Learning Efficient Exploration through Human Seeded Rapidly-exploring Random Trees

Max Zuo* (左绍文) zuo@gatech.edu
Logan Schick* lschick3@gatech.edu
Matthew Gombolay* matthew.gombolay@cc.gatech.edu
Nakul Gopalan* ngopalan3@gatech.edu

***Abstract*—Modern day computer games have extremely large state and action spaces. To detect bugs in these games' models, human testers play the games repeatedly to explore the game and find errors in the games. Such game play is exhaustive and time consuming. Moreover, since robotics simulators depend on similar methods of model specification and debugging, the problem of finding errors in the model is of interest for the robotics community to ensure robot behaviors and interactions are consistent in simulators. Previous methods have used reinforcement learning [8] and search based methods [6] including Rapidly-exploring Random Trees (RRT) to explore a game's state-action space to find bugs. However, such search and exploration based methods are not efficient at exploring the state-action space without a pre-defined heuristic. In this work we attempt to combine a human-tester's expertise in solving games, and the exhaustiveness of RRT to search a game's state space efficiently with high coverage. This paper introduces human-seeded RRT (HS-RRT) and behavior-cloning-assisted RT (CA-RRT) in testing the number of game states searched and the time taken to explore those game states. We compare our methods to an existing weighted RRT [18] baseline for game exploration testing studied. We find HS-RRT and CA-RRT both explore more game states in fewer tree expansions/iterations when compared to the existing baseline. In each test, CA-RRT reached more states on average in the same number of iterations as RRT. In our tested environments, CA-RRT was able to reach the same number of states as RRT by more than 5000 fewer iterations on average, almost a 50% reduction.**

I. Introduction

There is an explosion in model specifications for games, and simulators, be it for entertainment, or to save on physical learning interactions for robots. However, testing for these models is still performed manually which becomes challenging with large, continuous, state-action spaces. The problem of such model testing is well studied in computer games, which will be the focus of this paper. However, our algorithms can generalize to robotics simulators as they can learn and explore in continuous state-actions spaces. Testing a game's model or "play" involves extensive manual testing, where human players play the game multiple times and try to reach every possible state with the purpose of reporting bugs and inconsistencies. However, as games continue to grow in complexity, with larger action and state spaces, it is no longer feasible to test a game's model using human game play alone. In this paper we propose a method that learns heuristics for a search based algorithm using a few human demonstrations. These heuristics allow the search algorithms to automatically test other states in the game removing the necessity to test all possible scenarios in the game using human game plays.

Previously, search based methods have been used to find goals in a task quickly using their capacity to explore [9] [4] [7] [13]. However, traditional search based algorithms can take large amounts of time and compute power when a specific sequence of actions is required to advance. For example, in most games the action to *open a door* is allowed in all states. However, a door would only open as a result of this action when the agent has taken actions to pick up a key already, and be present next to a door. Search algorithms spend a lot of compute and time to discover these long sequence of actions. A human tester can select these actions rather trivially in comparison. Previous methods have attempted to use human capacity to advance in games using expert-programmed heuristic [6] or by using a pre-specified distribution of human preferences [18].

In this paper we present two novel algorithms that utilize human demonstrations to explore a game's search space. First, we present Human Seeded RRT (HS-RRT) which uses RRT from locations demonstrated by a human user. Second, we present Clone Assisted RRT (CA-RRT) which uses behavior cloning to generalize new locations from where RRT can be used to explore a game allowing generalization to unseen states. We compare these algorithms on their ability to find novel states in the least number of RRT node expansions. In our experiments, we find HS-RRT and CA-RRT require about half as many node expansions to search the entire search space when compared to existing algorithms on gym-minigrid domains. Our contributions in this work are:

1) HS-RRT: which uses human demonstrations as starting states to explore the a game's states rapidly.
2) CA-RRT: which uses behavior-cloning and RRT to explore game states randomly given novel unseen states.
3) Results showing CA-RRT outperforms previous weighted RRT baseline [18] by using fewer node expansions to explore the entire state space in two gym-minigrid domains: **DualHallway** 6836 expansions vs 12667, and **CascadingLockDoor** 6659 vs 12592.

