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

Alessandro Sestini^{1,2}, Joakim Bergdahl^{1}, Konrad Tollmar^{1}, Andrew D. Bagdanov^{2}, and Linus Gisslén^{1}
^{1}SEED - Electronic Arts (EA), ^{2}Università degli Studi di Firenze
{jbergdahl, ktollmar, lgisslen}@ea.com
{alessandro.sestini, 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. This paper proposes a new approach to automated game validation. Our method leverages a data-driven imitation learning 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 investigate the validity of our approach through a user study with industry experts. The survey results show that ours is indeed a valid approach to game validation and that data-driven programming would be a useful aid to reducing effort and increasing quality of modern playtesting. The survey also highlights several open challenges. With the help of the most recent literature, we analyze the identified challenges and propose future research directions suitable for maximizing the utility of our approach. For a video compilation of the results please visit `https://go.ea.com/kiwi`.

1 Introduction

Digital gaming has evolved from a fringe pastime into a main-stream, multi-billion dollar industry reaching 2.7B players in 2020 and with a yearly revenue of \$159B [16] that dwarfs many other entertainment industries. Modern games are often enormous and grow exponentially in size, complexity, and asset count with every iteration. 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 turnaround time – the time between game design and 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 creators. 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 to mitigate the need for manual validation in large games. Automated gameplay testing offers a fast and relatively cheap solution for testing games at scale, and this is often done by crafting model-based bots [45]. Another proposed solution from recent research is to train self-learning agents with reinforcement learning [18, 42, 17]. However, both of these approaches suffer from several drawbacks. Model-based agents, typically bots with hand-crafted behaviors (i.e. achieved by scripting), require a certain level of domain knowledge and programming skills that game designers do not necessarily have. Moreover, the lack of generalization in this method might render the agents unusable if changes are made in the game environment. Reinforcement learning agents, on the other hand, can not only learn to play the game without scripting but can also be easily retrained when the environment changes. However, reinforcement learning is sample inefficient and arguably requires a high level of expertise in machine learning to be effectively used (e.g. for properly crafting a well performing reward function). Reinforcement learning in general provides a low level of controllability as the agents will try to exploit the environment regardless of the intentions of the designer [34, 26]. Additionally, making a game compliant with a reinforcement learning training setup is a considerable engineering effort potentially requiring intrusive changes to the game's source code.

In this position paper we propose a data-driven approach for creating agent behaviors for automated game validation (illustrated in Figure 1). Unlike methods where agents are manually programmed or scripted, our method uses imitation learning, in particular the DAgger algorithm [37], to clone the behavior of the user. With this, we can leverage developer expertise while at the same time not requiring additional 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 days or sometimes weeks for manual playtests. This study is made to either validate or refute above claims. The research questions we are focusing on are: 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 use our approach. Further, we report on results from survey that explores how our method satisfies the requirements previously mentioned and gauges the interest in such a tool by professional developers. Based on the answers from the survey, we propose future research directions for improving game validation with imitation learning.

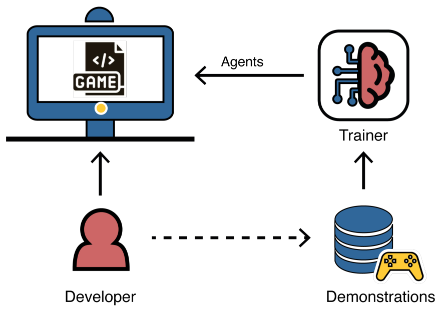


Figure 1: Our approach lets designers perform automated game validation directly during the design phase. The training is interactive as the algorithm allows designers to smoothly switch between designing the level, providing gameplay demonstrations, and getting feedback from the trained testing agents.

2 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 or random exploration [45, 23, 6]. Mugrai et al. [33] 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 model-based automated playtesting. Iftikhar et al. [25] proposed a model-based testing approach for automated, black-box functional testing of platform games, and Lovreto et al. [31] reported their experience with developing model-based automated tests for 16 mobile games. Other notable examples of model-based testing can be found in [44, 39, 8, 48]. However, when dealing with complex 3D environments these scripted or model-based techniques are not readily applicable due to the high-dimensional state space involved and poor generalization performance.

In parallel, many other works have used Reinforcement Learning (RL) to perform automated playtesting [36]. Zheng et al. [51] 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. [1] trained reinforcement learning agents to perform automated playtesting in 2D side-scrolling games, producing visualizations for level design analysis. Bergdahl et al. [4] proposed a study on the usefulness of using RL policies to playtest levels against model-based agents, while Gordillo et al. [18] used intrinsic motivation to train many agents to explore a 3D scenario with the aim of finding issues and oversights. Finally, Sestini et al. [42] 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. However, RL is known to be sample inefficient and leaves little control to the user over the final behavior of the policy [42, 34].

