathbf{E}_z[\mathbf{E}_{\tau\sim\pi_z}[\psi_l(\tau) - z]]]\}$.

3.2 Relative Metrics As Input

The way the metrics are encoded, when given to the agent as goals, is key here. Instead of aligning z (the agent's goal) to the absolute metrics values $\psi(\tau)$, it is aligned to the relative ones, w.r.t. the players distribution on that particular level: $\psi_l(\tau)$. It moves the issue of playing with a given play-style from a hard to define absolute point of view, into an easier relative one. It is difficult to define what can be considered a *high* frequency of shots, it is however easier to say if a given frequency of shots is among the *highest* over the players population.

Additionally, it improves greatly the usefulness of the agent at inference time, especially on new previously unseen levels. Giving the absolute metrics as the objective (i.e. $z \approx \psi(\tau)$) would have the agent learn how to achieve that specific goal, no matter the level design. During inference the agent would then still generate these exact metrics values, if the level allows it. This would render the agent quite useless for any automated testing procedure: on a new level the agent is tasked with shooting twice and it does it. We did not learn much, except that shooting twice on this new level is possible.

Instead giving the normalized metrics, relative to the players, as the objective (i.e. $z \approx \psi_l(\tau)$) means that the agent needs to learn to adapt its behavior to fit with the portion of the players represented by z. Since we assumed the data to be normally distributed and given than $P(X \sim \mathcal{N}(0, 1) < -1.96) = 0.025$, at inference time giving the target value $z = -1.96$ to the agent means shooting as much as the 2.5% of players that shoot the least. After playing one level with this z, the absolute number of shots done by the agent is then reported to the designers, which would correspond to the number of shots done by this portion of the players, had they played that new level.

3.3 Configurable Agent

The more complex the game the more numerous the play-styles, thus the more numerous the number of models to train. In order to solve this issue a single model is trained, much like in (Le Pelletier de Woillemont, Labory, and Corruble 2021). The objective z, which correspond to the normalized metrics $\psi_l(\tau)$ are given inputs to the model. Proceeding this way makes the agent configurable: the play-

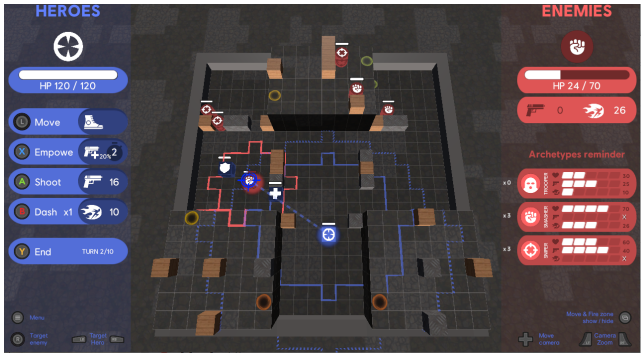

Figure 1: Screenshot of the Video Game

style we wish the agent to adopt can be chosen after training, at inference time. We only need to train a single model for all play-styles. At the beginning of each episode a new z is drawn and given as input to the agent. The agent has then the objective of matching this value by the end of the episode.

The reward function used to this end is defined as $r_t = d_{t-1} - d_t + [-d_t]_{t=T}$. If the agent does an action bringing it closer to its objective it will receive a positive reward corresponding to how much closer it got, much like in a navigation problem. The same applies if the chosen action moves it away from the objective. The objective of the agent is to end the episode as close as possible to z. This is why at the end of the episode the agent perceives a negative signal equal to how much distance to the target is left. Moreover, $\sum_{t=1}^{T} r_t = (d_0 - d_T) - d_T$: the cumulative reward on the whole episode is equal to the distance "*travelled*" towards the objective minus the distance left at the end.

4 Experimental Setup

4.1 Game Environment

The game environment used in this work is the same as in (Le Pelletier de Woillemont, Labory, and Corruble 2021). This environment presents a few advantages. It is complex enough to be able to have different play-styles and some of the challenges that come with training in a complex video game, but simple enough to have moderate computation requirements. Moreover it is developed using Unity's ML-Agents (Juliani et al. 2018), which allows to control the agent either with the Python programming language (for the RL aspect) or with a controller (for the human player aspect).

