Learning to Take Cover with Navigation-Based Waypoints via Reinforcement Learning

Tim Aris*, Volkan Ustun, Rajay Kumar****

*U.S. Army Combat Capabilities Development Command – Soldier Center (DEVCOM SC) Simulation and Training Technology Center (STTC), 12423 Research Parkway, Orlando, FL 32826,

**University of Southern California Institute for Creative Technologies, Playa Vista, CA, USA
timjaris@gmail.com, ustun@ict.usc.edu, kumar@ict.usc.edu

Abstract

This paper presents a reinforcement learning model designed to learn how to take cover on geo-specific terrains, an essential behavior component for military training simulations. Training of the models is performed on the Rapid Integration and Development Environment (RIDE) leveraging the Unity ML-Agents framework. This work expands on previous work on raycast-based agents by increasing the number of enemies from one to three. We demonstrate an automated way of generating training and testing data within geo-specific terrains. We show that replacing the action space with a more abstracted, navmesh-based waypoint movement system can increase the generality and success rate of the models while providing similar results to our previous paper's results regarding retraining across terrains. We also comprehensively evaluate the differences between these and the previous models. Finally, we show that incorporating pixels into the model's input can increase performance at the cost of longer training times.

Introduction

Reinforcement learning (RL) aims to produce optimal policies in given environments. While there has been significant progress, learning to navigate a 3D terrain remains challenging for RL systems.

The specific task chosen was for the agent to take cover in realistic environments from a stationary opponent or opponents by finding a location where the agent is not visible from the opponents' perspective. This paper expands on previous work (Aris et al. 2022) by introducing a waypoint movement system, and changing the number and location of enemies. Additionally, while the previous paper focused solely on raycast-based observations, similar to the LIDAR system in autonomous cars, this work analyzes the effect of including pixels into the observation.

The motivation behind this selection is to explore learning robust neural behavior models that can be used as a component of more complex behaviors in the Rapid Integration and Development Environment (RIDE), a military training simulation environment that can interface with the Unity game engine (Hartholt et al. 2021). Thus, agents were trained in the RIDE platform using geo-specific terrains and leveraging the ML-Agents framework within Unity (Juliani et al. 2018).

The previous paper involved a very granular action space. Agents (hereafter referred to as "fine-grained agents") would move very small distances per timestep, and episodes would have a large number of timesteps. This paper shows the movement of traversing waypoints, with shorter episodes and moving more distance per timestep, results in better generality and faster execution times. Furthermore, it shows that including pixels in the input can improve generality at the cost of training time.

Background

Many RL systems operating in 3D environments use pixels for observations. For example, DeepMind's open-ended learning agents (O.E.L. Team et al. 2021) take information about their goal and an object they may be holding, but the main observation is an RGB pixel image. Likewise, the primary observation in the MineRL environment (Guss et al. 2019) is a grayscale image.

One example of something similar to this paper is OpenAI's hide-and-seek environment (Baker et al. 2019), which uses LIDAR-inspired rays for agents to find ways to be occluded from other agents. However, their focus is on multi-agent scenarios rather than navigating real world terrain. Other work on analyzing raycasts in Unity include dodgeball (Gorgan et al. 2019), navigation (Alonso et al 2020), and self-driving cars (Hossain et al. 2019).

ML-Agents is a toolkit that facilitates the training of agents from within the Unity game engine. RIDE is a middleware that can utilize the Unity game engine to facilitate rapid development and prototyping of simulated environments. Furthermore, RIDE supports Machine

Copyright © 2023, by the authors. All rights reserved.

Learning experiments through the ML-Agents framework, allowing to directly train RL agents within the high-fidelity military training simulation environments based on geo-specific terrains.

Taking Cover

"Taking cover" is defined as having more than 24 out of the 27 fixed points on the agent occluded from a stationary opponent. In other words, the goal is to have an object in between the agent and the opponent such that the agent is no longer visible. An agent is also considered in cover if they are sufficiently far away from the opponent because it would be impractical for an opponent to shoot the agent from that distance. However, in the terrains that have been used, it's almost always faster to take cover by hiding behind an object. Finally, an agent is considered to have failed if they have not taken cover within the maximum number of timesteps for the episode, which is 87 for these experiments. This number was chosen to make a fair comparison between the agents, as it makes the total distance each type of agent can move by the end of the episode the same.