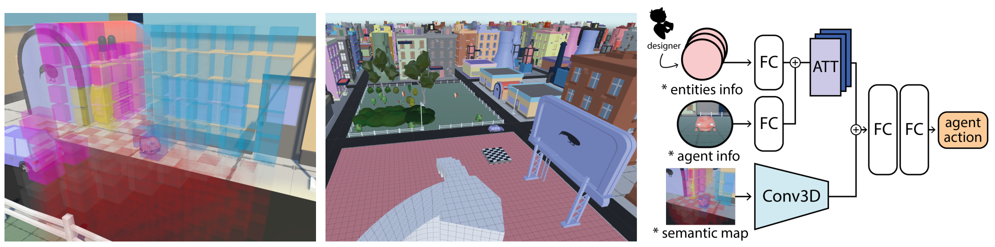


Figure 2: (a) Example of a semantic map used by the agent. Each cube describes the semantic integer value of the type of object at the corresponding position relative to the agent. (b) Overview of the environment used in this study. (c) Overview of the model architecture employed in this work.

Human-In-the-Loop Learning (HILL) refers to the practice of leveraging humans interactivity in the learning process of an RL agent [27]. This is often done through Imitation Learning (IL), that aims to distill a policy mimicking the behavior of an expert from a dataset of recorded demonstrations. Standard approaches are based on behavioral cloning that mainly use supervised learning [3, 28], while more advanced and computationally demanding techniques, like generative adversarial imitation learning, make use of adversarial training [22, 35, 14]. Most of the work done in the field of HILL focuses on creating more capable and sophisticated models using human interactivity [19, 49, 13]. However, 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 [6]. Zhao et al. [50] used an approach similar to behavioral cloning in which gameplay agents learn behavioral policies from the game designers. Harmer et al. [21] trained agents through a combination of IL and RL with multi-action policies for a first person shooter game. With IL we can leverage the designer's domain knowledge of the game, thus effectively adding HILL for controllability. Not only can we demonstrate the intended behavior, but also even correct it using new demonstrations with little additional training time. Compared to scripted behavior, IL provides the same level of generalization as RL. Compared to both RL and model-based bots, HILL calls for no or limited theory knowledge or programming skills as designers only have to demonstrate what they want the IL agent to do. With that said, we believe that IL is a better approach to game validation than the cited approaches.

3 Proposed Approach

Here we describe our approach to exploring the potential of data-driven methods for direct gameplay validation to designers.

Algorithm. We use an IL approach based on the DAgger algorithm [37], as shown in Figure 1. 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 the level, providing game demonstrations and getting feedback from trained testing agents. Our approach requires little training time and is more sample efficient than the baseline methods, as we see in Section 4. Designers can also provide corrections to faulty agent behaviors, resulting in a continuous and rapid feedback loop between designers and agents. Providing corrections is as easy as taking the controller back and playing the game one more time. Once they are satisfied with agent behavior, designers can stop providing demonstrations and watch the agent validate the game. Developers can then make any kinds of design changes and wait for agent feedback without recording any new 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 taking this into consideration. This approach, with minor observation tweaking, will transfer well to similar, but contextually 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 $\mathbb{R}^2$ 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. [42] which is general and performant. We illustrate an example of such a semantic 3D occupancy map used as input to the networks in Figure 2. 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 π_θ with parameters θ 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 producing the self-embedding $x_{\mathrm{a}} \in \mathbb{R}^d$, where d is the embedding size. The list of entities is passed through a separate linear layer with shared weights producing embeddings $x_{\mathrm{e}_i} \in \mathbb{R}^d$, one for each entity e_i in the list. Each of these embedding vectors is concatenated with the self-embedding, producing $x_{\mathrm{ae}_i} = [x_{\mathrm{a}}, x_{\mathrm{e}_i}]$, with $x_{\mathrm{ae}_i} \in \mathbb{R}^{2d}$. This list of vectors is then passed through a transformer encoder with 4 heads and final average pooling, producing a single vector $x_{\mathrm{t}} \in \mathbb{R}^{2d}$. In parallel, the semantic occupancy map $M \in \mathbb{R}^{s \times s \times s}$ is first fed into an embedding layer, transforming categorical representations into continuous ones, and then into a 3D convolutional network. The output of this convolutional network is a vector embedding $x_{\mathrm{M}} \in \mathbb{R}^d$ that is finally concatenated with x_{t} and passed through a feed forward network, producing an action probability distribution. The complete neural network architecture is shown in Figure 2(c).

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

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

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

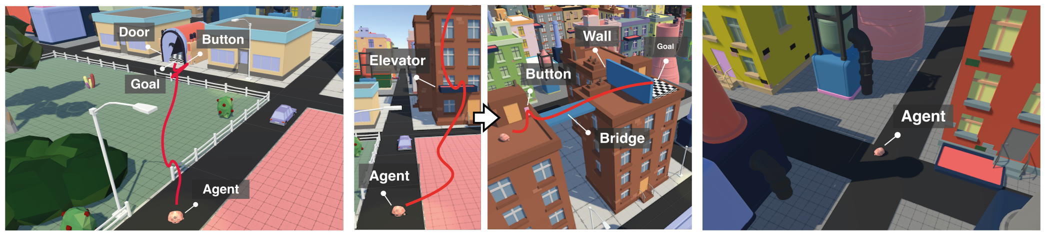
4 Experiments

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. In Table 1, we report quantitative results for each use case, comparing our approach to the RL baselines described below.

