Playtesting: What is Beyond Personas

Sinan Ariyurek, Elif Surer, Aysu Betin-Can
Graduate School of Informatics
Middle East Technical University
06800, Ankara, Turkey
{sinan.ariyurek, elifs, betincan}@metu.edu.tr

***Abstract*—Playtesting is an essential step in the game design process. Game designers use the feedback from playtests to refine their design. Game designers may employ *procedural personas* to automate the playtesting process. In this paper, we present two approaches to improve automated playtesting. First, we propose a goal-based persona model, which we call *developing persona*—developing persona proposes a dynamic persona model, whereas the current persona models are static. Game designers can use the developing persona to model the changes that a player undergoes while playing a game. Additionally, a human playtester knows which paths she has tested before, and during the consequent tests, she may test different paths. However, RL agents disregard the previously generated trajectories. We propose a novel methodology that helps Reinforcement Learning (RL) agents to generate distinct trajectories than the previous trajectories. We refer to this methodology as Alternative Path Finder (APF). We present a generic APF framework that can be applied to all RL agents. APF is trained with the previous trajectories, and APF distinguishes the novel states from similar states. We use the General Video Game Artificial Intelligence (GVG-AI) and VizDoom frameworks to test our proposed methodologies. We use Proximal Policy Optimization (PPO) RL agent during experiments. First, we show that the playtest data generated by the developing persona cannot be generated using the procedural personas. Second, we present the alternative paths found using APF. We show that the APF penalizes the previous paths and rewards the distinct paths.**

***Index Terms*—Reinforcement Learning, Player Modeling, Automated Playtesting, Play Persona**

I. Introduction

Game designers envision how a game will work during a play. As the game develops, it becomes increasingly difficult to predict how players will interact with the game. Playtesters help out this process by providing helpful feedback by playing the game. However, human playtesting introduces latency and additional costs to the process. Therefore, researchers proposed methods to automate the playtesting process [1] [2] [3].

The playtesting process may employ different players. These players will respond to the game differently, and they will generate unique play traces. The game designer can use these play traces to shape her game. In order to automate playtesting with different players, researchers replaced these playtesters with procedural personas. A procedural persona describes an archetypal player's behavior. Researchers used personas to playtest a Role-Playing Game [4] and a Match-3 [5] game. As a result, personas enabled distinct playstyles and helped to playtest a game like distinct players.

In order to realize the personas using RL agents, researchers used a utility function [6] to define the decision model of a persona. Researchers replaced the reward mechanism of the game with a utility function. However, this replacement makes personas bound to the utility function. Since the utility function was static, the realized personas were also static. Therefore, this approach is not flexible to create dynamic personas. For example, a player may change her objectives while playing the game. The current utility function cannot adapt to these changes.

Bartle [7] presents examples of these changes that a player can undergo while playing an Massively Multiplayer Online Role-Playing Game (MMO-RPG). We believe that the change in the playstyle occurs after accomplishing a goal. For example, a player may be interested in killing certain monsters or opening treasures in pursuit of finding a required item. This player is not necessarily a Monster Killer or Treasure Collector, but a player-driven by a goal. Therefore, we propose a goal-based approach to overcome this problem. The goal-based approach is previously used in automated video game testing agents [8] and was found more practical compared to non-goal-based approaches.

The goal-based persona model consists of multiple goals that are linked. Each goal consists of criteria and a utility function. The utility function serves the same purpose as in procedural personas. The criteria determine until which condition the current goal is active. When the current goal criteria are fulfilled, the next goal becomes active. The agent plays until the last goal criteria is fulfilled or until the end of the game. The game designer sets the criteria and utility functions of each goal. The goal structure enables the creation of dynamic personas. Additionally, this approach gives a more granularized control over a persona. The game designer can create variations of Monster Killer by setting different criteria. In order to playtest a casual Monster Killer, the game designer may set a health threshold as the criterion; and to playtest a hardcore Monster Killer, the game designer may set the percentage of monsters killed as the criterion.

Furthermore, the game designer may envision a game with various endings. In order to playtest her game, she utilizes and an Exit persona. The game designer analyzes the playtests and sees that the playtests only provide data for one of the possible endings. This shortcoming is not caused by the persona but by the inherent nature of RL algorithms. RL algorithms such as Deep Q-Network (DQN) [9], Proximal Policy Optimization (PPO) [10], and Monte Carlo Tree Seach (MCTS) [11] generate a single trajectory for a game. Though these algorithms may find different paths due to the random

initial state of the Neural Network (DQN and PPO) or due to the game's nondeterminism, finding distinct paths is not the primary objective of these algorithms.

Exploration methods in RL improve the agent's policy by motivating the agent to explore the environment. As the agent explores an environment, the agent improves its policy. The researchers proposed methods to motivate the agent to explore less visited states [12] [13] [14]. Exploration methods vastly improved the agent's score in Montezuma's Revenge [12]. Therefore, we propose our APF approach based on exploration methods. We penalize the agent when it visits states similar to the ones in the previous playtests. We reward the agent when the agent visits a novel state. We show how we augment an RL agent using the APF framework to generate new and unique playtests.

In this paper, we list the contributions as follows. Our first contribution is a goal-based play persona. A goal-based persona is more flexible and robust than the current persona models. We show how game designers can utilize the developing persona to empower the playtesting. Our second contribution is the Alternative Path Finder. We present a generic APF framework that can augment every RL agent. We use the GVG-AI [15] and VizDoom [16] environments to demonstrate our proposed methodologies.

This paper is structured as follows: Section II describes the examples and methodologies of related research. Our proposed methodology that consists of developing persona and APF is presented in Section III. Section IV describes our experimentation setup and Section V presents the results of these experiments. Section VI discusses the outcomes of the strategies used, their contributions and limitations. Lastly, Section VII concludes this paper.

II. Related Research

Playtesting is a methodology used in the game design process. Playtesters test a game, and feedback is collected from these playtesters. The game designers use this feedback to improve their game. As this process requires a human effort, researchers proposed methods to automate game playtesting. Powley et al. [1] coupled automated playtesting with a game development application. Gudmundsson et al. [2] trained a convolutional neural network to predict the most humane action in Candy Crush, and they used this network to assess level difficulty. Roohi et al. [3] used RL and a population model to determine level difficulty for Angry Birds Dream Blast. These approaches derive the automated playtesters from an individual player archetype. Nevertheless, during a playtest, there can be various playtesters resembling a different player archetype.

In playtesting, personas provide game designers information about how different player archetypes would play the game. Persona is a fictional character that represents a user type. Bartle [17] introduces a taxonomy of personas that are identified from a Multi-user Dungeon Game. The author acknowledges these four distinct personas as Socializers, Explorers, Achievers, and Killers. The author introduces a graph with axes that maps the players' interest in a persona. Bartle [7] extends this research by introducing development sequences for personas. The development sequences reveal how and why a player may change to a different persona. Tychsen and Canossa [18] present a study on collecting game metrics and how different personas can be identifiable by these metrics. The authors present the personas of the game Hitman Blood Assassin. The game identifies these personas: *Mass Murderer*, *Mass Murderer*, *Mad Butcher*, and *The Cleaner*. They argue that a persona can be recognized using the metrics collected from a play trace. These approaches focus on identifying different personas in a game.