The Agents

Agents were trained using the PPO algorithm (Schulman et al. 2017). The primary observation of the agent consists of Unity's rays, though pixel inputs were also included for one experiment. The shape of these rays is described in Figure 1. Image data is taken from 4 cameras surrounding the agent. Unlike fine-grained agents, rotation is not in the action space, so having 4 cameras guarantees that any nearby cover spot is visible to the agent. The observation space is stacked from the previous two timesteps.

The action space consists of the agent choosing which of 8 adjacent waypoints (cardinal directions and diagonals) to move to. There is no null action. When a waypoint is visited, the environment attempts to create adjacent waypoints in any direction that does not have one. If a waypoint in a direction does not exist, then that action will be pruned via action masking. Reasons for a missing waypoint can include: there was no valid navmesh position at that location, there was no valid path on the navmesh to get to that waypoint, or because the path found was far longer than the distance between the waypoints.

Upon successfully completing an episode, agents are rewarded with a quantity equal to 5 + 5X, where X is the percentage of the timesteps left in the episode. Such reward design gives a solid signal to distinguish just barely completing the episode before the time runs out and provides an additional incentive for speed. The agents also have a step penalty of 1/max_steps, so in practice, rewards range from -1 to just under 10. Exploratory experiments have shown that our models are not very sensitive to changes in reward structure, so we have kept this as the reward design.

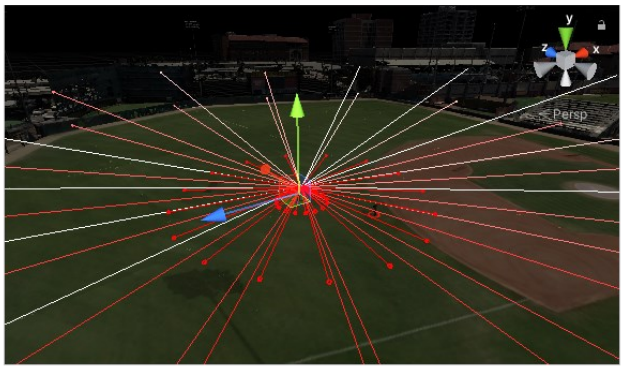

Figure 1: The rays that make up the majority of the agent's observations. These rays are represented to the agent as, for each ray, the length of the ray, and whether the ray hit the opponent, terrain, or nothing. The shape above was chosen for the virtue of being the most useful for a human player when the human tried the task with only rays visible.

The Environment

Agents were trained on the geo-specific, drone-captured terrains of the University of Southern California (USC) campus, Catalina island, and Razish village at the National Training Center at Fort Irwin, CA. The latter is used for military training. Agents spawn in 16x16 unit square areas on the terrain. In the close enemy scenario, enemies also spawn in that square.

Areas are chosen by sampling a random point on the Nav Mesh, checking whether the point is contiguous with the main mesh (to avoid roofs or places where occlusion is impossible), and checking if the square around the point is on the mesh to avoid agents spawning into walls. The time agents train is sped up to the extent possible without destabilizing the agent movements.

In the far enemies scenario, there are two conditions for where an enemy can spawn. One is that it can see the entire arena where the agent can spawn. The other is that there is a path of other enemy spawn points leading to the arena, because locations were generated using breadth first search. Calculating this in real time is intractable, so enemy locations are calculated beforehand and saved to a text file.

Experiments

This paper showcases five comparisons of agents.

1. It examines the effectiveness of differing waypoint distances.
2. It compares waypoint agents to the previous fine-grained agents.
3. It looks at the effectiveness of transfer learning of waypoint agents.
4. It compares the effectiveness of adding pixels to the observation space.
5. Finally, it compares the effectiveness of training with one near enemy versus three far enemies.

Experiments are done using curriculum learning (Narvekar et al. 2020). In our earlier work, we haven't observed an improvement in the final performance using curriculum learning. Yet, it provided a good metric to monitor progress and hence, we utilized it again in this effort.

Agents are tested on different starting locations on the terrains than they were trained on.