This video game environment is a discrete, turn-based, shooter-strategic, see Fig. 1. In this video game, two teams fight to the death on a 3D cell based board. This game is inspired in its gameplay elements from the "Mario + Rabbids" video game, developed by Ubisoft. The movements can be done in any directions toward an empty cell. Some portals (the circles in Fig. 1) are spawned across the board to allow characters to take short-cuts. In addition the board also consist of a series of covers behind which the characters can hide to avoid getting shot at.

A team of 3 heroes (in blue), controlled by an agent (or a human, during a play-test), face a team of a varying number of enemies (up to 7, in red) controlled by hand-crafted behavior trees, designed by us like any NPC would be in most video games, for a maximum of 10 turns. Every hero character has the capacity to move, shoot and stab (i.e. melee attack). They are all defined by a set of basic statistics: their health, their range of movement and of fire and their damages. Additionally, each hero a *super capacity*. One has the power to heal nearby allies, another has the ability to apply an empowerment to nearby allies thereby increasing the damage caused by their attacks for one turn and the third hero can apply a shield that will block one attack. Each of these *super capacity* has a two turns cool-down.

It is a turn-based game, meaning that when one of the team is playing, the other one is frozen in place. A full turn is not a single time-step, rather a full turn is many time-steps. So for example during one turn, the agent (controlling the whole hero team) can move around with hero number one, stab and shoot an enemy with the same hero, then shoot another enemy with hero number 3, and use the shield of hero number 2 before skipping the rest of its turn, effectively starting the enemy's turn. Each character has a maximum of one shot and one stab per turn. Once a character has shot it cannot move for the rest of the turn. The game ends once one of the two teams has been fully destroyed or if the game reaches the 10 turns limit, in which case the game is considered to be a draw.

The state returned by the environment is twofold: an image-like segmentation map of size 20x20x3 indicating what object is inside each cell (hero, enemy, cover, portal or nothing at all) and an array comprising the rest of the information needed: the number of turns left, the current stats for each hero and each enemy . The state size is 7414, as it is the concatenation of the flattened image-like segmentation map and vectorial inputs. Regarding the action space, there are three main group of actions per hero : movement, long and short range attacks, super use. To move, the agent selects a cell on which to send a hero, assuming this cell is within reach of the hero. We use a path-finding algorithm to then move the hero. For the attacks, the agent choose which of the enemies to attack, there is at most 8 enemies. So the total number of possible actions is 3 (number of heroes) x [20x20 (size of the map) + 8 (one shot per enemy) + 8 (one stab per enemy) + 3 (the number of supers)] + 1 (skip) = 1258. Note that not all actions are available for all heroes at all times. For example, the healer cannot use the shield ability or some cells might be out of reach for a hero. When an action is unavailable to the agent, it is simply masked, putting its probability to be selected to 0.

To gather players' data, a play session with 30 participants was organized. We designed a playlist of 10 levels that each player had to play. At the end of each level the player would go on to the next level, no matter the outcome. Playing all 10 levels took each participant roughly one hour of play-time. The game was introduced to them with a tutorial in the form of a PowerPoint document as well as a small video demonstration. No further interaction with the players took place to ensure that each player had the same level of information going in. The players only knew that this play session was

done in order to help automated test, there was no mention of ML. Most participants are not familiar with RL or with ML in general. Out of the 30 players, the data of 25 of them was used: the rest either did not play all the way to level 10 or skipped some levels. The first level was removed from the available data, as it mainly served as an introduction level where players where mostly testing the controllers and not engaging fully with the game. Levels 2 through 8 were kept as a training set (L_{Train}) and levels 9 and 10 as test levels (L_{Test}), which means the training set only contains 7 (levels, L_{Train}) * 25 (players, N_P) = 175 data points.

4.2 Training Procedure

To train the agent, we chose to use the ACER (Wang et al. 2017) algorithm. There are a few reasons why ACER was chosen. First, it is a discrete action algorithm which suits the problem well. It is an on-policy and an 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 (Schaul et al. 2015b). Another major reason for choosing ACER is the possibility to run multiple environments in parallel in an asynchronous fashion, all feeding the same buffer and training the same model. This is quite useful for training agents with an environment that is not perfectly stable and could crash. The inputs of the agent are both vectorial and convolution-based. The actions of the agent are also both vectorial (shots and stabs) and convolution-based (the movements of each hero). Therefore the neural architecture used was very similar to the one developed in (Gendre and Kaneko 2020). The neural network architecture treats the image-like inputs using 2D convolution layers and the vectorial inputs using dense layer. The features generated are then combined to produce both convolutional (for the movements) and vectorial (for the attacks, supers and skip) outputs for the actions.

