Proceedings of the Seventeenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2021)

Reinforcement Learning Methods to Evaluate the Impact of AI Changes in Game Design

Pablo Gutiérrez-Sánchez,[1] **Marco A. Gómez-Martín,**[2]
Pedro A. González-Calero,[2] **Pedro P. Gómez-Martín**[2]
[1]PadaOne Games, Calle Profesor Jose Garcia Santesmases, 9, 28040, Madrid, Spain
[2]Complutense University of Madrid, Madrid, Spain,
pablo.gutierrez@padaonegames.com, marcoa@fdi.ucm.es, pagoncal@ucm.es, pedrop@fdi.ucm.es

Abstract

Game development has become a long process that requires many professionals working on a project during several months or years. With this scenario the re-utilization of resources is crucial not only to alleviate the process but also to bring coherence into the final product. In this paper we focus on the reuse of NPCs and the problems it brings about. In particular it is common to have different breeds (or personalities) of NPCs that are placed on different levels on the game. The problem arises when their behaviors are fine-tuned to accommodate a specific level needs without taking into consideration that this change may alter their performance on previous already-tested levels. The paper presents the application of reinforcement learning together with behavior trees to automatically test if modifications to the AIs of a stealth game have an impact on the user experience. Our experiments reveal that this approach provides a way of diagnosing alterations in level gameplay that correspond to the effects observed by human testers.

Introduction

Quality control in modern video games can be a major challenge. Nowadays it is not only necessary to keep a strict and continuous control of technical failures or bugs that may arise during the development process, but also of new problems derived from unexpected changes in the playability of the different parts of the game. These changes can bring about various adverse effects such as preventing players from being able to complete previously solvable sections, altering the navigability of menus and environments, or modifying the difficulty perceived by the user, in dissonance with the experience originally conceived by the designers.

Quality assurance (QA) tasks usually involve an immense testing effort in which developers and players strive to detect and solve these problems. In light of this situation, in the last few years several research works have emerged with the aim of proposing strategies to improve the QA process in video games and reduce its cost. Many of these methods are aimed primarily at developing agents (usually by using deep reinforcement learning, DRL) that act as synthetic players capable of automating checks that may otherwise require numerous hours of manual testing, in an attempt to redirect the efforts of human testers towards less mechanical and more creative tasks.

In this paper we focus on automatic regression tests for detecting issues when modifications are made to AIs that are reused in multiple sections of the game. These changes may be done when fine-tuning the behavior of a NPC in a level without thinking about the implications of those small variations on other levels where the NPC was placed before.

Changes as simple as slightly modifying an enemy's movement speed could end up unexpectedly impacting how the player interacts with the game's levels. These changes need not be particularly drastic (such as as sudden violations of the completability of a level), and may boil down to the player taking more or less time to complete a part of the game, or exploiting a new way to beat a level that was not originally contemplated. All of these design alterations should not go unnoticed, but many of them are not usually straightforward to detect in typical testing environments.

In this paper we propose the application of different reinforcement learning methods to produce testing agents capable of interacting with a set of levels in a stealth game while collecting interaction statistics representative of the perceived gameplay in each level. The agents are afterwards used to test whether a change in the specification of an enemy type common to all levels induces significant changes in gameplay parameters collected by the agents before applying the modification.

Related Work

By testing we refer to the activity undertaken to evaluate the quality of a product and improve it by identifying its defects and problems (Abran 2004). Video games are complex software systems that must function correctly on different platforms with a range of configurations. The video game market is a very competitive one with buyers expecting increasingly more from them, which makes it unacceptable to release applications that are not robust or suffer from bugs. The robustness of a video game covers a wide spectrum of criteria, from the correct functioning of technical aspects such as performance or functional correctness to attributes such as the aesthetic soundness of the application. The validation of these criteria is a costly task in which a substantial part of the development effort of a project is invested, hence numerous strategies have been proposed in recent years in

an attempt to automate these tasks or reduce their associated workload. One of the simplest alternatives is the use of game segments recorded manually by human testers, which are subsequently used to check that the replayed sequences are still capable of completing the established objective (Ostrowski and Aroudj 2013). However, when the structure or the game environment is modified, the tests generated by these methods are no longer valid, and it is necessary to once again resort to human testers to re-record new sequences for the modified scenarios. This continuous obsolescence naturally leads to the proposal of alternatives that are capable of adapting dynamically to variations in the game environments, giving rise to the use of AI-based agents for testing.