Environment. The environment used in this work is shown in Figure 2(b). It is a 3D navigation environment with procedurally generated elements. 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 some interactable objects located around the map.

Training Setup. We compare our approach to two main baselines: Base-RL and Tuned-RL, both trained using the PPO algorithm with identical hyperparameters [40]. 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. 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 between the agent trajectories and the demonstrator ones.

Use case 1: Validating Design Changes. For the first use case, we investigate wheter an IL agent can validate in real-time the layout of a level. The goal in this experiment is to train an agent to follow



(a) (b) (c)

Figure 3: (a) Screenshot of the "*Validating a Complex Trajectory*" use case. (b) Screenshots of the "*Navigation*" use case. (c) Screenshot of the "*Validating a Complex Trajectory*" use case.

human demonstrations, then to change the level and see how the agent adapts to these changes. This will give developers a glimpse into the usefulness of our proposed approach when they are still in the design process: they can see how players would adapt accordingly to modifications in the level layout. In Figure 3(a), we give the details of the use case. The agent is tasked to navigate to a goal hidden behind a door that can be opened by interaction with a button. After training the agent, we test it by removing and adding new elements to the level. We argue, with the support of Table 1, that IL is more suitable than RL for use cases like this. IL is generally significantly faster than RL, and at the same time it gives the user much more control over the final agent behavior as the agent can be guided via demonstrations. 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 very difficult task – especially for non-experts in machine learning [26]. Moreover, defining a suitable reward function requires many adjustment iterations to the training setup that decreases the overall efficiency of the system. It is evident from the table that IL gives the same level of generalization as RL and it adapts easily to slight variations of the same environment.

Use case 2: Validating Complex Trajectories. For this use case we train our agent to reproduce a complex trajectory as shown in Figure 3(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. Here we are not interested in generalization, but only in the ability to replicate the expert behavior as quickly and closely as possible. In Table 1, we see how our interactive approach is much more suitable for our goals compared to the baselines. As the table shows, it takes a lot of time for the RL agent to train, which is problematic as efficiency is a key requirement. Even the Tuned-RL baseline is much more time consuming than our approach. Moreover, the RL agent will exploit the environment without taking into account the real intention of the designer. An interesting finding here is that the RL agent has found a different way to get to the goal location, which is not desirable for this specific use case but would be very useful for exploit detection.

Use case 3: Navigation. Navigation is one of the 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 we thus must compromise between navigation cost and quality. This is achieved in practice by either removing some of the navigation abilities or by pruning navigation mesh connections [2]. 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 test city that changes every 2 seconds until it reaches the goal location. A screenshot of this use case is given in Figure 3(c). Similar to the second use case, we want the agent to always follow the same path that is demonstrated. Table 1 summarizes the results. Since the city environment is large, training agents with RL would take a lot of 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 explore the environment finding different ways to arrive at the goal location. The Tuned-RL baseline, however, achieves slightly better results in this case. We must reiterate that this baseline is infeasible to replicate by a designer as it needs a specific reward function and many iterations. Even then, the overall result is not very different from our approach. As shown by the results from these experiments, the combination of these elements is more easily achievable with IL rather than RL.

	Use case 1			
	Success ↑	Time ↓	Generalization ↑	Imitation ↓
Ours	$\mathbf{0.95 \pm 0.00}$	$\mathbf{0.02}$ h	$\mathbf{0.96 \pm 0.05}$	$\mathbf{6.84 \pm 0.34}$
Base-RL	0.91 ± 0.02	5.00 h	0.90 ± 0.00	8.37 ± 0.27
Tuned-RL	0.95 ± 0.00	4.48 h	0.90 ± 0.02	8.13 ± 0.20
	Use case 2			
	Success ↑	Time ↓	Generalization ↑	Imitation ↓
Ours	0.90 ± 0.02	$\mathbf{0.06}$ h	-	$\mathbf{7.73 \pm 1.10}$
Base-RL	0.00 ± 0.00	18.18 h	-	28.11 ± 0.41
Tuned-RL	$\mathbf{0.92 \pm 0.03}$	13.56 h	-	9.53 ± 1.37
	Use case 3			
	Success ↑	Time ↓	Generalization ↑	Imitation ↓
Ours	0.81 ± 0.08	$\mathbf{0.22}$ h	0.78 ± 0.03	46.07 ± 1.03
Base-RL	0.00 ± 0.00	16.49 h	0.00 ± 0.00	80.22 ± 0.53
Tuned-RL	$\mathbf{0.86 \pm 0.05}$	4.12 h	$\mathbf{0.87 \pm 0.03}$	$\mathbf{16.79 \pm 2.12}$

Table 1: Quantitative results of our experiments. All values refer to the mean and standard deviation over 5 training runs. For a complete description of the use cases and baselines, see Section 4.

