Persona Agents from Playtraces and Emotion

Pedro M. Fernandes, Manuel Lopes, Rui Prada

INESC-ID and Instituto Superior Técnico, Univ. de Lisboa, Portugal
pedro.miguel.rocha.fernandes@tecnico.ulisboa.pt, manuel.lopes@tecnico.ulisboa.pt, rui.prada@tecnico.ulisboa.pt

Abstract

This paper proposes a novel pipeline for generating agents that simulate player behaviour. By clustering player traces and using evolutionary algorithms to evolve parametric agents to best represent those clusters, our pipeline creates persona agents that represent the behavioural space of players. We here propose clustering playtraces based on behaviour, emotional experience and a mixture of both. We implement the pipeline on a test bed game and using 182 collected player traces with both behavioural and emotional information, we demonstrate that our persona agents can generate diverse player-like behaviour both in the level used to evolve them but also in a previously unseen level. We further find that using emotional information leads to better behavioural coverage on both levels. Although on its early stages, our approach offers a new perspective on how game developers and testers can gather insights on player behaviour without having to rely on extensive user testing.

1 Introduction

How does a player play? For any sufficiently complex game, there is not a single answer. In the exact same game scenario, different players can behave in completely distinct ways. This makes the task of simulating player behaviour a complex one. In most games, it is unfeasible to simulate every possible sequence of actions a player might do. And even if possible, many sequences of actions might be unlikely to be chosen by a player, whereas others might be extremely common. Knowing which sequences of actions better reflect the behaviour and diversity of real players can help game developers and testers make better decisions.

The use of agents that are able to simulate diverse player behaviour can empower game developers and testers, allowing them to rely less on user testing, which can be a slow and costly endeavour. If game designers had access to such agents, they could use them to find areas in their maps that are seldom explored or challenges that have a high likelihood of being problematic. Testers could also deploy them to see if any bugs or crashes are introduced when changes are made to the game. Agents that behave like different players could also be used as the basis for realistic and diverse Non Playable Characters (NPCs).

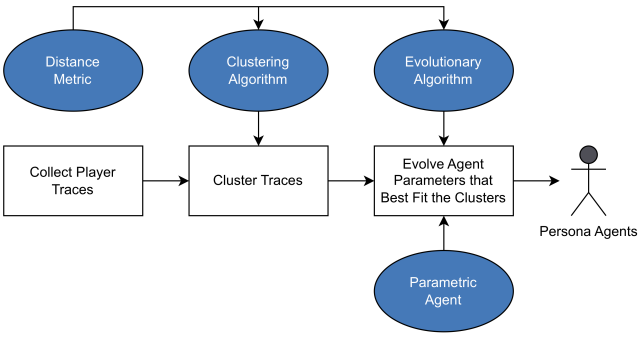

Figure 1: Diagram of the persona agent pipeline, identifying the 4 main components that need to be implemented: a distance metric; a clustering algorithm; an evolutionary algorithm; and a parametric agent.

Canossa et al. (Canossa and Drachen 2009) introduced the concept of persona, previously used in the area of Human Computer Interaction (Cooper 1999), to the world of game design. These personas were defined as "play-personas", representing different player behavior gathered from experience or observation.

In this paper, we propose a pipeline for generating persona agents, that is, agents whose behaviour represents a specific set of players. The process starts with a set of playtraces. To identify different player types, we cluster those traces in order to find sets of players that behave and/or experience similarly. We then use an evolutionary algorithm to evolve parametric agents to represent those clusters. By doing so, we obtain a set of persona agents.

The main contributions of this paper are: a modular pipeline for generating data driven persona agents that represent player behaviour; the use of emotional data to aid the creation of said persona agents; and the robustness of the agents, which work for different levels without any further training.

The paper is organized as follows. In Sec. 2, we give an overview of the areas and works that serve as the foundation for this paper. In Sec. 3, we describe the game used as proof of concept and the data collection process. Section 4 presents the implementation of the pipeline's main components, namely: the distance metric (Sec. 4.1); the clustering

algorithm (Sec. 4.2); the parametric agent (Sec. 4.3); and the evolutionary algorithm (Sec. 4.4). In Sec. 5 we present the methodology used to evaluate whether the persona agents can: 1)- represent the player traces from the corresponding cluster; 2)- generate diverse player-like behaviour; 3)- be used to represent player behaviour on a different level from the one they were evolved on. In Sec. 6, we discuss the obtained results and the limitations of the approach and finally, in Sec. 7, we conclude the paper.

2 Related Work

Previous work that is related to our approach falls within three main categories: clustering of player behavior; learning player behavior; and modeling player personas. In this section, we are going to cover some important work done in these areas.

Player clustering In order to learn or predict player profiles or personas, it is important to be able to group the players into clusters in such a way that all players within a cluster have similar behaviors. This makes having a metric for behavioural similarity an important component of the clustering process.

Osborn et al. (Osborn and Mateas 2014), proposed the "Gamalyzer metric", which builds upon the edit distance metric. This metric was capable of finding relevant sets of different player traces, with the advantage of being independent of the video game under analysis.