In (Hernández Bécares, Costero Valero, and Gómez Martín 2017), an AI played following the specification of a game given by a Petri net, making use of high-level actions, but required precise modeling of the level logic as well as manual implementation of the player's actions, whereas the techniques described in this paper do not require such an accurate understanding of the underlying game mechanics in order to be applied to a set of levels (being RL-based, our approach merely relies on the design of reward functions specifying how good an action in a given state). DRL and IL techniques used in (Pfau, Smeddinck, and Malaka 2017; Ariyurek, Betin-Can, and Surer 2021; Bergdahl et al. 2020) show promising results, but most of these efforts are generally focused on detecting technical errors, or verifying whether an automated agent succeeds in completing given testing objectives within certain acceptable margins (typically, the designer establishes an interval in which a set of parameters should lie and makes use of artificial players to play the game repeatedly while recording these parameters' metrics and checking that they are contained in those intervals), with few references to the detection of subtle gameplay modifications that may undermine game design plans. Our approach uses this methodology as a reference point, but expands it with the inclusion of hybrid BT-RL game-playing agents and statistical tests to evaluate the significance of a change in gameplay when altering a level, rather than just checking if the parameters are still within acceptable ranges.

Moreover, these methods are not always trivial to implement, often requiring a potentially daunting process of trial and error in the choice of training algorithm, reward allocation policy, model features, or the hyperparameters of the underlying neural networks. Fortunately, over the last few years, libraries for popular game engines have been appearing that greatly facilitate this process, with ML-Agents (Juliani et al. 2020) in the Unity 3D engine (Unity Technologies 2021a) being possibly one of the most well-known and actively maintained. Additionally, there has been work integrating reinforcement learning into hand-scripted control structures such as Behavior Trees (BTs) with the goal of narrowing learning problems to more controlled situations (Pereira and Engel 2015), thus reducing the time and effort needed to train agents. Taking ideas from these works, our experiments aim to make use of these advantages in the context of regression testing in game levels.

In (Holmgard et al. 2019) the use of procedural personas for level playtesting characterized by different utility functions employing a variant of the Monte Carlo Tree Search (MCTS) is proposed. These enable the modeling of decision-making processes of players with different goals, play styles and personal preferences in simple scenarios such as the levels in the 2D dungeon crawler used for evaluation. However, the specification of the utility functions to be used is left to the designer, and the method's suitability for more complex environments requires further confirmation. The idea is nonetheless relevant in this context, since reward functions can be roughly thought of as utility functions in RL, and these ideas combined with hybrid BTs to derive complex gameplay styles for the agents.

Lastly, there exist several works focused on facilitating the task of designing and adjusting the parameters of video game contents, both in the field of playtesting and in level balancing and design. (Gonzalez-Duque et al. 2020) proposes a method to obtain levels adjusted to a target difficulty based on the perception of an AI playing agent of maps generated by a trial-and-error algorithm. (Shaker, Yannakakis, and Togelius 2010) describes a procedure for creating custom levels for platform games using preference learning based on surveys administered to players after completing each level, while (Cook et al. 2021) introduces a tool to aid in the creation and understanding of procedural content generators with an emphasis on the analysis of their expressive range. These ideas can be used in parallel with our approach in order to evaluate how the changes they induce on levels actually affect gameplay perception.

Preliminaries

The following subsections introduce preliminary topics used in this work—RL, BTs, and RL-nodes in BTs.

Reinforcement Learning

Reinforcement learning (RL) is a process where an agent learns by trial and error from experience by interacting with its environment. In an RL problem, a distinction must be made between the agent, which interacts with its environment through actions, and the environment itself, which provides feedback and "rewards", or positive/negative reinforcement to the agent. RL algorithms usually take the formulation and formalism of Markov decision processes (MDPs) as a starting point. If we consider $s_t \in S$ as the state of the system at a given instant t, and $a_t \in A_{s_t}$ the action that the agent executes at that instant, where A_{s_t} is a possibly infinite set of admissible actions for the agent under state s_t, the agent's goal will be to learn a mapping between states and actions $\pi : S \to A = \cup_{s \in S} A_s$, that maximizes the expected long-run total reward from each state s:

$$G_t = E[\sum_{i=0}^{\infty} \gamma^i R_i] \tag{1}$$

where $\gamma \in (0, 1)$ is a discount factor that gives more weight to rewards earned in the short term than to those collected in a distant horizon, and R_i is the reward the agent receives at the i-th instant.