	Mean	**Median**	**Std**
Imitation	4.56	4.00	1.63
Generalization	6.50	7.00	0.73
Exploration	6.81	7.00	0.40
Personas	5.87	6.00	1.31
Efficiency	5.50	5.50	1.21
Controllability	4.18	4.00	1.68
Feedback	6.00	6.00	0.96
Fine tuning	5.62	6.00	1.50
Interpretability	6.50	7.00	1.03

Table 2: Survey results from characteristics questions. Participants could assign a value between 1 and 7 to each of the categories.

5 User Study

We performed a qualitative user study in the form of an online survey [1] with professional game and level designers, not only to assess validity and desirability of using our proposed approach but also to identify open opportunities for improving automated game validation. The data for the survey were collected from participants recruited using snowball sampling. Subjects were recruited among different game studios of varying sizes, with different background knowledge and workflows. The survey is composed of Likert questions with some additional open questions. We also let participants provide additional feedback to individual questions.

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 our 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 our proposed 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.

6 Survey Results

We received a total of 16 survey responses. The majority (71.4%) 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. 83.3% of the respondents do not use automatic playtesting and they rely on external people or systems for their validation. 72.2% of designers never rely on automated playtesting and only 38.9% have used at least once a scripted automated method 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 feeling is that designers would very likely use a tool like this as *"it definitely would be useful to speed up the typically time consuming*

[1]A video describing what was shown in the survey is available at: `https://vimeo.com/754718818/011e28a122`.

iterative design process" and *"a method like this can definitely speed up iteration, which is one of the main things any designer spends a lot of time"*. 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*" and they think the examples shown are "*realistic for some games such as platformers*".

Participants acknowledge the usefulness and the potential of the demonstrated method, even if it would need the proper adaptations for handling the specific game genre they are working on. Further, 81.2% of the respondents agree that automated validation of the level directly in the game editor is useful and 93.0% of them think the demonstrated agents could be useful for assisting in validating their content. Many respondents (41.1%) agree that a method like this adds value compared to scripted agents and some of them (23.5%) think it could completely replace scripting.

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 *"[he is] wondering about how effective this would be in genres that are more complicated mechanically"*. This was expected due to the preliminary nature of this study and that we do not cover every genre but focus on a few. However, a larger group (81.2%) think 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 daily workflow: one argues that "*human players play very differently than bots, can be difficult to only rely on AI when it comes to testing*" 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*". 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*".

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 2 summarizes these characteristics ratings. We claim that these results bring much value to our research: not only do we know that our initial hypothesis is supported by professional designers, but we also what they really want and why they are skeptical of using such an approach. This is important because, as we will see in Section 7, the values in Table 2 form very precise research directions that we encourage all of the game research community to consider in future contributions to this field. We give complete details on the survey results in Section C of the Appendix.

7 Future Directions

Here we propose research opportunities discovered by interviewing experts in the field. With this, we want to lower the bar for researchers who would like to contribute, expand, and/or improve the proposed approach, ultimately driving the research, and therefore the industry, forward.

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. With our 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 [24]. Generalization as a subject in self-learning agents has already been addressed in the literature, but the state-of-the-art approaches are not readily applicable for our purpose: most of them use either interactions with the game [22, 14, 46] or learn the inverse dynamics of the environment [32]. A possible future direction is to try data augmentation techniques used in offline reinforcement learning algorithms such as the work done by Sinha et al. [43]. Offline reinforcement learning has several connections to behavioral cloning [15]. It learns a policy using a pre-defined dataset without direct interactions with the environment, but in contrast with IL that supposes the data comes from an optimal policy, offline reinforcement learning can also leverage sub-optimal examples. This leads to trained agents being more general and allows a higher exploration level. With all this considered, we argue that generalization is far from solved for this use case.

Personas. One recurring feature requested by survey participants is the possibility of training the model for different behavior types, or so called *personas*. 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 exclusively in RL contexts. Works from de Woillemont et al. [9], Roy et al. [38], and Sestini et al. [41] have explored how to combine behavior styles with different reward functions which is not applicable to IL. The work by Peng et al. [35] is an example of how to combine IL and RL to create different animation styles. More research into training for different playstyles with only IL would allow designers to train more meaningful and diverse testing agents for validating gameplay in different ways.

Exploration. Participants frequently brought up the exploration aspect, i.e. the ability of agent to look beyond expert demonstrations in search for overlooked issues. Exploration is a well known problem in the RL literature [5], 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, more or less, precisely follow the expert demonstrations as well as explore beyond the expert behavior. Moreover, we would not use exploration to improve agents in the validation context, but we would like to exploit exploration to find bugs. Sestini et al. [42] 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 active game design tool.

Usability. The usability aspect is one of the most important for 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 [10] 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 [47, 42]. Additionally, a sentiment found in the survey is that for a tool like this to be usable, there is a need to minimize the effort imposed on the end-user. To address the latter, few-shot IL is an active topic that can potentially help to reduce the number of samples required. Works by Duan et al. [11] and Hakhamaneshi et al. [20] specifically addressed this problem within IL, mainly exploiting meta-learning techniques [12]. 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 offline reinforcement learning techniques [29, 7, 43]. We can leverage the recording process to not only store demonstrations, but all other interactions the agent performs while testing the level.