One of the most common methods of player clustering is the use of player behavior telemetry, meaning the use of player performance values, such as, for example, the time of the playing sessions. In Drachen et al. (Drachen et al. 2012) the authors perform cluster analysis on two very complex games, namely, "Tera" and "Battlefield 2: Bad Company 2". To do this, they used two clustering methods based on different algorithms: K-means and Simplex Volume Maximization. In the Battlefield game, both algorithms were capable of identifying a cluster with the veteran players, which have low representation in the population.

Another feature that can be used as means of clustering is how players navigate through a game level. Thawonmas et al. (Thawonmas et al. 2007) designed a clustering method based on game world landmarks. To find clusters, they calculated the Euclidean distance between the players according to their transition probabilities between the landmarks.

Besides the distance metric and the data used, one also has to consider the clustering algorithm itself. Drachen et al. (Drachen et al. 2014) compared several unsupervised clustering techniques. To compare them, the authors analyze the interpretable value of cluster profiles that are generated.

Understanding the need for clustering techniques that are straightforward to use and applicable to different types of games, Etheredge et al.(Etheredge, Lopes, and Bidarra 2013) proposed a generic clustering tool that can be easily applied by game designers to different games

A broader review and analysis of methods for clustering player behavior can be found in Bauckhage et al.(Bauckhage, Drachen, and Sifa 2014).

Learning/Modeling player behavior Regarding the topic of modeling players based on behavior, one relevant approach is through the use of Inverse Reinforcement Learning (IRL) (Ng, Russell et al. 2000). This algorithm class allows for the generation of reward functions based on observed player traces. An example of this approach can be seen in the work by Tastan et al. (Tastan and Sukthankar 2011). These authors propose an agent that generates policies for the competitive game Unreal Tournament. These policies are generated with rewards obtained from the IRL framework through the use of a linear solver. A limitation of this approach is that the obtained policies depend on the level structure that is being analyzed and therefore cannot be used in other levels.

Another work that also uses IRL can be seen in Lee et al. (Lee et al. 2014) where the authors applied apprenticeship learning in order to learn a platformer game, "Super Mario", from observing players. Other applications of IRL to learn different behaviours can be found in (Lee and Popović 2010) and (Almingol, Montesano, and Lopes 2013).

Tomai et al.(Tomai, Salazar, and Flores 2013) proposed a method for generating player-like behaviour based on learning Behavior Trees. Perez et al.(de Freitas, de Souza, and Bernardino 2018) also used the Behavior Trees technique, but in combination with grammatical evolution.

Bakkes et al.(Bakkes, Spronck, and van Lankveld 2012) presented and compared 4 different approaches, namely modeling based on player actions, based on player tactics, based on player strategies, and based on player profiling techniques. Hooshyar et al.(Hooshyar, Yousefi, and Lim 2018) presented a summary of the more relevant data-driven techniques and how they are comparable to theory-driven approaches.

Another field of research pertains to generating behaviour that is believably "human-like". Asensio et al. (Asensio et al. 2014) presented techniques based on genetic and evolutionary algorithms to create agents that are capable of passing the Turing test (Turing 2009) applied to observable behaviour. Another work that aimed at passing the Turing test can also be seen in Wang et al. (Wang et al. 2009).

Modeling player personas One relevant work regarding personas can be found in Holmgård et al. (Holmgård et al. 2018). In this paper, the authors implemented an approach based on psychological decision theory. In order to create different persona agents, termed procedural personas in the paper, they combine the Monte Carlo Tree Search (MCTS) algorithm with manually defined heuristics. Each heuristic introduces a bias on certain MCTS nodes, allowing the agent to have different "preferences" and follow different playstyles.

Another relevant work in this field is the one presented in Stahlke et al (Stahlke, Nova, and Mirza-Babaei 2019). In this paper, the authors propose an agent creation framework that is also based in heuristics. What is remarkable in this work is the addition of human qualities and limitations to agents, more precisely, visibility and memory. We will use a similar approach in this paper when defining the parametric agents, giving them knowledge of only the parts of the maps they have hitherto seen.

As an alternative to heuristics, Tychsen et al.(Tychsen and Canossa 2008) based their personas on video game metrics. These consisted of several data values that are obtained from the player interaction with the game world.

Holmgård et al. (Holmgård et al. 2016) created 3 types of evolving models, using neural networks to control them in order to generate human-like behaviour. Those models were, namely: the "personas", which evolved from the knowledge of game designers; the "clones", which evolved from observations of human players; and "specialized agents", which also evolved from observations but with the purpose of getting the representation as similar as possible to the player.

The use of emotional data to guide the creation of persona agents is not by itself novel. Barthet et al. used playtraces and emotional data to create persona agents (procedural personas) for a 3D car racing game leveraging the go-explore algorithm reinforcement learning algorithm (Barthet et al. 2022). They clustered players into four clusters, based on their skill, and then created agents that imitated behaviour, emotion, or both.