Behavior Trees

A Behavior Tree (BT) is a way of structuring policies or controllers in autonomous agents, such as robots or non playable characters in a video game (NPCs). In essence, a BT can be defined as a rooted tree in which the leaves correspond to execution nodes, associated to a specific task or condition check of the agent to be controlled, and the intermediate nodes take on the role of flow control nodes. BTs were initially conceived in the field of video game programming, with the first journal paper on BTs appearing in (Florez-Puga et al. 2009), and since then their use was progressively extended to the field of robotics. These constructs were originally developed with the motivation of promoting the production of modular, reusable, scalable, and easily understandable code when designing the behaviors of intelligent agents in the video game industry, thus trying to solve the problems commonly encountered with other classical mechanisms previously used, such as finite state machines (FSMs) or scripts.

RL Nodes in BTs

The above concepts can be combined giving rise to BTs with nodes supported by RL. The current literature contemplates two main types of RL nodes, depending on whether the learning is performed on a control or execution node.

- A *RL-control* node can be viewed as an extension of the usual control node, usually of type fallback (Pereira and Engel 2015), with the particularity that its children come to constitute the action space of a RL algorithm. Thus, given a state s, the node reorders its children according to the expected rewards of executing each of them (these estimates being learned during training).
- A *RL-execution* node contains a RL-learning algorithm with states, actions and rewards defined by the designer. The idea here is to employ the hierarchical structure of a BT to learn relatively small actions that are only executed under controlled conditions.

In any of the above cases, the main goal of combining BTs with RL is the implementation of actions and flows that are complex to program manually, but in specific regions of the agent's BT whose limited scope allows to dampen the so-called "curse of dimensionality" in learning sub-problems.

Methodology

In this paper we focus on the development of a preliminary automated testing methodology based on previously presented agents for a specific game environment that will be covered in detail in the experiment part of the paper.

This methodology starts from a specific level of our game, which we consider finished from a design and/or development point of view. For this level, we wish to monitor certain gameplay parameters in order to detect any relevant changes induced by a modification (at the global scope of the game) on the enemy AI as soon as possible, for instance by tuning its parameters to meet the requirements of a later level. To carry out this tracking, we train a set of "automatic players" using the techniques already described so that they learn to play that level "like humans". The training process culminates in the generation of benchmark statistics on the parameters of interest after having each agent play at the level of interest for a large number of times.

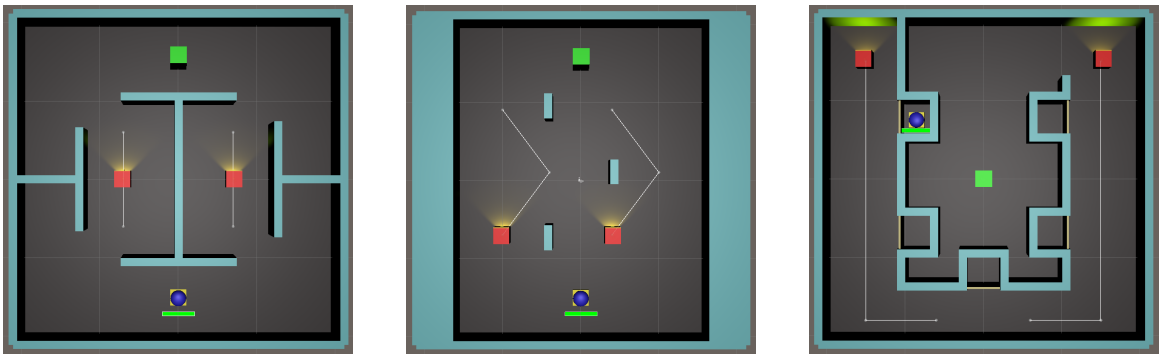
Once we have these references, it is possible to automatically perform a new round of gameplay and statistics compilation on the levels already developed to evaluate the possible impacts of the day's changes on their playability. With these statistics, we can verify if our agents are completing the levels with results equivalent to those recorded on the previous day. If this is statistically not the case, chances are that something has occurred, and we can issue a warning so that a member of the team can investigate the origin of this "alarm".

