Customizing Scripted Bots: Sample Efficient Imitation Learning for Human-like Behavior in Minecraft

Brian Broll
Microsoft Research
brian.broll@microsoft.com

Matthew Hausknecht
Microsoft Research
matthew.hausknecht@microsoft.com

Dave Bignell
Microsoft Research
david.bignell@microsoft.com

Adith Swaminathan
Microsoft Research
adswamin@microsoft.com

ABSTRACT

Although there have been many advances in machine learning and artificial intelligence, video games still define the behavior of characters through careful scripting. These scripted agents follow a fixed set of rules or behavior trees resulting in behavior which is predictable and discernibly different from human gameplay. We demonstrate how to combine imitation learning with scripted agents in order to efficiently train hierarchical policies. Integrating both programmed and learned components, these hierarchical polices improve upon the expressiveness of the original scripted agent, allowing more diverse and human-like behavior to emerge. Additionally, we demonstrate that learned policies also maintain behavior guarantees afforded by the scripted agent, allowing a developer to be confident in the behavior of the learned policy. We demonstrate this interplay between classical AI techniques and statistical machine learning through a case study in Minecraft, and explore open questions about the necessary and sufficient conditions to achieve a fruitful interplay between these two approaches.

KEYWORDS

Sequential decision making; Neural networks; Intelligent agents

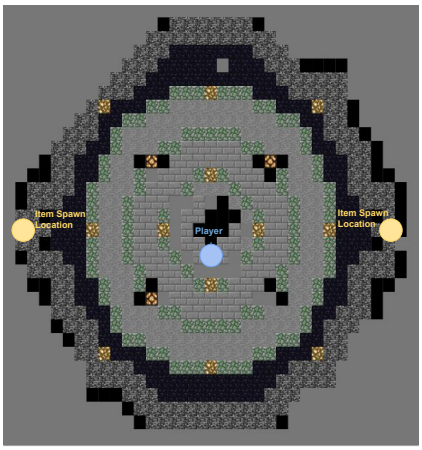


Figure 1: Minecraft Gladiator Arena: The player starts in the center of the room (shown in light blue) and must survive rounds of enemies. Enemies are spawned throughout the map while items are spawned in only two locations (shown in yellow).

1 INTRODUCTION

Despite the recent super-human performance of game playing agents such as AlphaZero [13], OpenAI Five [10], and CFR+ [2], artificial intelligence design in commercial video games today still relies on carefully designed rule-based agents. Integrating machine learned behaviors with existing rule-based agents has the potential to yield learned behaviors that are more flexible, robust, and human-like than their rule-based counterparts. However, there are many open questions about how to incorporate components from existing rule-based agents and how to define appropriate learning problems that encourage human-like behavior. This paper proposes a methodology for integrating scripted agents with learned behaviors to create robust, human-like agents.

Traditional scripted agents follow a fixed set of rules or behavior trees resulting in behavior which is predictable and discernibly different from human players. For example in Minecraft [7], the gladiator arena scenario depicted in Figure 1 pits the player against waves of enemies in an enclosed map. A scripted agent developed specifically for this scenario uses hand-coded path-finding, potential function-based movement, and a fixed priority list to decide its next action. While it is successful at defeating enemies and collecting items, its style and approach to the game is markedly different from human players. Humans employ a diverse set of strategies, intelligently avoid enemies, and use items in contextually appropriate ways (e.g. equipping a sword before targeting stronger enemies). Our goal is to enhance the scripted agent to elicit more diverse and human-like behavior.

Imitation learning (IL) is a natural solution to elicit human-like behavior. In theory, given access to a sufficiently large set of human trajectories, IL is capable of faithfully replicating human behavior. However, naive imitation learning at the level of player input/output (e.g. pixels and mouse/keyboard presses) not only requires a large volume of human data, but also fails to leverage the significant time investment spent by AI developers on the scripted agent. Furthermore, in many contexts, we need guaranteed behavior that is easy to enforce through rule-based scripts (e.g. inventory management in the Minecraft arena), but difficult to guarantee in an IL policy.

