Towards Informed Design and Validation Assistance in Computer Games Using Imitation Learning

Alessandro Sestini¹, Joakim Bergdahl¹, Konrad Tollmar¹, Andrew D. Bagdanov², Linus Gisslén¹

¹SEED - Electronic Arts (EA), ²Università degli Studi di Firenze
{aestini, jbergdahl, ktollmar, lgisslen}@ea.com
andrew.bagdanov@unifi.it

Abstract—In games, as in many other domains, design validation and testing is a significant challenge as systems are growing in size and manual testing is becoming infeasible. In this position paper we outline an approach to automated game validation based on an imitation learning technique, and provide an analysis of the potential benefits to automated game testing. The method leverages a data-driven technique, which requires little effort and time and no knowledge of machine learning or programming, that designers can use to efficiently train game testing agents. We evaluate the validity of our claim by conducting a user study with industry experts. The survey results presented in this paper demonstrate the potential of a data-driven approach to reduce effort and enhance the quality of game testing. Moreover, the survey reveals several open challenges. To this end, we analyze the identified challenges and provide a basis for further research and discussion, as well as to help guide the development of imitation learning for game testing.

Index Terms—Automated Playtesting, Imitation Learning, Game Design

I. INTRODUCTION

Modern games are often enormous and grow in size, complexity, and asset count with every year. Making games is a complex endeavour that can often involve thousands of developers, costing on the order of hundreds of millions of dollars. To ensure that a game plays as intended, and that each of its components work together, every time a level or level area is created or modified, there is a need for gameplay validation. Playtesting is one common procedure for assessing the quality of games. Testers check whether the game is complete, if it is fun and sufficiently challenging, or if it has problems like bugs or glitches. The process of game validation and testing is usually done either by an in-house quality verification (QV) team or by internal and external designated human testers. However, manual playtesting does not scale well with the size of games and the turnaround time – the time between game design and test feedback – is often long. To be effective, game validation should be performed as early as possible and ideally by those who actually create the content that requires testing: the game designers. For manual game testing, this is often limiting as bringing in human playtesters is expensive and inefficient. With automated game validation, it would be possible to bring in game playing agents directly into the development phase.

Recently, automated playtesting and validation techniques have been proposed as a way to reduce manual validation in large games. Automated gameplay testing is a fast and relatively cost-effective solution for testing games at scale, often achieved by crafting hand-scripted bots [1]. Additionally, recent research has proposed training self-learning agents with reinforcement learning [2, 3]. While these approaches are useful for exploring game scenes and exploiting environments to identify bugs and/or glitches in the geometry [4, 5], they have their drawbacks. Scripted agents, typically bots with hand-crafted behaviors, require a certain level of domain knowledge and programming skills that game designers may not possess; in addition, these agents may lack generalization and respond poorly to changes made in the game environment. Reinforcement learning agents are able to learn how to play the game without scripting, but they also require a high level of expertise in machine learning to be effectively used (e.g., creating a well-performing reward function). Furthermore, they are sample inefficient and provide a low level of controllability [2, 6]. Nonetheless, they present a promising avenue for automatically discovering and addressing various flaws in game systems at scale.

This position paper first proposes a data-driven approach for creating agent behaviors for automated game validation. The method proposed uses imitation learning, in particular the DAgger algorithm [7], to clone the behavior of the user. With this, we can leverage developer expertise without requiring knowledge of programming or machine learning. The feedback loop for the designers can therefore be rapid with a more efficient creation process than having to wait for days or sometimes weeks for manual playtests. This study is made to either validate or refute above claims by answering the following questions: can imitation learning be used as an effective game design tool? What improvements and research are needed to maximize its value as a tool? To explore this, we showcase the possible use cases for which level designers could utilize the approach. Secondly, we report on results from a survey that gauges the interest among professional game developers of using the proposed IL-based method as a tool for design assistance. Based on the answers from the survey, we propose future research directions for improving game validation with imitation learning.
II. RELATED WORK

Several studies have investigated the use of AI techniques for automated playtesting. Many of these works rely heavily on classical hand-scripted AI. A notable example is the work by Mugrai et al. [8], who developed an algorithm for mimicking human behavior to achieve more meaningful gameplay testing results, but also to aid in the game design process. Several approaches from the literature are based on scripted automated playtesting. Ifitikhar et al. [9] proposed a testing approach for automated, black-box functional testing of platform games, and Lovreto et al. [10] reported their experience with developing scripted automated tests for 16 mobile games. Other notable examples of scripted testing can be found in [11, 12, 13]. However, when dealing with complex 3D environments these techniques are not readily applicable due to the high-dimensional state space involved and poor generalization performance.

