Playtesting: What is Beyond Personas

Sinan Ariyurek ©, Elif Surer ©, Member, IEEE, and Aysu Betin-Can ©

Abstract—Playtesting is an essential step in the game design process. Game designers use the feedback from playtests to refine their designs. Game designers may employ procedural personas to automate the playtesting process. In this article, we present two approaches to improve automated playtesting. First, we propose developing persona, which allows a persona to progress to different goals. In contrast, the procedural persona is fixed to a single goal. Second, a human playtester knows which paths she has tested before, and during the consequent tests, she may test different paths. However, reinforcement learning (RL) agents disregard these previous paths. We propose a novel methodology that we refer to as alternative path finder (APF). We train APF with previous paths and employ APF during the training of an RL agent. APF modulates the reward structure of the environment, while preserving the agent’s goal. When evaluated, the agent generates a different trajectory that achieves the same goal. We use the general video game artificial intelligence and VizDoom frameworks to test our proposed methodologies. We use proximal policy optimization RL agent during experiments. First, we compare the playtest data generated by developing and procedural persona. Our experiments show that developing persona provides better insight into the game and how different players would play. Second, we present the alternative paths found using APF and argue why traditional RL agents cannot learn those paths.

Index Terms—Automated playtesting, play persona, player modeling, reinforcement learning (RL).

I. INTRODUCTION

GAME designers envision how a game will work during a play through. As the game develops, it becomes increasingly difficult to predict how players will interact with the game. Playtesters help out this process by providing 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]–[3].

Additionally, the playtesting process may employ players with distinct playstyles. These players will respond to the game differently, and they will generate different 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. This utility function was used as the reward function of the Q-learning agents. However, this replacement makes the agents bound to the utility function. Since the utility function is tailored for a specific decision model, the behavior of these agents is constant throughout the game. Therefore, the procedural personas approach is not flexible enough to create personas with developing decision models. For example, a player may change her objectives, while playing the game. Consequently, the decision model of this player cannot be captured by a utility function.

Bartle [7] presents examples of these changes that a player can undergo, while playing a massively multiplayer online role-playing game. We believe that the change in the playstyle occurs after accomplishing a goal. For example, a player may start a game by opening the treasures to find a required item and then killing monsters. This player chooses her actions like a Treasure Collector until she finds the desired item and becomes a Monster Killer. We propose a sequence of goals to model the decision-making mechanism of this player. The sequence-based approach was previously used in automated video game testing agents [8] and was found more practical than nonsequence-based approaches.

The developing 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 criterion 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 an agent that behaves like an exit persona and exercises this agent multiple times. Then, the game designer analyzes the trajectories generated by this agent and sees that all the trajectories provide data for only one of the possible endings. On the other hand, a human playtester would have generated trajectories that cover various endings. Thus, the shortcoming of automated playtests
is not caused by the exit 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 search (MCTS) [11] disregard the previous trajectories. Consequently, even if we train an agent with any of these algorithms and then evaluate the agent, and repeat this process numerous times, all the generated trajectories would be similar most of the time. However, the trajectories may be different due to the following reasons 1) the random initialization of the neural network (DQN and PPO), 2) $\epsilon$-greedy policy of DQN, 3) stochasticity of MCTS, and 4) the game’s nondeterminism. The critical point is that even if the agent generates a distinct trajectory, this result is not by design but by random chance.

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 [12–14]. Compared to the traditional exploration methods such as $\epsilon$-greedy, where exploration is achieved through randomness, these modern algorithms entice exploration logically. These algorithms learn to distinguish the unvisited states from the visited states, consequently, these algorithms guide the agents to less-visited states. As a result, exploration methods vastly improved the agent’s score, such as Montezuma’s Revenge [12].

On the other hand, APF knows the previous trajectories and guides the agent to learn to play differently from previous ones. For this purpose, APF penalizes the agent when the agent visits a similar state and rewards the agent when the agent visits a different state, compared to the states in the previous trajectories. APF employs the state comparison algorithms used in exploration algorithms. These state comparison algorithms are the backbone of exploration research, and researchers tested these algorithms in multiple games. We show how we build the APF framework to generate new and unique playtests and how APF augments any RL agent.