In conclusion, the work that will be presented on this paper joins together many different areas and approaches in order to create a pipeline for generating persona agents. The individual components of this pipeline are not revolutionary and many works have presented different versions of them. The creation of persona agents, also called procedural personas in literature, is equally not novel by itself. The approach proposed on this paper differs from all previous works by using parametric agents capable of playing different levels of the same game and by leveraging emotions during clustering to help the robustness of the behavioural coverage of the agents. With the proposed pipeline, we try to move closer to the goal of having persona agents capable of faithfully representing all player personas in any given level of a specific game, doing so without the need of any retraining when new levels are designed or old levels altered.

3 Game and Data Set

As a proof of concept, we used a simple 2D action-adventure test-bed game (Fig. 2). This game is inspired by top-down 2D games like the classic Legend of Zelda. The player controls a character that is armed with a sword and that can gather coins, which provide wealth, hearts, which provide health, and fight ghosts. The end goal for each level of the game is to find the stump of a tree. Once the player touches the stump, the tree grows and the level ends. This game was chosen for being easy to play while still allowing players to behave in a myriad of different ways.

We had 91 users play two levels of the game, the majority of which were first-year students of Psychology bachelor. The users received an explanation of the mechanics of the game and were given the liberty to play however they chose, with no time constraints. Throughout the play-through, key-pressing and other in-game data were collected. After playing each level, the players were further asked to self-annotate one of the three dimensions of the PAD emotional model (Russell and Mehrabian 1977). This model describes human emotions as locations in a 3-dimensional space, being the three dimensions of *Pleasure*, *Arousal*, and *Dominance*. Emotional data was collected to train a predictive model of emotion, not used for this paper, but also to study the use of emotional data for player clustering, which is done in this paper.

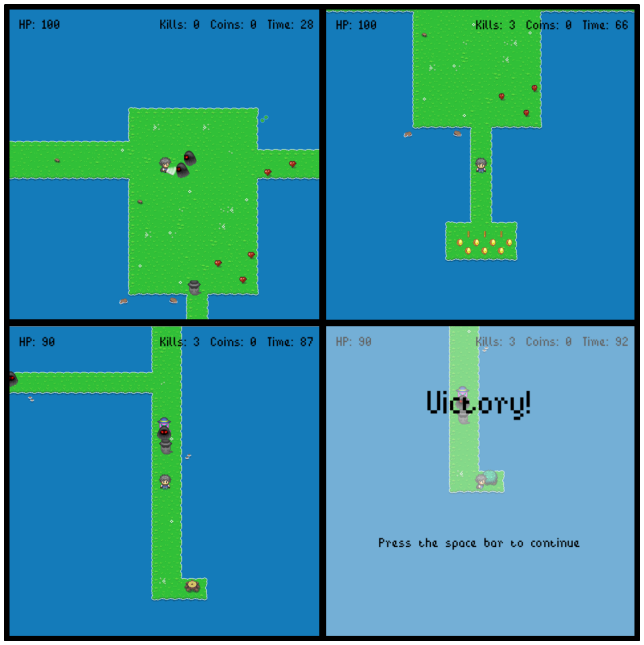

Figure 2: Screen shots of the used game. The player needs to traverse a field riddled with enemies, health-providing hearts, and coins (top right screenshot), in order to find a tree stump (shown on the bottom left screenshot). Once the player touches this stump, the tree grows, the game is over and the player wins (bottom right screenshot). If the player loses all its health points by touching enemies, then the game ends and the player loses. The player can use a sword to fight and kill the enemies (shown in the top left screenshot).

Each player was randomly assigned one of the three dimensions to annotate and given an explanation of the PAD model and their assigned dimension. They only continued with the annotation once they claimed to understand the dimension which they would be annotating. The annotation method used was after the fact and continuous, similar to the method described in (Lopes, Yannakakis, and Liapis 2017). After playing each level, the players watched a video of their play-through and were given the ability to rate their emotions, based on the assigned dimension, using the up and down arrows of the keyboard, denoting an increase or decrease of the dimension. In the process, a line showing the evolution of their assigned emotional dimension is displayed over the video. The collected play-through data along with the emotional annotations serve as the data set for both training and validating the pipeline here described.

4 The Persona Agent Pipeline

The main goal of this paper is to represent the whole diversity of player behaviors by creating a set of **persona agents**, each one representing the behaviour of a sub-set of play-

ers. To do so, we must first define these sub-sets of players, which will henceforth be called **persona clusters**. If a game already has a defined set of persona clusters, then this step is already accomplished and the sets can be used as they are. However, this will not be true for most games. To obtain persona clusters from data, we can run a clustering algorithm on collected traces from real players. By doing so, we will obtain clusters that contain traces that are similar to themselves.

Once we have the persona clusters, we need to design agents that represent each of these clusters: the persona agents. In this paper, we propose using an evolutionary algorithm and rule-based **parametric agents** (Sec. 4.3), being agents that have their behaviour defined by a set of parameters. By doing so, defining a persona agent means finding the set of parameters that characterizes the parametric agent that best represents a given persona cluster.