Our key insight is to distinguish between a player's high-level strategy and low-level execution. To successfully imitate a player's high-level strategy, we don't need to faithfully re-create each of their low-level actions. Instead we develop a hierarchical IL algorithm that (1) employs labeling functions to extract a player's high-level strategy; and (2) learns a controller to imitate the player's strategy

using subroutines recycled from the scripted agent. The complementary strengths of IL for high-level strategy and scripting for low-level execution combined in this way yield sample-efficient and human-like behavior. In a case study in Minecraft, this hybrid algorithm exhibits significantly more diverse strategies and human-like behavior than the baseline scripted agent using just 33 trajectories collected from non-expert players. The Minecraft Arena scenario, the scripted bot, anonymized human gameplay data and code to reproduce our hybrid bot, as well as videos showing its diverse human-like behavior are available online at https://sites.google.com/view/customizing-scripted-agents.

2 RELATED WORK

Scripted behaviors have a long and rich history of use in video game AI [11]; the most common approaches use hierarchical state machines and behavior trees. However, making these approaches more robust to emergent player behavior or elicit more human-like behavior can often take many thousands of man-hours [3]. This has spurred research in using machine learning techniques to speed up or replace traditional game AI (see [4] for a survey). Most of these techniques have focused on replacing the scripted behavior entirely. Instead of replacing the scripted agent, we recognize that there are many situations where scripted behaviors can be desirable and focus on fruitfully combining learned and scripted components. The closest in spirit to our work is dynamic scripting, an approach which combines ML and rules in an adaptive way as the player plays a game [14]. In dynamic scripting, a rule-based policy is created by the weighted sampling of rules from a rulebase. Weights for the rules are given by a value function trained from the reward signal observed from interactions in the game. Our approach however is a "top-down" approach that utilizes hierarchical IL to control scripted behavior – this design choice is more suitable than dynamic scripting whenever a scripted agent already exists.

Our approach builds on the rich literature on hierarchical IL [8] to achieve sample efficient learning. However, our "lower-level" policies arise as a consequence of the scripted behavior that may not match human behavior; this introduces novel challenges for hierarchical imitation learning that does not arise when lower-level policies are learned jointly, via e.g. reinforcement learning.

To address the mismatch between the low-level actions such as mouse movements present in player trajectories and the high-level behavior exposed by subroutines in the scripted agent, we borrow ideas from behavior cloning from observations and imitation learning from only states [9, 15, 16]. The core idea in these approaches is to learn a model that maps from observed state transitions to impute agent's actions that can cause the transition. We instead employ labeling functions to impute high-level strategic targets from player trajectories. A target is an entity in the world with which the player is trying to interact, such as an enemy to attack or an item to pick up. Player trajectories however do not, by default, have annotations that tell us the player's intended target; so we must infer the targets through other means. Labeling functions [12] are a data programming technique where domain heuristics are expressed in a program used to label data. They have been used successfully to bootstrap large training sets via weak supervision for classification tasks like knowledge base slot-filling; to our knowledge, our work is the first extension of their application to sequential decision-making tasks. In doing so, we employ a novel policy architecture to learn from the weak supervision provided by these functions – our policies use an LSTM to summarize long sequences of states like in DRQN [5] and employ shared weights and dynamic computation graphs as in DRRN [6].

3 A CASE STUDY IN MINECRAFT

We conducted experiments within the Minecraft Gladiator Arena using Malmo [7]. The Gladiator Arena is a survival game in which the player is spawned in a large room and must survive multiple rounds of combat, where various enemies and items are spawned throughout the map in each round. The items include weapons like swords and bows, armor, and health that assist the player in defeating enemies. Enemies have different types of attack such as melee (Zombie), ranged (Skeleton, Blaze), and self-destructive attacks (Creeper). Generally, the number and difficulty of the enemies increase as the levels progress.