In this article, we list the contributions as follows. Our first contribution is the developing persona. The developing persona is more flexible and capable 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 (APF). We present a generic APF framework that can augment every RL agent. We use the general video game artificial intelligence (GVG-AI) [15] and VizDoom [16] environments to demonstrate our proposed methodologies.

The remainder of this article is structured as follows: Section II describes the examples and methodologies of related research. We grouped the related research into four subsections: playtesting, personas in playtesting, automated playtesting, and exploration methods in reinforcement learning. The developing persona is based on the first three subsections. Next, APF is founded on the exploration methods in reinforcement learning. 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 article.

II. RELATED RESEARCH

A. Playtesting

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 (CNN) 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.

B. Personas in Playtesting

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 multiuser 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 identified by these metrics. The authors present the personas of the game Hitman Blood Assassin. The game identifies the following personas: 1) Mass Murderer, 2) Silent Assassin, 3) Mad Butcher, and 4) 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.

C. Automated Playtesting

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] extend 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 toward 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.

D. Exploration Methods in Reinforcement Learning

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 article, we address the shortcomings of the procedural persona with a multigoal oriented persona, the developing persona. Additionally, we recognize there may be alternative playtests that a persona may produce. We propose APF to discover those playtests.

In the following subsections, first, we introduce the developing persona. 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. Developing 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. The authors in [4] and [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. The authors in [4] and [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 article, we extend the procedural persona framework by introducing a multigoal persona.

We propose a multigoal persona to generate a more customizable playtester. We have two reasons that a multigoal 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, and the game designer had little control over these decisions [27]. Second, the previous approaches do not allow development in persona. Though procedural personas may realize the persona archetypes that Bartle [7] presented, procedural personas cannot realize the development sequences that Bartle also presents. For example, if the goal of the procedural persona is killing monsters, the procedural persona will always be a Monster Killer.

A multigoal 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 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 the 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. The developing persona maintains knowledge of interactions such as how many Monsters, Treasures have been killed or collected. Next to the interactions, the developing persona knows how much of its health is left. Developing persona uses this knowledge to check whether the current criteria are fulfilled. When all of the criteria of the current goal are fulfilled, the next goal becomes active. When there are no more goals, the training or the evaluation of the goal-based persona ends.

In this section, we have described the “sudden” transitions between goals, the previous goal becomes inactive, and the next goal becomes active immediately. However, this transition could also be “fuzzy.” The current goal and the next goal can be active simultaneously. A possible implementation of fuzzy transition may use the criteria fulfillment percentage. For example, when the criteria are completed at least 50%, the next goal could become active, while not deactivating the current goal. The persona would be rewarded from both of the utility functions. Whenever the persona fulfills the current goal completely, the next goal becomes the only active goal. Consequently, a fuzzy transition would create a smoother progression of playstyles.

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 toward its direction. The Avatar can slay Monsters by Attacking them. The monsters move randomly and kill 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.

A game designer may playtest a Monster Killer persona in the game shown in Fig. 1 and generate the following two developing 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 developing personas, as seen in Fig. 2. Next, she designs the utility functions of the goals, as seen in Table I. In order to realize these 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 developing personas is that developing personas introduces a framework to formalize how players change their goals over the course of playing a game.

### B. Alternative Path Finder

The actions of an RL agent are motivated based on the feedback received from an environment. As the agent is trained 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 train the same agent in the same environment multiple times, the learned policies will be similar. At the end of each training, we can evaluate 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 also falls short. For example, the game designer may want to see how different players complete a game with multiple endings. 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 closest ending. The other endings in the game will be neglected, and the game designer will only have playtest data that corresponds to one possible
end of the game. A preliminary solution to this problem is
masking the feedback from some of the endings. Thus, the agent
will generate a playtest toward a particular ending. However,
this solution requires additional tinkering, and there might be
additional playtests toward the same ending. Another subpar
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 solu-
tion 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 differ-
ently. 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 APF.

1) Measuring Similarity: A game can be formulated using a
Markov Decision process (MDP). MDP formulates the interac-
tion between an actor and the environment [28].

Suppose a human player or an agent played a game, and
we obtain the trajectory \( \tau = \{s_0, a_0, s_1, a_1, \ldots, s_n\} \) 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 \( \tau \), and we want this agent to generate a
trajectory different than \( \tau \). 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 recoding probability of a state \( s \),
\( p(s | \tau) \). 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 \( \tau \), then the prediction error
should be low, and if this transition does not exist in \( \tau \), then
the error should be high.