In order to automate the playtesting, researchers proposed techniques to realize the decision model of a persona. Holmgård et al. [6] used a utility function to realize the decision model of a persona. This utility function is used as the reward function for the Q-Learning agent. The agents are exercised in an environment called MiniDungeons. The agents produced play traces as if they are of a specific persona. Holmgård et al. [19] extended their previous work by substituting the Q-Learning agents with a neural network. The inputs to the neural network were hard-coded handpicked parameters. The authors used a genetic algorithm to find the weights of this neural network. They called their new method 'evolved agent'. Evolved agent required less training than the Q-Learning agent and was able to generalize to other levels better. Holmgård et al. [20] upgraded the environment to MiniDungeons 2. In this study, the authors proposed to generate personas using MCTS agents that use their proposed utility function. Their reasoning for using MCTS, especially Vanilla MCTS [11], was to provide faster data to the game designer. In Q-Learning and Evolved agents, these agents have to be trained first. Holmgård et al. [4] extends the MCTS by improving the selection method of MCTS. In their previous study, the authors state that the Mini Dungeons 2 game was too complex for Vanilla MCTS. Therefore, they model a new selection phase that is specifically tailored towards a specific persona. They accomplish this by evolving the UCB formula by a genetic algorithm. The authors crafted the fitness function of each persona. This fitness function also determined the fitness function of the evolutionary algorithm. The evolved UCB formula improved their results among every persona. Silva et al. [21] used personas to playtest the Ticket to Ride board game. The authors designed four different competitive personas to play the board game. The authors handcrafted a set of heuristics for each persona. They showed that personas revealed useful information that the game rules did not provide rules for two situations. Mugrai et al. [5] employed four different personas for Match-3 games. These personas are *Max Score*, *Min Score*, *Max Moves*, and *Min Moves*. The authors showed that these four personas could give the game designer valuable information about a level.

The main drawback of persona research is the utility function. First, the utility function is static and stays constant throughout the game. Therefore, the game designers cannot model players with development sequences [7]. Second, depending on the level layout, personas can execute a similar sequence [4]. Hence, the synthetic playtesters would provide ineffective feedback. Lastly, synthetic playtesters are realized

using RL agents. Since RL agents optimize the total accumulated reward, synthetic playtesters would not test all playable paths.

An RL agent explores the environment to learn which action yields the highest reward in a state. In order to learn this policy, the RL agent has to explore the environment. Intrinsically motivating an RL agent to explore novel states is an exploration problem. The researchers proposed different ways to make agents explore distinct states of the environment. Count-based approaches reward the less-visited states more than frequently visited states. Therefore, the agent becomes inclined to visit the less visited states. The count is formulated using a density model [12], a neural density model [22], a hash table [23], and exemplar models [13]. Another proposed approach is to augment the reward function by measuring the agent's uncertainty about the environment. Researchers measured the uncertainty using bootstrapped DQN [24], state-space features [14], and error of a neural function [25]. Additionally, researchers proposed approaches that explore the state space by optimizing the state marginal distribution to match a target distribution [26]. These exploration proposals intelligently incite the agent to explore the environment. The goal of exploration is not to find a unique way of playing but to find the best path every time we execute the RL agent. However, these methods can differentiate between similar states and new states. We base our APF proposal based on this accomplishment.

III. Methodology

In this paper, we address the shortcomings of the procedural persona approach, and we propose a goal-based approach, the developing persona. Furthermore, we recognize there may be alternative playtests that can help the game designer. We propose APF to discover those playtests.

In the following subsections, first, we introduce the goal-based persona approach. Afterward, we present the necessity for an APF and introduce the foundation of APF. Next, we show how we use the techniques in exploration field to implement the APF. Finally, we describe how to use APF with an RL agent.

A. Goal-based Persona

A persona reflects an archetypal player's decision model. In order to realize a persona, first, the persona's decision model should be translated to game conditions. Second, an actor should play according to this translation. Researchers [4] [5] proposed using a utility function to map the decision model to game conditions. This utility function replaces the reward mechanism of the environment and provides a tailored reward mechanism for each persona. Researchers [4] [5] used RL agents as actors. Consequently, these RL agents are akin to synthetic playtesters that represent the decision model of a persona. These playtesters, procedural personas, represented various personas such as the Monster Killer, Treasure Collector, and Exit personas. In this paper, we extend the procedural persona framework by introducing a goal-based persona.

Fig. 1: An example level created by GVG-AI framework.

We propose a goal-based persona to generate a more customizable playtester. We have two reasons that a goal-based persona would be beneficial for game designers. First, the game designer does not have granular control over the personas. For example, the game designer may want to playtest a monster killer persona that kills monsters until its health drops below a certain percent. However, when to cease killing monsters was left to the RL agent to decide in the previous approaches, and the game designer had little control over these decisions. Second, the previous approaches do not allow development in persona. If the procedural persona is a monster killer, the procedural persona will always be a monster killer. Bartle [7] highlights several development sequences for personas, and the previous approaches cannot realize these development sequences.

The goal-based persona is a procedural persona with a linked sequence of goals rather than a single utility function. A goal contains a utility function and a transition action to the next goal. If there is a single goal in the sequence, there is no need to define the transition. Hence, a goal-based persona with a single goal is equivalent to a procedural persona. The transition connects the goals, and the transition occurs depending on criteria. Game designers determine the criteria, and criteria hold conditions related to the game. For example, a criterion can be killing 50% of the monsters or exploring 90% of the game or having health less than 20% or the combination of these conditions. When the goal-based persona fulfills all of the conditions of the criteria, the transition occurs and activates the next goal. When there are no more goals, the goal-based persona ceases. Additionally, the transition between the goals can be sudden or can be fuzzy.

In Fig. 1, we created an example level to demonstrate the goal-based personas. In this example, the *Avatar* situated at bottom right corner can execute the following actions *Pass*, *Attack*, *Left*, *Right*, *Up*, and *Down*. The direction of the *Avatar* is shown by a pink triangle. If the direction of the *Avatar* and the action align, the *Avatar* moves one space in that direction, else the *Avatar* changes direction. When *Avatar* executes *Attack*, the *Avatar* slashes towards its direction. The *Avatar* can slay Monsters by Attacking them. The monsters

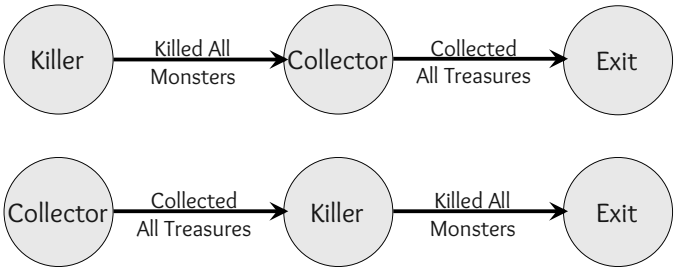


Fig. 2: Developing Persona

move randomly and kills the *Avatar* if they collide with the *Avatar*. There are also Treasure chests that *Avatar* can pick up by simply moving over them. Lastly, when the *Avatar* exits through the *Door*, the game terminates successfully.