Experiment

In this section, we will focus on applying the methodology described above to the development of automated tests for a simple demo game that includes some of the most common mechanics found in genres such as stealth games, in which the player navigates through levels patrolled by enemies that must be avoided while actively trying to attain a certain goal (often reaching a given point in the level). The rules of the designed game can be summarized in the following points:

1. The player is represented by a blue circular figure, with a continuous action space given by $A_t = \{(x, z), x, z \in [-1, 1]\}$, which determines the possible movement directions at each instant. The player has a fixed maximum velocity v^p, such that the velocity at each instant is given by $v_t^p = v^p \frac{(x_t, y_t)}{||(x_t, y_t)||}$.
2. The player starts each level placed on a yellow platform, to which they return after being defeated by an enemy. The victory condition is to reach the green platform located at a fixed point in the level. A player is defeated when their health points (HP) are reduced to 0, these being lowered by 1 each time the player is hit by an enemy bullet and starting at 10 HP initially. Some sample levels can be found in Fig. 1.
3. The enemies are shaped like red cubes and have an associated "patrol" path (white lines in the figures), which they cycle through until they detect the player in their range of vision, after which they proceed to chase and shoot at their target as long as it is alive or remains within detection range.
4. Some levels contain yellow walls that block the enemies' range of vision, but have no effect on the player's movement or visibility (effectively creating hiding areas). Fig. 1c contains an example of this mechanic.

This demo game was developed in the Unity 3D engine, with tests implemented using Unity's Test Framework (Unity Technologies 2021b), all of them being Play Mode tests on the different scenes to be verified. The training of the RL agents used in the tests was carried out using Unity's ML-Agents library, while the BTs of the enemies and the hybrid agents for the player were implemented with the help of the Behavior Bricks plugin for the engine (PadaOne Games 2021).



(a) *Enemy Avoidance* level (b) *Enemy Timing* level (c) *Hiding Areas* level

Figure 1: Example levels

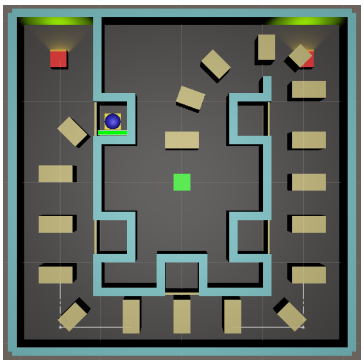


Figure 2: Artificial reward regions in the *Hiding Areas* level.

In the following subsections we describe the two training methodologies employed on the previously introduced demo game: first a method based on agents trained purely by RL, and then a variant employing RL-execution nodes on hybrid BTs in an attempt to simplify the problem and provide a different family of policies.

Pure RL Agents

We have already defined the action space A of our agent, but we still need to specify on the basis of what information these actions should be taken. We consider a total of 10 input features, distributed as follows:

- **Target location** (2 features). Components (x, z) of the distance vector between the player and the target platform, scaled to lay in the $[-1, 1]$ range.
- **Relative location of the enemies** (4 features per enemy). Components (x, z) of the distance vector between the player and each of the enemies in the level as well as the components (x, z) of the normal unit vector of the enemy's front face. The latter data provides information about the orientation of the enemy at any given time.

In addition to these features, we incorporate a total of 12 spatial sensors (*raycasts*) that allow us to identify objects close to the agent in radial directions. Here we consider the objects "goal", "enemy" and "wall".

Reward allocation conditions exhibit a strong heuristic component aimed at inducing the desired behavior by means of frequent stimuli, and can be summarized as follows:

- Positive rewards: (+2 units) for reaching the goal and finishing the level, and (+0.005 units) each time the agent manages to reduce its minimum distance to the goal.
- Negative rewards: (-2 units) for dying after being hit by an enemy bullet and finishing the level, (-0.25 units) for being hit by an enemy bullet, and (-0.1 units) every time the agent is detected by an enemy (at the instant it enters an enemy's vision range). Additionally, we add an existential negative reward (-0.001 units) for each action taken by the agent. This penalty is included to encourage agile flows and spur the agent to finish the episode as soon as possible to avoid accumulating too much negative reward (-5 units at the end of step 5000).

In some of the more complex levels, it can also prove helpful to incorporate artificial reward regions in the level that provide a modest stimulus (between +0.1 and +0.5 units) after being traversed for the first time, in order to encourage the agent to steer its trajectory through these regions. An example of this method to train an agent in the *Hiding Areas* level can be found in Fig. 2. We set a maximum training period of 750,000 agent steps per level with a maximum of 5,000 steps per training episode. Here we make use of PPO as our training algorithm of choice, using a neural network of two layers with 256 hidden units and a constant learning rate of 0.005.

Hybrid BT - RL Agents

In addition to agents trained purely by RL, we consider a second class of agents given by a hybrid model between manually programmed behaviors and flows learned by RL in the context of a BT. The idea here is to define a BT with a selector that alternates between a base flow ("navigate to goal" in our case) and learned actions within a RL node that is triggered upon entering a danger state, i.e. when an enemy enters the agent's range of vision ("evade enemy"). In this way it is possible to set the general goal, easily implementable, within a level-agnostic action node and let the RL algorithm handle the enemies under controlled conditions.