II. Related Work

Games and simulators are becoming pervasive in entertainment and automation. These systems require testing to verify their model specification. Automatic testing tools are becoming common to help in game testing and verification.

* *School of Interactive Computing, Georgia Institute of Technology*

Aghyad and Moataz [1] explore a wide breadth of these methods, including evolutionary algorithms, genetic algorithms, and graph search algorithms for searching game state spaces. The Ubisoft development teams for "Tom Clancy's: The Division" implemented bots to test the game servers and to test procedurally-generated levels in the game respectively [16]. Electronic Arts uses A* bots to automate goal-driven gameplay to find discrepancies and flaws within their games [6]. Curiosity-driven reinforcement learning [14] agents have also been used to improve gameplay coverage for testing [8]. Search algorithms like A* are good at finding paths to an end-goal given a pre-specified heuristic. Such end-goals and heuristics are hard to specify for large games.

Rapidly-exploring random trees (RRT) is traditionally used for path planning [12]. An RRT is biased to explore large unexplored regions in the state-space to find a path to the goal location [12]. By removing the goal configuration in search and simply focusing on expanding, RRT can be utilized to explore large state spaces *without an end goal*. Such exploration helps cover all possible states in the game helping test corner cases that can be reached by a player. RRTs have been used to explore Super Mario World [18] among other games [17]. Instead of trying to optimize game score or directly optimizing game state coverage, Zhan et al. [18] focused on maximizing diversity of game states visited in an analogous way to how some RL intrinsic motivation methods work [3] [14].

Some researchers have looked into using human input as guidance for search or planning algorithms [4] [9] [13] [7]: HA-RRT and Human-Guided RRT (HG-RRT*), for example, utilize human expert specifications for assisting exploration of a virtual urban environment and motion planning [13] [7]. Zhan et al. [18] find that human play-testers explore rapidly initially but exploration quickly fatigues, and over sufficient amount of time, RRT can overtake humans in number of states explored. In this work we aim to combine both these methods via learning heuristics for search algorithms. Even though a behavior cloning may not be able to complete an environment by itself, we believe the clone will be able to efficiently support the search process, similar to DeRRT* [11] and AlphaGo [15]. These previous methods used demonstrations to reduce the effective branching factor for search allowing an agent to find a goal fast. We, on the other hand, will use the demonstrations to seed novel RRT trees to search the state space rapidly for bugs, and we are not looking for an individual goal state but to ensure that there are no bugs in any of the states of a game or simulator.

III. Methodology

In this section we describe the two algorithms we developed HS-RRT: and CA-RRT. Previous algorithms that use RRT for exploration such as HA-RRT [13] and HG-RRT* [7] allow guidance from anywhere in the configuration space. They do so by allowing users to specify waypoints, avoidance points and drawing guidance paths. However, these methods do not allow direct modification or influence of the RRT trees and therefore cannot fully take advantage of human demonstrations. Furthermore, HG-RRT* waypoints or HA-RRT paths must be redefined or redrawn for every small change in the environment, leaving humans tethered to the search process. We aim to first allow direct influence of the RRT tree with HS-RRT, then decouple human oversight from the search/exploration process through CA-RRT.

A. HS-RRT

Human Seeded RRT (HS-RRT) directly uses a human tester's game play to improve coverage. Each visited state, or a subset of visited states, in the game play are then used as a collection of configurations to add to RRT's tree directly. Therefore, when RRT begins searching, the tree is no longer expanding from just the initial game state, but can expand from any of the configurations states demonstrated by a user in their game play. In order to make RRT explore widely instead of pathfinding to a particular goal, like Zhan et al. [18] we run RRT with no particular goal in mind, simply randomly sampling neighbors in the state space. This is not possible with RRT*, which requires a cost function w.r.t. to an end goal which we do not need for this problem [10]. The full HS-RRT method described in pseudocode in Algorithm 1. Notice that the RRT tree takes in all the user visited nodes as known configurations spaces to expand from in line 9 of Algorithm 1.