Other works have used Reinforcement Learning (RL) to solve the aforementioned problems [2]. Zheng et al. [14] proposed an on-the-fly game testing framework which leverages evolutionary algorithms, deep RL and multi-objective optimization to perform automated game testing. Agarwal et al. [15] trained reinforcement learning agents to perform automated playtesting in 2D side-scrolling games, producing visualizations for level design analysis. Bergdahl et al. [16] proposed a study on the usefulness of using RL policies to playtest levels against scripted agents, while Gordillo et al. [5] used intrinsic motivation to train many agents to explore a 3D scenario with the aim of finding issues and oversights. Finally, Sestini et al. [4] proposed the curiosity-conditioned proximal trajectories algorithm with which they test complex 3D game scenes with a combination of IL, RL and curiosity driven exploration. While one can leverage RL to exploit the environment for catching bugs and use exploration to create a more robust policy, at the same time it is commonly known to be sample inefficient and leaves little control to the user over the final behavior of the policy [6, 17].

Few approaches have applied IL to game testing agents. The Reveal-More algorithm uses demonstrations to guide exploration, although it does not use a learning algorithm [18]. Zhao et al. [19] used an approach similar to behavioral cloning in which gameplay agents learn behavioral policies from the game designers. Harmer et al. [20] trained agents through a combination of IL and RL with multi-action policies for a first person shooter game, although not in the scope of game testing. Similar to our work, the Falken [21] framework provides developers with the ability to train an ensemble of game-testing agents, each of which could effectively accomplish tasks of a few minutes each. Our approach differs from Falken as it is more general, game-engine agnostic and it uses a neural network architecture that can be applied to a variety of use-cases. In this paper, we claim that with IL, one can leverage the designer’s domain knowledge of the game and effectively add controllable agents for game testing. Not only can intended behaviors be demonstrated, but also corrected using new demonstrations with little additional training time. Machado et al. [22] supports this claim suggesting that agents following steps from a human trainer would help in cases where the agent needs to come-up with a strategy to overcome some challenges radically different from the one available. The experimental analysis in Section IV shows that IL is a viable alternative to RL for game testing, but requires minimal theory knowledge or programming skills compared to the latter.

III. PROPOSED APPROACH

Here we describe our approach to exploring the potential of data-driven methods for gameplay validation to designers. Algorithm. We use an IL approach based on the DAgger algorithm [7]. DAgger allows designers to train agents interactively by actually playing the game. The core of the algorithm is to let designers seamlessly move between designing a level, providing game demonstrations and getting feedback from trained testing agents. The approach requires little training time and is more sample efficient than the baseline methods, as seen in Section IV. Designers can also provide corrections to faulty agent behaviors, resulting in a continuous and rapid feedback loop. Providing corrections is as easy as taking the controller back and playing the game one more time. Once they are satisfied with the agent’s behavior, designers can stop providing demonstrations and watch the agent validate
the game. Developers can then make design modifications to
the game scene and quickly obtain feedback from agents with
minimal or no need for additional demonstrations.

State Space. For the approach to be effective, it must be
as general as possible in order to adapt to the many different
game genres and scenarios that designers may construct. Since
3D movement often is a crucial gameplay element of modern
video games, we focus on finding the best state representation
by taking this into consideration. This approach, with minor
observation tweaking, will transfer well to similar, but con-
textually different game modes. For setting up a behavior,
developers define a goal position in the game environment.
The spatial information of the agent relative to the goal is
composed of the $R^2$ projections of the agent-to-goal vector
onto the $XY$ and $XZ$ planes as well as their corresponding
lengths normalized to the available gameplay area. We also include
information about the agent indicating whether it can
jump, whether it is grounded or climbing, as well as any other
auxiliary data expected to be relevant to gameplay. The user
also specifies a list of entities and game objects that the agent
should be aware of, e.g. intermediate goals, dynamic objects,
enemies, and other assets that could be useful for achieving the
final goal. From these entities, the same relative information
is inferred as for the main goal position relative to the agent.
Lastly, the agent also has local perception. A semantic map
is used similar to the one used by Sestini et al. [4] which is
general and performant. An example of such a semantic 3D
occupancy map used as input to the networks is illustrated
in Figure 1(a). This map is a categorical discretization of the
space and elements around the agent, and each voxel in the
map carries a semantic integer value describing the type of
object at the corresponding game world position. The settings
of the semantic map can be configured by the designers.