A diagram of the pipeline for defining persona agents can be found in Fig. 1. To use this pipeline on any given game, four things need to be implemented: a distance metric, a clustering algorithm, a parametric agent, and an evolutionary algorithm. With these four components, player traces can be turned into persona agents. The quality and coverage of these persona agents will obviously depend on the implementation of the components as well as on the quantity and quality of collected player traces.

In the following sections, we will describe three different methods for implementing a distance metric (Sec. 4.1), describe the clustering algorithm (Sec. 4.2), the parametric agent used (Sec. 4.3), and the evolutionary algorithm (Sec. 4.4), which together allowed us to obtain persona agents from collected player traces.

4.1 Distance Metric

A distance metric is a method of obtaining a value of similarity between two traces. The lower the distance metric between two traces, the more similar those traces are. Defining such a metric that can be used in all situations is not a trivial task. Looking at observed behaviour, designers might be interested in looking at low level actions or high level decisions and interests. They can also be interested on the experience, or emotional traversal, that said players have when interacting with the game. Furthermore, they can be simultaneously interested in both.

We thus decided to take behaviour and player experience into consideration when defining distance metrics. We will begin by looking on how we can encode playtraces so that they can represent behaviour, emotional experience, or a mixture of both. We will then discuss how we can measure the distance between the encoded traces.

Behaviour For this proof of concept, we decided to consider that players that take different routes and chose different actions when playing a level of the game are behaving differently. Behaviour is thus here seen as the sequence of low level actions chosen by a player. We chose to consider the lowest action level, that is, the actions dictated by the keys being pressed at each of the game's time steps. A player game trace can therefore be represented by a string such as "dddwsd", meaning the player pressed the "d" key for the first three time steps, then the key "w" in the fourth, followed by the keys "s" and "d" in the fifth and sixth time steps respectively. This representation loses the contextual information present in the level being played, but is abstract enough to be used in most games.

Emotion To also take player experience into consideration and given the dataset used provided information about the emotional experience of the players, we decided to also transform this information into strings so that it could be used alongside the behavioural data. To do so, we took the continuous annotation of the PAD dimensions and transformed it into a sequence of 0s, 1s and 2s, where 0 signified the dimension was increasing, 1 signified the dimension remained unaltered and 2 signified the dimension was increasing. The string "000112" thus signified that the annotated dimension decreased for the first three time steps, then remained unaltered for two time steps and then increased in the last time step.

Behaviour and Emotion The behavioural and emotional information can be fused, creating strings with alternated actions and emotional state change. An example is the string "d0d0d0w1s1d2", a fusion of the previously used examples for the behavioural and emotional strings. This string represents a trace where the player presses the "d" key and has a decrease in the annotated emotional dimension for the first three time steps, then presses the "w" key and the emotional dimension remains unaltered during the fourth and fifth time step, during which the player presses the "s" key, until finally, in the sixth time step, the player presses the "d" key and has an increase in the annotated emotional dimension.

Levenshtein Distance To match the choice of using strings to represent traces, we implemented a distance metric based on the Levenshtein distance (Levenshtein et al. 1966) between the previously described string representations of traces. We therefore judged the similarity between two game traces based on the minimum number of edits, deletions and additions that are required to transform the string representation of one of the traces into that of another. For example, a trace represented by the string "dddwsd" has a Levenshtein distance of 4 to the trace represented by the string "wwddsdd".

4.2 Clustering Algorithm

Having settled on the distance metrics that would be used, we had a foundation to choose the clustering algorithm we would be employing to find the persona clusters. We did not have previous knowledge that allowed us to predict the number of persona clusters we could expect to find. Therefore, we had preference for using a clustering algorithm that did not require the number of clusters to be given as a parameter. We also required a clustering algorithms that did not rely on Euclidian distances and could be used with the distance metric we had previously defined.

With the aforementioned in mind, we decided to use the affinity propagation clustering algorithm (Frey and Dueck 2007), as implemented by the scikit-learn python library

(Pedregosa et al. 2011). This algorithm allows for any distance metric to be used and is able to define the number of clusters automatically.

4.3 Parametric Agent

A parametric agent is here defined as a rule-based agent whose behaviour is dictated by the values of a set of parameters, which can be thought of as the genome of the agent. Different sets of parameters will produce different behaviours. This encoding of the agent's behaviour into a set of parameters allows the use of evolutionary algorithms (Bäck and Schwefel 1993) as a way to find parametric agents that behave in a particular way. In this section, we will present our implementation of a parametric agent for the used game.

For the agent to represent player behaviour, we had to endow it with the ability to do that which players do. Namely, the agent had to be able to move and explore the map, fight enemies, collect items and reach the final objective. To achieve this, we began by endowing the agent with the ability to navigate the map.

