Towards Agent-Based Testing of 3D Games using Reinforcement Learning

Raihana Ferdous
rferdous@fbk.eu
Fondazione Bruno Kessler
Italy

Fitsum Kifetew
kifetew@fbk.eu
Fondazione Bruno Kessler
Italy

Davide Prandi
prandi@fbk.eu
Fondazione Bruno Kessler
Italy

Angelo Susi
susi@fbk.eu
Fondazione Bruno Kessler
Italy

ABSTRACT

Computer game is a billion-dollar industry and is booming. Testing games has been recognized as a difficult task, which mainly relies on manual playing and scripting based testing. With the advances in technologies, computer games have become increasingly more interactive and complex, thus play-testing using human participants alone has become unfeasible. In recent days, play-testing of games via autonomous agents has shown great promise by accelerating and simplifying this process. Reinforcement Learning solutions have the potential of complementing current scripted and automated solutions by learning directly from playing the game without the need of human intervention. This paper presented an approach based on reinforcement learning for automated testing of 3D games. We make use of the notion of curiosity as a motivating factor to encourage an RL agent to explore its environment. The results from our exploratory study are promising and we have preliminary evidence that reinforcement learning can be adopted for automated testing of 3D games.

KEYWORDS

reinforcement learning, game play testing, functional coverage

ACM Reference Format:
Raihana Ferdous, Fitsum Kifetew, Davide Prandi, and Angelo Susi. 2022. Towards Agent-Based Testing of 3D Games using Reinforcement Learning. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3551349.3560507

1 INTRODUCTION

The video game industry continues to grow and is very competitive [16]. Releasing excellent quality games and ensuring a top-notch user experience is of utmost importance in such a competitive environment.

Until recently game testing mostly relied on a manual or some scripted approach. As games grow in size and complexity, testing games using human participants or script based approaches face difficulty. Moreover, the diverse characteristics of video games [2] makes the design and development of a generic technique and tool for testing of relevant content and identification of potential anomalies extremely challenging. Complementing play-testing via autonomous agents seems to be promising as it has the potential to accelerate game testing at scale & breadth. Even though there is existing research toward developing automated play testing of games, there is still a need for continued research addressing the major difficulties [21].

1.

One of the difficulties concerns achieving high coverage because of the large combinatorial space of possible events and transitions, which makes testing all possible scenarios time-consuming and ineffective. Another difficulty is in achieving a wide range of application functionalities, because 2. some functionalities can only be reached through a specific sequence of events. To exercise such functionalities, we would need to deploy smart exploration strategies that go beyond a simple random exploration, as is done in random testing. While random testing is widely used in traditional software testing, in the context of 3D games we would need a better exploration strategy that can interact with the game intelligently. The use of autonomous test agents comes in with an advantage from this point of view. With the appropriate guidance and support, test agents have been shown to be effective in exploring 3D games [9, 22, 24].

Model-based approaches present an effective alternative where a formulation/modeling of game play testing is performed in such a way that different strategies, e.g., search-based testing, can be applied to generate tests automatically. In this case, high-quality models are extremely important to achieve good results. However, as is typical of model-based testing, producing such high-quality models is by itself challenging.

On the other hand, Reinforcement Learning (RL) based approaches have the potential to significantly improve automated game testing as they have the capability of learning directly by interacting with the dynamic and uncertain game environment without the explicit need of modeling it. Recent research on RL in computer games has mainly focused on constructing intelligent agents that can adapt to the behavior of players and to dynamically changing environments. RL agents have achieved great success in automated game playing, starting from totally random trials and finishing with sophisticated tactics and superhuman skills, including the ancient board game Go [26], classic Atari games [15], multiplayer online battle arena game Dota 2 [6], first-person shooter game [14], and real-time strategy (RTS) game StarCraft II [30]. Inspired by this remarkable result in game playing (also known as play testing), some research work [10, 32] has investigated the potential of using RL solutions beyond game play or play-testing, towards the direction of coverage.

In this paper, we focus on the suitability of using RL solutions for automated testing and providing functional coverage of computer games. The goal here is not to optimize a behaviour policy or obtain sophisticated game play skills, but rather to deploy an agent that learns how to advance in playing the game, and also to explore the game environment. Both abilities, to explore and to learn to play, are essential: with exploration, the agent will reveal enough game area and functionalities leading to better coverage, and with the ability to finish the game-play the agent will be skillful enough to survive and eventually to further explore the game.

The use of RL algorithms in automated testing of complex 3D environments (such as a 3D maze, first-person shooter game, real time strategy game, multiplayer online battle game, etc.) is more challenging than playing most Atari and board games as it involves a vast amount of critical thinking, problem-solving, and path planning. Unlike board games where a single frame of the environment provides an agent all the information needed to maximize its reward, in these games, the imperfect information of the agent due to its partial visibility of the environment makes efficient learning non-trivial [30]. Moreover, the large state-action space, long time span of the games, and delayed or sparse reward assignment pose challenges to effective RL solutions. To address these issues, careful attention is needed to represent states and actions and to define effective reward mechanisms in the environment.