Model. We build the neural network $\pi_\theta$ with parameters $\theta
with the goal of being as general and reusable as possible. The
way we define the structure of the network is to fundamentally
allow a higher level of usability by game designers. First, all
the information about the agent and goal is passed into a linear
layer with ReLU activation, producing the self-embedding
$x_0 \in R^d$, where $d$ is the embedding size. For the experiments,
a value of $d = 128$ was used. The list of entities is passed
through a separate linear layer with ReLU activation, same
size $d$ and shared weights producing embeddings $x_e \in R^d$,
one for each entity $e_i$ in the list. Each of these embedding
vectors is concatenated with the self-embedding, producing
$x_{ae_i} = [x_0, x_e]$, with $x_{ae} \in R^{2d}$. This list of vectors is then
passed through a single transformer encoder with 4 heads,
hidden size of 128 and final average pooling, producing a
single vector $x_1 \in R^{2d}$. In parallel, the semantic occupancy
map $M \in R^{s \times s \times s}$ is first fed into a learned embedding layer of
size 8 and Tanh activation, transforming categorical represen-
tations into continuous ones, and then into a 3D convolutional
network with three convolutional layers with 32, 64 and 128
filters, stride 2 and leaky ReLU activations. The output of
this convolutional network is a vector embedding $x_M \in R^d
that is finally concatenated with $x_1$ and passed through a feed
forward network with 2 linear layers of size 256 and ReLU
activations, producing an action probability distribution. The
complete neural network architecture is shown in Figure 1(b).

Given a demonstration dataset $D = \{\tau_i \mid \tau_i =
(s_0, a_0, ..., s_T, a_T)\}$, $i = 1, ..., N$ of $N$ trajectories $\tau_i$, each
composed of a sequence of state-action pairs $(s_k, a_k)$, we
update the network following:

$$\arg \max_\theta \mathbb{E}_{(s, a) \sim D}[\log \pi_\theta(a | s)]. \tag{1}$$

This objective aims to mimic the expert behavior which is
represented by the dataset $D$.

IV. EXPERIMENTS

In order to evaluate the performance of our approach, we
define a set of use cases that resemble typical situations that
game designers face in their daily workflow.

Environment. The environment used is constructed around
3D navigation with procedurally generated elements. The

Fig. 2: (a) Screenshot of the “Validating a Complex Trajectory” use case. The agent is tasked to navigate to a goal hidden
behind a door that can be opened by interaction with a button. In the left part, we can see the original environment with
the trajectory demonstrated to the agent (red line), while in the right part we can see the modified one. (b) Screenshots of
the “Navigation” use case. The agent has multiple intermediate goals: use the elevator, interact with the button to create the
bridge, destroy a wall by shooting it, and arrive at the goal location. (c) Screenshot of the “Validating a Complex Trajectory”
use case. The agent has to navigate through a procedural city. The assets of the environment are procedurally generated.
The environment is purposefully built like a mutable sandbox which can offer scenarios where a designer’s level design workflow can be simulated. Users can add and change the goal location, agent spawn positions, layout of the level, location of intermediate goals and the locations of dynamic elements in the map. In this environment, the agent has a set of 7 discrete actions: move forward, move backward, turn right, turn left, jump, shoot, and do nothing. In addition, the agent can use interactable objects located around the map. Figure 2 shows some screenshots of the environment.

Training Setup. We compare our approach to two main baselines: Base-RL and Tuned-RL, both trained using the PPO [23] algorithm with identical hyperparameters until reward convergence. Base-RL utilizes a naive reward function that gives a positive, progressive reward based on the distance to the goal. For Tuned-RL, a hand-crafted, dense reward function is used that is instead designed to lead the agent along a path, reaching multiple sub-goals before the final main goal. The reward for both is defined as:

$$R_t = \alpha(D_t(\text{agent, goal}) - D_{t-1}(\text{agent, goal})) + \mathbb{I}(D_t(\text{agent, goal}) < \epsilon),$$

where goal is the current goal (either a sub-goal or the main one) of the agent, $D_t$ is the distance from the agent to the current goal at timestep $t$, $\mathbb{I}(\cdot)$ returns 1 when its argument is true and 0 otherwise, and $\alpha$ and $\epsilon$ are hyperparameters.

The difference between Base-RL and Tuned-RL is that the former use only one global goal, while the latter has a set of different intermediate goals specifically defined to have the same behavior as the demonstrator. We use the same settings for all subsequent experiments. For each of the methods, we are interested in: the success rate of the agent (i.e. how many times it reaches the goal), training time (i.e. the time it takes to reach that success rate), generalization success rate (i.e. the success rate of the same agent in a different version of the environment), and imitation metric (i.e. how close the trajectories made by the agent are to the demonstrations). For the latter, we use the 3D Frechét distance [24] between the agent trajectories and the demonstrator ones.

