Selecting Subgoals using Deep Learning in Minecraft: A Preliminary Report

David Bonanno[1] Mark Roberts[2] Leslie Smith[3] David W. Aha[3]

[1]Naval Research Laboratory (Code 5557); Washington, DC | david.bonanno@nrl.navy.mil
[2]NRC Postdoctoral Fellow; Naval Research Laboratory (Code 5514); Washington, DC | mark.roberts.ctr@nrl.navy.mil
[3]Naval Research Laboratory (Code 5514); Washington, DC | {first.last}@nrl.navy.mil

Abstract

Deep learning is a powerful tool for labeling images in computer vision. We apply deep learning to select subgoals in the simulated game environment of Minecraft. Prior work showed that subgoal selection could be learned with off-the-shelf machine learning techniques and full state knowledge. We extend that work to learn subgoal selection from raw pixels. In a limited pilot study where a virtual chracter must overcome obstacles, we show that AlexNet can learn an effective policy 93% of the time with very little training.

1 Introduction

Deep Learning (DL) has played an integral role in the field of computer vision in the form of Convolutional Neural Networks (CNN) [Krizhevsky *et al.*, 2012] in recent years. These networks have utility in classification problems bridging the gap between the highly complex domain of pixels in an image and a rigid classification label understood by a human.

Deep Learning has recently been brought into the domain of video games including Atari [Mnih *et al.*, 2015] [Lipovetsky *et al.*, 2015]. This work combines DL with Reinforcement Learning (RL) to perform a different task; instead of focusing on which classification label to generate for a given image the network is tasked with accomplishing a goal. These goals, such as achieving a high score in a video game, are accomplished by selecting the appropriate sequence of actions (e.g., moving, blocking, jumping). The selection process can be either supervised (and trained by a human expert) or unsupervised. The initial research by [Mnih *et al.*, 2015] demonstrates how DL can be used as a part of a system for recognition and control in decision making processes.

While the DL+RL approach used was immensely successful at many of the arcade games, it favors environments that provide immediate and somewhat dense reward signals[Mnih *et al.*, 2015]. This research has shown poorer performance in games which involve long sequences of events including the Atari game Montseuma's Revenge. Here the agent must learn to pick up a key to open a door or that it must use a torch to explore the pyramid. Hierchical planning can buttress this weakness by providing a more sparse decision space for selecting among subgoals, thus abstracting away such long sequences of actions and focusing exploration toward subgoals that are known to be useful[1]. Making decisions at the subgoal level requires a controller that translates the chosen subgoal into a sequence of actions.

In this paper, we focus on the problem of classifying the game images in Minecraft to overcome simple obstacles by selecting subgoals such as walking, creating bridges, destroying obstacles, and building stairs. We build on previous work by Roberts et al. (2016) who show that subgoal selection can be learned using a supervised learning method and full state information. Here, we learn the mapping from *raw images* to select a best subgoal using the guidance of an "expert". This technical approach momentarily leads us toward classification and away from RL, but we identify in our future work discussion our plans for returning to a DL+RL framework and learn this selection policy using RL. We find that a DL method can learn a subgoal-selection policy that is 87.1% accurate. In some cases the failures resulted from occlusions that we are working to address. Our results show that combining DL with hierarchical planning successfully leverages the strengths of both.

2 Goal Reasoning and ACTORSIM

We augment the model of online planning and execution by Nau (2007) with a goal reasoning loop (see Figure 1). Our work builds on a recent model of goal reasoning that is based on Goal-Task Network (GTN) planning [Alford *et al.*, to appear]. GTN planning is a hybrid model by Alford et al. (2016) that merges hierarchical task network planning [Nau *et al.*, 2003] with hierarchical goal network planning [Shivashankar *et al.*, 2013]. Nodes in a *gtn* can be either a goal (i.e., a state to achieve) or a task to perform. Thus GTN planning provides a natural formalism for representing knowledge in a way that can decompose complex tasks into combinations of subgoals or subtasks and was the basis for a recent formal model and semantics for goal reasoning by Roberts et al. (2016). For this paper we focus more on a simple goal-task network (see Figure 3).