Some objectives z are easier than others to reach. For example, shooting as little as the players who shot the least is easier than shooting as much as the players who shot the most. This is the reason why a curriculum-based approach was used to sample the play-modes objectives z at each episode. Moreover, automatic curriculum approaches can improve performances of multi-goal agents (Portelas et al. 2020) . The approach used here is the modeling of absolute learning progress with Gaussian mixture models (ALP-GMM) developed by Portelas et al..

Realistically, training a RL agent on only 7 levels is usually not enough for the agent to be able to generalize well. Moreover, in most video games it is usually possible to create more levels, simply by changing the topology, the enemies team composition, or the characters' stats (e.g. health or damage). In this work, the low amount of levels used to train is not due to the low number of levels available to the RL framework, but rather due to the low number of levels the players played on. Finding a way to incorporate additional levels, even without human data, into the training procedure should be a focus in future work.

Three environments in parallel (each participating in the training of the same model) were used, each running around 12,500 episodes, training the same model. It is equivalent to 24 hours given our computational setup, which is a reasonable constraint to aim for, in real-life use-case of game production. Our setup is a single computer with a 12 core CPU and a NVIDIA GeForce GTX 1070 GPU.

5 Evaluation And Results

The CARMI agent is trained on 5 metrics: the number of shots, the number of stabs, the number of shots under an empower, the number of heal made and the number of shield used. All these metrics are expressed as a number per turn, so not the number of shots, but the number of shots per turn for example. The first two metrics represent the overall attack strategy of the play-style, while the other three represent the use of the *super capacity* available.

We will measure the quality of our agent on two aspects :

1. Does the space of play-modes generated by the agent cover well the distribution of human players ?
2. Is the agent capable of generating human-like metrics on new and unseen levels

Three models are trained : CARMI, CARI and WinOnly. All three models train on the same levels, have the same state and action space and use the same neural network architecture.

The coefficients of the reward function used by CARI are sampled uniformly in intervals including both positive and negative values, to have the possibility to both encourage and discourage the agent to perform certain actions. This baseline measures what diversity driven agent produces, without taking into account the closeness between its diversity and the diversity in the players play-styles. The WinOnly model has a sparse binary win/loss reward function. This serve as a baseline to how useful this most commonly used approach would be to give feedback to designers on difficulty assessment, using only a win driven agent, without taking into account the diversity of play-styles.

5.1 Play-Style Coverage

In this section we report the coverage of play-styles for both the CARI and the CARMI agent, and compare them with the players. We report the results solely on two metrics, due to space constraints: the number of shots per turn and the number of stabs per turn. The CARI agent used here was trained on the same metrics as the CARMI agent. To measure the coverage possible by the CARI and the CARMI agent we ran 2500 episodes with random $w \sim \mathcal{U}(W)$ for CARI and $z \sim \mathcal{N}(0, 1)$ for CARMI. These results are reported in Fig 2. Additionally, we report the Kullback-Leibler (KL) divergence and the Jensen-Shannon (JS) divergence between each of the models (CARMI, CARI and WinOnly) and the Players, both on the train and on the test levels, on the joint distribution of all metrics in Table 2.

The CARMI agent is indeed very much capable to cover the same space as the players. It even generates play-styles not seen in the players population. This too can be an interesting feedback to give to designers as to what is achievable in these levels.