Multitiple 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. Most of the current research in IL focuses on single agents learning from a single teacher. There are a few examples of multi-agent IL. Harmer et al. [21] used IL in a multi-agent game, but they do not really address the problem of a multi-agent system, while Le et al. [30] 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, and we encourage researchers to follow this research direction.

8 Conclusion

In this position paper we claim that data-driven programming via imitation learning is a suitable approach for real-time validation of game and level design. We propose an imitation learning approach and we 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 model-based scripted behaviors. We also performed a user study with professional game and level designers from diverse game studios and game genres. We asked participants to assess the desirability and improvement opportunities of using such an approach in their daily workflow. Moreover, we asked designers 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 have an automated way to test and validate their games. Along with our preliminary results we demonstrate that the data-driven approach we proposed is a potential candidate for achieving such objectives. The study also highlights challenges and the gap that exists between techniques from the literature and their actual use in the game industry. For this reason, we propose a series of research directions that will help such approaches move beyond an impractical tool to an effective game design tool. We hope that our recommendations will encourage the game research community to contribute to or to expand on this research.

References

[1] Shivam Agarwal, Christian Herrmann, Günter Wallner, and Fabian Beck. Visualizing AI playtesting data of 2D side-scrolling games. In *2020 IEEE Conference on Games (CoG)*, 2020.

[2] Eloi Alonso, Maxim Peter, David Goumard, and Joshua Romoff. Deep reinforcement learning for navigation in AAA video games. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI-21)*, 2021.

[3] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

[4] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. Augmenting automated game testing with deep reinforcement learning. In *IEEE Conference on Games (CoG)*, 2020.

[5] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*, 2018.

[6] Kenneth Chang, Batu Aytemiz, and Adam M Smith. Reveal-more: Amplifying human effort in quality assurance testing using automated exploration. In *2019 IEEE Conference on Games (CoG)*, 2019.

[7] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[8] Chang-Sik Cho, Kang-Min Sohn, Chang-Jun Park, and Ji-Hoon Kang. Online game testing using scenario-based control of massive virtual users. In *2010 The 12th International Conference on Advanced Communication Technology (ICACT)*, 2010.

[9] Pierre Le Pelletier de Woillemont, Rémi Labory, and Vincent Corruble. Configurable agent with reward as input: a play-style continuum generation. In *2021 IEEE Conference on Games (CoG)*, 2021.

[10] Jeff Druce, Michael Harradon, and James Tittle. Explainable artificial intelligence (XAI) for increasing user trust in deep reinforcement learning driven autonomous systems. *arXiv preprint arXiv:2106.03775*, 2021.

[11] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Ho Jonathan, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

[13] Julius Frost, Olivia Watkins, Eric Weiner, Pieter Abbeel, Trevor Darrell, Bryan Plummer, and Kate Saenko. Explaining reinforcement learning policies through counterfactual trajectories. *arXiv preprint arXiv:2201.12462*, 2022.

[14] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[15] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[16] N Gilbert. Number of gamers worldwide 2021/2022: Demographics, statistics, and predictions, 2020.

[17] Linus Gisslén, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, and Konrad Tollmar. Adversarial reinforcement learning for procedural content generation. In *2021 IEEE Conference on Games (CoG)*, 2021.

[18] Camilo Gordillo, Joakim Bergdahl, Konrad Tollmar, and Linus Gisslén. Improving playtesting coverage via curiosity driven reinforcement learning agents. In *2021 IEEE Conference on Games (CoG)*, 2021.

[19] Lin Guan, Mudit Verma, Sihang Guo, Ruohan Zhang, and Subbarao Kambhampati. Explanation augmented feedback in human-in-the-loop reinforcement learning. *arXiv preprint arXiv:2006.14804*, 2020.

[20] Kourosh Hakhamaneshi, Ruihan Zhao, Albert Zhan, Pieter Abbeel, and Michael Laskin. Hierarchical few-shot imitation with skill transition models. *arXiv preprint arXiv:2107.08981*, 2021.

[21] Jack Harmer, Linus Gisslén, Jorge del Val, Henrik Holst, Joakim Bergdahl, Tom Olsson, Kristoffer Sjöö, and Magnus Nordin. Imitation learning with concurrent actions in 3D games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 2018.

[22] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Proceedings of the 30st International Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[23] Christoffer Holmgård, Michael Cerny Green, Antonios Liapis, and Julian Togelius. Automated playtesting with procedural personas through MCTS with evolved heuristics. *IEEE Transactions on Games*, 11(4): 352–362, 2018.

[24] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[25] Sidra Iftikhar, Muhammad Zohaib Iqbal, Muhammad Uzair Khan, and Wardah 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.

[26] Mikhail Jacob, Sam Devlin, and Katja 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.