Waypoint Distance

As can be seen in Figure 2, for models tested outside of their home terrain, reward tends to increase as the waypoint distance increases. There is a tradeoff, however: if the distance is too large, then the likelihood of the agent finding a cover spot that is far from the closest cover spot increases. Additionally, the human normalized score of the agents tends to decrease with larger waypoint distances. Waypoint distance 4 is the highest value that has positive values, meaning the performance is due to the quality of the model, not the ease of the environment. Thus, the distance 4 was chosen as the smallest distance with close-to-optimal performance.

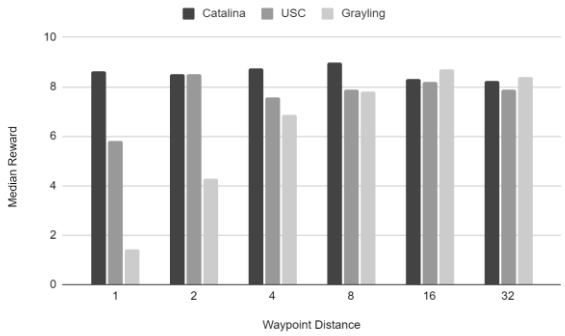

Figure 2: Average reward of models versus the distance between waypoints they were tested and trained with. Models were tested in the Catalina terrain.

Comparison to Previous Agents

Compared to fine-grained agents, waypoint agents generalize to other terrains better. As can be seen in Table 1, on terrains they weren't trained on, waypoint agents have both a higher success rate and a lower number of timesteps to find cover.

However, there is a downside to either waypoint agents or the far enemies scenario. When retraining from a difficult terrain to an easier one, fine-grained agents will have a significantly lower training time compared to an uninitialized agent. In contrast, waypoint agents trained with far enemies take the same amount of time to reach peak performance whether or not they were trained in another terrain first. This effect can be seen in Figure 3.

Table 1: Comparison between models with fine grained actions and models with waypoint actions. The first number is the median finish time, the last number is the success rate, and for the waypoint models, the middle number is the equivalent number of timesteps if they were moving at the same speed as the fine grained agents.

	USC Test Set	Catalina Test Set	Razish Test Set
USC Fine Actions	932 (83%)	2294 (80%)	3282.0, (54%)
Catalina Fine Actions	1481 (72%)	819 (91%)	5000.0, (13%)
Razish Fine Actions	5000.0, (42%)	1146.5, (75%)	1608.0, (79%)
USC Waypoints	18, 1034, (85%)	10, 575, (91%)	38.5, 2213, (79%)
Catalina Waypoints	19, 1092, (79%)	9, 517, (98%)	39.0, 2241, (70%)
Razish Waypoints	20.0, 1149, (70%)	11.0, 632, (91%)	19.0, 1092, (95%)

Near versus Far Agents

Next is comparing waypoint agents trained with one nearby enemy to three far enemies. Previous experiments, both in this paper and the previous one, consisted of agents taking cover from enemies that spawn very close to the agent's starting location. Thus, to make behaviors more in line with taking cover in realistic conditions, the far enemy scenario was created.

As can be seen in Table 2, when tested with far enemies on the same terrain they were trained, far models generally outperform near models. Far-enemy-agents outperformed their near-enemy-agent counterparts by +1%, +5%, and +14% on Catalina, USC, and Razish respectively. However, on any terrain the model wasn't trained on, there is no consistent difference in the performance of the near and far models. Far models are comparable to near models in near-enemy scenarios.

This shows that training with far enemies makes the agents more robust to changes in enemy positions, but that effect is overshadowed by the difficulty of switching to an unfamiliar terrain.