### Use case 1: Validating Design Changes

For the first use case, we investigate whether an IL agent can validate the layout of a level in real-time. The goal in this experiment is to train an agent to follow human demonstrations, then to change the level layout and see how the agent adapts to these changes. This will give developers an insight into the usefulness of our proposed approach when they are still in the design process: they can see how agents would adapt accordingly to modifications in the level layout. The agent is tasked to navigate to a goal hidden behind a door that can only be opened by interaction with a button. After training the agent, we test it by removing and adding new elements to the level, e.g. removing existing layout elements and adding several new obstacles within the path to the goal. In Figure 2(a), we give details of the use case. For this use case, we collect 6 episodes of demonstrations (for a total of 2700 timesteps at 50 frames per seconds). The Tuned-RL baseline is trained with Equation 2 but with 2 sub-goals. As Table I demonstrates, IL provides faster training times than RL, while simultaneously granting the developer more control over the final agent behavior. As seen in this table, we could improve controllability and training time of the RL solution (e.g. Tuned-RL), but this requires reward shaping, which is a non-trivial task – especially for non-experts in machine learning [25]. The table also reveals that IL yields the same level of generalization as RL and is able to quickly adapt to minor modifications of the environment.

### Use case 2: Validating Complex Trajectories

For this use case we train an agent to reproduce a complex trajectory as shown in Figure 2(b). The agent has multiple intermediate goals: use the elevator, interact with the button to create the bridge, destroy a wall by shooting it, and arrive at the goal location. This type of test is beneficial for designers to validate their design, or check if all interactable elements are functioning as expected (e.g. if the button always creates the bridge without problems). Here we are not interested in generalization, but rather the ability to replicate the expert behavior as quickly and closely as possible. For this use case, we use 5 episodes of demonstrations (for a total of 4800 timesteps at 50 frame per seconds). The Tuned-RL baseline is trained with Equation 2 but with 4 sub-goals. In Table I, it can be seen that the proposed interactive approach is more suitable for the goals compared to the baselines. As the table shows, for this use case IL is not only more sample efficient than RL, but also less time consuming than even the Tuned-RL baseline.

### Use case 3: Identifying Alternative Behaviors

For this use case we identify alternative agent behaviors. The IL agent is trained to reach a goal (e.g. the elevator) as quickly and closely as possible. For this use case, we collect 5 episodes of demonstrations (for a total of 2500 timesteps at 50 frames per seconds). The Tuned-RL baseline is trained with Equation 2 but with 3 sub-goals. As Table I demonstrates, IL provides faster training times than RL, while simultaneously granting the developer more control over the final agent behavior. As seen in this table, we could improve controllability and training time of the RL solution (e.g. Tuned-RL), but this requires reward shaping, which is a non-trivial task – especially for non-experts in machine learning [25]. The table also reveals that IL yields the same level of generalization as RL and is able to quickly adapt to minor modifications of the environment.

| Use case 1 | Use case 2 | Use case 3 |
|------------|------------|------------|
| **IL (Ours)** | **Base-RL** | **Tuned-RL** |
| Success ↑ | Time ↓ | Generalization ↑ | Imitation ↓ |
| 0.95 ± 0.00 | 0.02 h | 0.96 ± 0.05 | 6.84 ± 0.34 |
| 0.91 ± 0.02 | 5.00 h | 0.90 ± 0.00 | 8.37 ± 0.27 |
| 0.95 ± 0.00 | 4.48 h | 0.90 ± 0.02 | 8.13 ± 0.20 |
| **IL (Ours)** | **Base-RL** | **Tuned-RL** |
| Success ↑ | Time ↓ | Generalization ↑ | Imitation ↓ |
| 0.90 ± 0.02 | 0.00 h | - | 7.73 ± 1.10 |
| 0.00 ± 0.00 | 18.18 h | - | 28.11 ± 0.41 |
| 0.92 ± 0.03 | 13.56 h | - | 9.53 ± 1.37 |
| **IL (Ours)** | **Base-RL** | **Tuned-RL** |
| Success ↑ | Time ↓ | Generalization ↑ | Imitation ↓ |
| 0.81 ± 0.08 | 0.22 h | 0.78 ± 0.03 | 46.07 ± 1.03 |
| 0.00 ± 0.00 | 16.49 h | 0.00 ± 0.00 | 80.22 ± 0.53 |
| 0.86 ± 0.05 | 4.12 h | 0.87 ± 0.03 | 16.79 ± 2.12 |
of shooting at the final wall, the agent uncovered a distinct route, different from the intended one, to conclude the episode. While this behavior would be suitable for exploits detection, it is not desirable for this particular use case, demonstrating how RL and IL can complement each other in an actual testing scenario.