TABLE I: Utility weights for the goals

	Goal Names		
Game Event	Killer	Collector	Exit
Death	-1.0	-1.0	-1.0
Exit Door			1.0
Monster Killed	1.0		
Treasure Collected		1.0	

A game designer may playtest a Monster Killer persona in the game shown in Fig. 1 and generates the following two personas. First one kills the *Monsters* and then collects the *Treasure* as trophy. Second one collects the *Treasure* hoping to gain an advantage against the *Monsters* and then kills the *Monsters*. In order to realize the aforementioned personas, the game designer designs two goal-based personas, as seen in Fig. 2. Next, she designs the utility functions of these goals, as seen in Table I. In order to realize the goal-based personas as playtesters in a game, the game designer can employ any RL agent. When the agent finishes training, the game designer can use the agent for playtesting. The importance of the goal-based persona is that these playtests cannot be achieved by a single utility function.

B. Alternative Path Finder

The actions of an RL agent are motivated based on the feedback received from an environment. As the agent trains in an environment, the feedback will shape the agent's policy. When the training is over, the agent will behave according to the learned policy. Additionally, if we exercise the same agent in the same environment multiple times, the learned policies will be similar. At the end of each training, we can test the trained agent in the same environment to obtain trajectories. These trajectories will be similar as the learned policies were similar. On the other hand, the game designer might be interested in seeing different playstyles.

In order to diversify the learned policies, one has to change the feedback mechanism of the environment. Procedural personas [4] [5] accomplish this by rewiring the feedback mechanism by a utility function. An agent representing a persona will learn a different policy than another agent that represents a different persona. However, when the game designer wants to see different playstyles within the same persona, the procedural persona approach falls short. For example, the game designer may want to see how different players complete a game with multiple exits. To model these players, she trains an agent that mimics the Exit persona, and she analyzes the trajectory from this agent's execution. Nevertheless, the resultant trajectory of this persona will be the path to the nearest Exit. The other Exits in the game will be neglected, and the game designer will only have playtest data that corresponds to one possible ends of the game. A preliminary solution to this problem is masking the feedback from some of the Exits. Thus the agent will generate a playtest towards a particular Exit. However, this solution requires additional tinkering, and there might be additional solutions towards the same Exit. Another solution is that the game designer would apply randomness to the agent's actions or add random noise to the input to diversify the trajectories. However, randomness does not guarantee that the agent will generate different playtests. Therefore, this solution also does not give complete control to the game designer.

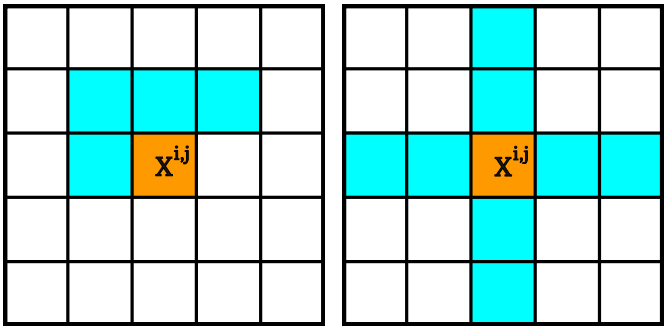
On the other hand, with human playtesters, the game designer could have asked a playtester to play differently. The playtester already knows which paths or particular states she has visited before, so she uses this past knowledge to play the game differently. Therefore, the source of this problem is that the current agent does not know what the previous agents did in the prior runs. Every playtester which an RL agent represents generates a playtest anew. In order to solve this problem, we propose Alternative Path Finder.

1) Measuring Similarity: A game can be formulated using a Markov Decision Process (MDP). MDP formulates the interaction between an actor and the environment [27]. Markov Decision Process is a tuple (S, A, T, R) where S is the set of states, A is the set of actions, $T : S \times A \times S \rightarrow [0, 1]$ is the state transition probability matrix, and $R : S \times A \rightarrow \mathbb{Q}$ is the *reward function*.

Suppose a human player or an agent played a game, and we obtain the trajectory $\tau = \{s_0, a_0, s_1, a_1, ..., s_n\}$ where s corresponds to a state, a corresponds to an action, and the subscripts denote the state or action at time t. We want to train an agent that knows τ, and we want this agent to generate a trajectory different than τ. Therefore, we need to calculate a measure to represent the similarity of these two trajectories. We propose two different methods to calculate the similarity. First method is to calculate the visitation probability $p(s|\tau)$. If $s \in \tau$, then the probability should be high and if $s \notin \tau$, then the probability should be low. Second method is calculating the prediction error of a dynamics model $q((s_t, a_t, s_{t+1})|\tau)$. If the transition s_t, a_t, s_{t+1} exists in τ, then the prediction error should be low, and if this transition does not exist in τ, then the error should be high.

In the rest of this paper, we swap the state s with observation o, which the RL agent sees. In most of the frameworks such as GVG-AI [15] and VizDoom [16], the observation seen by the RL agent corresponds to a frame f.

2) From Visitation Probability to Intrinsic Feedback: Bellemare et al. [12] used Context Tree Switching (CTS) [28] to intrinsically motivate an RL agent for exploration. CTS uses a filter to evaluate the recoding probability of a pixel. The filter



(a) L-shaped Filter (b) +-shaped Filter

Fig. 3: Filters mask the pixels around the orange pixel, the data from white pixels are blocked, and the data from the cyan pixels are supplied. Finally, CTS uses the information gathered from cyan pixels to predict the recoding probability of the orange pixel.

used by the authors and in our experiments is shown in Fig. 3(a) and Fig. 3(b), respectively. The filter gathers information around a pixel and CTS uses this information to predict this pixel. When this operation is done for every pixel of an image, the recoding probability of an image is calculated.

Consequently, CTS does not directly calculate the visitation probability of a frame but a frame's recoding probability, fortunately these two probabilities are directly proportional.