We have used a *curiosity-based* reward scheme that has the ability to become a powerful exploration mechanism to facilitate the discovery of solutions for complex, sparse or long time span tasks. Specifically, the scheme is beneficial in this context where we aim to maximize the coverage. It is to note that defining a common notion of coverage for 3D games itself is a difficult job due to the diverse characteristics and complexity of games. Hence, recent research has mainly focused on revealing larger game space with the assumption that a good exploration indicates higher game space coverage and also higher possibility of discovering anomalies [5, 10, 32]. In this work, we concentrate on identifying coverage metrics significant to provide functional coverage of a 3D maze game and the strategy to measure them.

The approach is implemented in a prototype tool RLbT that applies the curiosity-based RL solution for automated testing of games by maximizing coverage. Empirical evaluation is carried out by applying RLbT on a 3D game called Lab Recruits and comparing the curiosity-based approach with a couple of alternative baselines. Results are promising where the curiosity-based RL is effective in achieving reasonable levels of coverage, in particular on larger and complex game scenarios/levels.

The main contributions of this work is:

- an approach showing the feasibility of using RL in automated testing of games with the aim of maximizing coverage (evaluation is performed through two research questions defined in Section 5);
- publicly available artifacts (tool, data) that enable reproducibility of results and facilitate further research.

The rest of the paper is organised as follows: Section 2 introduces the running example used throughout the paper. Section 3 presents the background concept, Section 4 shows our proposed

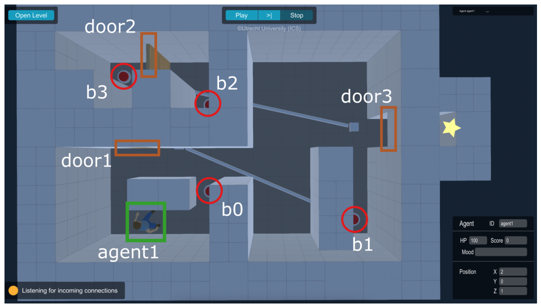

Figure 1: Level buttonDoors in Lab Recruits.

reinforcement learning based approach in game testing. Experimental results are presented in Section 5 and related work is discussed in Section 6. Section 7 concludes, and outlines future work.

2 ENVIRONMENT EXAMPLE

This section introduces Lab Recruits[1], a 3D game developed for experimenting with intelligent agents. Lab Recruits allows the definition of mazes, a set of rooms and corridors connected by doors. Each door is opened/closed by one or more buttons, and each button activates one or more doors. Scenarios could further be enriched by adding different furniture (e.g., tables) as well as simulating fire hazards. The player's goal is to find a path to reach a certain room by opening doors in the correct order. The game can be played by both humans and artificial agents [23]. Lab Recruits levels are defined as csv (comma-separated value) human-readable files allowing researchers to specify levels of variable complexity.

As a running example, Figure 1 shows a level of Lab Recruits named buttonDoors. The level consists of three doors, door1, door2, and door3, and four buttons, b0, b1, b2, and b3. Door door1 is activated by buttons b1, b2, and b3, while door2 and door3 are connected only to b2. b0 is not connected to any door, therefore pressing it has not effect in the game. The goal of the Agent agent1 is to reach the room marked with a star, and therefore to open door3. A possible path requires agent1 to press b1 to open door1 and then b2 to open door3. Since b2 also acts on door1, at this point agent1 is not able to reach door3, but need to traverse door2 and press b3 to open door1. Now, agent1 walks through door1 and door3, finally reaching the destination. Even if the layout of the level is simple, the path to reach the final room is non trivial, in particular for automated testing.

3 BACKGROUND

3.1 Reinforcement Learning (RL)

Reinforcement Learning [27] approach is a machine learning method suitable for solving problems where the decision making is successive and the goal/objective is long-term. RL concerns about learning the optimal behavior in an environment to obtain maximum reward. This optimal behavior is learned through interacting with the environment and observing of the responses/feedbacks. In RL, the sequential decision-making problem is defined as a Markov decision process (MDP) which is defined as a four-tuple (S, A, T, R),

[1]https://github.com/iv4xr-project/labrecruits

where S is a set of states, A is a set of actions, T is a transition function (probability of transitioning from state s to state s' after taking action a), R indicates the reward function $R(s, a)$ which specifies the reward of choosing action a in state s.

3.2 Q-learning Algorithm

Q-Learning [31] is a value-based, model-free Reinforcement Learning algorithm that finds the optimal policy indirectly by training a value or action-value function (i.e., Q-function) that knows the value of each state or each state-action pair. The simple form of storing the optimal Q-function is in a tabular form known as a Q-table. The Bellman Equation [8] is used to determine/update the value of an action in a particular state (i.e., the goodness of an action). From state s, a choice of an action a leads to a new state s' with a reward $R(s, a)$. The value function $Q(s, a)$ is updated using the following equation which uses the current state, the reward associated with that state, and, the maximum expected reward of the new state for all actions. Discount factor γ determines the importance of future rewards. The learning rate α determines to what extent newly acquired information overrides old information.

$$Q(s, a) = Q(s, a) + [\alpha(R(s, a) + \gamma \max Q'(s', a') - Q(s, a))]$$

4 METHODOLOGY

Play testing involves finding a sequence of actions to achieve a desired goal in the game. This can be framed as a reinforcement learning task where an *agent* learns an optimal policy to achieve its goal by trial-and-error to maximize the expected reward as a result of its interaction with an unknown *environment*. Here, the environment is what an agent interacts with (e.g., the game under test), a *state* is the agent's observation of the environment, the *action* is a set of possible decisions that the agent can make in a given state, and the *reward* is the feedback by which the success or failure of an action can be measured with regard to achieving some goal (e.g., reaching the treasure room, achieving points, etc.).