**Use case 3: Navigation.** Navigation is one of the most fundamental aspects of many modern video games. The traditional scripted way to handle navigation is to pre-bake and use navigation meshes in combination with classical pathfinding algorithms. However, using a navigation mesh becomes intractable in many practical situations, and it requires compromising between navigation cost and quality. This is achieved in practice by either removing some of the navigation abilities or by pruning navigation mesh connections [26]. Moreover, every time designers change something in the layout, the navigation mesh needs to be regenerated. For this use case, we consider a complex navigation task where the agent has to navigate through a city that the designers have not completely finished yet. To simulate this, we first record demonstrations in an unfinished city, and then we evaluate the trained agent within a procedural test city that changes its asset and scenario elements every 2 seconds until it reaches the goal location. This requires an agent that is able to not only follow the provided demonstrations, but also to generalize to different elements. The combination of these two capabilities is non-trivial with a navigation mesh agent only. A screenshot of this use case is given in Figure 2(c). For this use case, 6 episodes of demonstrations were used (for a total of 12600 timesteps at 50 frames per second). Table I summarizes the results. Since the city environment is large, training agents with RL would take more time compared to guiding them along the correct path. Moreover, even if they enjoy a similar level of generalization, the corresponding RL agents will try to exploit the environment finding different ways to arrive at the goal location. Similar to use case 2, the Tuned-RL baseline is trained with Equation 2 but with 5 sub-goals that define waypoints leading to the goal. This baseline yields slightly better results in this case, achieving a similar success rate while exhibiting improved generalization and imitation capabilities. This is likely due to the hand-engineered reward being designed to be close to the original demonstrations, allowing for generally good results in the imitation task as confirmed by the other use cases. However, all of this comes with the cost of sample inefficiency and requiring an engineered reward function.

**Ablation Study.** We perform a series of tests to better understand the performance of the model used in this work. In particular, we claim that the combination of the transformer network and the 3D convolutional network with the learned embeddings described in Section III is an efficient way to encode environment information. For these experiments, we trained different agents using the same set of demonstrations and the same number of epochs, but with three different ablations. All of them start from the architecture described in Section III, and each replaces one component of the original one: a two-layer MLP with size of 256 and ReLU activations replacing the transformer network, a two-layer MLP of size 256 with ReLU activations replacing the 3D convolutional network, and a one-hot encoding replacing the learned embedding. Table II shows that the combination of these components provides a significant boost to performance compared to using the single ones.

| ID     | Role          | Genre(s) | YoE | MLK |
|--------|---------------|----------|-----|-----|
| P01    | Level Designer| FPS      | 15  | Low |
| P02    | Level Designer| FPS      | 11  | None|
| P03    | Level Designer| Racing   | 3   | Very Low |
| P04    | Level Designer| RPG      | 6   | High |
| P05    | Level Designer| Racing   | 1   | Very Low |
| P06    | Level Designer| RG      | 4   | Very Low |
| P07    | Game Designer | RPG      | 9   | Low |
| P08    | Game Designer | Sport    | 3   | Very Low |
| P09    | Game Designer | Sport    | 4   | Very Low |
| P10    | Game Designer | Match3   | 1   | Very Low |
| P11    | Level Designer| RPG      | 15  | Very Low |
| P12    | Level Designer| FPS      | 21  | Low |
| P13    | Gameplay Designer| RPG | 15  | Very Low |
| P14    | Game Designer | RPG      | 22  | None |
| P15    | Level Designer| FPS, racing | 10 | Very Low |
| P16    | Level Designer| RPG      | 5   | Very Low |
| P17    | Design Director| Match3 | 17  | None |
| P18    | Level Designer| FPS      | 13  | Low |

**TABLE III: Summary of participants.** Abbreviations: YoE (years of experience), MLK (machine learning knowledge), FPS (first person shooter), RPG (role-playing game).
The survey was divided into four sections: the first asks participants of some background information; the second asks them about their current game validation workflow, in particular if they use manual or automated playtesting; the third, being the main part of the survey, asks participants what they think about the proposed solution, if they would use it in their games, what characteristics an agent/approach like this should have to help them in their game and level design work, if they think IL would help them create better games; and the fourth contains optional questions about possible use cases and future directions they see that we had not considered.

Since one of the main focuses of this paper is to assess what improvements and research are needed to maximize the value of using such an approach, one important question in the third section of the survey asks participants to evaluate various characteristics that an agent should display for automated content evaluation. These characteristics are: Imitation - the agent can exactly replicate the demonstrations; Generalization - the agent can adapt to different variations of the same situation; Exploration - the agent can explore beyond the demonstrations and find bugs and issues; Personas - the agent can have different types of behaviors; Efficiency - the agent must use as few demonstrations as possible; Controllability - having control over the agent behavior vs complete autonomy; Feedback - the agent gives feedback when the amount of demonstration data is enough; Fine tuning - behaviors can be fine tuned after initial training; Interpretability - the agent can inform when and why it fails.