	Metrics	Cluster 1		Cluster 2		Cluster 3		All		
		Player	CARMI	Player	CARMI	Player	CARMI	Player	CARMI	WinOnly
Train	**Stabs**	0.6 (±0.1)	0.6 (±0.1)	1.3 (±0.1)	1.1 (±0.1)	0.9 (±0.2)	0.9 (±0.1)	1.0 (±0.1)	0.9 (±0.0)	1.1 (±0.0)
	Shots	1.7 (±0.1)	1.5 (±0.1)	2.1 (±0.1)	1.9 (±0.0)	1.7 (±0.2)	1.6 (±0.1)	1.9 (±0.1)	1.7 (±0.0)	0.9 (±0.0)
	Empower	0.2 (±0.1)	0.1 (±0.0)	0.8 (±0.1)	0.7 (±0.0)	0.5 (±0.1)	0.4 (±0.1)	0.5 (±0.1)	0.5 (±0.0)	0.0 (±0.0)
	Heal	0.3 (±0.0)	0.2 (±0.0)	0.3 (±0.0)	0.2 (±0.0)	0.1 (±0.0)	0.0 (±0.0)	0.2 (±0.0)	0.2 (±0.0)	0.2 (±0.0)
	Shield	0.1 (±0.0)	0.1 (±0.0)	0.2 (±0.0)	0.1 (±0.0)	0.1 (±0.0)	0.1 (±0.0)	0.1 (±0.0)	0.1 (±0.0)	0.2 (±0.0)
	% Win	75 (±11)	52 (±5)	95 (±4)	77 (±3)	87 (±10)	67 (±7)	88 (±4)	68 (±2)	33 (±1)
	% Lost	0 (±0)	22 (±4)	0 (±0)	18 (±3)	0 (±0)	20 (±6)	0 (±0)	19 (±2)	8 (±1)
	% Draw	24 (±11)	25 (±4)	4 (±4)	4 (±1)	12 (±10)	12 (±5)	11 (±4)	12 (±2)	58 (±2)
Test	**Stabs**	1.3 (±0.4)	0.9 (±0.2)	2.1 (±0.9)	1.3 (±0.2)	1.4 (±2.2)	1.4 (±0.2)	1.7 (±0.4)	1.2 (±0.1)	1.6 (±0.1)
	Shots	2.1 (±0.2)	2.0 (±0.2)	2.2 (±0.3)	2.2 (±0.1)	1.9 (±0.6)	1.8 (±0.2)	2.1 (±0.2)	2.1 (±0.1)	1.2 (±0.1)
	Empower	0.2 (±0.1)	0.2 (±0.1)	0.8 (±0.3)	0.6 (±0.1)	0.7 (±0.4)	0.3 (±0.1)	0.5 (±0.2)	0.5 (±0.1)	0.0 (±0)
	Heal	0.3 (±0.1)	0.2 (±0.0)	0.3 (±0.1)	0.2 (±0.0)	0.1 (±0.1)	0.1 (±0.0)	0.2 (±0.1)	0.2 (±0.0)	0.1 (±0.0)
	Shield	0.1 (±0.1)	0.1 (±0.0)	0.2 (±0.1)	0.1 (±0.0)	0.1 (±0.1)	0.1 (±0.0)	0.2 (±0.1)	0.1 (±0.0)	0.5 (±0.0)
	% Win	100 (±0)	75 (±13)	100 (±0)	80 (±8)	75 (±25)	52 (±20)	95 (±8)	74 (±7)	33 (±5)
	% Lost	0 (±0)	21 (±13)	0 (±0)	13 (±7)	0 (±0)	43 (±20)	0 (±0)	20 (±6)	0 (±0)
	% Draw	0 (±0)	2 (±2)	0 (±0)	6 (±5)	25 (±25)	4 (±4)	4 (±8)	4 (±3)	66 (±5)

Table 1: Mean and 95% confidence interval of key-metrics for each cluster for the players and the CARMI agent, on the Train and Test Levels. In bold the metrics used to train the CARMI agent and are reported as the average per turn.

		WinOnly	CARI	CARMI
Train	KL	10.17	9,34	**6,69**
	JS	0.76	0,73	**0,60**
Test	KL	9,53	**9,38**	9,74
	JS	0,80	0,80	**0,78**

Table 2: All metrics joint normalized distribution divergence between players and agents

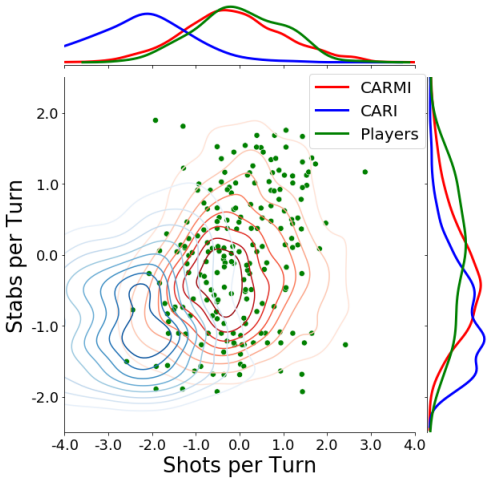

Figure 2: Coverage of CARI, CARMI and the players. The metrics here are normalized by the players distribution, each point corresponding to an episode.