Algorithm 1 HS-RRT

```
1: function HS-RRT(env, initial_configuration)
2:     visited ← {initial_configuration}
3:     while ¬done do   ▷ Collect human trajectory and to visited configurations
4:         action ← human.read_action()
5:         env.step(action)
6:         current_configuration ← env.conf()
7:         visited ← visited ∪ {env}
8:     end while
9:     RRT(env, visited)
10: end function
```

B. CA-RRT

Because HS-RRT requires collecting human trajectories and using human-visited states as seed nodes in its tree, HS-RRT can only be run on an instantiation of a game where a human has played on, limiting our ability to test different initial conditions or even slightly changed game configurations: we would need to collect a human trajectory in each version of an edited game. To ensure that our agent can explore unseen environments we developed CA-RRT, which is a (behavioral) Clone-Assisted RRT searching algorithm. CA-RRT starts with collecting a set of human-created trajectories, once, to train a behavioral clone using the state-action pairs from those trajectories. Before RRT runs on the environment, we use the behavior cloned policy to rollout for some number of timesteps similar to HS-RRT and use the clone-collected

trajectory of state-action pairs to seed our search. Because CA-RRT produces its own trajectories, there is no reliance on a human operator to seed the search each time as is the case with HS-RRT. CA-RRT can therefore run with different initial configurations and on environments where the clone was not trained on at all, as it does not need a human player to collect trajectories after the initial training phase. During RRT, the clone agent produces the action probabilities for every state which RRT uses as a prior to sample from.

During evaluation of CA-RRT, we still wish to fully explore the range of game states, not simply the expected trajectory learned by the behavioral cloning training. Therefore, we use a Laplace smoothing over the action probability distribution, and the factor α increases after every step allowing the agent to choose actions more uniformly.

IV. Experiments

Our work is evaluated in the OpenAI Gym [2] library with custom gym-minigrid [5] environments built to be extremely challenging if not given a detailed and extrinsic reward function. We compare the results of the HS-RRT method and our CA-RRT method to the vanilla RRT method from Zhang et al. [18]. The new environments are publicly available in a forked repository.

A. Gym-minigrid

Gameplay-testing aims to evaluate and test as many reachable game states as possible, if not all of them. In many environments, the number of game states varies from one play to another. Often times the number of game states can be immeasurably large: the game state could be a continuous space, or the branching factor given the number of possible actions explodes exponentially. In order to conduct controlled experiments with calculable ground truth state coverage, we built a set of our own environments on top of the gym-minigrid [5] library with a countable number of states. Gym-minigrid [5] describes a family of lightweight environments built for tasks such as reinforcement learning. Every environment has the same basic building blocks: the agent, doors - which may require a key to open, walls, balls, etc. The agent can only pick up one item (e.g. key, ball) at a time. The agent is provided with a partial observation of the state of the game from its perspective, much like how a first-person game limits what the human player can view. We also created a new, tougher building block which we employ and recommend using, the **MultiLockDoor**, which requires several keys to open.

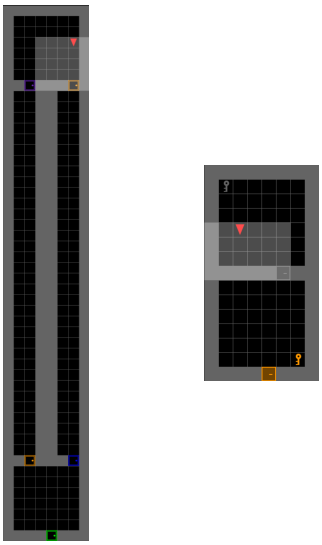


Fig. 1: The environments tested on. From the left: **DualHallway-v0**, **CascadingLockDoor-v0**.

1) DualHallway Environment: The first environment experimented in is the **DualHallway** environment which invites a playing agent to open a few doors to reach the end goal room through paths that can include two possible locked hallways. In order to cover a large portion of the game state space effectively, a game play testing agent must efficiently explore all hallways and rooms, which will require the agent to use the *toggle* action. These doors are not locked. Door location, door color, agent orientation, and agent position are all randomly assigned. See example in Figure 1.