In the rest of this article, 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 \( o \) seen by the
RL agent corresponds to a frame \( f \).

2) From Recoding Probability to Intrinsic Feedback: Belle-
mare et al. [12] used context tree switching (CTS) [29] to
intrinsically motivate an RL agent for exploration. CTS uses
a filter to evaluate the recoding probability of a pixel. The filter
used by the authors and in our experiments is shown in Fig. 3(a)
and (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.

In order to use the recoding probability to differentiate be-
tween the novel frames from similar frames, we need a boundary
probability value. We refer to this probability as \( p_{\text{min}} \) [see (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 \( p_{\text{min}} \) 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 recoding probability than \( p_{\text{min}} \)

\[
p_{\text{min}} = \min\{p(f_0|\text{CTS}), p(f_1|\text{CTS}), \ldots, p(f_n|\text{CTS})\}
\]

\[ \text{s.t. } f_{0..n} \in \tau_0, \ldots, \tau_n. \]  \hspace{1cm} (1)

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

\[
p_{\text{new}} > p_{\text{min}} : \text{feedback} = \frac{\beta}{1 + \log \frac{p_{\text{new}}}{p_{\text{min}}}} - \beta
\]

\[
p_{\text{new}} \leq p_{\text{min}} : \text{feedback} = \beta - \frac{\beta}{1 + \log \frac{p_{\text{new}}}{p_{\text{min}}}}. \hspace{1cm} (2)
\]

We use (2) to calculate the additional reward signal. This for-
mula yields maximum \( \beta \) reward when \( p_{\text{new}} \to 0 \) and minimum
\( -\beta \) when \( p_{\text{new}} \to 1 \). This additional reward signal provides
a negative feedback for visiting similar states and positive feed-
back 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 intrin-
sically motivate an RL agent for exploration. ICM is a neural
network (NN) architecture that learns to predict the environ-
ment 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 CNN to encode the states into state features, \( \phi(s_t) =
\text{CNN}(s_{t+1}) \). 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
\( \phi(s_t), a_t, \hat{\phi}(s_{t+1}) \), the prediction 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 \( q_{\text{mean}} \) [see (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 toward the transitions that exist in the given trajectories and a worse prediction toward 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

\[
q_{\text{mean}} = \text{mean}(\text{ICM}(f_0, a_0, f_1), \ldots, \text{ICM}(f_{n-1}, a_{n-1}, f_n))
\]

\[
s.t. f_0, f_n \in \tau_0, \ldots, \tau_n
\]

\[
s.t. a_0, a_{n-1} \in \tau_0, \ldots, \tau_n.
\]

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 \( f_{\text{new}} \). First, we calculate the prediction error of this transition, \( q_{\text{new}}=\text{ICM}(f, a, f_{\text{new}}) \). If \( q_{\text{new}} \) is greater than \( q_{\text{mean}} \), this indicates that this transition is less likely to exist in the previous trajectories. If \( q_{\text{new}} \) is less than \( q_{\text{mean}} \), this indicates that this transition is likely to exist in the previous trajectories

\[
q_{\text{new}} > q_{\text{mean}} : \text{feedback} = \beta - \frac{\beta}{1 + \log \frac{q_{\text{mean}}}{q_{\text{new}}}}
\]

\[
q_{\text{new}} \leq q_{\text{min}} : \text{feedback} = \beta - \frac{\beta}{1 + \log \frac{q_{\text{new}}}{q_{\text{mean}}}} - \beta. \tag{4}
\]

We use (4) to calculate the additional reward signal. This formula yields maximum \( \beta \) reward when \( q_{\text{new}} \to 0 \) and minimum \( -\beta \) when \( q_{\text{new}} \to \infty \). 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 either APFCTS or APFICM. Before we start training an agent, we first train the APF with the previous trajectories as described in Section III-B2 or III-B3. At this point, we have an APF module that discerns the states or transitions. Afterward, when a new state and a new reward are observed from the environment during the training, these observations first enter the APF. APF modulates the reward signal by adding a penalty or reward by using (2) or (4).