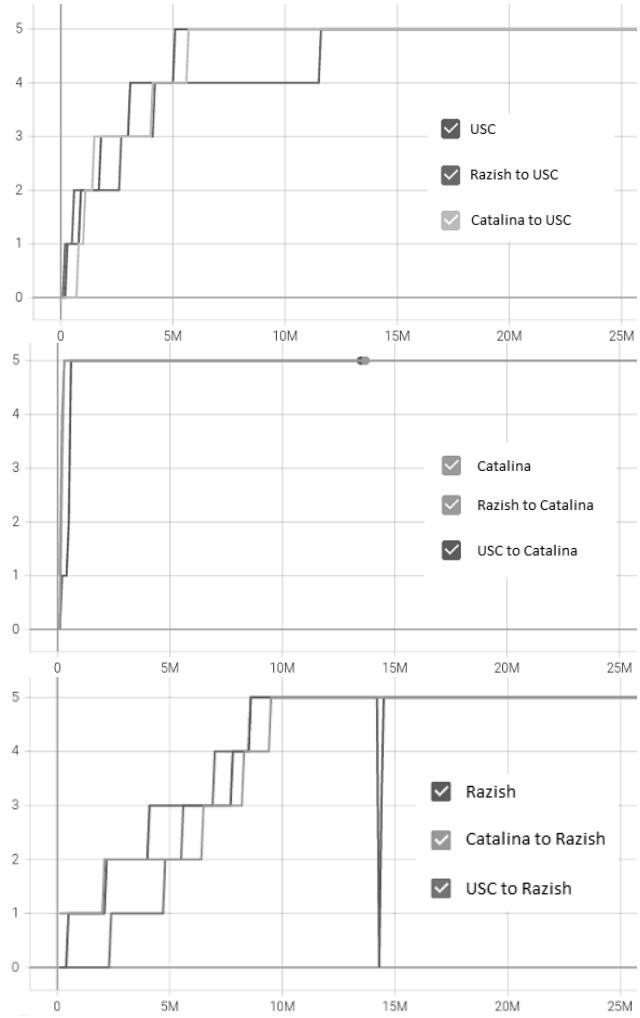

Figure 3: The lesson number of the models over time, as trained on the three terrains. The lesson number sets the number of areas the agent trains in, from 4 to 128.

Table 2: A comparison of agents trained with 1 nearby enemy to agents trained with 3 far enemies. Both close and far agents were trained on the three terrains, then tested on each combination of terrain and enemy location. The bolded numbers are the experiments that are also used for the comparisons in Table 3.

	USC Close Set	USC Far Set
USC Close Model	18.0, 1034, (85%)	27.0, 1552, (77%)
USC Far Model	16.0, 920, (88%)	**25.0, 1437, (82%)**
Catalina Close Model	19.0, 1092, (79%)	34.0, 1954, (67%)
Catalina Far Model	20.0, 1149, (71%)	**37.0, 2126, (59%)**
Razish Close Model	20.0, 1149, (70%)	30.0, 1724, (61%)
Razish Far Model	21.0, 1207, (72%)	**36, 2069, (63%)**

	Catalina Close Set	Catalina Far Set
USC Close Model	10.0, 575, (91%)	18.0, 1034, (86%)
USC Far Model	9.0, 517, (91%)	**20.0, 1149, (81%)**
Catalina Close Model	9.0, 517, (98%)	14.0, 805, (94%)
Catalina Far Model	8.0, 460, (98%)	**14.0, 805, (95%)**
Razish Close Model	11.0, 632, (91%)	19.5, 1121, (82%)
Razish Far Model	11.0, 632, (88%)	**21.0, 1207, (75%)**

	Razish Close Set	Razish Far Set
USC Close Model	38.5, 2213, (79%)	65.0, 3736, (59%)
USC Far Model	35.0, 2011, (78%)	**54.0, 3103, (63%)**
Catalina Close Model	39.0, 2241, (70%)	55.0, 3161, (61%)
Catalina Far Model	34.0, 1954, (74%)	**49.0, 2816, (63%)**
Razish Close Model	19.0, 1092, (95%)	23.0, 1322, (76%)
Razish Far Model	19.0, 1092, (96%)	**25.0, 1437, (90%)**

Pixels

When including pixels into the far enemies scenario, agents show an increase in generalization at the cost of training times. As shown in Table 3, the success rate of the pixels-and-rays models was within 5% of the ray models on the terrain they were trained on. In 5 out of 6 times the models were tested on a different terrain, the pixels-and-rays models outperformed the ray models by over 5%.

The ray agents trained at an average rate of 700,000 timesteps per hour, while the pixels-and-rays agents trained at a rate of 80,000 timesteps per hour.

Table 3: A comparison of models with ray observations and models with both rays and pixels. Like before, the first 3 numbers are the median finish time, equivalent finish time for fine-movement agents, and success rate. The 4th number in the bottom half is the net gain in success rate from adding pixel inputs to the model.