In order to use the recoding probability to differentiate between the novel frames from similar frames, we need a boundary probability value. We refer to this probability as p_{min} (see Eq. 1). First, we train a CTS model using all of the frames in trajectories. Then we calculate the recoding probability of every frame in this trajectory. Next, we set the p_{min} equal to the minimum of all these recoding probabilities. As CTS is a learning-positive model, every frame from these trained trajectories will have a higher recording probability than p_{min}.

$$p_{min} = \min(p(f_0|\text{CTS}), p(f_1|\text{CTS}), ..., p(f_n|\text{CTS})) \quad s.t.\ f_{0..n} \in \tau_0, ..., \tau_n \tag{1}$$

When an agent or a human player plays the game, the actor will receive a new frame f_{new}. First, we calculate its recoding probability p_{new}=$p(f_{new}|\text{CTS})$. If p_{new} is smaller than p_{min}, this indicates that this frame provides new information and if p_{new} is greater than p_{min}, this indicates that this frame does not provides new information. Next, the magnitude of the information depends on how close p_{new} is to p_{min}. We use this difference to calculate the amount of reward or penalty.

$$\begin{aligned} p_{new} > p_{min} &: feedback = \frac{\beta}{1 + \log \frac{p_{new}}{p_{min}}} - \beta \\ p_{new} \leq p_{min} &: feedback = \beta - \frac{\beta}{1 + \log \frac{p_{min}}{p_{new}}} \end{aligned} \tag{2}$$

We use Eq. 2 to calculate the additional reward signal. This formula yields maximum β reward when $p_{new} \to 0$ and minimum $-\beta$ when $p_{new} \to 1$. This additional reward signal provides a negative feedback for visiting similar states and positive feedback for visiting novel states. We refer to the APF method that uses CTS internally as APFCTS.

3) From Predicting Dynamics to Intrinsic Feedback: Pathak et al. [14] used the Intrinsic Curiosity Module (ICM) to intrinsically motivate an RL agent for exploration. ICM is a Neural Network (NN) architecture that learns to predict the environment dynamics and uses the prediction error as the intrinsic motivation. ICM has two NNs called as forward model and inverse model. The forward model predicts the next state features $\phi(s_{t+1})$ using the current state features $\phi(s_t)$ and current action a_t. The inverse model predicts the current action a_t using the current state features $\phi(s_t)$ and the next state features $\hat{\phi}(s_{t+1})$. ICM uses Convolutional Neural Network (CNN) to encode the states into state features, $\phi(s_t)$ = CNN(s_{t+1}). The prediction error is the difference between the predicted next state features $\hat{\phi}(s_{t+1})$ and extracted next state features $\phi(s_{t+1})$. Therefore, if the agent has seen the transition $\phi(s_t), a_t, \phi(s_{t+1})$, the prediciton error will be low, and if not, the prediction error will be high.

In order to use the prediction error to differentiate between the novel frames from similar frames, we need a boundary value. We refer to this value as q_{mean} (see Eq. 3). First, we initialize an empty ICM architecture. Next, we use transfer learning to set the weights of CNN encoders, and then we freeze the weights of CNN. The source can be the CNN layers of the RL agent, or if the agent also used ICM, we can use ICM's CNN layers. Afterward, we use the previous trajectories to train the forward and inverse models of ICM. At the end of the training, we have an ICM model that has a better prediction towards the transitions that exist in the given trajectories and a worse prediction towards the transitions that do not exist. Lastly, we replay the previous trajectories, gather all of the prediction errors, and calculate the *mean* of all the prediction errors. We do not calculate the *max* of all the prediction errors as the ICM may not improve the predictions for every transition or make prediction errors. Therefore, *max* would be a poor choice for a boundary value.

$$q_{mean} = \text{mean}(\text{ICM}(f_0, a_0, f_1), ..., \text{ICM}(f_{n-1}, a_{n-1}, f_n)) \quad s.t.\ f_{0..n} \in \tau_0, ..., \tau_n \quad s.t.\ a_{0..n-1} \in \tau_0, ..., \tau_n \tag{3}$$

When an agent or a human player plays the game, the actor executes action a on frame f. As a result, the actor sees a new frame f_{new}. First, we calculate the prediction error of this transition, q_{new}=ICM(f, a, f_{new}). If q_{new} is greater than q_{mean}, this indicates that this transition is less likely to exist in the previous trajectories. If q_{new} is less than q_{mean}, this indicates that this transition is likely to exist in the previous trajectories.

$$\begin{aligned} q_{new} > q_{mean} &: feedback = \beta - \frac{\beta}{1 + \log \frac{q_{new}}{q_{mean}}} \\ q_{new} \leq p_{min} &: feedback = \frac{\beta}{1 + \log \frac{q_{mean}}{q_{new}}} - \beta \end{aligned} \tag{4}$$

We use Eq. 4 to calculate the additional reward signal. This formula yields maximum β reward when $q_{new} \to 0$ and minimum $-\beta$ when $q_{new} \to \infty$. We use this additional feedback signal to reward the novel transitions and to penalize similar transitions. We refer to the APF method that uses ICM internally as APFICM.

4) APF Architecture: We augment the traditional Agent and Environment interaction by adding a new box. This augmented architecture is shown in Fig. 4. The APF corresponds to an APFCTS or an APFICM. Before an agent starts to interact with the environment, we train the APF with the previous trajectories as described in Section III-B2 or Section III-B3. At this point, we have an APF module that discerns the states or transitions. When a new state and a new reward is observed from the environment, these observations first enter the APF. APF modulates the reward signal by adding a penalty or reward by using the Eq. 2 or Eq. 4.

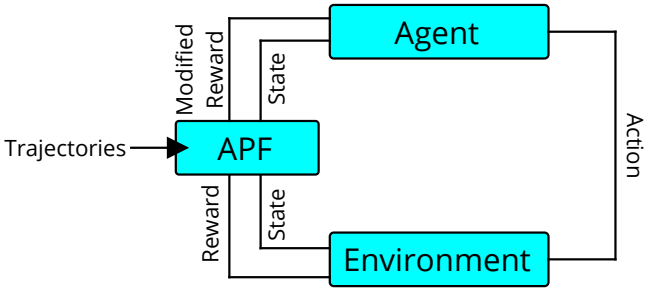


Fig. 4: Alternative Path Finding Architecture.

The one drawback of this approach is that the feedback is unbounded. Since the feedback is infinite, the agent may visit a novel state repeatedly to get a positive reward. This problem may lead an agent to get stuck in a state [25]. The second drawback is that some portion of the game may be strict, offering no alternative paths such as Super Mario Bros. [14]. Consequently, APF will penalize this portion of the game, naively thinking there may be alternative paths.

We propose a solution for each of these drawbacks. For the first drawback, we propose to put a cap on the total reward and penalty that APF provides. This solution limits the infinite feedback, and this process operates as follows: if a state is distinct, APF clamps the reward by the positive cap pos_{cap}. Then, APF yields this clamped reward and updates the positive cap by subtracting the clamped. Once the positive cap is exhausted, the additional reward that APF provides becomes zero. We also apply the same principles for the penalty by providing a negative cap, neg_{cap}. Hence, this solution prevents an agent stuck in a novel state. For the second drawback, we propose to cut these portions from the collected trajectories. Consequently, APF will not penalize the agent, as APF will be blind for this portion of the path.