Since a player would not have access to the full map when playing a level of the game, we decided not to give knowledge of the full map to the agent. The agent would thus have to construct a map of the level as it explored, much as a player would. This is not to say that giving the agent access to the full map cannot be a viable solution. This can be done, for example, to simulate a player that is playing a level for a second time and already knows where things are located.

A map is built for the agent as it traverses the game. This is done by creating a square object in the game of a set dimension. The dimension of this square corresponds to the granularity of the internal map. We iterate this square over all non-overlapping positions on the field of view of the agent, checking at each iteration if the square collides with any object. If no collision is detected, the position of the square is added to a "free positions graph". All adjacent nodes of this graph are connected. In this way, the agent can run classical path-finding algorithms on this graph, like Dijkstra's algorithm (Dijkstra et al. 1959), in order to navigate the level.

Having a graph of the visited locations of the map, the agent is able to navigate by finding the shortest path to a given known location and following it. The agent can also reach any object it previously found, including enemies. Endowing the agent with the ability of fighting enemies was only a question of having the agent move towards the closest enemy and begin attacking when the enemy was within reach. Our agent was thus able to do everything required to play the game: it knew how to act. It had, however, no policy, that is, no way of knowing when it should do any of these things nor how it should explore the level. This is solved by the parametric nature of the agent. Each parametric agent will have a set of parameters, the value of which will dictate the chosen actions of the agent. The parameters are the following:

1. $\mathbf{O}_p$ - Objective Parameter.
2. $\mathbf{F}_p$ - Fighting Parameter.
3. $\mathbf{C}_p$ - Currency Parameter.
4. $\mathbf{H}_p$ - Health Item Parameter.
5. $\mathbf{E}_p$ - Exploration Method Parameter.

Each of these parameters can have a value $\in [0, 100]$. The first four parameters encode the preference the parametric agent will give to reaching the objective, fighting enemies, gathering currency and gathering health items, respectively. The fifth parameter encodes the method of exploration that the agent will pursue. The remainder of this section will be dedicated to explaining how these parameters dictate the behaviour of the agent.

At any given moment, the agent can follow any of five macro-actions:

- **Reach Objective** - Go to the final objective (if the objective has been found).
- **Fight** - Fight the closest enemy (if any enemy is visible).
- **Gather Currency** - Move towards the closest currency item (if any currency item has been found and not gathered).
- **Gather Health Items** - Move towards the closest health item (if any health item has been found and not gathered).
- **Explore** - Continue exploring the level and finding new locations (if the level has not been fully explored).

Each one of these macro-actions is given a value according to the following equations:

$$V(\textbf{Reach Objective}) = \frac{\mathbf{O}_p}{distance(objective)} \quad (1)$$

$$V(\textbf{Fight}) = \frac{\mathbf{F}_p}{distance(enemy)} \quad (2)$$

$$V(\textbf{Gather Currency}) = \frac{\mathbf{C}_p}{distance(currency)} \quad (3)$$

$$V(\textbf{Gather Health Items}) = \frac{\mathbf{H}_p}{distance(health\_item)} \quad (4)$$

$$V(\textbf{Explore}) = \lambda \times \xi \quad (5)$$

with $distance(x)$ returning the distance to the nearest object of type x, λ being an exploration constant set as 0.5 for this implementation and:

$$\xi = \begin{cases} 0, & \text{if all locations in the map have been discovered} \\ 1, & \text{otherwise} \end{cases} \quad (6)$$

At each moment, the parametric agent will choose to pursue the macro-action with the greatest value, thus guided by the distance to the several game objects and its parameters.

The only parameter not yet mentioned, $\mathbf{E}_p$, dictates how the agent explores the level. When the agent decides to explore, it has to choose which of the boundary nodes on its internal map graph it should visit so as to expand its knowledge of the map. Boundary nodes are the nodes that lie at the limit of the agent's map graph whose neighbours have not been discovered, be them reachable positions or walls.

The agent will choose to explore the node that minimizes the following distance, d:

$$d = \sqrt{\left(i_x + \frac{p_x * \mathbf{E}_p}{\frac{\mathbf{MaxE}_p}{2}} - n_x\right)^2 + \left(i_y + \frac{p_y * \mathbf{E}_p}{\frac{\mathbf{MaxE}_p}{2}} - n_y\right)^2} \quad (7)$$

where (i_x, i_y) is the initial position of the player/agent on the level, (p_x, p_y) is the current position of the player, (n_x, n_y) is the position of the node that is being considered for exploration and $\mathbf{MaxE}_p$ is the maximum value that $\mathbf{E}_p$ can take.

In practice, if the value of $\mathbf{E}_p$ is close to 0, the agent will explore the node closest to the initial position, resembling a breadth first search algorithm. If $\mathbf{E}_p$ is close to 50, the agent will explore the node that is closest to itself. If $\mathbf{E}_p$ is close to 100, the agent will explore the node that is furthest away from both itself and the initial position.

We have thus described the parametric agent and how its parameters dictate its behaviour. In the next section, we will describe the evolutionary algorithm used to find the sets of parameters that will be used for the persona agents.