The one drawback of this approach is that the feedback is unbounded. Since the feedback is infinite, the agent may loop over novel states or get stuck in a novel state [25]. The agent may visit a novel state repeatedly to get a positive reward and forget the actual task in the environment. 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 \( pos \). Then, APF yields this clamped reward and updates the positive cap by subtracting the clamped reward. 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, \( neg \). This solution limits the agent looping over distinct states or getting stuck in a state like the noisy TV problem [25]. Furthermore, as the total reward and penalty are known beforehand, this solution also simplifies the design of the utility function for personas. 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 decoding probability.

Last but not least, though we presented the APF on top of exploration methods CTS and ICM, APF may also be formulated on other exploration methods such as exemplar models [13]. As APF depends on methods used in exploration, we need to draw a line between exploration and APF. The goal of exploration methods is to increase the agent’s knowledge about its environment during training. So that when we evaluate, this agent delivers top performance in this environment. The goal of APF is to help the agent to discover the different performances without changing the agent’s goal. Therefore, during training, APF modulates the reward structure so that the old performances are penalized, and different performances are rewarded.

IV. EXPERIMENTS

In this article, 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 $14 \times 20$ grid-size, and consists of an Avatar, Exits, static 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 2000 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 distinct feedback for the following interactions killing a Monster, getting killed by a Monster, collecting a Treasure, and colliding with a Door. Additionally, the actor receives constant negative feedback of 0.001 for every step taken.

The second testbed game is a Doom level, shown in Fig. 7. The game has a $1600 \times 832$ grid size, and consists of an Avatar, Exit, Monsters, Treasures, and Walls. The human player or an agent controls the Avatar (see Fig. 6). 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 feedback if the actor interacts with the Door. Additionally, the actor receives 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 [30]. We also tested the proposed persona with other 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 A1, and the hyperparameters of APF techniques are shown in Table A2. Lastly, as the first game is deterministic, we evaluated the trained agent once. On the
other hand, as the second and the third games are stochastic, we evaluated the trained agent 1000 times. Furthermore, we noticed that our training was more consistent whenever we used an exploration algorithm such as CTS or ICM. Consequently, we had to restart the training in the first game.

GVG-AI environment sends an observation with shape $160 \times 112 \times 4$, we downscale this observation to $80 \times 56$ 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 a $42 \times 42$, 3-bit grayscale image, and calculate the recoding probability of this observation. 

We created four different procedural personas and five different developing personas. The four procedural personas are Exit, Monster Killer, Treasure Collector, and Completionist. The utility weights of these procedural personas is given in Table II. We chose these procedural personas from [4], and we drew inspiration from these personas to make their developing persona counterparts. The five developing 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 IV, the utility function of the goals are given in Table III, and the criteria of these goals are shown in Table V.

V. RESULTS

In this study, we asked the following research questions.

1) How does a goal-based persona perform compared to a procedural persona? 

   a) Diversity of playtests generated by personas.

   b) Agreement between interactions performed and Persona’s decision model.

2) Which additional paths can be discovered with APF?

A. Experiment I: Procedural vs Goal-Based Personas

Table VI 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. Lastly, the Developing Completionist kills more Monsters and collects more Treasures than every other persona. However, Developing Completionist misses the Monster and the Treasure below the starting position. We see all procedural personas interact with a small region of the level, whereas the developing personas interact with a broader region. Therefore,
TABLE VII
INTERACTIONS OF PERSONAS PERFORMED BY THE PPO + CTS RL AGENT IN EXPERIMENT I

| Game Event | Trained path | Tested paths |
|------------|--------------|--------------|
| Trained path | 1 | 2 | 3 | 4 | 5 | 6 |
| Monster Killer | 3 | 0 | 1 | 2 | 2 | 1 |
| Treasure Collector | 0 | 3 | 1 | 2 | 2 | 1 |
| Completionist | 2 | 3 | 1 | 2 | 2 | 1 |

The first row shows the total discounted reward without APFCTS. For the rows with a path number, the number indicates which path we used to train the APFCTS. The values under tested paths show the total discounted reward that the agent receives when APFCTS modulates the environment reward. The bold values demonstrate the found paths when we execute the PPO + CTS + APFCTS agent.