We introduced two different APF approaches as each has its advantages and disadvantages. The advantage of APFCTS is that the CTS model can be trained from a trajectory that consists of a few frames. However, APFICM is more data-intensive compared to APFCTS. Furthermore, APFICM requires a previously trained agent for transfer learning, which is not required for APFCTS. Nevertheless, as APFCTS operates directly on pixels, a slight noise in a frame would decrease the visitation probability.

IV. Experiments

In this paper, we used two different environments to test our proposals, GVG-AI [15] and VizDoom [16]. We describe the environments and the experimental setup in this section.

The first testbed game is created using the GVG-AI framework, shown in Fig. 5. The game has a 14×20 grid-size, and consists of an *Avatar*, *Exits*, *Monsters*, *Treasures*, and *Walls*. The human player or an agent controls the *Avatar*. The game lasts until the *Avatar* goes to one of the *Exits*, or gets killed by a *Monster*, or until 200 timesteps. The action space consists of six actions *No-Op*, *Attack*, *Left*, *Right*, *Up*, and *Down*. GVG-AI framework is extended to run a game with more than one *Door*. The actor receives a distinct feedback for the following interactions killing a *Monster*, getting killed by a *Monster*, collecting a *Treasure*, and colliding with a *Door*.

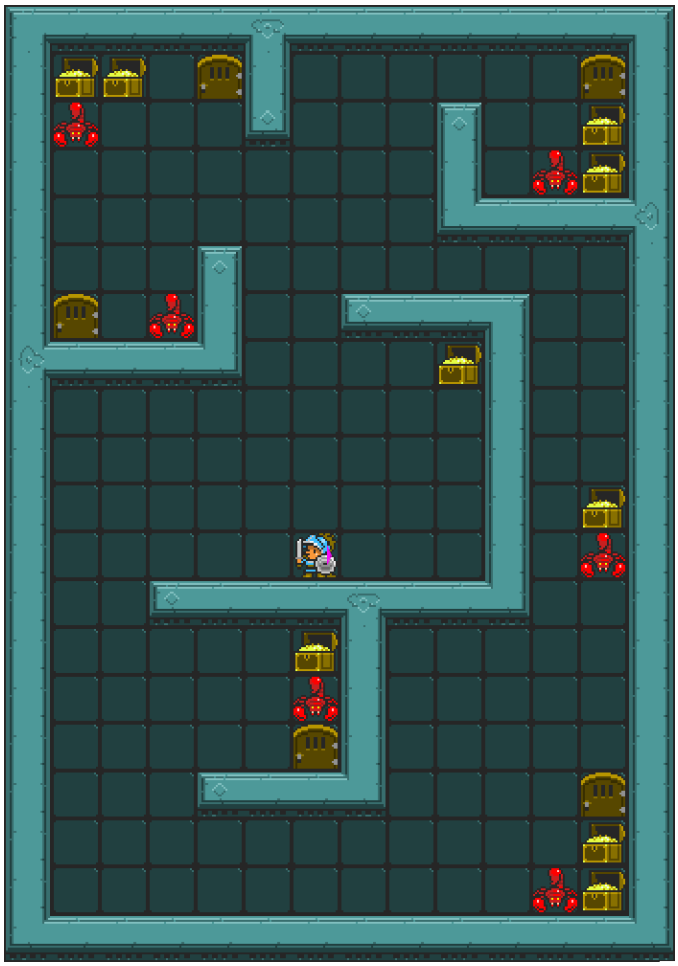


Fig. 5: Map of the first testbed game.

The second testbed game is a Doom level, shown in Fig. 7. The game has a 1600×832 grid size, and consists of an *Avatar*, *Exit*, *Monsters*, *Treasures*, and *Walls*. The human player or an agent controls the *Avatar*. The game lasts until the *Avatar* goes to the *Door*, or gets killed by a *Monster*, or until 2000

timesteps. The action space consists of seven actions *Attack*, *Move Left*, *Move Right*, *Move Up*, *Move Down*, *Turn Left*, and *Turn Right*. The actor receives a distinct feedback for the following interactions killing a *Monster*, getting killed by a *Monster*, collecting a *Treasure*, and colliding with the *Door*. Additionally, the actor receives a constant negative feedback of 0.001 for every step taken.

Fig. 6: Doom in-game snapshot.

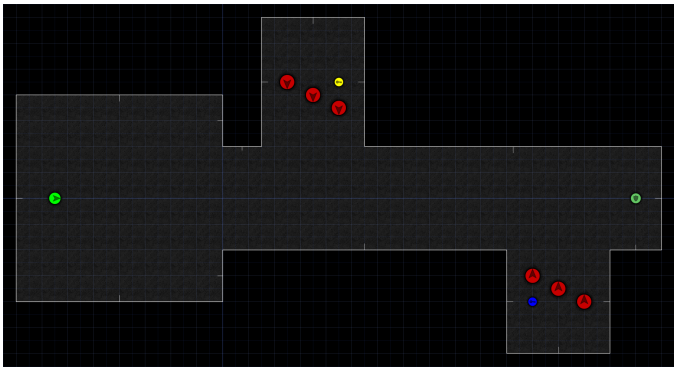


Fig. 7: Map of second testbed game.

The third testbed game is another Doom level, shown in Fig. 8. The game has a 1664×704 grid size, and consists of an *Avatar*, an *Exit*, and *Walls*. The human player or an agent controls the *Avatar*. The game lasts until the *Avatar* goes to the *Door*, or until 2000 timesteps. The action space consists of seven actions *Attack*, *Move Left*, *Move Right*, *Move Up*, *Move Down*, *Turn Left*, and *Turn Right*. The actor receives a feedback if the actor interacts with the *Door*. Additionally, the actor receives a constant negative feedback of 0.001 for every step taken.

We experiment with the procedural and goal-based personas in the first and second testbed games. We test the APF in the first and third testbed games. We used the same random seed during the APF experiment to properly test the APF method. We use PPO [10] agent in all of the experiments. For the PPO+CTS, PPO+ICM, PPO+APFCTS, and PPO+ICM+APFICM, we change the base PPO implementation slightly. The base PPO implementation is from the Stable-Baselines project [29]. We also tested with other

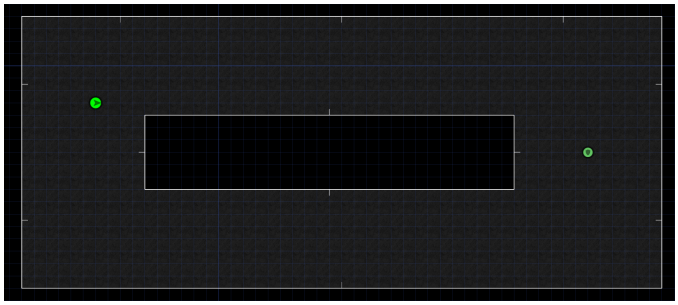


Fig. 8: Map of third testbed game.

RL agents during the initial experiments, and we found that PPO requires less hyperparameter tuning, so we used PPO in all of our experiments. The hyperparameters of PPO agents are presented in Table II, and the hyperparameters of APF techniques are shown in Table III.