4.4 Evolutionary Algorithm

Given a set of persona clusters (Sec. 4.2) and a parametric agent (Sec. 4.3), we now need a method of finding the set of parameters of the parametric agent that dictate a behaviour that most resembles the traces found on each persona cluster. To do so, we will use an evolutionary algorithm.

For each persona cluster, we initialize a population of 100 parametric agents with random sets of parameters. At each generation, all of the agents play the same level of the game. Their actions are recorded and compared to the action traces of all the members of the chosen persona cluster. This comparison is done using the average distance metric (Sec. 4.2) to all the members of the cluster. This average distance is used to assign a fitness to each of the agents: the greater the average distance between the agent trace and the cluster traces, the lower the fitness of the agent. All fitness values will therefore be negative, with the highest possible fitness being 0, when the traces compared are exactly the same.

Once every agent has been assigned a fitness, the 30% with the highest fitness are chosen to reproduce and be a part of the next generation. The reproduction is done by randomly selecting two parent agents and randomly mixing their parameters to create a new child agent (uniform crossover). With a 20% probability, each of the parameters can suffer a mutation and take a random value. The top 30% of the population thus reproduces between itself until the population has once again 100 agents. This process is repeated for 100 generations. The agent with the set of parameters that allow it to have the best fitness in the last population is then chosen as the persona agent.

5 Results

We ran a set of experiments to evaluate whether the persona agents created using the previously presented pipeline can: 1)- represent the player traces from the corresponding cluster; 2)- generate diverse player-like behaviour; 3)- be used to represent player behaviour on a different level from the one they were evolved on.

The experience was done as follows:

1. We clustered the player traces collected from level A into clusters that represent different sets of players. We created three different sets of clusters, one for each distance metric discussed in Sec. 4.1: behaviour, emotion, and both.
2. We used the evolutionary algorithm described in Sec. 4.4 to obtain persona agents that represent each of the clusters for the three sets obtained from the previous step. From this, we obtained three sets of persona agents.
3. We ran those agents on level A
4. Taking the persona agent traces, we assigned them to the clusters that had been obtained in step 1. Each agent was assigned to the cluster to which it was closest (average behavioural distance to all members of the cluster). The behavioural distance had to be used for all agents given that the agents did not produce emotional data. The results of this can be found on the top half of Table 1.
5. Finally, we ran the persona agents on level B and assigned them to the player clusters that were found on this level (obtained using the same method described in step 1). The results of this can be found on the bottom half of Table 1.

For the sake of brevity and clarity, we will henceforth refer to the persona agents obtained from clustering using the behavioural distance metric, the emotional distance metric and the mixed distance metric as, respectively, **behavioural agent**, **emotional agent** and **mixed agent**.

The clustering algorithm on level A found 9 clusters when using the behavioural distance metric, 8 clusters using the emotional distance metric and 10 clusters using the mixed distance metric. We thus evolved 9 behavioural agents, 8 emotional agents and 10 mixed agents: one agent to represent each cluster.

We find in Table 1 the coverage and accuracy obtained in level A from the persona agents obtained from clustering using the three distance metrics described in Sec. 4.1. We here judge coverage by the percentage of clusters that the persona agents were able to represent (row "% Represented"). We consider that a persona agent represents a cluster when that agent is assigned to that cluster. Accuracy is assessed by looking at the average behavioural distance of the persona agent trace to all the members of the clusters the persona agent represents (row "Avg. Dist. to Rep. Cluster"). To contextualize the results, the average distance between player traces in level A is 156.

We can observe that despite obtaining the highest coverage, the emotional agent fares considerably worse in terms of accuracy when compared to the other two agents in level A. The behavioural agent obtains the highest accuracy but a low coverage and the mixed agent is able to get the most balanced results.

We found that the persona agents were seldom assigned to the clusters from which they were evolved. This happened for the three types of agents. For example, only 4 of the 10 mixed agents were assigned to the original clusters.

Level A				
Distance Metric	**# Clusters**	**# Clusters Represented**	**% Represented**	**Avg. Dist. to Rep. Cluster**
Behaviour	9	4	44%	86
Emotion	8	6	75%	154
Mixed	10	7	70%	112

Level B				
Distance Metric	**# Clusters**	**# Clusters Represented**	**% Represented**	**Avg. Dist. to Rep. Cluster**
Behaviour	7	3	43%	116
Emotion	9	5	56%	70
Mixed	12	5	42%	57

Table 1: Results for level A and level B, showing the number of clusters, the number of represented clusters, the percentage of represented clusters and the average distance to represented clusters for the persona agents created using the three different types of distance metrics during the clustering phase of the pipeline.

The results for level B can be found in the lower half of Table 1. The results here are considerably different from the ones obtained on level A, as would be expected given that the agents were evolved using traces from level A and not from level B. The emotional agent is the one with the greatest coverage, followed by the behavioural agent and the mixed agent with very similar coverage values between them. In terms of accuracy, the mixed agent is the one with the best result, having the lowest average distance to the represented cluster. Next comes the emotional agent and lastly, the behavioural agent. To put the accuracy values in perspective, the average behavioural distance between all player traces in level B is 118.