TABLE VIII
TOTAL DISCOUNTED REWARD WITHOUT APFCTS AND WITH APFCTS

| Tested paths | Trained path |
|--------------|--------------|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0.86 | 0.78 | 0.84 | 0.84 | 0.86 | 0.76 |
| 2 | 0.82 | 0.81 | 0.98 | 0.98 | 0.98 | 0.98 |
| 3 | 0.98 | 0.98 | 0.74 | 0.86 | 0.82 | 0.88 |
| 4 | 0.99 | 0.86 | 0.98 | 0.86 | 0.96 | 0.86 |
| 5 | 0.98 | 0.98 | 0.81 | 0.85 | 0.76 | 0.87 |
| 6 | 0.99 | 0.87 | 0.87 | 0.88 | 0.88 | 0.60 |

C. Experiment III: Personas in Doom

We experimented with different personas in the second testbed game, a Doom level (see Fig. 7). The interaction results are presented in Table IX, and all of the personas behave similarly to their specifications. The Exit persona always finishes the game, and in some of the evaluations, Exit persona kills a Monster but never collects a Treasure. The Monster Killer persona generally kills all of the Monsters, rarely collects a Treasure, and habitually finishes the game. Developing Killer is similar to Monster Killer but kills half of the Monsters and rarely dies. They both collect a single Treasure, kill the least Monsters, and die the most. The Completionist, Developing Completionist, and Developing Casual Completionist personas behave similarly, but minor differences exist. The Developing Casual Completionist always finishes the level but usually cannot collect the second Treasure, but in doing so, rarely die and cannot finish the level.

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 X. 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.

VI. DISCUSSION

In this article, 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 used by game designers 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 starting position of the agent is close to the Door, the agent may neglect the Treasures. Conversely, if the Door is positioned after the Treasures, the agent is likely to interact with most of the Treasures. We saw this dilemma 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 exploration into the agents that realize these personas, the set of actions executed by these personas became different. Furthermore, 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].

The problem with the utility function is that the utility function is an amalgamation of multiple goals. Hence, depending on the level composition and RL agent’s hyperparameters, the procedural persona represents one of those playstyles. In Experiment I, we believe that the Developing Completionist fits better with the idea of a “Completionist” persona than the procedural Completionist. Developing persona addresses this problem by introducing a sequence of goals. Consequently, a game designer may use the developing persona to choose which playstyle she wants to playtest carefully.

Another advantage of developing personas over procedural personas is that developing personas support 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—a Treasure Collector—after fulfilling a criterion. These development sequences were mentioned by Bartle [17], but development sequences were impractical, while using a single utility function. Consequently, this behavior performed by Developing Raider was missing in procedural personas. On the other hand, another important aspect of playtesting is the ability to generate playtraces as if a human would. In this article, we used handcrafted utility functions, however, these utility functions could have been extracted from human playtest data by inverse reinforcement learning [28]. This alteration might help the RL agent to generate a playtest that is more humanlike [8], [31].

In addition to the GVG-AI environment, we conducted experiments on the Doom environment. To the best of our knowledge, this article is the first study to playtest personas in a 3-D environment. In 2-D environments, the authors in [4] and [5] employed MCTS RL agent to realize personas. Nevertheless, MCTS would be an ineffective choice for 3-D environments, and MCTS would underrepresent the persona. Consequently, we used the PPO agent in Experiments III and IV, as PPO is a competent agent used by OpenAI [32]. In Experiment III, we see that the PPO agent realized the decision models of personas properly. From the results in Table IX, we interpret that a player has to kill a Monster to finish this level. The level is hardest for Treasure Collector and Developing Collector as they have to kill a Monster to collect the Treasures. We see an interesting fact about the game when we compare the Developing Casual Completionist and the Developing Completionist personas. The former never dies but collects only a single Treasure, whereas the latter seldom dies but collects both of the Treasures. From this data, we understand that collecting the second Treasure causes the death of the player. As the Developing Casual Completionist fears losing her health more, this persona finds collecting the second Treasure risky. Furthermore, comparing the Killer and the Developing Killer personas shows that the latter die more than the former. This comparison unravels another fact about this level. If a player engages in combat to kill Monsters, then this player should kill as much as possible. Otherwise, this player is likely to die, such as the Developing Killer. On the other hand, the Developing Casual Completionist also kills as much Monsters as a Monster Killer. This indifference indicates that the game may not be challenging enough for a hardcore player.