VI. Survey Results

We received a total of 18 survey responses. The majority (11/18) of the participants have more than 5 years of experience in level design, with a median of 7.5 years. All the participants are level, game, or gameplay designers. The general machine learning knowledge of the participants range from very low to low. 16/18 of the respondents do not currently use automated playtesting and they rely on external people or systems for their validation. 14/18 of designers never rely on automated playtesting and only 6/18 have used a scripted automated method at least once in their daily work. They work on different game projects and game genres and they use different tools in their level design workflow, but all respondents have knowledge of game engine editors. The general consensus is that designers would likely use a tool like this as “it definitely would be useful to speed up the typically time consuming iterative design process [P05]” and “a method like this can definitely speed up iteration, which is one of the main things any designer spends a lot of time [P04]”. Moreover, they can relate to the demonstrated example as “it is not quite like how designers are building their levels, but it is not far off [P06]” and they think the examples shown are “realistic for some games such as platformers [P01]”.

Participants acknowledge the usefulness and the potential of the demonstrated method, even if it would need proper adaptation for handling the specific game genre they are working on. Further, 15/18 of the respondents agree that automated validation of the level directly in the game editor is useful and 16/18 of them think the demonstrated agents could be useful for assisting in validating their content. 9/18 agree that a method like this adds value compared to scripted agents and some of them (5/9) think it could completely replace scripting, while 5/18 could not answer this question as they had never used other types of automated methods.

Some designers are skeptical about the complexity of the examples shown in the survey and they would like to see how well the approach works in more complicated games. One participant says that “[they are] wondering about how effective this would be in genres that are more complicated mechanically [P05]”. This was expected due to the preliminary nature of this study and that we do not cover every genre but focus on one. However, a larger group (15/18) thinks that complexity was high enough which reinforces the notion that the environment is relevant to a significant part of the community. Designers also describe the challenges of using such a tool in their workflow: one argues that “human players play differently than bots, can be difficult to only rely on AI when it comes to testing [P02]” or “demonstrations take time, which would push busy level designers to not use it. Demonstrating how to solve specific issues – e.g. recover from a mistake – should not be on the level designer [P04]”. Not all game creators are fully convinced by the approach because “the demonstration video showed a process that does not really show a level designer anything new or unexpected [P06]”.

After describing their doubts, we asked participants to evaluate each of the characteristics delineated in the previous section to specify which ones are the most important to improve the approach. Table IV summarizes these characteristics ratings. This not only shows that our initial hypothesis is supported by professional designers, but also outlines their preferences and doubts about using such an approach. This is important because, as we will see in Section VII, the values in Table IV form precise research directions that we encourage all of the game research community to consider in future contributions to this field. As we detail in Section VII, based on the results in Table IV, we can generally say that a combination of IL and RL could be an effective approach for automated game testing.

VII. Future Directions

This section proposes research opportunities discovered by interviewing experts in the field.

Generalization. One of the most important requests is to have an agent that not only imitates expert behaviors, but that is more representative of the unpredictable nature of players. Moreover, “if we can create many simulations, with various amounts of generalization it could be a key to finding issues [P07]”. With the proposed general purpose neural network and state space we mitigate this problem, but in many cases agents trained under one set of demonstrations are not able to adapt to larger variations [27]. Generalization as a subject in self-learning agents has already been addressed in literature, but state-of-the-art approaches are not readily applicable for.
the stated purpose: most of them use either interactions with the game [28, 29, 30] or learn the inverse dynamics of the environment [31]. A possible future direction is to try data augmentation techniques used in Offline Reinforcement Learning (ORL) algorithms such as the work done by Sinha et al. [32]. ORL has several connections to imitation learning [33]. It learns a policy using a pre-defined dataset without direct interactions with the environment, but in contrast to IL which supposes the data comes from an optimal policy, ORL can also leverage sub-optimal examples. This leads to trained agents being more general and allows for a higher exploration level.

**Personas.** One recurring feature requested by survey participants is the ability to train the model for different behavior types: “if this system would allow giving them different behaviors I could see more value in using it [P12]“. The aim is to have various behaviors similar to the multi-modal nature of how humans play, in order to create more meaningful agents. The research community has addressed the problem of creating personas many times, but to our knowledge exclusively in RL contexts. Works from de Woillemont et al. [34], Roy et al. [35], and Sestini et al. [36] have explored how to combine behavior styles with different reward functions which is not applicable to IL. The work by Peng et al. [37] is an example of how to combine IL and RL to create different animation styles. More research into achieving different playstyles with only IL would allow designers to train more diverse testing agents for validating gameplay in different ways.