6 Discussion

In the beginning of Sec. 5, we enumerated three research questions. Having now presented the results of the study, let us try and answer them. Firstly, we wanted to find if the persona agents created using the proposed pipeline accurately represented the player traces from the corresponding cluster. We believe that based on the results, the answer to that question is no regarding the presented implementation of the pipeline. The persona agents did not reliably represent of the individual clusters that evolved them, since the majority of the persona agents did not produce traces that would belong to their corresponding cluster.

Secondly, we wanted to find if the persona agents were able to generate diverse player-like behaviour. To this question, we can give a more positive answer. Although the persona agents did not faithfully represent the clusters used to evolve them, they were able to represent a majority of the player traces (emotional and mixed agent in level A and emotional agent in level B). The behaviour generated can be considered diverse, as the persona agent traces were classified as belonging to different clusters, and player-like, as the distance between the persona agent traces and the traces in the cluster they were assigned to was always inferior to the average distance between player traces.

Finally, we wanted to know whether the persona agents evolved in one level could be used on a different level of the same game. The answer to this question is also positive. The persona agents evolved using traces from level A were successfully used to represent clusters of player traces from level B. However, the resulting coverage and accuracy was, as expected, lower than the one obtained on the original level.

It could be argued that the best performing persona agent was the mixed agent, which had information regarding both the actions and the emotional state of the players. It showed a good balance between coverage and accuracy in level A and had the best accuracy in level B. Surprisingly, the emotional agent had the best coverage in level B, having also a good accuracy. A possible explanation for this is that the emotional traversal of a level is a more abstract representation of players' preferences than low level actions, providing a better transfer of information regarding play-style between different levels.

The overall results obtained are far from ideal in terms of coverage and accuracy, but they do show that the generated persona agents are able to represent a fraction of player traces, even when used in a level different from the one the agents had been evolved in. Improvements on any of the several modules of the implemented pipeline, such as improving the clustering algorithm or having parametric agents capable of more complex behaviour, should improve the final trace coverage and accuracy of the final persona agents.

The way in which we have implemented the pipeline has a number of shortcomings. The metric for behavioural distance used (Sec. 4.1) considers that any difference in low-level actions represents different behaviour. If two players that play in very similar ways happen to randomly choose a different path in an intersection, then using the Levenshtein distance between their traces will consider the players as behaving differently, although the different sequences of actions stem from a random choice and not from an intrinsic different way of playing. This metric is therefore blind to the

intentions and motivations of the players, focusing only on the sequence of actions chosen. An alternative could be to transform the low-level actions into high level pursuits, such as considering that any movement that is not directed to any specific visible object is exploration, regardless of the direction chosen. The data used when generating traces based on the emotional experience of players is also not representative of the complete experience that the player had and lacked contextual information.

The parametric agents implemented were simple agents with few parameters. Players can behave in ways that our agents would not be able to mimic even if all possible combinations of parameters were tried. One approach to mitigate this problem is making the parametric agent more complex, allowing for more intricate behaviour. This could be done by expanding the number of parameters and the amount of inputs considered when making decisions. Conditional parameters could also be used, dictating the influence that certain parameters or in game variables have on the strength of other parameters. One example could be having a parameter that translates the impact that the health of the agent has on its willingness to fight. This added complexity will, however, likely come at the cost of needing more generations to have the evolutionary algorithm converge in a set of parameters for each cluster.

Given the lack of emotional data from the persona agents, we were forced to measure the similarity between the persona agent traces and the player clusters using the behavioural distance metric. A possible venue of future work is the exploration of using emotional predictive models to have persona agents also generate emotional traces. We could thus use all three distance metrics when evaluating the solution.

Although the main goal is the automatic simulation of player behaviour, the approach here proposed still requires an initial set of playtraces. This means that the approach requires data collection or data from previously done data collection rounds. Notwithstanding, once the persona agents have been obtained, they can be used for the remainder of the game's life-cycle without requiring any further data collection.

The pipeline presented in this paper could be used to obtain persona agents for any game, provided there were enough player traces available and that the main components of the pipeline were implemented with the particular game and objectives in mind. But there is still room for improvement, as implementing each of the components anew requires a considerable amount of work. Designing parametric agents from scratch can be burdensome, for example. One way to improve this would be to define a generic parametric agent, such that it could be used for several different games, requiring only a number of functions to be implemented. This generic agent could already have all the logic for turning parameters into different behavioural patterns, with only the functions representing each individual pattern needing to be implemented. We could thus create a persona agent framework by having a number of distance metrics, clustering algorithms and evolutionary algorithms already connected to such a generic parametric agent and ready to use.

7 Conclusion

In this paper, we have presented a pipeline for the creation of level robust persona agents based on a set of player traces with both behavioural and emotional information. Following the proposed approach, we used the traces collected from 91 users playing two levels of a 2D Zelda-like game to create persona agents capable of generating diverse player-like behaviour in both the level they were evolved in and in a different one.