2) CascadingLockDoor Environment: This environment incorporates the **MultiLockDoor** object. There two rooms, where the gameplaying agent is asked to reach the end of the second room. In each room, there is a key which corresponds to the door at the end of the room, which the gameplaying agent must pick up and use on its corresponding door. The first key must be used on the second lock as well. See example in Figure 1.

B. Experiment Setup

For both of these environments, a single trajectory by a human player was collected and used to train a crude behavioral cloning model. We then run RRT, HS-RRT, and CA-RRT on each of these environments and measure how quickly each algorithm was able to saturate, i.e., explore all states of, the game state space. RRT* [10] is not explored in here as the main goal is not to find the optimal path through a configuration space, but to rather explore through the entire space. We can now directly calculate the upper limit of reachable states by counting the number of discrete possible reachable game map locations. Game map coverage can still be used in a continuous state space, if we were to compute a game state hash to represent whether or not a state/discretized map location has been reached before.

The CA-RRT policy is a convolutional neural network with an input of images with the size $(56, 56, 3)$. The architecture used is a convolutional layer, followed by a residual block, repeated, followed by two fully-connected layers both with 16 output nodes, and a final layer with an output size of 7, the number of actions allowed. A dropout layer is used with a dropout rate of 10%. Any similar learner would have been mostly equivalent in effectiveness. For each experiment, the agent position and orientation, door color and position, and key position are all permuted. Therefore, CA-RRT and RRT were never evaluated on the exact instantiation of an environment seen during training time. For **DualHallway** environments, the Laplace smoothing factor was $\alpha = 0.1$ which increased at a rate of 10^{-5} every iteration, and for **CascadkingLockDoor**, $\alpha = 0.1$ with a growth rate of 10^{-3}. Once $\alpha \geq 0.5$, CA-RRT defaulted to hand-defined weights. The hand-crafted weights used may not be optimal. However, the weights were designed by domain knowledge, and empirically chosen for their performance.

We differentiate between versions of RRT where actions are sampled uniformly versus RRT where actions are sampled over a user-defined distribution. Building a user-defined distribution requires domain knowledge of the problem, and is explored in prior works [18].

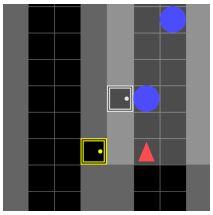


Fig. 2: **HeavyBall** and distractor doors in the **DualHallway** environment

Distractors: For some **DualHallway** experiments, during test time we introduce new challenges for our behavior cloning agent it hasn't seen before to test CA-RRT's generalization capabilities. First, we flip the hallway onto its side by rotating the whole thing by $90°$. We also narrow the entire environment by one cell. This is the **SidewaysDualHallway** setup. Next, we introduce a setup with 5 distractor doors at random locations in random colors along the middle wall. This is the **DualHallway + Distractor** setup. Finally, on top of the distractor doors we place 5 *immovable* **HeavyBall** items randomly into the environment. This is the **DualHallway + Distractors & Obstacles** setup. This setup reduces the number of total states possible as a heavy ball can be in the path of exploration (see Figure 2).

Note: HS-RRT cannot be run in the **SidewaysDualHallway**, **DualHallway + Distractors**, or **DualHallway + Distractors & Obstacles** setups as only one trajectory from one instantiation of the original **DualHallway** setup was collected. We introduce these additional setups mainly as a method for testing generalization abilities for CA-RRT.

V. Results

HS-RRT was evaluated solely on the one instantiation of the environment where human data was collected. All other methods were evaluated on a large set of permutations of the environment: no two runs for CA-RRT or RRT were evaluated on the same instantiation. Even with just one trajectory from a human game player, CA-RRT improves the effectiveness with which we explore a game's state space. In all figures, colored areas represent 5th to 95th percentile, and the solid colored line represents the median.