**Exploration.** Participants frequently brought up the exploration aspect, i.e. the ability to look beyond expert demonstrations in search for overlooked issues: “the improvisation will show you more likely what some players might try to do [P13]”. Exploration is a well known problem in RL literature [38], but exploration in IL is, to our knowledge, largely unexplored. This is mainly due to the conflicting nature of an agent that must both learn to precisely follow the expert demonstrations while simultaneously explore beyond the expert behavior. Moreover, one would not use exploration to improve agents in the validation context, but rather exploit it to find bugs. Sestini et al. [4] provided a good example of how to leverage both IL and RL to train agents to both follow demonstrated behaviors but also to explore in search for issues. However, this type of solution is still too sample inefficient to be used as an interactive design tool.

**Usability.** The usability aspect is one of the most important to participants. This includes both providing useful information to designers about the models they are training, and the ease-of-use of the tool. Recent techniques from the explainable RL [39] research community could be used to address the challenge of interpreting behaviors of the agent. To improve upon this, one can use techniques from game analytics research and gameplay visualization [4, 40]. Additionally, a sentiment found in the survey is that for a tool like this to be usable, the effort imposed on the end-user needs to be minimal. To address the latter, few-shot IL is an active topic that can potentially help reducing the number of samples required. Works by Duan et al. [41] and Hakkamanseshi et al. [42] specifically addressed this problem within IL. However, these approaches require many preliminary training iterations and it is unclear how this could be applied to a game in development that is unstable and not yet finished. One way to both improve the agent’s quality and the usability of the tool is to leverage the already cited ORL techniques [32, 43, 44]. We can leverage the recording process to not only store demonstrations, but all other interactions the agent performs while testing the level.

**Multiple Agents.** Many designers noted that most modern video games are multi-agent systems, and in order to thoroughly test these environments we need to train multiple interacting agents. One participant says: “I would like to be able to train agents on different teams and have them battle. [P01]”. Most of the current research in IL focuses on single agents learning from a single teacher, and there are only a few examples of multi-agent IL. Harmer et al. [20] used IL in a multi-agent game, but they do not address the problems of a multi-agent system, while Le et al. [45] proposed a joint approach that simultaneously learns a latent coordination model along with the individual policies. We believe there is still a long way to go for meaningful multi-agent IL, especially for this use case.

VIII. DISCUSSION AND LIMITATIONS

In this position paper we argue that data-driven behaviors via imitation learning is a suitable approach for real-time validation of game and level design. We propose an imitation learning approach and investigate its performance, focusing on three different design validation use cases. Our experiments demonstrate how this type of approach can satisfy many of the requirements of an effective game design tool in comparison to approaches utilizing reinforcement learning or scripted behaviors. A user study was also performed with professional game and level designers from diverse game studios and game genres. The participants were asked to assess the desirability and improvement opportunities of using such an approach in their daily workflow. Moreover, designers were asked what characteristics they would want from a data-driven tool for creating autonomous agents that validate their design. The user study highlights the desire of designers to automatically validate their games and with the preliminary results we demonstrate that the data-driven approach proposed is a potential candidate for achieving such objectives. The study also highlights challenges and the gap that exists between techniques from literature and their actual usability in the industry. For this reason, this paper proposes a series of research directions that will help shape such approaches into effective tools for game design.

Despite the successes of this preliminary approach and the user study, there are still several limitations to consider. The user study only included 18 participants that viewed a video of the tool’s functionalities. To understand the full potential of imitation learning for video game design, a larger sample of participants that can actually use and test the tool is needed. Due to technical limitations, we were not able to conduct such
a test, and it was left as a potential user study in the future. Additionally, in our use cases, we tested the approach with navigation and interaction tasks only. In the future, we plan to use our approach for more complex gameplay scenarios, such as team combat or racing games.

REFERENCES

[1] S. Stahlke, A. Nova, and P. Mirra-Babaei, “Artificial players in the design process: Developing an automated testing tool for game level and world design,” in Annual Symposium on Computer-Human Interaction in Play (CHI Play), 2020.

[2] C. Politowski, Y.-G. Guéhéneuc, and F. Petrillo, “Towards automated video game testing: Still a long way to go,” arXiv preprint arXiv:2202.12777, 2022.

[3] L. Gisslén, A. Eakins, C. Gordilho, J. Bergdahl, and K. Tollmar, “Adversarial reinforcement learning for procedural content generation,” in Conference on Games (CoG), 2021.

[4] A. Sestini, L. Gisslén, J. Bergdahl, K. Tollmar, and A. D. Bagdanov, “Automated gameplay testing and validation with curiosity-conditioned proximal trajectories,” IEEE Transactions on Games, 2022.

[5] C. Gordilho, J. Bergdahl, K. Tollmar, and L. Gisslén, “Improving playtesting coverage via curiosity-driven reinforcement learning agents,” in Conference on Games (CoG), 2021.