The modular pipeline here presented is a stepping stone in the direction of robust persona agents capable of representing player personas on any given level without the need for agent retraining. Further research on the several modules of the pipeline as well as on the pipeline architecture itself is still required in order to fully achieve that goal.

Acknowledgments

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UIDB/50021/2020 and 2020.05865.BD.

References

Almingol, J.; Montesano, L.; and Lopes, M. 2013. Learning multiple behaviors from unlabeled demonstrations in a latent controller space. In *International conference on Machine learning*, 136–144. PMLR.

Asensio, J. M. L.; Peralta, J.; Arrabales, R.; Bedia, M. G.; Cortez, P.; and Peña, A. L. 2014. Artificial intelligence approaches for the generation and assessment of believable human-like behaviour in virtual characters. *Expert Systems with Applications*, 41(16): 7281–7290.

Bäck, T.; and Schwefel, H.-P. 1993. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1): 1–23.

Bakkes, S. C.; Spronck, P. H.; and van Lankveld, G. 2012. Player behavioural modelling for video games. *Entertainment Computing*, 3(3): 71–79.

Barthet, M.; Khalifa, A.; Liapis, A.; and Yannakakis, G. 2022. Generative personas that behave and experience like humans. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 1–10.

Bauckhage, C.; Drachen, A.; and Sifa, R. 2014. Clustering game behavior data. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3): 266–278.

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

Cooper, A. 1999. The inmates are running the asylum. In *Software-Ergonomie'99*, 17–17. Springer.

de Freitas, J. M.; de Souza, F. R.; and Bernardino, H. S. 2018. Evolving Controllers for Mario AI Using Grammar-based Genetic Programming. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. IEEE.

Dijkstra, E. W.; et al. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271.

Drachen, A.; Sifa, R.; Bauckhage, C.; and Thurau, C. 2012. Guns, swords and data: Clustering of player behavior in computer games in the wild. In *2012 IEEE conference on Computational Intelligence and Games (CIG)*, 163–170. IEEE.

Drachen, A.; Thurau, C.; Sifa, R.; and Bauckhage, C. 2014. A comparison of methods for player clustering via behavioral telemetry. *arXiv preprint arXiv:1407.3950*.

Etheredge, M.; Lopes, R.; and Bidarra, R. 2013. A generic method for classification of player behavior. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 9, 2–8.

Frey, B. J.; and Dueck, D. 2007. Clustering by passing messages between data points. *science*, 315(5814): 972–976.

Holmgård, C.; Green, M. C.; Liapis, A.; and Togelius, J. 2018. 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. 2016. Evolving models of player decision making: Personas versus clones. *Entertainment Computing*, 16: 95–104.

Hooshyar, D.; Yousefi, M.; and Lim, H. 2018. Data-driven approaches to game player modeling: a systematic literature review. *ACM Computing Surveys (CSUR)*, 50(6): 1–19.

Lee, G.; Luo, M.; Zambetta, F.; and Li, X. 2014. Learning a Super Mario controller from examples of human play. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. IEEE.

Lee, S. J.; and Popović, Z. 2010. Learning behavior styles with inverse reinforcement learning. *ACM transactions on graphics (TOG)*, 29(4): 1–7.

Levenshtein, V. I.; et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, 707–710. Soviet Union.

Lopes, P.; Yannakakis, G. N.; and Liapis, A. 2017. Ranktrace: Relative and unbounded affect annotation. In *2017 Seventh International Conference on Affective Computing and Intelligent Interaction (ACII)*, 158–163. IEEE.

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

Osborn, J. C.; and Mateas, M. 2014. A game-independent play trace dissimilarity metric. In *FDG*.

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

Russell, J. A.; and Mehrabian, A. 1977. Evidence for a three-factor theory of emotions. *Journal of research in Personality*, 11(3): 273–294.

Stahlke, S.; Nova, A.; and Mirza-Babaei, P. 2019. Artificial Playfulness: A Tool for Automated Agent-Based Playtesting. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–6.

Tastan, B.; and Sukthankar, G. 2011. Learning policies for first person shooter games using inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 7, 85–90.

Thawonmas, R.; Kurashige, M.; Chen, K.-T.; et al. 2007. Detection of Landmarks for Clustering of Online-Game Players. *Int. J. Virtual Real.*, 6(3): 11–16.

Tomai, E.; Salazar, R.; and Flores, R. 2013. Simulating aggregate player behavior with learning behavior trees. In *Proceedings of the 22nd Annual Conference on Behavior Representation in Modeling & Simulation*.

Turing, A. M. 2009. Computing machinery and intelligence. In *Parsing the turing test*, 23–65. Springer.

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.

Wang, D.; Subagdja, B.; Tan, A.-H.; and Ng, G.-W. 2009. Creating human-like autonomous players in real-time first person shooter computer games. In *Twenty-First IAAI Conference*.