A. DualHallway

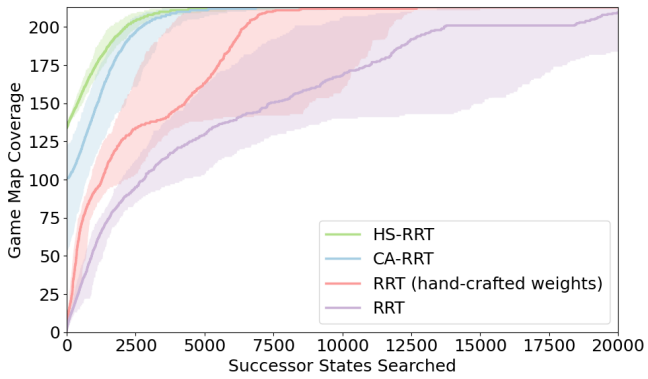
For the experiment on the **DualHallway** original setup (see Figure 3a), HS-RRT is seeded with a human-curated trajectory, which in the original setup totalled 134 out of the 213 total states. CA-RRT, with no reliance on instantiations of the environment involving human-produced trajectories, was able to roll out and visit an average of 101 states in the original setup to seed RRT search in the original setup, and on average saturated the game space a full 5831 iterations (46% fewer iterations) before weighted RRT.

In the **SidewaysDualHallway** setup (see Figure 3b), CA-RRT average performance dips, where the average number of states visited during rollout was 83, and fully saturated the game state space only 1617 iterations (18% fewer iterations) before weighted RRT did on average. However, only 94% of weighted RRT experiments were able to saturate the game state space in $20,000$ iteration compared to all 100% of CA-RRT experiments.

For both the **Distractors** (Figure 4a) and **Distractors & Obstacles** (Figure 4b), RRT with hand-crafted weights' performance curves changed minimally, but CA-RRT's performance takes a noticeable hit. When only adding distracting doors, we see CA-RRT saturates the game space in 7033 fewer iterations than weighted RRT (54% fewer iterations) CA-RRT was to roll out nearly as far as in previous environments (16 states on average for **Distractors & Obstacles**). On average, the CA-RRT algorithm saturates the state space more efficiently than the other two RRT methods, but its advantage throughout time has diminished significantly. Nevertheless, our experiments find in the **Distractors & Obstacles** setup CA-RRT saturates the game state space 4636 (33% fewer iterations) steps before weighted RRT does on average, suggesting weighted RRT struggled to expand to the final last few states more than CA-RRT.

B. CascadingLockDoor

These graphs show the difficulty of reaching the second room (locked door is 37th state: see y-value in Figure 5a and Figure 5b). The first plateau in Figure 5a is where the RRT methods got stuck trying to unlock the first room to go to the second room. Median statistics: HS-RRT took the shortest amount of time, saturating the game state space in 1269 iterations (90% fewer iterations than weighted RRT). The CA-RRT algorithm reached the second room in 5933 fewer iterations (47% fewer iterations) than weighted RRT.



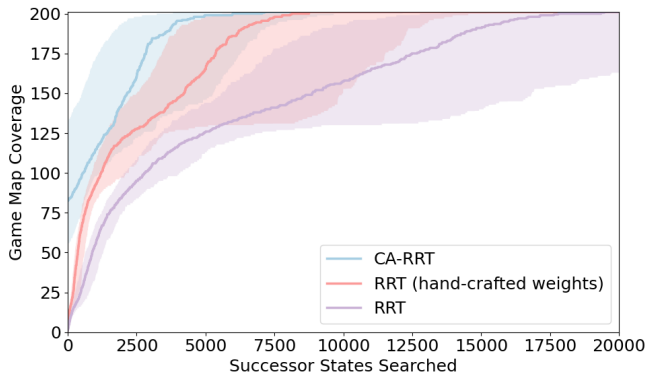
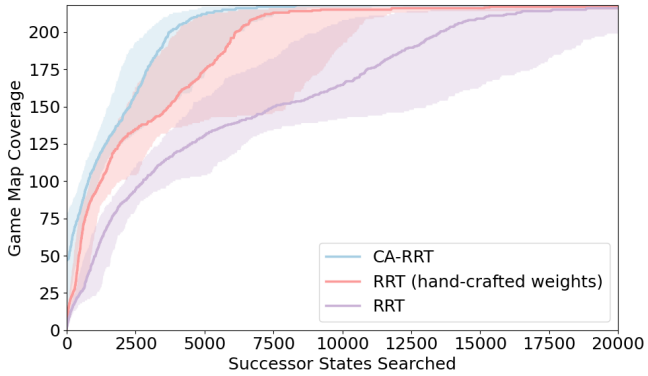


Fig. 3: The plots show the total number of novel reachable states explored w.r.t. iterations of RRT expansions. (a) **DualHallway** - original setup. HS-RRT performs better in this task compared to all other algorithms because it is being tested on the same environment instance repeatedly. (b) **SidewaysDualHallway**. CA-RRT performs better in this task as it can generalize to novel environment instances with completely different orientations because it uses the human demonstrations to learn heuristics for search.