The RL node in this case considers the same features for the states s_t as those described in the previous subsection, but we modify the reward assignments to enforce the agent's expected goal in this context (avoid taking damage from enemy gunfire) with a negative reward (-1 unit) each time the agent is hit by an enemy bullet, while keeping the previous stimuli of +2 units for completing the level or -2 units for dying, respectively.

For the training of these agents we set again a limit of 750,000 training steps per level with 5,000 maximum steps per episode, and the same neural network architecture and hyperparameters as in the pure reinforcement learning case. However, unlike in the non hybrid case, the greater detachment of the trained behavior from the structure of the level often allows to initialize the weights of the policy to be trained with those obtained in the training of a previous level, on the assumption that the strategy to avoid enemies does not vary excessively between scenes. Applying this type of initialization generally results in more agile learning at any given level. Another aspect to note here is that since the RL node is only active under danger conditions, the actual time required to train a hybrid model is always practically higher than that needed in a pure RL case, assuming that the number of training steps remains unchanged for both models: these agents only truly train during the time when the RL nodes are active in the behavior tree, which can lead to situations of strong wastage of training time (for instance if the agent infrequently enters the enemy's range of vision). Although we have not focused on this issue in this article, it could be addressed in future work by designing, for example, training environments that favor the activation of certain nodes, or by directly training these actions under controlled situations (where the relevant nodes are always active). A pending matter, however, is to evaluate these techniques in more complex games and nested behavior trees in order to analyze how this affects the time and training difficulties of each agent.

Testing Process

Once we have agents that are able to interact with a level in a satisfactory way, it is possible to proceed to the next testing phase. At this stage we will be interested in extracting a sample of relevant execution parameters for a given number of attempts of the agent on the considered level. In our particular case, we set a number of repetitions $N = 100$ per agent and level, in each of which we record statistics regarding the HP with which the agent finishes each run (in range $[0, 10]$) and the number of steps needed to complete the level (in range $[0, 5000]$). These samples can be stored as references against which to test future changes in the level.

In the context of this paper, we select 3 levels representative of the general mechanics of the game, on which we train different agents using the techniques explained above until we obtain policies that are able to complete each level in an acceptable way. The concept of acceptability here adopts an admittedly subjective meaning: an agent must be able to complete the level consistently, but at the same time it is desirable that its learned policy does not outperform an average human in skill.

Having selected reference agents for a level and recorded a sample of contrast statistics, it is possible to perform a series of Student's t-tests (or Welch's t-tests if we do not assume variance equality) after incorporating any change in the game in order to check whether any of the statistics considered varies significantly (for the reference agent) after introducing said change. To do this, it is sufficient to collect a new execution sample for the agent on the modified level and perform a parameter-by-parameter t-test to assess whether the means of the reference and test sets are significantly different from one another after the change ($\alpha = 0.05$). In our case, we assume that the variances of the two groups do not necessarily need to be equal, and therefore we resort to Welch's t-tests (Lu and Yuan 2010). While we have limited ourselves to these tests for this experiment, they may not be applicable to different scenarios or parameters from more complicated games and mechanics where normality assumptions may not hold. Under these situations, nonparametric, more broadly applicable tests such as Mann-Whitney may be better suited to test for changes and nuances in gameplay. Under this scheme, we will evaluate the following changes for each level and agent:

- Increase the enemy patrol speed by 40 percent.
- Expand the enemy's field of vision, with a 35 percent greater detection distance and 20 additional degrees of peripheral visibility.
- Increase the enemy's gun firing rate, making the enemy capable of firing at twice the original speed (double the number of bullets per unit of time).

From a level design perspective, it would be expected that the above changes would amount to a substantial increase in the difficulty of the environments, especially in those levels where a good degree of control over the player is necessary whilst dodging enemies. In particular, we hypothesize that an increase in the enemy's firing rate should lead to significant reductions in the player's final HP in all levels, while the other two changes could possibly involve higher completion times should agents be forced to exercise more caution, or lower final HP for agents which display more aggressive navigation patterns.

Results

Below we present the results of applying each type of agent on the test levels with the described alterations. Table 1 depicts the mean statistics for the parameters to be compared in a benchmark run on the unmodified levels, whereas table 2 captures the results of running a Welch's t-test to contrast the difference in means between the reference and the agent samples after applying each type of change at each level, although we will limit ourselves to including only statistically significant entries for the sake of conciseness. We declare that a variation in a level fails a test on a parameter if the corresponding p-value is less than $\alpha = 0.05$, highlighting the associated value in the test results tables accordingly.