GVG-AI environment sends an observation with shape $160 \times 112 \times 4$, we downscale this observation to 80×56 and then convert the observation into grayscale. Afterward, we stack the most recent four observations, and lastly feed the stacked observations to the agent. For CTS used in PPO+CTS and APFCTS, we process the observation into we 42×42, 3-bit grayscale image, and calculate the recoding probability of this observation. Doom environment sends the observation with shape $160 \times 120 \times 1$, we resize this observation to $84 \times 84 \times 1$, and we feed the agent and the APFICM this resized observation.

TABLE II: Hyperparameters of PPO Agents

	Agents		
Hyperparameters	PPO	PPO+CTS	PPO+ICM
Policy	CNN	CNN	CNNLstm
Horizon	256	256	64
Num. Minibatch	8	8	8
GAE (λ)	0.95	0.95	0.99
Discount (γ)	0.99	0.99	0.999
Learning Rate (α)	5×10^{-4}	5×10^{-4}	5×10^{-4}
Num. Epochs	3	3	4
Entropy Coeff.	0.01	0.01	0.001
VF Coeff.	0.5	0.5	0.5
Clipping Param.	0.2	0.2	0.1
Max Grad. Norm.	0.5	0.5	0.5
Num. of Actors	16	16	32
CTS Beta (β)	-	0.05	-
CTS Filter	-	L-shaped	-
ICM State Features	-	-	256
ICM Beta (β)	-	-	0.2

TABLE III: Hyperparameters of APF Techniques

Hyperparameters	APFCTS	APFICM
pos_{cap}	0.4	0.1
neg_{cap}	-0.4	-0.4
APF Beta (β)	0.01	0.01

We created four different procedural personas and five different goal-based personas. The four procedural personas are

Exit, Monster Killer, Treasure Collector, and Completionist. The utility weights of these procedural personas are given in Table IV. The five goal-based personas are Developing Monster Killer, Developing Treasure Collector, Developing Raider, Developing Completionist, and Developing Casual Completionist. The development sequences of these personas are presented in Table VI, the utility function of the goals are given in Table V, and the criteria of these goals are shown in Table VII.

TABLE IV: Utility weights for procedural personas. Exit (E), Monster Killer (MK), Treasure Collecter (TC), and Completionist (C).

	Personas			
Game Event	(E)	(MK)	(TC)	(C)
Reaching an Exit	1	0.5	0.5	
Killing a Monster		1		1
Collecting a Treasure			1	1
Dying	-1	-1	-1	-1

TABLE V: Utility weights for the goals. Killer (K), Collecter (Col), Exit (E), and Completionist (Com).

	Goal Names			
Game Event	(K)	(Col)	(E)	(Com)
Death	-1	-1	-1	-1
Exit Door			1	
Monster Killed	1			1
Treasure Collected		1		1

TABLE VI: Sequences for the developing personas.

Hyperparameters	**Development Sequence**
Dev. Killer	Killer -> Exit
Dev. Collector	Collector -> Exit
Dev. Raider	Killer -> Collector -> Exit
Dev. Completionist	Completionist -> Exit
Dev. Casual Completionist	Casual Completionist -> Exit

TABLE VII: Criteria of the goals. Killer (K), Collecter (Col), Completionist (Com), and Casual Completionist (Cas. Com.).

	Goal Names			
Criterion	(K)	(Col)	(Com)	(Cas. Com.)
Monsters Killed	50%		100%	
Treasure Collected		50%	100%	
Remaining Health				50%

V. Results

In this study, we asked the following research questions.

- How does a goal-based persona perform compared to a procedural persona?
- Which additional paths can be discovered with APF?

A. Experiment I: Procedural vs Goal-based personas:

Table VIII presents the interactions done by seven different personas. The Exit persona directly goes to the *Door*, which is four spaces below the *Avatar*. The other three procedural personas also go to the same *Door*, but also collecting the *Treasure* and killing the *Monster* on the way. The developing killer persona defeats all of the *Monsters* on the upper half of the level. The developing collector persona collects four of the *Treasures* on the upper half of the level. The developing raider is a combination of developing killer and developing collector, consequently kills the *Monsters* and then collects the *Treasures* in the upper half of the level. We see all procedural personas interact with a small region of the level, whereas the developing persona interacts with a broader region. Therefore, we conducted the same experiment for procedural personas with PPO + CTS RL agent. Table VIII displays the interactions performed by procedural personas when the agent explores the environment. We see that the interactions performed by procedural personas fit better to each persona's decision model.

TABLE VIII: Interactions of Personas performed by the PPO RL agent in Experiment I.

	Game Event		
Personas	Monsters Killed	Treasures Collected	Door
Exit			1
Monster Killer	1	1	1
Treasure Collector	1	1	1
Completionist	1	1	1
Dev. Killer	3	0	1
Dev. Collector	1	4	1
Dev. Raider	3	4	1

TABLE IX: Interactions of Personas performed by the PPO + CTS RL agent in Experiment I.

	Game Event		
Personas	Monsters Killed	Treasures Collected	Door
Monster Killer	2	0	1
Treasure Collector	0	3	1
Completionist	2	3	1

B. Experiment II: Alternative paths found in GVG-AI:

We used the path found by the Exit persona in Experiment I to train APFCTS, and then we trained the PPO + CTS + APFCTS agent in GVG-AI first testbed game. For each path obtained from the PPO + CTS + APFCTS agent, we repeated the experiment. First, an APFCTS is trained using one of the obtained paths, and then we used this trained APFCTS to train a PPO + CTS + APFCTS agent. The paths identified at the end of the process are shown in Fig. 9. Table X shows the total of discounted rewards, and the bold values demonstrate the found paths when we train the CTS with the trained path. The first row demonstrates the total discounted reward without using APFCTS, and we see paths one and five leading the other paths. When we use APFCTS, we see that the reward

of playing the same path decreases by at least 0.1, and the reward of distinct paths increases by at least 0.1.

TABLE X: Total Discounted Reward without APFCTS and with APFCTS. The first row shows the total discounted reward without APFCTS. For the rows with a number, we train the APFCTS and calculate the discounted reward by using CTS. The bold values demonstrate the found paths when we execute the PPO + CTS + APFCTS agent by training APFCTS with the trained path.