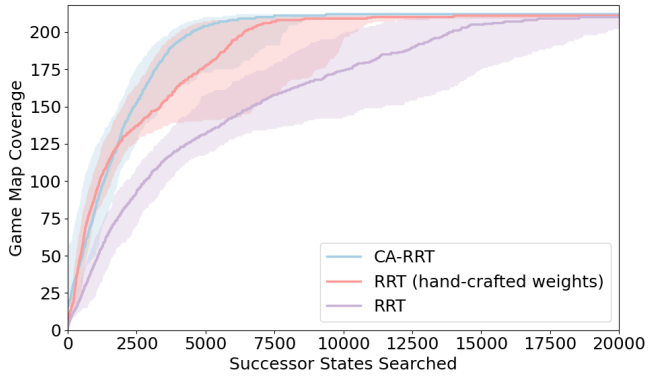


Fig. 4: The plot shows the total number of novel reachable states explored w.r.t. iterations of RRT expansions. (a) **DualHallway + Distractors**. CA-RRT performs better in this task given its ability to generalize. HS-RRT cannot be run as it is tethered to the environment instance where demonstrations were collected. (b) **DualHallway + Distractors & Obstacles**. CA-RRT performs better in this task than weighted RRT on average but advantage is reduced.

RRT without weights was not able to reach the second room even after $20,000$ iterations in the average case, remaining trapped.

CA-RRT, weighted RRT, and vanilla RRT all have drastically varying ranges of performance, with CA-RRT's 5th percentile performance worse than the median weighted RRT performance. Neither CA-RRT nor weighted RRT were able to solve the environment's last locked door in the 5th-95th percentile range, although CA-RRT was able to solve it in 2% of experiments, and weighted RRT did not solve it once. CA-RRT shows an improvement over weighted RRT, but still leaves significant more room to improve upon showing that these domains are challenging and require novel methods.

VI. Discussion

When evaluated on challenging custom gym-minigrid environments, it was found that both HS-RRT and CA-RRT were able to reach more meaningful states when compared to the baseline weighted RRT [18] algorithm. It does, however, remain possible to define more optimal hand-crafted weights, as we only chosen empirically. We showed that even with one human trajectory, the efficiency of exploration of a game's state space can be improved using CA-RRT. Using our method, developers could also target specific sections of the game for bug testing by initializing HS-RRT with trajectories from those areas.

However, there are several areas where CA-RRT falls short. First, real compute time: because our environments are very light, RRT iterations are inexpensive. Further, since CA-