The Actor Simulator, ACTORSIM (Figure 2), implements the goal lifecycle and GTN semantics. It complements existing open source planning systems with a standardized im-

[1]Although we do not consider temporal planning in this work, it is one direction for future work.

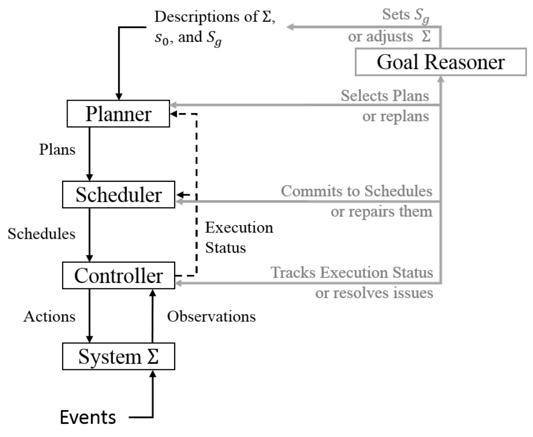

Figure 1: Relating goal reasoning with online planning, where the GRPROCESS can modify the objectives of the system.

plementation of goal reasoning and also provides links to simulators that can simulate multiple agents interacting within a dynamic environment. The **Core** provides the interfaces and minimal implementations of the platform. It contains the essential abstractions that apply across any simulator. This component contains information about Areas, Locations, Actors, Vehicles, Symbols, Maps, Sensors, and configuration details. The **Planner** contains the interfaces and minimal implementations for linking to existing open source planning systems. This component unifies Mission Planning, Task Planning, Path Planning, and Motion Planning. It currently includes simple, hand-coded implementations of these planners, although we envision linking this component to many open source planning systems. **Connector** links to existing simulators directly or through a network protocol. Currently supported simulators include George Mason University's MASON[2] and two computer game simulators: StarCraft and Minecraft. We envision links to common robotics simulators (e.g., Gazebo, ROS, OpenAMASE), additional game engines (e.g., Mario Bros., Atari arcade, Angry Birds), and existing competition simulators (e.g., RDDLSim). We plan to eventually link ACTORSIM to physical hardware. **Coordinator** (not shown in the figure) houses the interfaces that unify all the other components. This component contains abstractions for Tasks, Events, Human interface Interaction, Executives (i.e., Controllers), and Event Notifications. It uses Google's protocol buffers[3] for messaging between distributed components. The **Goal Refinement Library** is a standalone library that provides goal management and the data structures for transitioning goals throughout the system. It contains the default implementations for goals, goal types, goal refinement strategies, the goal memory, domain loading, and domain design.

3 The Game of Minecraft

Here we discuss how goal reasoning can be used in the video game Minecraft. This video game contains complex environments in terms of both imagery and planning. We also define the ACTORSIM Connector, a tool which allows us to apply goal reasoning techniques to this game.

3.1 Applications of Goal Reasoning in Minecraft

We study goal reasoning in Minecraft, a popular game where a human player moves a character, named Steve, to explore a 3D virtual world while gathering resources and surviving dangers. Managing the complete game is challenging. The character holds a limited inventory to be used for survival. Resource blocks such as sand, dirt, wood, and stone can be crafted into new items, which in turn can be used to construct tools (e.g., a pickaxe for mining or a shovel for digging) or structures (e.g., a shelter, house, or castle). Some blocks are dangerous to the character (e.g., lava or water). Hostile non-playing characters like creepers or skeletons, generally called mobs, can damage the characters health. Steve can only fall two blocks without taking damage. We focus on the problem of navigating obstacle courses. The set of possible choices are staggering; for navigating a 15x15 maze in Minecraft, Abel et al. (2014) estimate the state space to be nearly one million states.