As mentioned in Section 1, use of RL solutions in play testing of complex 3D environments (e.g., 3D maze as `Lab Recruits` game) is challenging. The 1. imperfect/partial information of the agent regarding the environment complicates effective learning. For example, in `buttonDoors` level (as shown in Figure 1) of `Lab Recruits`, determining the state and reachable entities are based on the observation range and position of the agent, something that changes constantly with the agent's movements. At any time t, an agent only has partial information of the level according to its observation range. Thus, the agent is not always immediately aware of the impact of its action (i.e., a door is open due to the pressing of a button) simply because the entity is out of its visibility range. In such a context, there is a risk of not assigning a reward to a space-action pair accordingly, which hinders the learning procedure.

Another key challenge is the large state-action space due to the long time span and complexity of the game. To address this issue, the definition of what comprises a state and an action in the environment is significant. Specifically, abstractions are needed to represent states in a complex game environment, but not to the degree that we lose control and observability of the environment. In some research work, states are defined as the 3D position of the agent at a given point in time. Keeping track of these visit counters on a continuous 2 space, however, quickly becomes intractable. Here, we propose to keep state space discrete by staying at a higher level of abstraction while representing states and actions. Thus, a state should consist of relevant features only; i.e., entities that can be controlled/interacted by the agent and entities that can affect an agent.

3. Delayed or sparse reward assignment is also a hurdle for effective use of reinforcement learning in many 3D game environments where rewards extrinsic to the agent are extremely delayed, sparse, or even non-existing. Usually, in environments with dense rewards, the rewards are received fairly frequently during the training time, making it easy to learn an optimal policy. While, in sparse reward environments, rewards may only be received after many sub-goals are completed, making it difficult or impossible for an agent to learn an optimal policy based on the reward signals alone. For example, in the `buttonDoors` level of `Lab Recruits` (Figure 1), an agent is only rewarded if it reaches the room marked with a star. However, that room is far away from the agent's starting point, so most of the rewards will be zero. As rewards act as feedback for the RL agent, if it does not receive any, the knowledge of which action is beneficial (or not) will not be updated and it will take much longer or forever to learn an optimal policy. A common solution to this problem is reward shaping/crafting meaning to make rewards dense and more suitable for learning. However, reward shaping requires complete knowledge of the environment. Hence, annotating each environment with hand-designed dense rewards is not scalable and generic. One way of tackling this sparse reward problem is to use the notion of an intrinsic curiosity, an agent's natural desire to explore the environment to understand how things work and reduce the uncertainty about the environment [4, 19]. This tactic of using curiosity is more generic as it can be applied to diverse environments. Here, we have proposed to use curiosity as a motivating factor for the agent to explore the environment. Along with the intrinsic reward an agent receives from the game environment, we propose to provide incentives to the agent for exploring new states while penalizing it for exploiting previously visited states only. The notion of curiosity encourages the agent to explore the environment which leads to better coverage and at the same time helps to learn useful skills or achieve environmental specific goals.

As it is extremely difficult to construct a sufficiently accurate environment model for complex games, we have used *Q-learning algorithm*, a model free RL strategy which is applicable in different environments and can readily react to new and unseen states.

4.1 Curiosity based Q-Learning solution

This section details using RL algorithms in testing of `Lab Recruits` and similar 3D games. We have used an agent based testing infrastructure [23] to interact with `Lab Recruits`, where the intelligent agent can follow goal driven planning and reasoning which helps it towards effective navigation in complex interaction space. Thus we can rely on the navigation capability of the agent and provide it with high-level RL goals (e.g., open *door3*) to solve.

We have used the Q-learning algorithm that intends to find the optimal policy by training a action-value function containing the value of each state-action pair. We have focused on using a tabular Q-learning solution to keep the design simple. Such modeling is

Entity	Property
door1	*id=door1, type=door, isOpen=false...*
door3	*id=door3, type=door, isOpen=false...*
b0	*id=b0, type=button, isOn=false...*
b1	*id=b1, type=button, isOn=false...*

Table 1: A state from agent's observation

not trivial, in particular, it requires special attention to reduce the state space as well as consideration to ensure convergence of the learning process which are described in the following subsections.