In Experiment II, we prepared a game that consists of five Doors. We found that —without APF— the Exit persona would
take either the first or the fifth path shown in Fig. 9. The lengths of these paths are the same and shorter than every possible path that ends with a Door. Consequently, in the first row of Table VIII, we see that the first and the fifth path share the highest score. Furthermore, in Experiment IV, we saw that —without APF— the Exit persona would take the first path (see Fig. 10). Since this path is the closest toward the Door, and, therefore, playing this path yields a higher score compared to the other path, shown in Table X.

We proposed APF to let RL agents discover these additional paths shown in Figs. 9 and 10. A human playtester would have played these paths, but without APF, the Exit persona would overlook them as these paths yield a lower score. Hence, the game designer would not have any playtest data for other endings. Tables VIII and X show insight on how APF achieves this feat. APF modulates the reward signal of the environment. When the agent tries to learn a similar path, the agent is penalized, and when the agent tries to learn a distinct path, the agent is rewarded. This reward modulation is the reason how APF promotes finding distinct paths. The game designer can exercise the APF to get a distinct path and then study this path to improve her game. Afterward, she can exercise the APF to generate as many paths as she needs. However, the game designer might be interested in examining the play traces that could have come from human playtesters. We could employ an auxiliary NN trained to select the best humanlike action given an observation [2]. Nevertheless, carefully combining this NN with APF is a topic of another study.

On the other hand, an alternative path is a subjective concept. Every human playtester may think of another way to represent the Exit persona. In Table VIII, we see that when we train the APFCTS with the second path, the score of the first path decreases, and the score of the sixth path increases. According to APFCTS, the first and second paths are more similar than the second and sixth paths (see Fig. 9). However, one might argue that the first and second paths are distinct as they reach different Doors, and the second and sixth paths are similar as they reach the same Door. Though APFCTS is objective in finding alternative paths, these alternative paths are “subjectively” different for the game designers. The objectivity of APFCTS and APFICM comes from the recording probability of a frame and the dynamics prediction error, respectively.

Additionally, we found that APFICM is more robust compared to APFCTS. We also experimented with APFCTS in Doom. However, CTS calculated the recording probability of some frames 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, researchers employed curiosity to increase the playtesting coverage of an RL agent [33]. Though we promoted APF to find distinct paths, APF may help game tester agents [8]. Coverage is crucial for testing, and APF increases coverage by finding distinct paths.

**Limitations and 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 benefit from these 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 article focuses 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 alternative 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 various personalities in developing personas to generate unique playtests. Furthermore, our experiments indicate that developing personas provide information to game designers that procedural personas cannot provide. Furthermore, we show that automated playtesting can be extended to 3-D environments using state-of-the-art RL algorithms.

We proposed APF to discover alternative 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 persons 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 3-D environments such as Minecraft [34].

**APPENDIX A**

**HYPERPARAMETERS USED IN EXPERIMENTS**

| Hyperparameters | PPO   | PPO+CTS | PPO+ICM |
|-----------------|-------|---------|---------|
| Policy          | CNN   | CNN     | CNNLSTM |
| Timesteps       | 1e8   | 1e8     | 2e8     |
| 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 \times 10^{-4} | 5 \times 10^{-4} | 5 \times 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 A2

| Hyperparameters | APFCTS | APFICM |
|-----------------|--------|--------|
| $p_{pos cap}$   | 0.4    | 0.1    |
| $n_{neg cap}$   | -0.4   | -0.4   |
| APF Beta ($\beta$) | 0.01   | 0.01   |

### References

[1] E. J. Powley, S. Colton, S. Gaud, R. Saunders, and M. J. Nelson, “Semi-automated level design via auto-playtesting for handheld casual game creation,” in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.

[2] S. Gudmundsson et al., “Human-like playtesting with deep learning,” in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[3] S. Roohi et al., “Predicting game difficulty and churn without players,” in *Proc. Anna. Symp. Comput.-Human Interact. Play*, New York, NY, USA, 2020, pp. 585–593.

[4] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, “Automated playtesting with procedural personas with evolved heuristics,” *IEEE Trans. Games*, vol. 11, no. 4, pp. 352–362, Dec. 2019.

[5] L. Mugrai, F. Silva, C. Holmgård, and J. Togelius, “Automated playtesting of matching tile games,” in *Proc. IEEE Conf. Games*, 2019, pp. 1–7.