5.2 Human-Like Play-Style Emulation

To measure the capacity of the agent to emulate human-like play-style one must first define the play-styles used. To this end, a Gaussian mixture model (GMM) (Reynolds 2015) clustering is trained on the players training normalized summary data: $\psi_{L_{Train}} := \{\psi_l(\tau_{l,.})\}_{l \in L_{Train}}$, which yields C clusters, each representing a portion of the players. In other words, $\{\mu_c, \Sigma_c\}_{c \in [1:C]} \leftarrow GMM(C; \psi_{L_{Train}})$. These clusters are what is being used to sample the adequate z in order to emulate players' play-styles..

For CARMI, a number of targets per level is sampled from the distribution of each cluster. For each level $l \in [1 : N_L]$ (including L_{Test}) and each cluster $c \in [1 : C]$, z is sampled following $\mathcal{N}(\mu_c, \Sigma_c)$ and the episode is run using the trained policy π_z on level l. Using this sampling method, the absolute metrics generated by CARMI are compared to what is observed in the players' data, on the train and test levels.

Note that the 7 train levels are used to fit the clustering GMM algorithm. The 2 test levels are used for testing. The human data available on these 2 test levels is not used to re train the clustering or the CARMI agent. So, in effect, neither the agents, nor the clustering, use the test levels for anything else than evaluation.

We do not compare results with CARI here for 2 reasons. First, there is no straightforward way to transform the samples z from the clusters into w to feed the CARI agent. Second, even if there were, it is obvious from the big gap in Fig. 2 between the CARI and the players, that the CARI agent would never be able to emulate the players accurately. We do however, compare those results with a pure RL model trained with the sole objective of winning, called "WinOnly". This model is here to compare with what is possible using "*classical*" RL. Indeed, training a model solely to win and using it to inform designers on balancing issues is not unheard of in the industry. Note that neither the training of CARMI nor the training of the clustering have seen the test levels nor the players' data on those levels, these are truly previously unseen levels. These results are available in

Table 1.

The first thing to notice is the capacity of the agent to match the desired metrics on the train levels, it is a good indicator that the model did converge. Also it is worth noting that the agent is overall capable to generalize those play-styles to unknown levels, in the test set. Looking at what the CARMI agent has produced, the feedback that we would have given the designers would have been that the new levels will have the players shoot and dash more, but that the use of the *super capacities* would remain somewhat stable. These feedback would have been correct since they match what is observed in the player's data for the test levels. Had we used the WinOnly model to give feedback to the designers we would have been right about the shots and dash, we would however have missed completely the feedback on the *super capacities*. This goes to show that encouraging models to fit the players play-modes allows to create mode meaningful feedback from automated tests.

The other very interesting result are the win rates. Overall the players perform better on the new level: from 88% to 95% of game won. The same is observed with the agent: from 68% to 74%. But even more interesting are the win rates of the clusters between the train and the test levels. The players in clusters 1 and 2 both perform better in the test levels, and this increase in performance is mirrored by the agent emulating these two clusters. However the players in cluster 3 perform worse: from 87% to 75%. The agent, when emulating this third cluster, also performs worse. Indeed, regarding difficulty assessment, given what the CARMI agent has produced, we would have concluded to the designers that the new levels are overall easier, expect for the third cluster. Additionally, as for the play-modes, the win rates of the WinOnly model is not reliable and doesn't allow to draw correct conclusions.

6 Conclusion And Discussion

We have developed a new agent capable, with one single training phase, to generate a continuum of play-styles which includes the players' ones, using limited human summarized data. We have also developed and demonstrated the effectiveness of a straightforward sampling strategy able to generate human-like play-styles on new levels reliably. This approach can provide very meaningful feedback to level designers. This approach makes no assumption as to the type of environment, or RL algorithm used, making it easily usable in many different contexts.