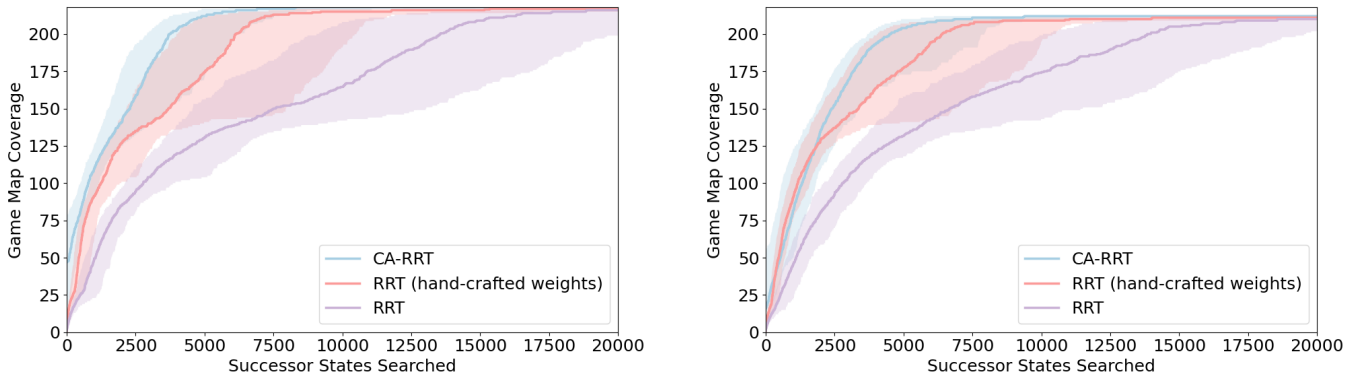


Fig. 5: The plot shows the total number of novel reachable states explored w.r.t. iterations of RRT expansions. (a) **CascadingLockDoor**. HS-RRT performs better in this task since human demonstrations initiailize RRT with access to both rooms immediately. (b) **CascadingLockDoor**. Same data as (a), but presenting only CA-RRT and weighted RRT for clarity.

RRT relies on a behavior cloning agent to produce action distributions at every step, using CA-RRT to explore may result in longer real compute time. In our experiments, when CA-RRT performed poorly, it took more than twice as long as RRT with hand-crafted weights running on a laptop. There are steps which can be taken that can reduce compute time (batch compute on GPU, store agent predictions, etc) which we have not yet explored. As the number of iterations correlates directly with the size of the tree computed, it is also possible that the cost of loading/saving all searched states becomes unsustainable.

VII. Conclusion

As games continue to grow in size, the work required to find bugs and flaws in the game state space grows, motivating for efficient automated game testing. Previous literature has explored the use of methods like weighted RRT for game testing [18]. As demonstrated, vanilla RRT can easily produce a trapped agent if not assisted in some form. We find HS-RRT and CA-RRT to be more useful in covering game state space for testing compared to the baseline RRT method proposed in previous literature [18]. While the focus of this paper is efficient wide state space exploration and saturation, we hope that an adapted CA-RRT method, like other similar human-guided RRT methods [9] [4] will be able to also help robots find goal conditions more efficiently.

References

[1] Aghyad Mohammad Albaghajati and Moataz Aly Kamaleldin Ahmed. Video game automated testing approaches: An assessment framework. *IEEE Transactions on Games*, pages 1–1, 2020.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[4] Américo De Jesús Caves Corral Caves. *Human-automation collaborative RRT for UAV mission path planning*. PhD thesis, Massachusetts Institute of Technology, 2010.

[5] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

[6] Fernando de Mesentier Silva, Igor Borovikov, John Kolen, Navid Aghdaie, and Kazi Zaman. Exploring Gameplay With AI Agents. *arXiv e-prints*, page arXiv:1811.06962, November 2018.

[7] Néstor García, Raúl Suárez, and Jan Rosell. Hg-rrt*: Human-guided optimal random trees for motion planning. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–7, 2015.

[8] Camilo Gordillo, Joakim Bergdahl, Konrad Tollmar, and Linus Gisslén. Improving playtesting coverage via curiosity driven reinforcement learning agents. *arXiv preprint arXiv:2103.13798*, 2021.

[9] Alina Griner. *Human-RRT collaboration in Unmanned Aerial Vehicle mission path planning*. PhD thesis, Massachusetts Institute of Technology, 2012.

[10] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[11] Yen-Ling Kuo, Andrei Barbu, and Boris Katz. Deep sequential models for sampling-based planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6490–6497. IEEE, 2018.

[12] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[13] Siddhartha S. Mehta, C. Ton, Michael J. McCourt, Z. Kan, E.A. Doucette, and W. Curtis. Human-assisted rrt for path planning in urban environments. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 941–946, 2015.

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

[15] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[16] Tommy Thompson. The secret ai testers inside tom clancy's the division, 2020.

[17] Jonathan Tremblay, Pedro Torres, and Clark Verbrugge. An algorithmic approach to analyzing combat and stealth games. pages 1–8, 08 2014.

[18] Zeping Zhan, Batu Aytemiz, and Adam M Smith. Taking the scenic route: Automatic exploration for videogames. *arXiv preprint arXiv:1812.03125*, 2018.