	Tested Paths					
Trained Path	1	2	3	4	5	6
-	**0.86**	0.78	0.84	0.84	**0.86**	0.76
1	0.76	0.77	**0.98**	**0.98**	**0.98**	**0.98**
2	0.82	0.61	**0.98**	**0.99**	**0.98**	**0.98**
3	**0.98**	**0.98**	0.74	0.86	0.82	0.88
4	**0.99**	**0.98**	0.86	0.72	0.86	0.86
5	**0.98**	**0.98**	0.81	0.85	0.76	0.87
6	**0.99**	**0.98**	0.87	0.87	0.88	0.60

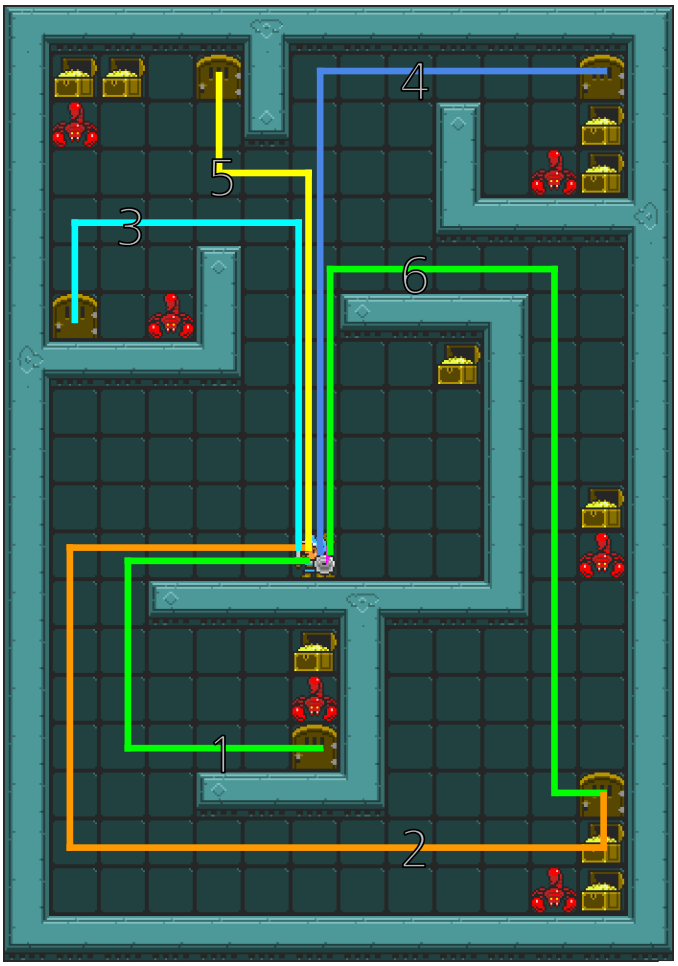


Fig. 9: Paths found by Exit persona with PPO and with PPO + CTS + APFCTS.

C. Experiment III: Personas in Doom:

We experimented with five different personas in second testbed game, a Doom level (see Fig. 7). The interaction results for the personas are given in Table XI. In Doom, Completionist, Developing Completionist, and Developing Casual Completionist make the same interactions. The Exit persona directly goes to the *Door*, and Monster Killer finishes the level after killing the *Monsters*.

TABLE XI: Interactions of Personas in Experiment III.

	Game Event		
Personas	Monsters Killed	Treasures Collected	Door
Exit			1
Monster Killer	6		1
Completionist	6	2	1
Dev. Completionist	6	2	1
Dev. Cas. Compl.	6	2	1

D. Experiment IV: Alternative paths found in Doom:

We trained an exit persona in the third testbed game using PPO + ICM agent. The first path shown in Fig. 10 is the trajectory taken by the exit persona. We trained an APFICM using this first path, and then we trained a new exit persona using PPO + ICM + APFICM agent. The new exit persona played the second path. The total discounted reward obtained by these two exit personas is shown in Table XII. As the first path consists of 52 steps, whereas the second path consists of 77 steps, the total reward of the first path is higher than the second. However, applying APFICM, we increase the total reward obtained from the second path and decrease the total reward obtained from the first path.

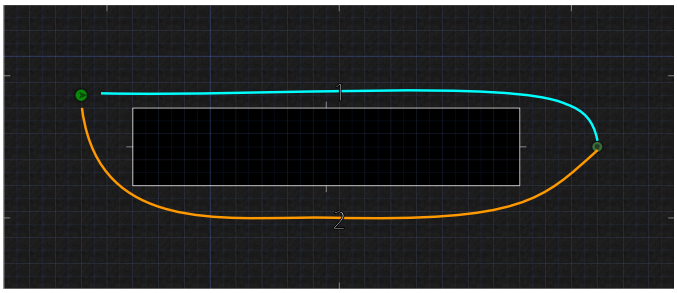


Fig. 10: Paths found by Exit persona with PPO and with PPO + ICM + APFICM.

TABLE XII: Total Discounted Reward without APFICM and with APFICM. First row shows the total discounted reward without APFICM. For the rows with a number, we train the APFICM and calculate the discounted reward by using ICM. The bold values demonstrate the found paths when we execute the PPO + ICM + APFICM agent by training APFICM with the trained path.

	Tested Paths	
Trained Path	1	2
-	**0.79 ± 0.015**	0.67 ± 0.013
1	0.51 ± 0.015	**0.77 ± 0.016**

VI. Discussion

In this paper, we presented an advancement for procedural persona, goal-based persona and introduced a method to let RL agents discover different paths, APF. We experimented with these methods in GVG-AI and Doom environments.

Procedural personas and developing personas are two methods that a game designer may use to automate the playtesting process. One drawback of the procedural personas originates from the utility function. A utility function realizes the decision model of a persona. For example, a treasure collector receives positive feedback from finishing the level and collecting a treasure. However, if the agent starts close to the *Door*, the agent may not interact with any of the *Treasures*. Conversely, if the *Door* stands after all of the *Treasures*, the agent is likely to interact with most of the *Treasures*. We saw this problem in Experiment I. Without any exploration technique, the procedural personas monster killer, treasure collector, and completionist executed the same set of actions. When we integrated CTS exploration, these procedural personas executed a different set of actions. These new sets of actions were more fitting to their decision model. This problem is also seen in the MCTS agent playtesting the MiniDungeons 2 game [4]. However, we were unaffected by this problem while using developing personas. The problem is that the utility function embodies multiple playstyles within the same persona. Therefore, depending on the level composition and RL agent's hyperparameters, the procedural persona represents one of those playstyles. A game designer may use the developing persona to choose which playstyle that she wants to playtest carefully.

A game designer may use developing personas to represent playstyles that involve alteration. For example, in Experiment I, the developing raider killed the *Monsters* and then collected the *Treasures*. The developing raider starts the game as a monster killer and becomes a different persona after fulfilling a criterion. Afterward, the developing raider becomes a treasure collector. These development sequences were mentioned by Bartle [17], but development sequences were impractical while using a single utility function. Moreover, the game designer may want to cluster collected human playtraces according to their persona's [6]. If a human playtester has changed her behavior during the playtest, this player would be incorrectly classified without developing persona.

In addition to the GVG-AI environment, we conducted experiments on the Doom environment. To the best of our knowledge, this paper is the first work to playtest personas in a 3D environment. In 2D environments, the researchers [4][5] employed MCTS RL agent to realize personas. MCTS would be an ineffective choice for 3D environments, and MCTS would underrepresent the persona. We used the PPO agent in all our experiments, as PPO is a competent agent used by OpenAI [30]. In Experiment III, we see that the decision models of personas are expressed correctly. The exit persona does not interact with any game events other than the *Door*. All other personas besides exit persona kill all of the *Monsters*, and completionist personas also collect all of the *Treasures*. We observe that the developing casual completionist persona is indifferent to other completionist personas. This indifference indicates that the game may not be challenging enough for a casual player.