4.1.1 Representation of States and Actions. In defining a state of `Lab Recruits` game we propose to stay at a high level of abstraction to limit the state space. Hence, a state in `Lab Recruits` game environment is constructed from the agent's observation of the environment at a certain moment. An agent has a visibility range and its observation of the environment at time t includes `Lab Recruits` game features/entities according to the agent's position and visibility range. A state at time t comprises a set of entities from the agent's observation including their properties. For example, the `buttonDoors` level in Figure 1) contains 7 feature entities: three doors, `door1`, `door2`, and `door3`, and four buttons, `b0`, `b1`, `b2`, and `b3`. Each entity has some properties such as its type (e.g., entity type, dynamic/static, interactable/not interactable, etc), status (e.g., position, velocity, etc), and history (e.g., change of status). In this level, entities like buttons and doors have properties to indicate if they are interacted with or not (e.g., for a button, property 'isOn' is false if it is not pressed, and for a door, property 'isOpen' is false if it is closed). From its current position (as in Figure 1), the agent (here, name "agent1") can only see `door1`, `b0`, `b1`, and `door3`. It cannot see the entities `door2`, `b2`, and `b3` as they are out of the visibility range of the agent residing behind `door1`. At this point the state s obtained from the agent's observation of the environment is shown in Table 1. Given a state s, the set of possible actions for the agent consists of the interactable entities such as buttons (e.g., a table maybe present in the environment but it is not an interactable entity). From the state s in Table 1, the set of actions includes: {press `b0`, press `b1`, check `door1` status (open/closed), check `door3` status (open/closed)}. Executing an action leads to the next state. For example, pressing `b1` will open `door1` and `door2` and `b3` will become visible to the agent. Thus, this action will lead the agent from state $s = \langle \texttt{b0}(isOn = false), \texttt{b1}(isOn = false), \texttt{door1}(isOpen = false), \texttt{door3}(isOpen = false)\rangle$ to reach state $s_1 = \langle \texttt{b0}(isOn = false), \texttt{b1}(isOn = true), \texttt{door1}(isOpen = true), \texttt{door3}(isOpen = false), \texttt{door2}(isOpen = false), \texttt{b3}(isOn = false)\rangle$. One thing to note regarding the interaction with an interactable entity is that an agent can only do that if the entity is physically reachable from its current position.

4.1.2 Reward Function. Given an action that causes a transition from state s to $s1$, we formulate the reward proportional to the transition's novelty. Here, along with the intrinsic reward that the agent receives from the environment, the agent is also provided with incentives for its curiosity in exploring the environment. This formulation of the reward function will help the agent not only to figure out how to maximize the environment reward but also to be curious about the environment. We provide a low and decreasing reward (e.g., penalty) for revisiting previously explored states, while a high reward is for reaching new areas or triggering new actions. Therefore, we keep track of the game states that have been visited by an agent and their frequency. The reward is computed based on how distant the current observation is from the most similar observation in the memory. The agent receives a higher reward for seeing observations which are not yet represented in memory. An agent is only considered to have entered a new state once its dissimilarity to any previously visited state is larger than a threshold. We have used the Jaccard coefficient [13] to measure the similarity between states.

4.1.3 State similarity. A state in `Lab Recruits` game consists of the observation of an agent at time t. As the observation of the agent relies on its position and visibility range, the agent rarely sees the same state twice. For example, even if the agent returned to the same room, it may see this room from a different angle compared to an already stored state in the Q-learning table. Hence, checking for an exact match could be meaningless. Also, making a new state entry for each observation will soon make the Q-table so large that it will take a long time to converge, i.e., to learn values for each state-action pair. To this end, we adopt a state similarity measure while updating the Q-table. When an agent gets a new observation/state, instead of making a new entry, it first looks for a similar existing state in the Q-table. In the case when a similar state is observed in Q-table, the agent considers it as current state and chooses action accordingly.

4.1.4 Q-learning algorithm. Though Q-learning an off-policy algorithm, during the learning process, an agent may follow a learning policy which leads towards an optimal policy. An agent may greedily select actions based on the information that it has acquired, i.e, performs what is known as *exploitation*. While exploitation is useful, it could lead to premature results as the agent will be biased by what it has observed and not be able to effectively *explore* the environment. Hence, striking the right balance between exploration and exploitation is crucial. Considering this, we have used *Decayed Epsilon-Greedy* as the initial policy to balance exploration and exploitation. In particular, it allows the agent to explore more when it does not have enough information about the environment, i.e., at the beginning of the learning. Once the agent does gather enough information for interacting optimally with the environment, the policy allows it to exploit the information. The process is controlled by the ϵ parameter which we set initially to a value and at each episode we *decay* it by a factor, hence allowing the agent to exploit the knowledge it has learned so far. The decaying factor is calculated by the number of learning episodes. The algorithm for curiosity based Q-learning is presented in Algorithm 1 and 2. With sufficient amount of training, the output of this algorithm is the optimized Q-table that is the reference table for the agent to select the best action based on the highest Q-value from a state.

5 EVALUATION

In this section, we present the exploratory experiment we carried out to get insight into the feasibility of the proposed reinforcement learning solution and the prototype tool `RLbT` which we used for the experiments.