We note here that from the perspective of the hybrid agent the level whose gameplay is most affected by the introduced changes is "Enemy Avoidance" (Fig. 1a), which is to be expected, as it is the one that requires the highest degree of

Level ID	Agent Type	Av. final HP	Av. comp. time
Enemy Avoidance	Hybrid-RL	9.57	61.45
	Pure RL	9.54	53.73
Enemy Timing	Hybrid-RL	9.73	53.63
	Pure RL	9.9	32.17
Hiding Areas	Hybrid-RL	6.35	216.38
	Pure RL	9.8	127.53

Table 1: Reference Run Statistics

Level ID	Agent type	Change type	Tested parameter	Welch's t results			
				New Mean	*t*	*df*	*p-value*
Enemy Avoidance	Hybrid-RL	Patrol Speed	Completion Time	68.98	-2.56	185.89	**0.011**
Enemy Avoidance	Hybrid-RL	Field of vision	Final HP	9.13	2.92	174.16	**0.003**
Enemy Avoidance	Hybrid-RL	Field of vision	Completion Time	67.42	-2.10	190.13	**0.036**
Enemy Avoidance	Hybrid-RL	Firing Rate	Final HP	9.1	3.06	171.14	**0.002**
Enemy Timing	Pure RL	Patrol Speed	Completion Time	31.67	2.35	185.45	**0.02**
Enemy Timing	Pure RL	Firing Rate	Final HP	9.67	3.75	156.49	**0.0**
Hiding Areas	Hybrid-RL	Patrol Speed	Final HP	8.82	-4.52	170.88	**0.0**
Hiding Areas	Hybrid-RL	Patrol Speed	Completion Time	166.62	8.92	168.28	**0.0**
Hiding Areas	Pure RL	Patrol Speed	Final HP	8.71	6.49	115.43	**0.0**
Hiding Areas	Pure RL	Patrol Speed	Completion Time	132.03	-2.49	119.83	**0.013**
Hiding Areas	Pure RL	Field of vision	Final HP	9.31	5.27	154.37	**0.0**
Hiding Areas	Pure RL	Firing Rate	Final HP	9.57	2.84	173.02	**0.004**

Table 2: Test results after changes - Statistically Significant entries

control when dodging enemies in a fairly limited space. In particular, tests indicate that expanding the enemy's field of vision has a significant effect on both completion time and final HP at the end of the level (the player ends up with less health and takes longer to finish). This is not the case for the purely reinforcement-trained agent, whose higher overall skill level leads to the test detecting no significant differences in any of the statistics evaluated. After manual testing to evaluate these changes from a third, human perspective this time, it becomes apparent that all three modifications result in a noticeable increase in difficulty when traversing the enemy regions of the level, meaning that the tests using the hybrid model succeed in warning about these changes.

In the case of the "Enemy Timing" level (Fig. 1b), we observed a somewhat opposite effect: while the mixed agent does not detect significant changes with any of the modifications, the pure RL agent perceives a slight reduction in the time required to complete the level by increasing the enemies' patrol speed (probably due to the fact that this allows to advance the instant in which the enemies have moved enough to be able to cross the central corridor without being spotted), and an increase in the damage received by increasing the enemies' firing rate (< 0.5 difference). The human tests in this case lead to the conclusion that the changes introduced do not notably affect the playability of the level, in accordance with the hybrid model's test results.

Lastly, in the "Hiding Areas" level (Fig. 1c), both agents perceive a significant change in the two evaluated parameters when increasing the patrol speed, but in different ways. The hybrid agent perceives a marked reduction in difficulty (finishing with over 2 HP more than without the change, and in slightly over half the time), while the pure reinforcement agent's average final HP decreases, with a slight yet significant increase in the time needed to complete the level. If we observe the strategies employed by each agent in this environment, we realize that the hybrid agent applies a policy in which it waits for the enemy to modify its position to a safe configuration before advancing, something that is accelerated by increasing the patrolling speed, allowing for more frequent and safer advances. The pure RL agent, on the other hand, follows a much more aggressive policy in which it attempts to complete the level as fast as possible relying solely on its dodging skills, leading to an increase in perceived difficulty. From the human point of view, the difficulty of the level is notably reduced and navigation is clearly simplified when following a strategic plan that exploits the hiding areas, as opposed to the effect that was initially expected to be induced (harder level overall).