In Experiment I, we prepared a game that consists of five *Doors*. We found that the exit persona taking either the first or the fifth path (see Fig. 9). The lengths of the first and the fifth paths are the same and shorter than every possible path that ends with a *Door*. Consequently, an RL agent representing an exit persona is likely to select either the first or the fifth path. We proposed APF to let RL agents discover these additional paths shown in Fig. 9. APF learns the previously played path and adjusts the reward mechanism of the environment to let RL agents discover distinct paths. When we examine the paths shown in Fig. 9 and their respective scores in Table X, we see that APF penalizes the previously played path, and rewards the distinct paths. Furthermore, we also see the same result in Experiment IV, when we study the paths shown in Fig. 10 and their respective scores in Table XII. Therefore, a game designer may use APF to identify different paths and improve her game by using this information.

On the other hand, an alternative path is a subjective concept. Every human playtester may think of another possible way of representing the exit persona. In Table X, we see that when we train the APFCTS with the first or the second path, the score of both of these paths decreases. Next, when we train the APFCTS with the third, fifth, or sixth path, the score of the fourth path increases slightly. According to APFCTS, the first and second paths are more similar than the fourth and fifth paths. The subjectivity of APFCTS and APFICM is based on the recoding probability of a frame and the dynamics prediction error, respectively. However, this subjectivity does not impede APFCTS or APFICM from finding distinct paths. On the other hand, as CTS operates on pixels, we found that the recoding probability of some frames in the Doom environment is calculated as 0. Furthermore, we observed that for our experimentation setup the plus-shaped filter in Fig. 3(b) yielded better results than the original CTS filter in Fig. 3(a). Lastly, though we promoted APF to find distinct paths, APF can benefit game tester agents [8] since increasing the coverage of tests is crucial in testing, and APF increases coverage by finding distinct paths.

Limitations & Challenges: The performance of developing and procedural persona is dependent on the RL algorithms. If the RL algorithm cannot play a game, the game designer could not use automated playtesters. Furthermore, our APF proposals are based on exploration algorithms. The performance of APF in an environment is linked to how well the exploration algorithm would perform in this environment.

VII. Conclusion

This paper focused on the problem of providing additional tools to game designers for playtesting. In this regard, we proposed developing persona, a direct successor to procedural personas. Furthermore, we presented a novel method to help RL agents to discover unique trajectories, APF. We introduced two APF approaches, APFCTS and APFICM.

Our results show that developing personas are a successor of procedural personas. A game designer can embody vari-

ous personalities in developing personas to generate unique playtests. Furthermore, we show that automated playtesting can be extended to 3D environments by using state-of-the-art RL algorithms.

We proposed APF to discover unique paths in an environment. We based APF on exploration research techniques and proposed two methodologies to implement APF, APFCTS, and APFICM. In our experiments in GVG-AI and Doom environments, we found that APF ensures that the same path is not generated again.

In the future, we would like to experiment with different personas using APF. Next, APFICM can be improved by substituting the linear layer with an LSTM layer. This substitution will provide path information rather than state transition information. Lastly, we would like to experiment with other 3D environments such as Minecraft [31].

REFERENCES

[1] E. J. Powley, S. Colton, S. Gaudl, R. Saunders, and M. J. Nelson, "Semi-automated level design via auto-playtesting for handheld casual game creation," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[2] S. Gudmundsson, P. Eisen, E. Poromaa, A. Nodet, S. Purmonen, B. Kozakowski, R. Meurling, and L. Cao, "Human-like playtesting with deep learning," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 08 2018, pp. 1–8.

[3] S. Roohi, A. Relas, J. Takatalo, H. Heiskanen, and P. Hämäläinen, *Predicting Game Difficulty and Churn Without Players*, ser. CHI PLAY '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 585–593.

[4] C. Holmgard, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas with evolved heuristics," *IEEE Transactions on Games*, pp. 1–1, 2018.

[5] L. Mugrai, F. Silva, C. Holmgård, and J. Togelius, "Automated playtesting of matching tile games," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–7.

[6] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Generative agents for player decision modeling in games," in *Proceedings of the 9th International Conference on the Foundations of Digital Games (FDG)*, 2014.

[7] R. Bartle, "Virtual worlds: Why people play," *Massively Multiplayer Game Development 2*, vol. 2, pp. 3–18, 01 2005.

[8] S. Ariyurek, A. Betin-Can, and E. Surer, "Automated video game testing using synthetic and humanlike agents," *IEEE Transactions on Games*, vol. 13, no. 1, pp. 50–67, 2021.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[11] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, March 2012.

[12] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 1471–1479.

[13] J. Fu, J. D. Co-Reyes, and S. Levine, "EX2: exploration with exemplar models for deep reinforcement learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 2577–2587.

[14] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 2778–2787.

[15] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, 2019.

[16] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A Doom-based AI research platform for visual reinforcement learning," in *IEEE Conference on Computational Intelligence and Games*. Santorini, Greece: IEEE, Sep 2016, pp. 341–348, the best paper award.

[17] R. A. Bartle, "Hearts, clubs, diamonds, spades: Players who suit MUDs," http://www.mud.co.uk/richard/hcds.htm, 2019.

[18] A. Tychsen and A. Canossa, "Defining personas in games using metrics," in *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, ser. Future Play '08. New York, NY, USA: ACM, 2008, pp. 73–80.

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

[20] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Monte-carlo tree search for persona based player modeling," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[21] F. de Mesentier Silva, S. Lee, J. Togelius, and A. Nealen, "Ai-based playtesting of contemporary board games," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*. ACM, 2017, p. 13.

[22] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 2721–2730.

[23] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "#exploration: A study of count-based exploration for deep reinforcement learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 2753–2762.

[24] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep exploration via bootstrapped DQN," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 4026–4034.

[25] Y. Burda, H. Edwards, A. J. Storkey, and O. Klimov, "Exploration by random network distillation," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[26] L. Lee, B. Eysenbach, E. Parisotto, E. P. Xing, S. Levine, and R. Salakhutdinov, "Efficient exploration via state marginal matching," *CoRR*, vol. abs/1906.05274, 2019.

[27] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[28] M. G. Bellemare, J. Veness, and E. Talvitie, "Skip context tree switching," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014, pp. 1458–1466.

[29] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[30] B. Baker, I. Kanitscheider, T. M. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[31] M. Johnson, K. Hofmann, T. Hutton, D. Bignell, and K. Hofmann, "The malmo platform for artificial intelligence experimentation," in *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI - Association for the Advancement of Artificial Intelligence, July 2016.