Algorithm 1 Curiosity based Q-learning algorithm

```
1:  Input : Learning rate, $\alpha \in (0,1]$, Discount factor, $\gamma \in (0,1]$,
    Epsilon-greedy policy, $\epsilon \in (0,1]$, $numofEpisode$, $maxAction$
2:  Initialize Q table, $Q$
3:  $decayfactor \leftarrow \epsilon / numofEpisode$
4:  for each episode do
5:      while not reached final destination or $maxAction$ do
6:          Observe current state $S$
7:          Choose action $A$ from $S$ using learning policy
8:          Execute action $A$, observe new state $S'$
9:          Get reward, $R \leftarrow GetReward(S,A, S')$
10:         Update Q table : $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
11:         Set $S \leftarrow S'$
12:     end while
13:     Set $\epsilon \leftarrow \epsilon - decayfactor$
14: end for
```

Algorithm 2 *GetReward(S,A, S')*

```
1: Input : State $S$, Action $A$, new State $S'$, $statevisitthreshold$
2: if $S'$ is final state then
3:     Get Max reward
4: end if
5: if $S'$ is not final state then
6:     Calculate similarity of $S'$ and $S$
7:     Calculate $visitcount$ of $S'$
8:     Get reward for visiting new state or penalty for $visitcount > statevisitthreshold$
9: end if
```

5.1 Prototype

The proposed approach is implemented in the prototype tool `RLbT` relying on the BURLAP[2] library for reinforcement learning. The agent based testing infrastructure used to interact with `Lab Recruits` is based on the openly available agent interface implementation[3] which in turn relies on an agent programming library[4] of the iv4xr framework. Our implementation of `RLbT` is open source[5].

5.2 Environment - Systems under test

We carried out our experiments on five *levels* of `Lab Recruits` with different characteristics that allow us to get insight into the applicability of reinforcement learning.

- `buttonDoors`: a small maze shown in Figure 1.
- `4-room`: this level contains 16 buttons and 8 doors distributed across 4 rooms. The layout of the level is in such a way that the agent can fairly easily spot the buttons and eventually observe their effects (i.e., doors that open/close) as well.
- `8-room`: similar to `4-room` but with eight rooms, hence increasing the complexity in terms of the entities to be interacted with. Here, 32 buttons and 16 doors are distributed across the 8 rooms (see Figure 2). The goal is to open the door in the last room (the room at the far end opposite to the agent position in Figure 2).
- `LargeMaze_1`: a randomly generated level where the number of entities is comparable to `8-room` (19 buttons and 14 doors distributed across 14 rooms connected by long corridors) but the *physical space* covered by the level is significantly larger, as shown in Figure 3. This means that the agent needs to travel a long distance to get from one entity to another. This level poses a different type of difficulty to reinforcement learning also because observing the effect of an action (e.g., a button pressed) is difficult as the corresponding door that is opened/closeds may not be immediately visible as it resides in a different room across a long maze of corridors.
- `LargeMaze_2`: a randomly generated maze like `LargeMaze_1` with 29 buttons and 19 doors distributed across 19 rooms.

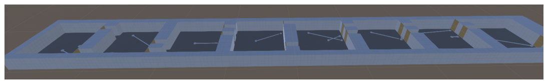
Figure 2: `8-room` level of `Lab Recruits`

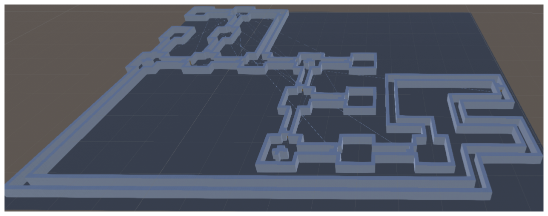
Figure 3: `LargeMaze_1` level of `Lab Recruits`

5.3 Experimental Setup

The experiment aims to assess the feasibility of reinforcement learning for the automated testing of 3D games. Consequently, we formulate the following research questions to guide our experimental evaluation:

RQ1 - Suitability of RL how suitable is RL based approach for automated testing of 3D games?

RQ2 - Coverage how suitable is RL for providing functional coverage in the context of game testing?

To assess the feasibility and effectiveness of reinforcement learning solutions in automated game testing and coverage, we compare the proposed `curiosity-based RL` solution with two alternative baseline solutions. First is `sparse-reward RL`, a classic RL approach with only intrinsic sparse reward, where an agent receives positive feedback only when it reaches its goal, otherwise nothing. The second baseline approach is the pure random solution, where the agent takes decisions randomly.

5.3.1 Parameter Settings. Several parameters control different aspects of `RLbT`. Some of the parameters are related to reinforcement learning, and some are related to the SUT (`Lab Recruits`) and the test agent used to interact with it. We have performed a set of preliminary experiments to determine suitable values for some of the important parameters. Based on the preliminary experiments,

[2]http://burlap.cs.brown.edu/
[3]https://github.com/iv4xr-project/iv4xrDemo
[4]https://github.com/iv4xr-project/aplib
[5]https://github.com/iv4xr-project/iv4xr-rlbt

Parameter	button Doors	4-room	8-room	Large Maze_1	Large Maze_2
num. episodes	100	300	500	500	500
action per episode	40	80	120	130	130
cycles per action	20	60	90	90	90
initial ϵ-value	0.5	0.5	0.5	0.5	0.5
learning rate (α)	0.25	0.25	0.25	0.25	0.25
discount rate (γ)	0.6	0.6	0.6	0.6	0.6

Table 2: Parameter settings RL solutions

we selected parameter values that represent a reasonable trade-off between effectiveness and efficiency without much loss in the generalizability of RLbT. We report the values of the important parameters used in our experiments in Table 2.