Improvements should be made to incorporate more play-modes in order to capture more of the players play-styles. We argue that the more numerous the play-modes, the smaller the differences in win rates. Another limitation of this approach is the number of level used for training being limited by the available human data. Including randomly generated levels (or at least some variations of the training levels), lacking human data, into the training procedure would also allow the agent to generalize better between the train and the test levels. Furthermore, we notice that the differences between the agent and the players are bigger at the level of the clusters rather than at the level of the whole population. Including the players' clusters distribution in the learning phase would probably increase the accuracy of the agent when emulating each of these clusters. Simply put, using the clustering during both learning and inference would allow a better approximation of the clusters by the agent.

Moreover, this method relies on metrics computed over a whole episode (i.e. a whole game). So, for games with very long episodes (e.g. RTS games) there are many ways to reach the desired metrics. This might produce unexpected results. For example, in a RTS game with a metric measuring the amount of resources gathered, the CARMI agent could gather the desired amount of resources in an unexpected way. This could indeed produce misleading feedback for the designers. One way to counter balance this is to increase the number of target metrics used and their variety to capture more gameplay aspects. The more numerous the number of metrics, the more constrained the agent should be. Studying the effect of a varying number of metrics should be done in the future.

A Additional Results

A.1 Play-Modes Distributions

In Table 1, we reported only the mean and 95% confidence interval of each play-mode for the players' clusters and their CARMI counterpart. Here we report the full distributions. These results can be seen in Fig. 3. The data that produced Table 1 is the same displayed in Fig. 3.

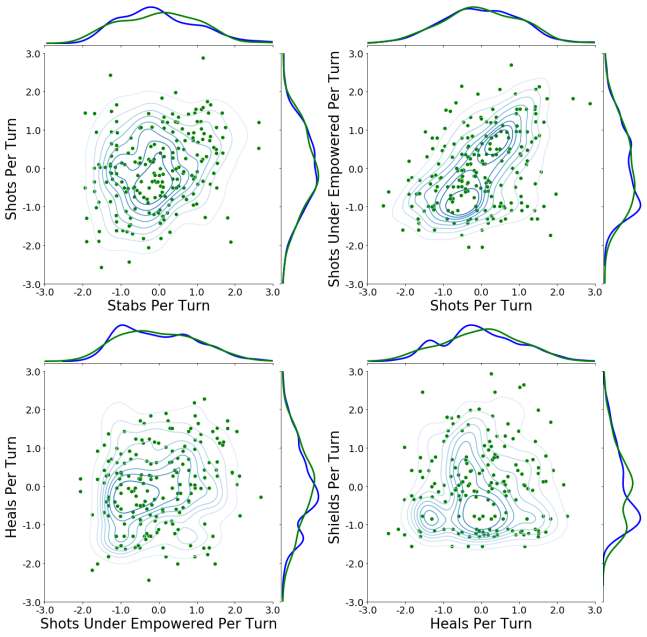

Figure 3: Distribution over the play-modes of players (in green) and CARMI (in blue) normalized summary data, following the sampling strategy described in section 5.2

These graphs showcase that our model is capable to cover the space of players play-style and that the very simple sampling strategy is adequate to have the agent emulate the players' play-style.

A.2 Learning Curves

We display the evolution of the episodic reward throughout the training for both the CARI and the CARMI agent, in the Fig. 4.

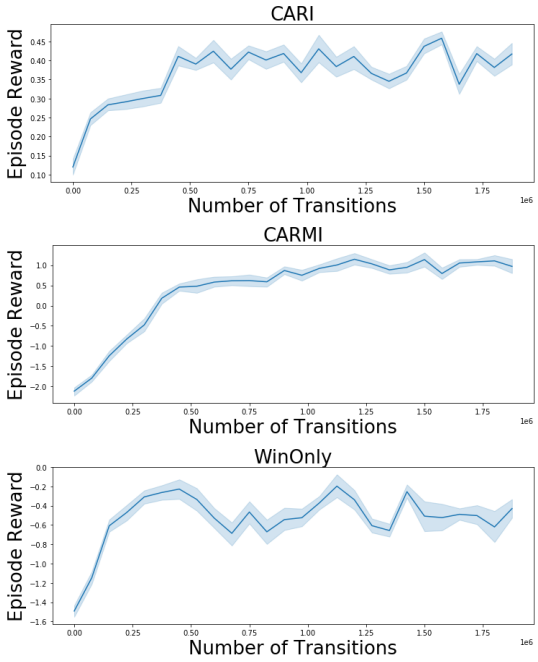

Figure 4: Evolution of the CARI, CARMI and WinOnly agent episodic reward.