[27] Jakob Karalus and Felix Lindner. Accelerating the convergence of human-in-the-loop reinforcement learning with counterfactual explanations. *arXiv preprint arXiv:2108.01358*, 2021.

[28] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: the TAMER framework. In *Proceedings of the 5th international conference on Knowledge capture*, 2009.

[29] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[30] Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated multi-agent imitation learning. In *International Conference on Machine Learning (ICML)*, 2017.

[31] Gabriel Lovreto, Andre T Endo, Paulo Nardi, and Vinicius HS Durelli. Automated tests for mobile games: An experience report. In *17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2018.

[32] Juarez Monteiro, Nathan Gavenski, Roger Granada, Felipe Meneguzzi, and Rodrigo Barros. Augmented behavioral cloning from observation. In *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020.

[33] Luvneesh Mugrai, Fernando Silva, Christoffer Holmgård, and Julian Togelius. Automated playtesting of matching tile games. In *2019 IEEE Conference on Games (CoG)*, 2019.

[34] Open AI. Faulty reward functions in the wild, 2016. URL `http://example.com/`.

[35] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. AMP: adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics*, 40(4), 2021.

[36] Cristiano Politowski, Yann-Gaël Guéhéneuc, and Fabio Petrillo. Towards automated video game testing: Still a long way to go. *arXiv preprint arXiv:2202.12777*, 2022.

[37] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *2011 International Conference on Artificial Intelligence and Statistics (ICAIS)*, 2011.

[38] Julien Roy, Roger Girgis, Joshua Romoff, Pierre-Luc Bacon, and Christopher Pal. Direct behavior specification via constrained reinforcement learning. *arXiv preprint arXiv:2112.12228*, 2021.

[39] Christopher Schaefer, Hyunsook Do, and Brian M Slator. Crushinator: A framework towards game-independent testing. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[41] Alessandro Sestini, Alexander Kuhnle, and Andrew D Bagdanov. Policy fusion for adaptive and customizable reinforcement learning agents. In *2021 IEEE Conference on Games (CoG)*, pages 01–08. IEEE, 2021.

[42] Alessandro Sestini, Linus Gisslén, Joakim Bergdahl, Konrad Tollmar, and Andrew D Bagdanov. CCPT: Automatic gameplay testing and validation with curiosity-conditioned proximal trajectories. *arXiv preprint arXiv:2202.10057*, 2022.

[43] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. S4RL: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning (CORL)*, 2022.

[44] Samantha Stahlke, Atiya Nova, and Pejman Mirza-Babaei. Artificial playfulness: A tool for automated agent-based playtesting. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019.

[45] Samantha Stahlke, Atiya Nova, and Pejman Mirza-Babaei. Artificial players in the design process: Developing an automated testing tool for game level and world design. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play (CHI Play)*, 2020.

[46] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[47] Günter Wallner and Simone Kriglstein. Visualization-based analysis of gameplay data–a review of literature. *Entertainment Computing*, 4(3):143–155, 2013.

[48] Gang Xiao, Finnegan Southey, Robert C Holte, and Dana Wilkinson. Software testing by active learning for commercial games. In *Conference on Artificial Intelligence (AAAI)*, 2005.

[49] Haoyu Xiong, Quanzhou Li, Yun-Chun Chen, Homanga Bharadhwaj, Samarth Sinha, and Animesh Garg. Learning by watching: Physical imitation of manipulation skills from human videos. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[50] Yunqi Zhao, Igor Borovikov, Fernando De Mesentier Silva, Ahmad Beirami, Jason Rupert, Caedmon Somers, Jesse Harder, John Kolen, Jervis Pinto, Reza Pourabolghasem, et al. Winning isn't everything: Enhancing game development with intelligent agents. *IEEE Transactions on Games*, 2020.

[51] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *34th IEEE/ACM International Conference on Automated Software Engineering*, 2019.

A Method Comparison

In this section we better delineate the different common approaches to automated testing and justify why we decide to use Imitation Learning (IL).

Reinforcement Learning. Reinforcement Learning (RL) for game validation is known to be sample inefficient. For example, the work proposed by Gordillo et al. [18] required an average of two days of training before thorough coverage of a game scene is achieved. Even if sample inefficiency can be mitigated by combining RL with IL similar to the curiosity-conditioned proximal trajectory algorithm [42], it is still challenging to practically applyt RL in production. Moreover, RL typically grants little control to the user over the final behavior of the policy as the trained agents will inherently exploit the reward function regardless designer intention [34]. We can alleviate this problem with reward shaping, but this requires considerable knowledge of RL and machine learning in general, making it impractical for the general user. Even given these problems, RL still has some advantages. If designers prefer exploitation and exploration over imitation, then RL is preferable; and similar to IL, it does not require any programming knowledge to train.