[6] C. Holmgård et al., “Generative agents for player decision modeling in games,” in *Proc. 9th Int. Conf. Found. Digit. Games*, 2014. [Online]. Available: https://dblp.org/rec/conf/fdg/HolmgardLTY14.html?view=bibtex

[7] R. Bartle, “Virtual worlds: Why people play,” *Massively Multiplayer Game Develop.*, vol. 2, pp. 3–18, Jan. 2005.

[8] S. Aryurek, A. Betin-Can, and E. Surer, “Automated video game testing using synthetic and humanlike agents,” *IEEE Trans. Games*, vol. 13, no. 1, pp. 50–67, Mar. 2021.

[9] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015.

[10] J. Schulman et al., “Proximal policy optimization algorithms,” 2017. [Online]. Available: https://dl.fbaipublicfiles.com/imagenet_models/arXiv2017_postprint.pdf

[11] C. B. Browne, “A survey of Monte Carlo tree search methods,” *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[12] M. G. Bellemare et al., “Unifying count-based exploration and intrinsic motivation,” in *Proc. Adv. Neural Inf. Process. Syst. 29th: Annua. Conf. Neural Inf. Process. Syst.*, 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 *Proc. Adv. Neural Inf. Process. Syst. 30th: Annua. Conf. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017, pp. 1471–1479.

[14] D. Pathak et al., “Curiosity-driven exploration by self-supervised prediction,” in *Proc. 34th Int. Conf. Mach. Learn.*, 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 Trans. Games*, vol. 11, no. 3, pp. 195–214, Sep. 2019.

[16] M. Kempka, M. Wydmuch, G. Ruc, J. Toczew, and W. Jaskowski, “ViZDoom: A doom-based AI research platform for visual reinforcement learning,” in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 341–348.

[17] R. A. Bartle, “Hearts, clubs, diamonds, spades: Players who suit MUDs,” 1996. Accessed: Jul. 30, 2019. [Online]. Available: http://www.mud.co.uk/richard/hcds.htm

[18] A. Tychsen and A. Canossa, “Defining personas in games using metrics,” in *Proc. Conf. Future Play: Res., Play, Share, Ser.*, Future Play, 2008, pp. 73–80.

[19] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, “Evolving personas for player decision modeling,” in *Proc. IEEE Conf. Comput. Intell. Games*, 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 *Proc. 11th Artif. Intell. Interactive Digit. Entertainment Conf.*, 2015, pp. 8–14.

[21] F. de M. Silva et al., “AI-based playtesting of contemporary board games,” in *Proc. 12th Int. Conf. Found. Digit. Games*, 2017, Art. no. 13.

[22] G. Ostrovski et al., “Count-based exploration with neural density models,” in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2721–2730.

[23] H. Tang et al., “EG: Exploration by random network distillation,” in *Proc. 7th Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=SJNqoXeYbe

[24] Y. Burda et al., “Exploration by random network distillation,” in *Proc. 8th Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=SJNqoXeYbe

[25] L. Lee et al., “Evaluating unsupervised exploration via sparse global memory,” in *Proc. 9th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=JWcCCL9LNM

[26] J. A. Brown, “Towards better personas in gaming: Contract based expert systems,” in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 540–541.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[28] M. G. Bellemare, J. Veness, and T. Talvitie, “Skip context tree switching,” in *Proc. 31th Int. Conf. Mach. Learn.*, 2014, pp. 1458–1466.

[29] A. Hill et al., “Stable baselines,” 2018. [Online]. Available: https://github.com/hill-a/stable-baselines

[30] B. Tastan and G. Sukthankar, “Learning policies for first person shooter games using inverse reinforcement learning,” in *Proc. 7th AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2011, pp. 85–90.

[31] B. Baker et al., “Emergent tool use from multi-agent autocurricula,” in *Proc. 8th Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=iclr/BakerKMWPMM20.html?view=bibtex

[32] C. Gordillo et al., “Improving playtesting coverage via curiosity driven reinforcement learning agents,” in *Proc. IEEE Conf. Games*, 2021, pp. 1–8.

[33] M. Johnson et al., “The Malmo platform for artificial intelligence experimentation,” in *Proc. 25th Int. Joint Conf. Artif. Intell.*, AAAI - Assoc. Advanc. Artif. Intell., 2016, pp. 4246–4247.