In this specific scenario, we collected 33 human demonstrations from players of different skill levels. Across these trajectories we observed a variety of strategies employed by the players (both successful and unsuccessful). Strategies include user preference for ranged versus melee weapons and different combat styles. Another interesting dimension of playstyle among the collected human trajectories is the various strategies for collecting items or attacking enemies: some players preferred to directly attack enemies, while others would hide and collect items in order to fight more effectively.

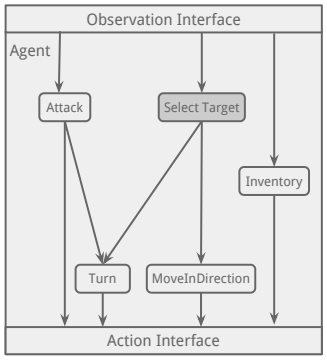


Figure 2: A scripted agent for playing the Arena challenge. The agent consists of five subroutines: attacking, selecting the current target, managing the inventory, turning, and movement. Shaded in dark grey, the Select Target subroutine is replaced with a learned controller.

Scripted Agent: In contrast to human players, the scripted agent depicted in Figure 2 uses hand-coded subroutines to perform low-level pathfinding, inventory management, combat, and target selection. Specifically, the scripted agent always attacks if an enemy is within melee range. Next, it always interacts with its inventory to equip the strongest available armor and weapon, to use any items which restore health if needed, and to discard any undesirable items. Finally the `SelectTarget` subroutine (Figure 2) selects the next target to move toward in the level such as a weapon or enemy. The

default `SelectTarget` subroutine uses a static ranking over all possible entities to choose the next target.

Movement: The movement of the scripted agent is determined using potential functions [1] which repel the agent from enemies and static obstacles and attract the agent to its desired target. If there are no targets present, the agent is attracted to "home squares" at the top and bottom of the arena. Finally, regardless of target, if there is an enemy that is getting close to being within attack range, the bot faces toward the enemy.

3.1 Defining a Learned Controller

Within the context of the Arena, the scripted agent, and the behaviors observed in the human trajectories, we create a learned controller to replace the `SelectTarget` subroutine in the original scripted agent (shown in Figure 2). Of the various subroutines present in the scripted agent, we chose to replace `SelectTarget` because high-level target selection maps well onto the space of human strategies and can naturally be used to describe the strategies employed by different human players. In particular, much of the diversity in human behavior can be explained by describing which entities they decide to interact with and the order in which they collected items and attacked enemies.

3.2 Deriving Supervision from Player Trajectories

To construct the training data from player trajectories, we must impute targets for the `SelectTarget` controller. There are many conceptually valid ways of doing this:

Trajectory Matching: The first and simplest approach relies on some way to measure similarity between two gameplay trajectories. At any game state in a player trajectory, we can impute every possible target and collect how the scripted agent continues from that state. Then, the behavior that is closest to the human continuation is chosen according to the similarity metric and the corresponding target is imputed. In situations where humans and agents use different action APIs to interact with the game, the similarity can be computed using just the game observations collected in the trajectories. This approach is a non-starter because it is computationally infeasible and relies heavily on a robust similarity metric – coding or learning the similarity metric can be harder than training the controller!

Inverse Modeling: The second approach relies on learning an inverse model [16]. Suppose we randomly perturb the targets selected by the agent and collect the resulting bot behaviors. On this dataset, the targets are known and a model can be trained to predict the targets given observations as context. However, there is no correspondence with human trajectories during training – so it is unlikely that the trained model generalizes to impute good targets on human trajectories. This approach is also computationally expensive – having to randomize over all possible target manipulations to construct a bot dataset is very expensive when what we want are the targets that roughly correspond to human gameplay.

Labeling Functions: Our approach is computationally cheap, naturally achieves correspondence with human trajectories, and doesn't require a hard-to-learn similarity metric. We use labeling functions [12] to impute targets for the `SelectTarget` controller. At any game state in a player trajectory, the labeling function takes the human continuation from that state as input. The length of the future states is a hyper-parameter δ: this hyper-parameter is set to approximate the maximum frequency of target changes in a player trajectory. The output of the labeling function serves as the target that the learned controller must predict. Consequently, if δ is small then we require a more fine-grained alignment between the game observations a bot might see if it continued from a state and what the player experiences from that state.