5.3.2 Experimental protocol. For both the RL solutions, Curiosity-based and Sparse-reward, we train an agent over a level with identical parameter settings as presented in Table 2. After the training/learning period, a Q-Table is obtained containing all the state-action pair values that the agent observed during the learning process. After training, we performed a test game-play session on a level of `Lab Recruits`, where the agent is guided by the Q-table regarding the best action to take from a state to maximize reward to achieve its goal. We have measured the performance and coverage based on the results of the test game-play sessions. For the random solution, we allow the agent to play the game for each level of `Lab Recruits` for an identical number of episodes as mentioned in Table 2. Here, the agent follows a random policy, thus choosing an action randomly from any state.

5.4 Results

In this section, we present the results of our experiments. As stated earlier, our experiments are exploratory and the results reported here have to be interpreted accordingly. We first discuss the results related to the goal oriented exploration followed by the coverage oriented exploration.

5.4.1 Goal Oriented. With goal oriented exploration, the aim is to learn the best way to achieve the specified goal in the game. For the levels of `Lab Recruits` we used in our study, this translates to activating a sequence of buttons that open various doors until the specified target goal is reached, in our case a specific door is opened. We have run the experiments on `buttonDoors` and `4-room` level. For `buttonDoors` the goal is to open *door3* and for `4-room` level the goal is to open *door16*. It is noticed that the agent is able to effectively learn the optimal sequence of actions needed to achieve the goal. For example, in `4-room` level the agent learns to reach the destination within 4 action sequences.

5.4.2 Coverage Oriented. In this work, we have focused on providing functional coverage for 3D games. In our experiments, we concentrate on identifying coverage metrics important for `Lab Recruits` game and the approach to measure the quantitative value of coverage achieved by our explorative agent.

- **Entity coverage** - percentage of observed/interacted entities (with all possible properties) in a level of `Lab Recruits` game. For example, the `buttonDoors` level (as shown in Figure 1) of the `Lab Recruits` game features seven entities (i.e., three doors and four buttons). A door can be observed in two statuses, thus having two properties *Open* and *Closed*. A button can have two properties, *pressed* and *not-pressed*. Table 3 presents the entities with all possible properties in `buttonDoors` level of the `Lab Recruits` game. Hence, the ratio of entity coverage for a level in `Lab Recruits` game is measured by the entities observed by an agent during testing with the total entities with all possible properties.
- **Entity Connection Coverage**- In a level of `Lab Recruits` game, doors are usually connected with buttons. For example, Table 4 presents connections between doors and buttons in the `buttonDoors` level. This metric measures the ratio of connection satisfies in a level.

Entity	Properties	
b0	*pressed*	*not-pressed*
b1	*pressed*	*not-pressed*
b2	*pressed*	*not-pressed*
b3	*pressed*	*not-pressed*
door1	*open*	*closed*
door2	*open*	*closed*
door3	*open*	*closed*

Table 3: Properties of entities available in level `buttonDoors` in `Lab Recruits`

Buttons	Connecting Doors		
b0		*not connected*	
b1		*door1*	
b2	*door1*	*door2*	*door3*
b3		*door1*	

Table 4: Entities interaction in level `buttonDoors` in `Lab Recruits`

With the coverage oriented mode, we aim to maximize the ratio of entity coverage metric. In this mode, the RL agent is not geared towards reaching a specific entity, but rather towards *covering all the entities in a level*. For example, in coverage oriented mode in `buttonDoors` level of `Lab Recruits`, the goal of the agent is to cover entities shown in Table 3 with all their possible properties. Measuring the coverage ratio in `Lab Recruits` game is not straight forward. Particularly, measuring the quantitative value of `Entity Connection Coverage` is difficult due to the partial observability issue of the agent. To this end, we follow a probabilistic approach to measure the `Entity Connection Coverage` ratio.

Figure 4 and 5 show the per episode entity coverage achieved during the training phase for `8-room` and `LargeMaze_1` level. Note that curiosity-based RL shows good coverage result in a small and straightforward level like `8-room`, while it shows significant improvement in achieving entity coverage compared to sparse-reward RL and random solution in `LargeMaze_1` level.

We have run a test game-play session to measure the quantitative value of coverage. The result is compared only between two RL solutions as the test session is guided by the respective learned