Scripted Bots. By scripted bots we mean bots modelled with programming methods, i.e. the standard way of creating game AI bots. To program autonomous bots, designers must have relatively high domain knowledge of the level design, but also scripting skills to successfully craft the behaviors of the bots. Moreover, even if they have very fine control over the final behavior of the bots, the scripted agents will quickly become sub-optimal, and even unusable, when faced with design changes in the environment. The setup time also greatly depends on the complexity of the game and game scene. For the reasons listed above, we argue that this approach does not satisfy the requirements listed in Section 1 of the main paper.

Imitation learning (IL). IL is more sample-efficient than RL and allows training of agents in minutes or only few hours [42]. With this method we only need to leverage the designer's domain knowledge of the game, thus effectively adding humans-in-the-loop for superior controllability. Not only can we demonstrate the intended behavior, but we can even correct it using new demonstrations with little additional training time. Compared to scripted behavior, IL provides the same level of generalization as RL as the policy model should be able to generalize to some extent to unseen observations. However, compared to both RL and model-based bots, IL calls for no or limited knowledge of theory or programming skills as designers only have to demonstrate what they want the IL agent to do. With IL we can mitigate some of the drawbacks of RL-trained and scripted agents.

B Training Details

To optimize the policy network we used the DAgger algorithm [37]. The number of demonstrations used for training the policy depends on the use case: we started with an average of 4 episodes for the first iteration of training, then we correct the behavior of the agent with all the necessary demonstrations. Most of the hyperparameter values were chosen after many preliminary experiments made with different configuration: learning rate $\mathrm{lr} = 0.0005$, batch size $\mathrm{bs} = 512$, and number of epochs $\mathrm{e}_0 = 300$ for the first DAgger iteration, and $\mathrm{e}_i = 100$ for the subsequent ones. We found this set of values to work well for all use cases. All training was performed on a single machine with an NVIDIA GTX 1080 GPU with 8 GB VRAM, a 6-core Intel Xeon W-2133 CPU, and 64 GB of RAM.

C Survey Results

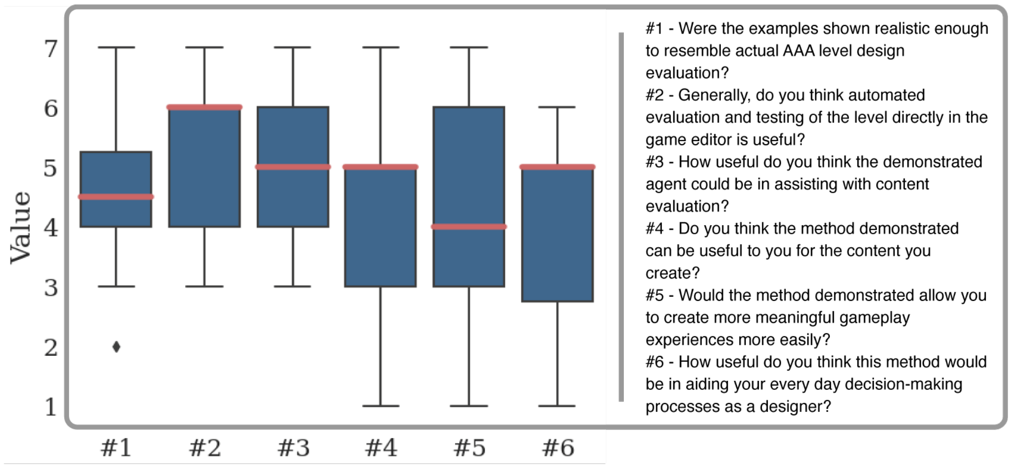
In this section we give additional details about the results obtained through our user study with professional industry experts. In Table 3 we report the details of all survey participants. In Table 4 we report all the questions in the user survey. In Figure 4 and Figure 5 we report some of the most interesting results obtained.

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	RPG	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

Table 3: Summary of participants. Abbreviations: YoE (Years of Experience), MLK (Machine Learning Knowledge), FPS (First Person Shooter), RPG (Role-Playing Game).

N°	Question	Type
1	What is your area of expertise?	Open
2	What game or franchise do you work on?	Open
3	What is your level of machine learning knowledge?	Multi
4	How many years of experience do you have in the video game industry?	Open
5	What tools/software do you most commonly use in your everyday work?	Open
6	Is any part of the level design evaluation process automatic in your current workflow?	[Yes, No]
7	Have you used scripted automated method in your daily work?	[Yes, No]
8	How often do you rely on automated rather than manual playtesting?	Multi
9	Were the examples shown realistic enough to resemble AAA level design evaluation?	[1, 7]
10	Generally, do you think automated evaluation and testing of the level directly in the game editor is useful?	[1, 7]
11	How useful do you think the demonstrated agent could be in assisting with content evaluation?	[1, 7]
12	What are the important characteristics that an agent for automated content evaluation should have? Please rank the options below	-
12*	Imitation (the agent can exactly replicate the demonstrations)	[1, 7]
12*	Generalization (the agent can adapt to different variations of the same situation)	[1, 7]
12*	Exploration (the agent can explore beyond the demonstrations and find bugs and issues)	[1, 7]
12*	Personas (the agent can have different types of behaviors)	[1, 7]
12*	Efficiency (the agent must use as few demonstrations as possible)	[1, 7]
12*	Controllability (having control over the agent behavior vs complete autonomy)	[1, 7]
12*	Feedback (the agent gives feedback when the amount of demonstration data is enough)	[1, 7]
12*	Fine-Tuning (behaviors can be fine tuned after initial training)	[1, 7]
12*	Interpretability (the agent can inform when and why it fails)	[1, 7]
13	Any other important characteristics that come to mind?	Open
14	Do you think the method demonstrated can be useful to you for the content you create?	[1, 7]
15	Would the method demonstrated allow you to create more meaningful gameplay experiences more easily?	[1, 7]
16	What are the potential problems you see in using a method like this?	Open
18	How does the demonstrated solution compare with scripted agents?	Multi
19	How useful do you think this method would be in aiding your everyday decision-making processes as designer?	[1, 7]
20	Do you see other use cases for this method?	Open
21	What other features would you like to see in a method like this?	Open
21	Do you have any other feedback, comment or idea for us?	Open