Concretely, the labeling function observes the player's position across δ time-steps and infers the target to be the entity that the player moved furthest toward. The player's displacement from s_i to $s_{i+\delta}$ is computed and projected onto the displacement between the player and each possible entity. The entity to which the player moved most toward is then considered the target. Because human players aren't always moving towards a target (in fact sometimes they aren't moving at all), we introduced an additional target "self," which indicates that the player is stationary during the δ time-steps. Throughout the paper, δ was set to 10 frames (0.5 seconds). Thus, our labeling function provides a weak supervisory signal that approximates the target that a player may be going after at the given state, by peeking 0.5 seconds into the future.

3.3 Training the Controller

After labeling the human trajectories, the controller is then trained using behavior cloning to imitate the targets imputed by the labeling function. In order to capture sufficient context to represent human behavior, the learned controller is a recurrent neural network which selects the current target from the list of valid entities at the given state. The neural network architecture accepts two variable length inputs, the observation sequence O and the valid entities $e_0 \ldots e_n$ extracted from the final observation (shown in Figure 3). The encoder LSTM A, generates a fixed size embedding of the given trajectory of observations. This embedding, z, is then concatenated with each valid entity and provided as input to the target evaluator, B. Finally, the logits computed for the entities are converted to a valid probability distribution via the softmax layer.

This dynamic architecture was necessary since the number of and type of targets can change from step to step.

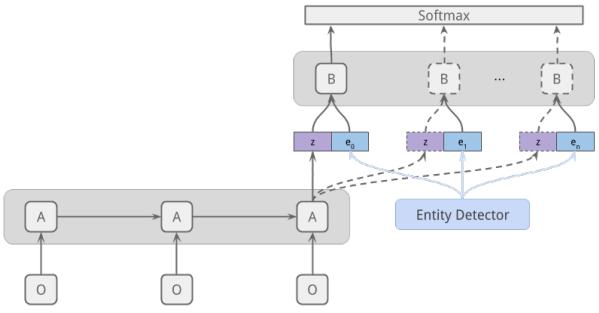


Figure 3: Policy network architecture

3.3.1 State and Entity Representation. The states were represented by the internal game state encoded to include the number of entities killed, damage taken, damage dealt, life, score, and player orientation. The states also included counts of the numbers of items

and entities in different regions of the map. Entities were encoded using orientation, health, and a one-hot encoding for the entity type.

3.3.2 Training. Mini-batches were stratified by the total number of valid entities in the last state of the sequence to ensure that the valid entity count was fixed for each mini-batch. The datasets were resampled to address class imbalance ($\tilde{4}5\%$ of the samples were labeled "self"). The policy was trained for 50,000 iterations ($\tilde{8}5$ epochs) using a batch size of 128 and a sequence length of 25. The learning rate was 0.00001 for the ADAM optimizer. The training, validation, and testing datasets were created using a 70/20/10 split of the human trajectories and were sampled to preserve the distribution of classes. Code to replicate training of our controller is available online at https://sites.google.com/view/customizing-scripted-agents.

3.4 Summary of the Approach

For reference, the approach used in vanilla imitation learning is shown in Figure 4. In vanilla imitation learning, the objective is to learn the direct policy (shown in dark blue) which maps the states to the agent actions given the human demonstrations. In the proposed approach, a learned controller is introduced into an existing scripted agent to allow us to leverage both the flexibility of imitation learning and the behavioral guarantees afforded by the remaining scripted logic. A labeling function is defined to derive a weak supervisory signal from the human demonstrations. The targets imputed by the labeling function are used to train the `SelectTarget` controller via behavior cloning. As the target space is much smaller than the original agent action space, the sample complexity of the imitation learning problem is reduced and becomes tractable while needing fewer human demonstrations. After we have trained the controller, we then use the hybrid agent to emit actions when being driven by the targets selected by the controller.