	USC Test	Catalina Test	Razish Test
USC Rays	**25.0, 1437, (82%)**	**20.0, 1149, (81%)**	**54.0, 3103, (63%)**
Catalina Rays	**37.0, 2126, (59%)**	**14.0, 805, (95%)**	**49.0, 2816, (63%)**
Razish Rays	**36, 2069, (63%)**	**21.0, 1207, (75%)**	**25.0, 1437, (90%)**
USC Rays and Pixels	27.0, 1552, (81%), ***(-1%)***	16.0, 920, (89%), ***(+8%)***	44.0, 2529, (74%), ***(+11%)***
Catalina Rays and Pixels	35.0, 2011, (68%), ***(+8%)***	18.0, 1034, (91%) ***(-4%)***	62.0, 3563, (54%), ***(-9%)***
Razish Rays and Pixels	37.0, 2126, (71%), ***(+9%)***	30.0, 1724, (81%), ***(+6%)***	28.0, 1609, (95%), ***(+5%)***

Discussion and Conclusion

This paper demonstrates that waypoint movement can be a powerful tool for RL agents to find cover. It shows that waypoint agents generalize to new terrains better than fine-grained agents, at the cost of no longer benefiting from retraining. It shows that training with more far enemies can increase the robustness of the models, though that aspect doesn't generalize to new terrains. Finally, it shows that including pixel inputs substantially increases the models ability to generalize at the cost of almost 10x training times.

A note on how we see waypoints agents being implemented in a practical setting: rather than have the agents trace the waypoint path, querying the RL model whenever it is at a waypoint, instead we can split the agent into two: the RL "ghost" agent, and an A* pathing agent. The more complex agent that calls the Take Cover behavior would instantiate a "ghost agent," which could then move once per frame. Thus, with 60 frames a second, it can usually find a cover spot in under 1.5 seconds, and the more complicated agent can traverse to that location using the navmesh. This method offers the benefits of an RL agent finding a coverspot without the drawback of inefficient pathing. In other words, from the human trainee's perspective, the opponent or ally will take a quicker and more direct path to taking cover.

Future work will include changing the reward function to be more proportional to the agent's likelihood of being shot. Currently, if the nearest cover spot requires moving directly past the enemy, there is no incentive for the agent not to do that. Other future work will involve multiple agents taking cover (with punishments if the agent obstructs an ally's sight of an enemy), changing the pixel inputs to a depth map, and letting the agent change its posture.

Acknowledgments

The project/effort/work depicted here was or is sponsored by the U.S. Army Research Laboratory (ARL) under contract number W911NF-14-D-0005. Statements and opinions expressed and content included do not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

Alonso, E., Peter, M., Goumard, D. and Romoff, J., 2020. Deep reinforcement learning for navigation in aaa video games. arXiv preprint arXiv:2011.04764.

Aris, T., Ustun, V. and Kumar, R., 2022, May. Learning to Take Cover on Geo-Specific Terrains via Reinforcement Learning. In The International FLAIRS Conference Proceedings (Vol. 35).

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B. and Mordatch, I., 2019. Emergent tool use from multi-agent autocurricula. arXiv preprint arXiv:1909.07528.

Gorgan, D., 2019. Performance Analysis in Implementation of a Dodgeball Agent for Video Games. International Journal of User-System Interaction, 12(4), pp.225-240.

Guss, W.H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M. and Salakhutdinov, R., 2019. MineRL: A large-scale dataset of Minecraft demonstrations. arXiv preprint arXiv:1907.13440.

Hartholt, A., K. McCullough, E. Fast, A. Reilly, A. Leeds, S. Mozgai, V. Ustun, and A. S. Gordon. 2021. "Introducing RIDE: Lowering the Barrier of Entry to Simulation and Training through the Rapid Integration & Development Environment". 2021 Virtual Simulation Innovation Workshop.

Hossain, S. and Lee, D.J., 2019. Autonomous-driving vehicle learning environments using unity real-time engine and end-to-end CNN approach. The Journal of Korea Robotics Society, 14(2), pp.122-130.

Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., ... & Lange, D. (2018). Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M.E. and Stone, P., 2020. Curriculum learning for reinforcement learning domains: A framework and survey. The Journal of Machine Learning Research, 21(1), pp.7382-7431.

Schulman J., Wolski F.,Dhariwal F., Radford A., and Klimov O. 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Team, O.E.L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M. and McAleese, N., 2021. Open-ended learning leads to generally capable agents. arXiv preprint arXiv:2107.12808.