Additionally, in order to evaluate how often these tests would alert about potentially non-existent changes, we consider their repeated application on environments that remain unchanged with respect to the initial reference and compute the proportion of cases in which such tests generate a positive result (false positives). For each level and type of agent, we repeat the corresponding test 100 times and record the proportion of cases in which the test reports a (mistaken)

Level ID	Agent Type	False positive rate (final HP)	False positive rate (comp. time)
Enemy Avoidance	Hybrid-RL	5%	4%
	Pure RL	13%	4%
Enemy Timing	Hybrid-RL	0%	0%
	Pure RL	9%	1%
Hiding Areas	Hybrid-RL	1%	2%
	Pure RL	1%	1%

Table 3: False positive rates per configuration and parameter

modification of the parameters of interest. The results can be found in table 3. We observe that the hybrid model generally yields lower fail rates in almost all cases in comparison with the pure reinforcement model, which may be the result of the lower degree of freedom that hybrid agents have in their decision making, given that part of their behavior is comprised of hand-programmed flows. However, while this is certainly noteworthy, it should be emphasized that false positives in this context are rather mild in severity, insofar as, in the worst case, they would simply force an unnecessary revision of the level.

Conclusions

This paper focused on the problem of creating testing agents to detect significant changes in two different aspects of gameplay on a demo with stealth genre mechanics. We proposed two accessible alternatives to obtain agents capable of playing different levels of the game, the first one through pure RL, and the second through hybrid models that combine RL action nodes within hand-coded control structures in the form of BTs. After obtaining satisfactory agents for each level and training method, a first simulation step was carried out in which the agents played their respective levels repeatedly, recording relevant statistics regarding their perception of the execution, namely HP remaining at the end of the level and time taken to complete it. Subsequently, after introducing various changes in the behavior of the enemies, we again performed simulations with the agents on all levels to re-collect performance metrics in the modified environments. Lastly, the reference samples were compared with those of the post-change simulations by applying a means-contrast test on each of the parameters of interest to detect statistically significant changes in level gameplay.

Our results show that the automated tests performed using this methodology can assist in locating and explaining changes of interest in the base of evaluated levels. If the tests after modification fail for any of the agents, the designer can analyze the results to check whether they conform to the predicted effect, if this was intended, or proceed to manually investigate the level where the test failed to search for the origin of the variation in the statistics. This makes it possible to narrow down the search for unexpected effects to those levels reported by the tests, or to evaluate whether a design change has the desired impact on the monitored parameters.

The conducted experiments seem to suggest that hybrid-type agents, despite their apparently lower general skill at level completion, are more sensitive to changes in level perception that match those perceived by human testers than their purely RL-trained counterparts. On the other hand, although the use of hybrid agents involves certain drawbacks, such as a certain inefficiency during training due to the lack of continuous activation of the RL nodes, it also offers several other advantages that we believe could prove convenient in the testing and design process. Possibly the clearest benefit is that the use of hybrid agents comes hand in hand with a greater degree of control over the applied policy, allowing to replace and/or support parts of it with manually programmed behaviors, or even to retrain critical flow blocks without having to sacrifice the totality of the learned behavior (as would occur in the case of pure reinforcement). On the other hand, the modular structure of behavior trees enables the use of policies with a great variety and depth of customization, constructing flows that follow a certain reward or utility function locally and not only globally, as is the case when specifying such functions in agents trained with a single RL policy.

In the future, we would like to validate our approach by applying this methodology on a wider spectrum of environments as well as on modifications varying in nature and extent, in order to demonstrate that it is able to detect changes of distinct types in level gameplay. Additionally, since this study has not made any attempts to achieve agents that exhibit human behavior, it remains as future work to incorporate mechanisms that allow replicating either flows/traces recorded by designers or testers (through imitation learning), or that emulate designer-devised player archetypes (through reward functions adjusted to different play styles, for example, along the lines of the work of (Holmgard et al. 2019)). Now, while current agents do not attempt to emulate human behaviors, we consider that a positive on a change test still has value in that it is indicative of a perceptual alteration of at least one policy that, while it may or may not be representative of how a human approaches the problem at hand, does, in any case, denote a structural change in the level. Lastly, we consider as future work to include training based on procedural level generation (e.g., adapting ideas from works such as (Justesen et al. 2018)) for the sake of improving the degree of generalization of the produced agents, which is a topic that has not been dealt with in this first approximation.

References

Abran, A., ed. 2004. *Guide to the software engineering body of knowledge, 2004 version: SWEBOK ; a project of the IEEE Computer Society Professional Practices Committee*. Los Alamitos, Calif.: IEEE Computer Society. ISBN 9780769523309. OCLC: 934432015.