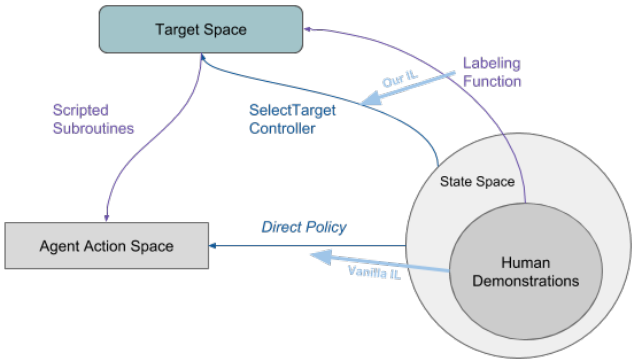


Figure 4: An overview of the approach to perform sample efficient imitation learning leveraging an existing scripted agent. The standard approach to imitation learning is also shown for reference.

4 EMPIRICAL EVALUATION

We conduct extensive experiments to test the performance of the hybrid agent using a learned controller and scripted subroutines across several dimensions.

(1) Does the learned controller generalize to select good targets on a held out set of human trajectories? [Section 4.1]
(2) Does the hybrid agent exhibit more diverse and human-like behavior? Section 4.2 compares the distribution of different strategies used during the first round of the Arena scenario. Section 4.4 compares the aggregate environment occupancy of the player across trajectories generated from both bots and the human.
(3) Can we get guaranteed behavior from the hybrid agent just like we can from the scripted agent? [Section 4.3]
(4) How does the mismatch between humans' low-level behavior and the scripted subroutines affect the bot's overall performance in the Arena scenario? [Section 4.5]

4.1 Does the Controller Generalize Across Human Trajectories?

Evaluation is performed by executing the policy network both on fixed length sequences as well as using all previous observations. When using all previous observations, we expect the controller to struggle since it must remember the salient information from state sequences in its LSTM units. As a baseline, we present the performance of both random guessing and the scripted agent's target selection logic for selecting the current human target as imputed by the labeling function on the test set. Random guessing results in an accuracy of 5.15% whereas the original scripted target selection logic has an accuracy of 13.65%. The learned script-controller has an overall accuracy of 73% when evaluated on fixed length ($N = 25$) sequences and 60% when evaluated using all prior observations. These accuracies mean the controller more closely matches the human target selection than the original scripted agent.

This suggests that the learned controller is indeed able to identify the target a player is likely to go after given the recent gameplay context, when evaluated in the kinds of situations a player might face.

4.2 Are the Agent's Strategies Diverse?

In this section, we evaluate the distribution of strategies used by the original agent, hybrid agent, and human players. In this context, a strategy is considered to be the order in which the player collects (or discards) any items and kills the enemies. To simplify the exposition, we only report the strategy of the player during the first round.

In the first round of the Arena, there are two items and three enemies spawned. The items include a "Stone Sword" and an "Iron Chestplate" and all three enemies are zombies which have a melee attack but no ranged attacks. One common strategy among the human trajectories was to first collect the sword then kill all the enemies. Conversely, the original scripted agent always prioritizes collecting the items then returns to an open portion of the map and fights the zombies. Although this is the default behavior of the scripted agent, stochasticity in the environment and its ability to fight any enemy within range sometimes results in enemies being killed while it is trying to collect the items.

Figure 5 shows the distribution of strategies for the first round among the collected human demonstrations (in the distribution on the left). This shows that the most common strategy was to first collect the sword and then kill all the enemies. There were also

Figure 5: Distribution of strategies for the first round of the Arena: Strategies are determined by the order of player events including picking up items, discarding items, and killing enemies. The following abbreviations are used: Kill enemy, Pick up item, and Discard item. When picking up or discarding items, the item is either Sword or Chestplate. The ATTACK_ONLY strategy is when the player attacks the enemies without any item and dies before killing a single enemy.

many other strategies including collecting both items (sword and chestplate) before killing the enemies and even a strategy in which the players beat the enemies without the use of any of the items. As these trajectories included non-experts, there are also a number of different strategies which are ineffective such as "ATTACK_ONLY" in which the player died without collecting any items or killing enemies.