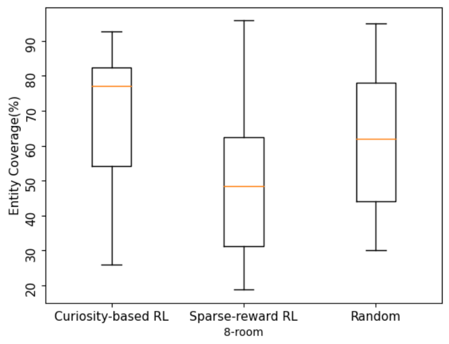

Figure 4: Per episode entity coverage on `8-room` level

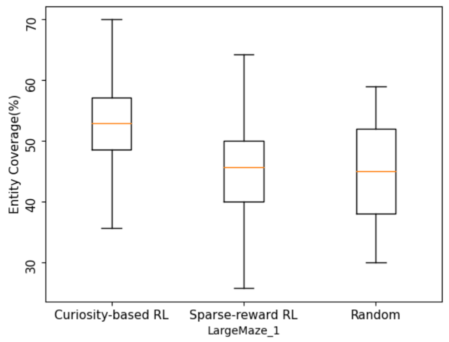

Figure 5: Per episode entity coverage on `LargeMaze_1` level

Q-table. The random solution is not included here as there is no learning process involved in random exploration. For the game-play testing session we also use the same budget (i.e., action per episode, and cycle per action) as training as shown in Table 2. Our goal is to observe the coverage ratio obtained by both RL solutions in limited budget. Coverage results obtained from our experiments are presented in table 5. It is noticed that for a simple and small level like `buttonDoors` both curiosity-based and sparse-reward RL achieve high/full coverage. While curiosity-based has shown high potential to obtain better coverage ratio in a large and complex level like `LargeMaze_1`. Though both RL solutions obtain low coverage for `LargeMaze_2` level, this may be because the learning duration was not enough to acquire an optimal Q-table.

Overall, with the results that we observed on the five levels of `Lab Recruits`, we can confirm that the proposed approach for mapping the game environment into a reinforcement learning problem is promising. Given the fact that the reinforcement learning agent does not have direct interaction with the game, but rather through an autonomous test agent, and that the environment is partially observable, the results obtained are quite encouraging and pave the way for further experimentation with more complex levels of `Lab Recruits` and eventually other 3D games with more complex interaction spaces.

The proposed RL solution has the potential to discover inconsistency and design issue that is valuable to the game designer. For

Environment/ Level	Curiosity RL		Sparse Reward RL	
	Entity Cov	Connect Cov	Entity Cov	Connect Cov
`buttonDoors`	100%	100%	100%	100%
`4-room`	100%	80%	60%	50%
`8-room`	92%	60%	80%	54%
`LargeMaze_1`	80%	50%	40%	37%
`LargeMaze_2`	55%	30%	40%	20%

Table 5: Coverage measure of difference levels of `Lab Recruits` game

example, analyzing the log of the learning process, we have identified two issues related to the placement of a button and the logical connection between a door and button in `LargeMaze_2` level.

6 RELATED WORK

There have been studies [1, 18] on the reinforcement learning solution to automated test generation for Graphical User Interface (GUI) applications. These studies demonstrate how to detect fatal exceptions and achieve high code and activity coverage. To date, a couple of studies have investigated the use of automatic exploration techniques to maximize game state coverage. The authors of [3] introduce a self-learning mechanism to the testing of a First Person Shooter game using deep reinforcement learning (DRL). The Wuji framework [32] employs an RL policy together with evolutionary multi-objective optimization to encourage exploration and high game state coverage in two commercial combat games. [29] investigates the possibility of using RL for load testing video games, i.e., to examine the game's performance. [5] has used Deep RL to increase test coverage,in particular, to find exploits, test map difficulty, and to detect common problems that arise in the testing of first-person shooter (FPS) games. Curiosity has been a subject of research in Reinforcement Learning for a long time [4]. One of the first attempts to consider reinforcement learning agents with a sense of curiosity is [7, 19], where curiosity is formulated as the error in an agent's ability to predict the consequence of its actions. The curiosity-powered agent learns how to interact with the environment by curiosity alone and able to learn skills to finish the game-play [11, 12, 17, 20, 25, 28]. Authors of [10] address the problem of automatically exploring and testing 3D games using RL. They use the notion of curiosity as a motivating factor for the agents to learn the complex navigation mechanics required to reach the different areas around the map, thus providing higher state coverage.

7 CONCLUSION

We have presented an approach based on reinforcement learning for automated testing of 3D games and results from an exploratory experimental study we have carried out on `Lab Recruits`, a 3D game. Our objective is to assess the suitability of reinforcement learning for the purpose of testing 3D games by remaining at a higher level of abstraction when defining the states and actions of the reinforcement learning environment. To this end, we have proposed a curiosity driven reinforcement learning approach with a reward mechanism that enables the reinforcement learning agent to explore the space of interactions in the game. The reward function encourages the discovery of previously unseen states and discourages immobility and revisiting of already seen states. The results from our exploratory study are promising in that we have preliminary evidence that reinforcement learning can be adopted for automated testing of 3D games. In our experiments we used a few levels of `Lab Recruits`, hence the results are not generalizable to a wider range of games. However the results shed light on the potential of curiosity driven reinforcement learning for the purpose of automated testing with a higher level of abstraction in the definition of states and actions. Furthermore, the proposed approach could be easily adapted for testing of other 3D games, provided that there is an implementation of the testing agent interface.

As part of our future work, there are a number of aspects that we would like to work on, and some of them are currently ongoing. We would like to explore further by applying `RLbT` to more levels in `Lab Recruits`, and eventually apply it to other 3D games from the real world, such as Space Engineers[6]. On the other hand, from an empirical perspective, we would like to perform comparative studies to assess the effectiveness of `RLbT` with respect to other testing approaches applicable to 3D games (e.g., [9]).

ACKNOWLEDGMENTS

The work presented is part of the iv4XR H2020 EU project funded under grant number: 856716

REFERENCES

[1] David Adamo, Md Khorrom Khan, Sreedevi Koppula, and Renée Bryce. 2018. Reinforcement learning for android gui testing. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*. 2–8.