Ariyurek, S.; Betin-Can, A.; and Surer, E. 2021. Automated Video Game Testing Using Synthetic and Humanlike Agents. *IEEE Transactions on Games* 13(1): 50–67. ISSN 2475-1510. doi:10.1109/TG.2019.2947597.

Bergdahl, J.; Gordillo, C.; Tollmar, K.; and Gisslén, L. 2020. Augmenting Automated Game Testing with Deep Reinforcement Learning. In *2020 IEEE Conference on Games (CoG)*, 600–603. doi:10.1109/CoG47356.2020.9231552. ISSN: 2325-4289.

Cook, M.; Gow, J.; Smith, G.; and Colton, S. 2021. Danesh: Interactive Tools For Understanding Procedural Content Generators. *IEEE Transactions on Games* 1–1. ISSN 2475-1502, 2475-1510. doi:10.1109/TG.2021.3078323. URL https://ieeexplore.ieee.org/document/9426419/.

Florez-Puga, G.; Gomez-Martin, M.; Gomez-Martin, P.; Diaz-Agudo, B.; and Gonzalez-Calero, P. 2009. Query-Enabled Behavior Trees. *IEEE Transactions on Computational Intelligence and AI in Games* 1(4): 298–308. ISSN 1943-068X, 1943-0698. doi:10.1109/TCIAIG.2009.2036369. URL http://ieeexplore.ieee.org/document/5325892/.

Gonzalez-Duque, M.; Palm, R. B.; Ha, D.; and Risi, S. 2020. Finding Game Levels with the Right Difficulty in a Few Trials through Intelligent Trial-and-Error. In *2020 IEEE Conference on Games (CoG)*, 503–510. Osaka, Japan: IEEE. ISBN 9781728145334. doi:10.1109/CoG47356.2020.9231548. URL https://ieeexplore.ieee.org/document/9231548/.

Hernández Bécares, J.; Costero Valero, L.; and Gómez Martín, P. P. 2017. An approach to automated videogame beta testing. *Entertainment Computing* 18: 79–92. ISSN 1875-9521. doi:10.1016/j.entcom.2016.08.002. URL https://www.sciencedirect.com/science/article/abs/pii/S1875952116300234.

Holmgard, C.; Green, M. C.; Liapis, A.; and Togelius, J. 2019. Automated Playtesting With Procedural Personas Through MCTS With Evolved Heuristics. *IEEE Transactions on Games* 11(4): 352–362. ISSN 2475-1502, 2475-1510. doi:10.1109/TG.2018.2808198. URL https://ieeexplore.ieee.org/document/8295256/.

Juliani, A.; Berges, V.-P.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2020. Unity: A General Platform for Intelligent Agents. *arXiv:1809.02627 [cs, stat]* URL http://arxiv.org/abs/1809.02627. ArXiv: 1809.02627.

Justesen, N.; Torrado, R. R.; Bontrager, P.; Khalifa, A.; Togelius, J.; and Risi, S. 2018. Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation. *arXiv:1806.10729 [cs, stat]* URL http://arxiv.org/abs/1806.10729. ArXiv: 1806.10729.

Lu, Z.; and Yuan, K.-H. 2010. *Welch's t test*, 1620–1623. 10.13140/RG.2.1.3057.9607.

Ostrowski, M.; and Aroudj, S. 2013. Automated Regression Testing within Video Game Development. *GSTF Journal on Computing (JoC)* 3(2): 10. ISSN 2010-2283. doi:10.7603/s40601-013-0010-4. URL http://www.globalsciencejournals.com/article/10.7603/s40601-013-0010-4.

PadaOne Games. 2021. BehaviorBricks. http://bb.padaonegames.com/. Accessed: 2021-04-16.

Pereira, R. d. P.; and Engel, P. M. 2015. A Framework for Constrained and Adaptive Behavior-Based Agents. *arXiv:1506.02312 [cs]* URL http://arxiv.org/abs/1506.02312. ArXiv: 1506.02312.

Pfau, J.; Smeddinck, J. D.; and Malaka, R. 2017. Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving. In *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, 153–164. Amsterdam The Netherlands: ACM. ISBN 9781450351119. doi:10.1145/3130859.3131439. URL https://dl.acm.org/doi/10.1145/3130859.3131439.

Shaker, N.; Yannakakis, G.; and Togelius, J. 2010. Towards Automatic Personalized Content Generation for Platform Games. Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010.

Unity Technologies. 2021a. Unity Real-Time Development Platform | 3D, 2D VR & AR Engine. https://unity.com/. Accessed: 2021-04-10.

Unity Technologies. 2021b. Unity Test Framework | 1.1.24. https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html. Accessed: 2021-04-10.