There are fewer strategies employed by the scripted agent (shown in the central plot in Figure 5) which is to be expected given the inflexible logic used for selecting the current target. As the environment is stochastic and the scripted agent will always attack an enemy that is within range, there is actually more diversity in the behavior than might otherwise be expected. The individual strategies share common characteristics as the agent always collected the sword ("PS") before the armor ("PC"). Additionally, the first three strategies (accounting for 77.21% of the strategies employed by the scripted agent) only vary in whether the player collected the items and lost (the second strategy) or was able to kill an enemy before collecting the armor (the third strategy).

The distribution of strategies employed by the agent using the learned target selection is presented in the right distribution from Figure 5. The modified agent exhibits a larger set of strategies than the original scripted agent including strategies unique to the human demonstrations as well as strategies unique to the scripted bot. The top two strategies are bot prominent strategies employed in the human demonstrations ("PS_K_K_K" and "ATTACK_ONLY"). The third most common strategy, "PS_K_PC_K_K," is also the third most common strategy exhibited by the scripted agent and is not shown in any of the human demonstrations.

The total variation between the distribution of the scripted agent's strategies (viewed as a discrete probability distribution) and the human strategies is 0.8375; whereas the total variation between the distribution of the hybrid agent's strategies and the human strategies is 0.4487. This suggests that the hybrid agent indeed produces more diverse, human-like behavior.

4.3 Can the Agent Produce Guaranteed Behavior?

In this section, the hybrid agent is evaluated with respect to preserving two behavior guarantees provided by the scripted sub-policies: the use of health restoring items and discarding useless items. As the training demonstrations were collected from non-experts, some of the demonstrations contain examples violating the desired behavior guarantees. Consequently, standard imitation learning approaches cannot hope to consistently produce behavior which complies with the desired guarantees much less provide guarantees about the agent's behavior. To evaluate these behavior guarantees, we defined two related constraints to check for behavior violations on the trajectories of the hybrid agent and the human demonstrations.

As we would like to ensure that the agent uses health restoring items when needed, a constraint was defined which ensured that the agent did not have any unused health recovery items in its inventory when it died. Five of the 33 human demonstrations (15.15%) violated this constraint. As expected, none of the trajectories collected from the scripted agent nor the hybrid agent violated this constraint.

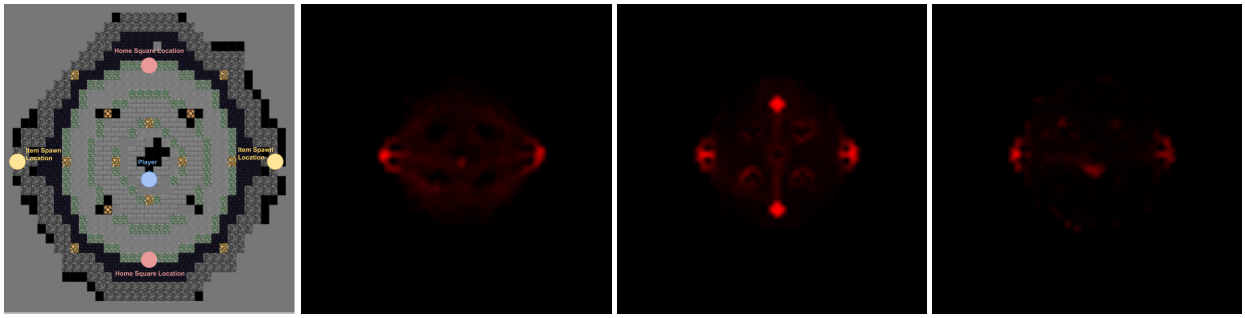
As items are automatically picked up when they are touched by the agent, the second constraint was defined more flexibly and simply checked if the agent discarded the useless items within 1 minute. This constraint was violated by 23 of the 33 human demonstrations (69.70%). Furthermore, the majority (68.86%) of the states from the human demonstrations included inventories containing useless items. As with the previous constraint, neither the scripted agent nor the hybrid agent violated this constraint.