Table 4: All the questions presented to industry experts through the survey. The *Type* column refers to the type of answer allowed for each question: *Open* allowed participants to answer with a free-text response; *Multi* allowed participants to choose from list of answers; *[Yes, No]* allowd participants to answer either yes or no; and "*[1, 7]*" allowed participants to answer with a value between 1 and 7, where 1 generally means "*totally disagree*" and 7 means "*totally agree*".



#1 - Were the examples shown realistic enough to resemble actual AAA level design evaluation?
#2 - Generally, do you think automated evaluation and testing of the level directly in the game editor is useful?
#3 - How useful do you think the demonstrated agent could be in assisting with content evaluation?
#4 - Do you think the method demonstrated can be useful to you for the content you create?
#5 - Would the method demonstrated allow you to create more meaningful gameplay experiences more easily?
#6 - How useful do you think this method would be in aiding your every day decision-making processes as a designer?

Figure 4: Boxplot of answers to some important questions from the survey. The boxes represent the area between the lower and upper quartiles. The red lines represent the median of the values, and the whiskers represent the minimum and maximum of the distributions (except for the outliers that are visualized with black dots).

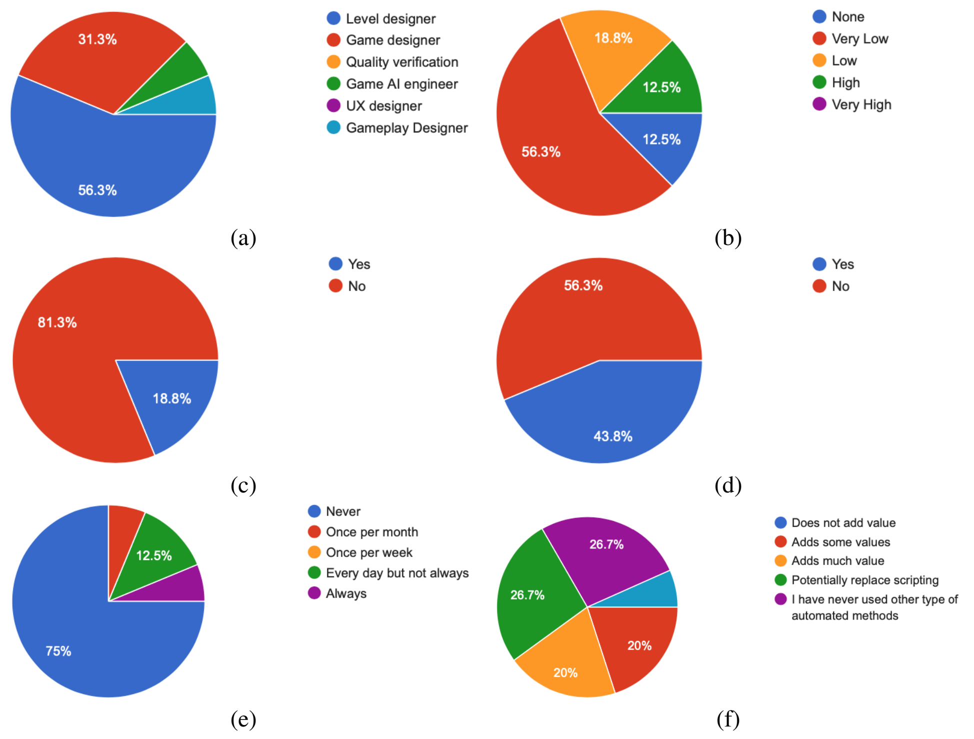


Figure 5: Pie charts of answers to some of the most important questions in the survey. (a) The question "*what is your area of expertise?*"; (b) The question "*what is your machine learning knowledge?*"; (c) The question "*is any part of the level design evaluation process automatic in your current workflow?*"; (d) The question "*have you used scripted automated method in your daily work?*"; (e) The question "*how often do you rely in automated rather than manual playtesting?*"; and (f) The question "*how does the demonstrated solution compare with scripted agents?*".