[6] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Goyal, and T. Hester, “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis,” Machine Learning, vol. 110, no. 9, pp. 2419–2468, 2021.

[7] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in 2011 International Conference on Artificial Intelligence and Statistics (ICAIAS), 2011.

[8] L. Mugrai, F. Silva, C. Holmgård, and J. Togelius, “Automated playtesting of matching tile games,” in Conference on Games (CoG), 2019.

[9] S. Hukhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, “An automated model based testing approach for platform games,” in 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2015.

[10] G. Lovreto, A. T. Endo, P. Nardi, and V. H. Durelli, “Automated tests for mobile games: An experience report,” in 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), 2018.

[11] S. Stahlke, A. Nova, and P. Mirza-Babaei, “Artificial playfulness: A tool for automated agent-based playtesting,” in Conference on Human Factors in Computing Systems, 2019.

[12] C. Schaefer, H. Do, and B. M. Slator, “Crushinator: A framework towards game-independent testing,” in 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013.

[13] G. Xiao, F. Southei, R. C. Holte, and D. Wilkinson, “Software testing by active learning for commercial games,” in Conference on Artificial Intelligence (AAAI), 2005.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.

[15] M. Jacob, S. Devlin, and K. Hofmann, “It’s unwieldy and it takes a lot of time. Challenges and opportunities for creating agents in commercial games,” in 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2020.

[16] E. Alonso, M. Peter, D. Gouraud, and J. Romoff, “Deep reinforcement learning for navigation in AAA video games,” in in International Joint Conference on Artificial Intelligence (IJCAI), 2021.

[17] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, “Adversarial attacks on neural network policies,” arXiv preprint arXiv:1702.02284, 2017.

[18] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS), 2016.

[19] Y. Fu, K. Luo, and S. Levine, “Learning robust rewards with adverserial inverse reinforcement learning,” in International Conference on Learning Representations (ICLR), 2018.

[20] F. Torabi, G. Warnell, and P. Stone, “Behavioral cloning from observation,” in International Joint Conference on Artificial Intelligence (IJCAI), 2018.

[21] J. Monteiro, N. Gavenski, R. Granada, F. Meneguzzi, and R. Barros, “Augmented behavioral cloning from observation,” in 2020 International Joint Conference on Neural Networks (IJCNN), 2020.

[22] S. Sinha, A. Mandlekar, and A. Garg, “S4RL: Surprisingly simple self-supervision for offline reinforcement learning in robotics,” in Conference on Robot Learning (CORL), 2022.

[23] S. Fujimoto and S. S. Gu, “A minimalist approach to offline reinforcement learning,” Advances in Neural Information Processing Systems (NeurIPS), 2021.

[24] P. L. P. de Weillenmont, R. Labory, and V. Corballé, “Configurable agent with reward as input: a play-style continuum generation,” in Conference on Games (CoG), 2021.

[25] J. Roy, R. Girgis, J. Romoff, P.-L. Bacon, and C. Pal, “Direct behavior specification via constrained reinforcement learning,” arXiv preprint arXiv:2112.12228, 2021.

[26] A. Sestini, A. Kuhnle, and A. D. Bagdanov, “Policy fusion for adaptive and customizable reinforcement learning agents,” in Conference on Games (CoG), IEEE, 2021, pp. 01–08.

[27] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, “AMP: adversarial motion priors for stylized physics-based character control,” ACM Transactions on Graphics, vol. 40, no. 4, 2021.

[28] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” in International Conference on Learning Representations (ICLR), 2018.

[29] J. Druce, M. Harradon, and J. Tittle, “Explainable artificial intelligence (XAI) for increasing user trust in deep reinforcement learning driven autonomous systems,” arXiv preprint arXiv:2106.03775, 2021.

[30] G. Wallner and S. Kriegstein, “Visualization-based analysis of gameplay data—a review of literature,” Entertainment Computing, vol. 4, no. 3, pp. 143–155, 2013.

[31] Y. Duan, M. Andrychowicz, B. Stadie, H. Jonathan, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, “One-shot imitation learning,” Advances in Neural Information Processing Systems (NeurIPS), 2017.

[32] K. Hakkamaheshi, R. Zhao, A. Zhan, P. Abbeel, and M. Laskin, “Hierarchical few-shot imitation with skill transition models,” arXiv preprint arXiv:2107.08981, 2021.

[33] A. Kumar, A. Zhou, G. Tucker, and S. Levine, “Conservative Q-learning by active learning for commercial games,” in Conference on Games (CoG), 2020.

[34] G. Wallner and S. Krigstein, “Visualization-based analysis of gameplay data—a review of literature,” Entertainment Computing, vol. 4, no. 3, pp. 143–155, 2013.