4.4 Can the Learned Controller Affect Other Scripted Subroutines?

The behavior of the agents and humans were also evaluated with respect to the occupancy of the map across all trajectories visualized as heatmaps in Figure 6. This represents another dimension of playstyle in which the scripted agent is distinct from the human players. Although this form of evaluation does not capture the specific context or subgoal of the player, it provides insight into the aspects of behavior across trajectories.

All the heatmaps show a moderate degree of occupancy for points on the left and right of the map. These points correspond to the item spawn locations; their high visitation suggests that both the humans and the agents have a tendency to collect these items. The human demonstrations (shown in the left heatmap) show a somewhat uniform occupancy for the locations between the items with a slightly higher occupancy in the center where the player is spawned.

The heatmap for the scripted agent (shown in the center heatmap) shows a high frequency of occupancy for the left, right, top and



(a) Gladiator Arena (b) Human Demonstrations (c) Scripted Agent (d) Learned Agent

Figure 6: Heatmap of player occupancy information given a top-down view of the scenario for the human demonstrations, scripted agent, and learned agent. Frequently occupied locations are shown in red. Item spawn locations are shown on the left and right and the "home square" location (used by the scripted bot) is shown on the top and bottom of the arena map.

bottom points of the map. The high occupancy locations in the top and bottom of the map are locations known to the scripted bot and used as a default location when there are no other targets to pursue. This ensures that the agent will not move to a location where it may be cornered by enemies. The degree to which these four locations are occupied by the scripted agent illustrates the consistency of its behavior across trajectories which is distinct from the behavior shown in the human demonstrations. In contrast, the hybrid agent's occupancy map does not prioritize the top and bottom locations and visually mimics the occupancy of human demonstrations.

4.5 Is the Agent's Overall Performance Affected?

The overall performance of an agent in a trajectory is evaluated by recording the level the player was able to reach. The performance of the humans, scripted agent, and hybrid agent across the collected trajectories is shown in Figure 7. The distribution of human performance shows a distinct split in ability in which some demonstrations were able to complete all 7 levels whereas many others were defeated on levels 2-4. The scripted agent showed a similar trend around levels 2 and 4 but had significantly fewer trajectories reaching level 7 (12%). The hybrid agent showed some performance degradation as it was defeated more often during level 2 and seldom reached the 4th level and beyond.

The second level introduces multiple ranged units which often defeat the players. The successful human demonstrations often employ a number of different behaviors such as strafing, dodging, and using ranged weapons themselves for which the bot does not have any corresponding subroutines. As a result, the use of a more human-like `SelectTarget` controller can hinder the bot's performance when human-like subroutines are necessary for the current target to be achievable.

5 DISCUSSION

We have presented a robust and sample efficient approach for enhancing the behavior of existing scripted agents. We defined a controller within such scripted agents and developed an efficient technique for deriving supervisory signal from player trajectories without requiring fine-grained alignment between the human and bot trajectories. This approach was demonstrated within Minecraft and was shown to exhibit more human-like behavior with respect to gameplay strategies and map occupancy.

There are also a number of remaining open questions pertaining to this approach. There are challenges in evaluating human-like behavior, especially in the context of a hybrid agent as we would like to evaluate the aspects of behavior affected by the learned meta-controller. In the presented hybrid agent, the impact of the meta-controller had a direct impact on the order in which the player interacted with the items and enemies in the surroundings (strategy). These strategies provided a compact representation of the trajectory which was heavily influenced by the learned component allowing comparison and quantitative evaluation. However, this evaluation is specific to the given task and parameterization of the hybrid agent and does not necessarily generalize to other scenarios or hybrid agents. Additionally, an important property of the labeling function