References

Alonso, E.; Peter, M.; Goumard, D.; and Romoff, J. 2020. Deep Reinforcement Learning for Navigation in AAA Video Games. *CoRR*, abs/2011.04764.

Bergdahl, J.; Gordillo, C.; Tollmar, K.; and Gisslén, L. 2021. Augmenting Automated Game Testing with Deep Reinforcement Learning. *CoRR*, abs/2103.15819.

Canossa, A.; and Drachen, A. 2009. Patterns of Play: Play-Personas in User-Centred Game Development. In *DiGRA Conference*.

Chane-Sane, E.; Schmid, C.; and Laptev, I. 2021. Goal-Conditioned Reinforcement Learning with Imagined Subgoals. In *ICML*.

Cooper, A. 2004. *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity (2nd Edition)*. Pearson Higher Education. ISBN 0672326140.

Gendre, Q.; and Kaneko, T. 2020. Playing Catan with Cross-dimensional Neural Network. *CoRR*, abs/2008.07079.

Holmgård, C.; Green, M.; Liapis, A.; and Togelius, J. 2019. Automated playtesting with procedural personas through MCTs with evolved heuristics. *IEEE Transactions on Games*, 11(4): 352–362.

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

Holmgård, C.; Liapis, A.; Togelius, J.; and Yannakakis, G. N. 2014. 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. ISBN 978-0-9913982-2-5.

Jacob, A. P.; Wu, D. J.; Farina, G.; Lerer, A.; Bakhtin, A.; Andreas, J.; and Brown, N. 2021. Modeling Strong and Human-Like Gameplay with KL-Regularized Search. *CoRR*, abs/2112.07544.

Juliani, A.; Berges, V.; Vckay, E.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2018. Unity: A General Platform for Intelligent Agents. *CoRR*, abs/1809.02627.

Kaelbling, L. P. 1993. Learning to Achieve Goals. In *IN PROC. OF IJCAI-93*, 1094–1098. Morgan Kaufmann.

Kangasrääsiö, A.; and Kaski, S. 2017. Inverse Reinforcement Learning from Summary Data. *CoRR*, abs/1703.09700.

Le Pelletier de Woillemont, P.; Labory, R.; and Corruble, V. 2021. Configurable Agent With Reward As Input: A Play-Style Continuum Generation. In *2021 IEEE Conference on Games (CoG)*, 1–8.

Ng, A. Y.; Russell, S. J.; et al. 2000. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, 2.

Portelas, R.; Colas, C.; Hofmann, K.; and Oudeyer, P. 2019. Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments. *CoRR*, abs/1910.07224.

Portelas, R.; Colas, C.; Weng, L.; Hofmann, K.; and Oudeyer, P. 2020. Automatic Curriculum Learning For Deep RL: A Short Survey. *CoRR*, abs/2003.04664.

Reynolds, D. 2015. *Gaussian Mixture Models*, 827–832. Boston, MA: Springer US. ISBN 978-1-4899-7488-4.

Roy, J.; Girgis, R.; Romoff, J.; Bacon, P.; and Pal, C. J. 2021. Direct Behavior Specification via Constrained Reinforcement Learning. *CoRR*, abs/2112.12228.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015a. Universal Value Function Approximators. In Bach, F.; and Blei, D., eds., *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 1312–1320. Lille, France: PMLR.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015b. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Sestini, A.; Gisslén, L.; Bergdahl, J.; Tollmar, K.; and Bagdanov, A. D. 2022. CCPT: Automatic Gameplay Testing and Validation with Curiosity-Conditioned Proximal Trajectories. *arXiv e-prints*, arXiv:2202.10057.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; et al. 2017. Mastering the

game of go without human knowledge. *nature*, 550(7676): 354–359.

Tychsen, A.; and Canossa, A. 2008. Defining Personas in Games Using Metrics. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, 73–80. ISBN 9781605582184.

Vinyals, O.; Babuschkin, I.; Chung, J.; Mathieu, M.; Jaderberg, M.; Czarnecki, W.; et al. 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii. Accessed: 2022-08-28.

Wang, Z.; Bapst, V.; Heess, N.; Mnih, V.; Munos, R.; Kavukcuoglu, K.; and de Freitas, N. 2017. Sample Efficient Actor-Critic with Experience Replay. arXiv:1611.01224.