[2] Thomas H Apperley. 2006. Genre and game studies: Toward critical approach to video game genres. *Simulation and Gaming* 37, 1 (2006), 6–23.

[3] Sinan Ariyurek, Aysu Betin-Can, and Elif Surer. 2019. Automated video game testing using synthetic and human-like agents. *IEEE Transactions on Games* (2019).

[4] Arthur Aubret, Laetitia Matignon, and Salima Hassas. 2019. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976* (2019).

[5] Joakim Bergdahl, Camilo Gordillo, Konrad Tollmar, and Linus Gisslén. 2020. Augmenting automated game testing with deep reinforcement learning. In *2020 IEEE Conference on Games (CoG)*. IEEE, 600–603.

[6] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[7] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. 2018. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355* (2018).

[8] Avinash K Dixit, John JF Sherrerd, et al. 1990. *Optimization in economic theory*. Oxford University Press on Demand.

[9] Raihana Ferdous, Fitsum Meshesha Kifetew, Davide Prandi, I. S. W. B. Prasetya, Samira Shirzadehhajimahmood, and Angelo Susi. 2021. Search-Based Automated Play Testing of Computer Games: A Model-Based Approach. In *Search-Based Software Engineering - 13th International Symposium, SSBSE 2021, Bari, Italy, October 11-12, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12914)*, Una-May O'Reilly and Xavier Devroey (Eds.). Springer, 56–71. https://doi.org/10.1007/978-3-030-88106-1_5

[10] Camilo Gordillo, Joakim Bergdahl, Konrad Tollmar, and Linus Gisslén. 2021. Improving Playtesting Coverage via Curiosity Driven Reinforcement Learning Agents. *arXiv preprint arXiv:2103.13798* (2021).

[11] Oliver Groth, Markus Wulfmeier, Giulia Vezzani, Vibhavari Dasagi, Tim Hertweck, Roland Hafner, Nicolas Heess, and Martin Riedmiller. 2021. Is curiosity all you need? on the utility of emergent behaviours from curious exploration. *arXiv preprint arXiv:2109.08603* (2021).

[12] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. 2016. Vime: Variational information maximizing exploration. *Advances in neural information processing systems* 29 (2016).

[13] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. 1. *New phytologist* 11, 2 (1912), 37–50.

[14] Guillaume Lample and Devendra Singh Chaplot. 2017. Playing FPS games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[16] Newzoo. 2021. Global games market report. https://newzoo.com/insights/articles/the-games-market-in-2021-the-year-in-numbers-esports-cloud-gaming/

[17] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. 2017. Count-based exploration with neural density models. In *International conference on machine learning*. PMLR, 2721–2730.

[18] Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, and Xuandong Li. 2020. Reinforcement learning based curiosity-driven testing of android applications. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 153–164.

[19] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*. PMLR, 2778–2787.

[20] Alexandre Péré, Sébastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer. 2018. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. *arXiv preprint arXiv:1803.00781* (2018).

[21] Cristiano Politowski, Fabio Petrillo, and Yann-Gaël Guéhéneuc. 2021. A survey of video game testing. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE, 90–99.

[22] Rui Prada, I. S. W. B. Prasetya, Fitsum Meshesha Kifetew, Frank Dignum, Tanja E. J. Vos, Jason Lander, Jean-Yves Donnart, Alexandre Kazmierowski, Joseph Davidson, and Pedro M. Fernandes. 2020. Agent-based Testing of Extended Reality Systems. In *13th IEEE International Conference on Software Testing, Validation and Verification, ICST 2020, Porto, Portugal, October 24-28, 2020*. IEEE, 414–417. https://doi.org/10.1109/ICST46399.2020.00051

[23] ISWB Prasetya, Mehdi Dastani, Rui Prada, Tanja EJ Vos, Frank Dignum, and Fitsum Kifetew. 2020. Aplib: Tactical Agents for Testing Computer Games. In *International Workshop on Engineering Multi-Agent Systems (EMAS)*. Springer, 21–41.

[24] I. S. W. B. Prasetya, Mehdi Dastani, Rui Prada, Tanja E. J. Vos, Frank Dignum, and Fitsum Meshesha Kifetew. 2020. Aplib: Tactical Agents for Testing Computer Games. In *Engineering Multi-Agent Systems - 8th International Workshop, EMAS 2020, Auckland, New Zealand, May 8-9, 2020, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12589)*, Cristina Baroglio, Jomi Fred Hübner, and Michael Winikoff (Eds.). Springer, 21–41. https://doi.org/10.1007/978-3-030-66534-0_2

[25] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. 2018. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274* (2018).

[26] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[27] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[28] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. 2017. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems* 30 (2017).

[29] Rosalia Tufano, Simone Scalabrino, Luca Pascarella, Emad Aghajani, Rocco Oliveto, and Gabriele Bavota. 2022. Using Reinforcement Learning for Load Testing of Video Games. *arXiv preprint arXiv:2201.06865* (2022).

[30] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. 2017. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782* (2017).

[31] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[32] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 772–784.

[6]https://store.steampowered.com/app/244850/Space_Engineers/