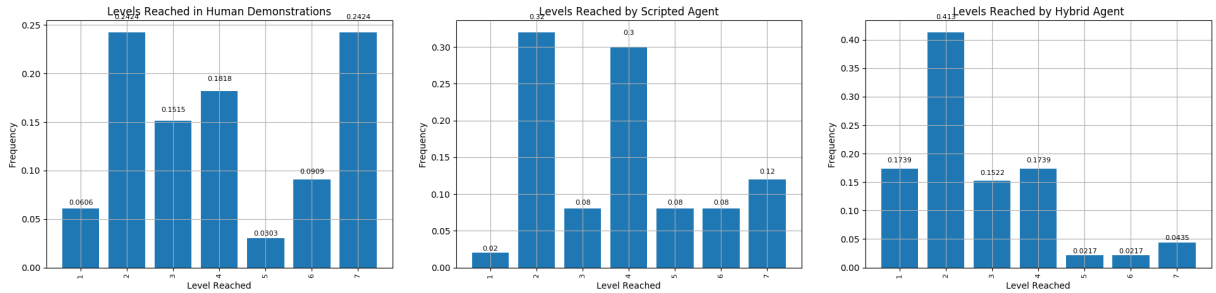


Figure 7: Distribution of the last level reached in the human demonstrations, scripted agent, and hybrid agent.

is the use of the δ parameter to enable alignment between the human and agent data at a higher granularity. This avoids alignment issues between the human actions and the scripted subroutines, but we leave a careful study of the effect of δ for future work.

In the presented empirical evaluation, the accuracy of the labeling function on the hybrid agent's trajectories is rather low (31.85%). This surprisingly low consistency between an agent's `Select-Target` parameter and observed behavior prompts a number of related questions. Is consistency required between the labeling function and the scripted options?

Sample efficient imitation learning could enable customizing the hybrid agents to individual players or playstyles and exposing meaningful parameters to enable developers to tune the playstyles of these hybrid agents.

REFERENCES

[1] D.M. Bourg and G. Seemann. 2004. *AI for Game Developers: Creating Intelligent Behavior in Games*. O'Reilly Media, Sebastopol, CA. https://books.google.com/books?id=cKKGAgAAQBAJ

[2] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. 2015. Heads-up Limit Hold'em Poker is Solved. *Science* 347, 6218 (January 2015), 145–149.

[3] M. Cerny, T. Plch, M. Marko, J. Gemrot, P. Ondracek, and C. Brom. 2017. Using Behavior Objects to Manage Complexity in Virtual Worlds. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 2 (2017), 166–180.

[4] Leo Galway, Darryl Charles, and Michaela Black. 2008. Machine learning in digital games: a survey. *Artificial Intelligence Review* 29, 2 (01 Apr 2008), 123–161.

[5] Matthew J. Hausknecht and Peter Stone. 2015. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposia*. The AAAI Press, Palo Alto, CA, 29–37.

[6] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2016. Deep Reinforcement Learning with a Natural Language Action Space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Assocation for Computational Linguistics, Berlin, Germany, 1621–1630.

[7] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. 2016. The Malmo Platform for Artificial Intelligence Experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press, Palo Alto, CA, USA, 4246–4247. http://dl.acm.org/citation.cfm?id=3061053.3061259

[8] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daumé, III. 2018. Hierarchical Imitation and Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, Stockholmsmässan, Stockholm Sweden, 2917–2926. http://proceedings.mlr.press/v80/le18a.html

[9] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. 2018. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. Curran Associates, Inc., Red Hook, NY, 1118–1125.

[10] OpenAI. 2018. OpenAI Five. https://blog.openai.com/openai-five/. (2018).

[11] Steven Rabin. 2017. *Game AI Pro 3: Collected Wisdom of Game AI Professionals*. A K Peters/CRC Press, Boca Raton, FL, USA.

[12] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data Programming: Creating Large Training Sets, Quickly. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., Red Hook, NY, 3567–3575.

[13] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[14] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. 2006. Adaptive game AI with dynamic scripting. *Machine Learning* 63, 3 (01 Jun 2006), 217–248. https://doi.org/10.1007/s10994-006-6205-6

[15] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. AAAI Press, AAAI Press, Palo Alto, CA, USA, 4950–4957.

[16] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Generative adversarial imitation from observation. (2018). arXiv:1807.06158v3