Researchers have recently begun using the Minecraft game for the study of intelligent agents [Aluru *et al.*, 2015]. In previous work, researchers developed a learning architecture called the Brown-UMBC Reinforcement Learning and Planning (BURLAP) library, which they implemented in their variant of Minecraft, BURLAPCraft [Abel *et al.*, 2015] BURLAPCraft allows a virtual player to disregard certain actions that are not necessary for achieving goals such as navigating a maze.

Similar to that research, we task the GRPROCESS, acting as a virtual player, with controlling Steve to achieve the goal of navigating to a gold block through an obstacle course. However, our technical approach differs from prior research. Our aim is to develop a GRPROCESS that can incorporate increasingly sophisticated goal-task networks and learned experience about when to apply them. At a minimum, this requires thinking about how to compose action primitives into tasks that the GRPROCESS can apply and linking these tasks into a gtn. Thus, we construct these tasks and build a gtn that uses them.

Figure 3 shows the gtn consisting of a top goal of moving to the gold block and the four descriptive subgoals that help the character lead to that objective. These subgoals do not contain operational knowledge. For example, preconditions on actions ensure that Steve will not violate safety by falling too far or walking into a pool of lava or water. For moving toward the goal, the block at eye level must be air, the block stepped on cannot be lava or water, and Steve cannot fall more than a height of two blocks. A staircase requires a wall with a height of two blocks and the ability to move backwards in order to place a block. Mining is only applicable if the obstacle has a height of three blocks.

The order of subgoal choice impacts performance. For example, suppose the subgoal to step forward is selected when lava is directly in front of Steve. Steve's Controller disallows this step because it violates safety and the subgoal will fail, which will require additional goal reasoning to resolve the failure.

[2]`http://cs.gmu.edu/~eclab/projects/mason/`
[3]`https://developers.google.com/protocol-buffers/`

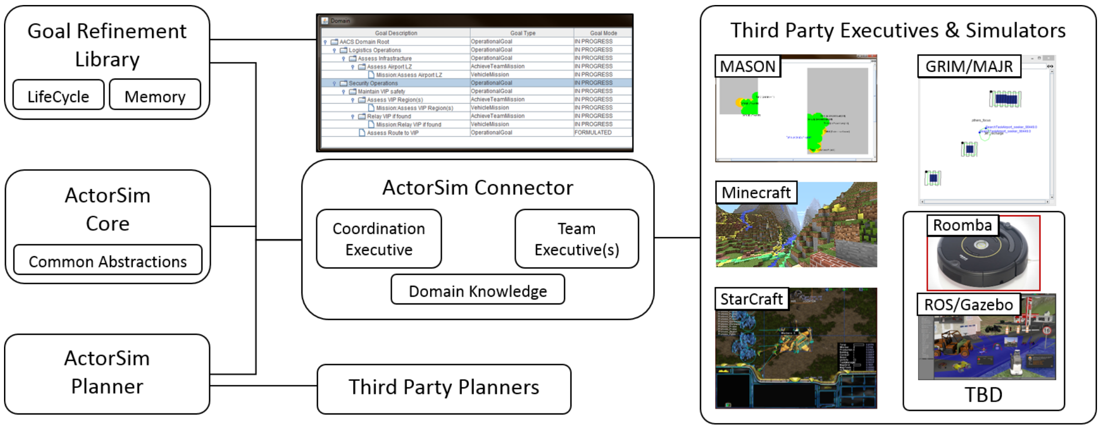

Figure 2: The Component Architecture of ACTORSIM.

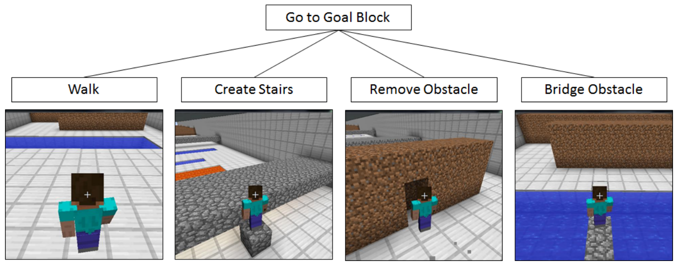

Figure 3: The *gtn* for the GRPROCESS in our study.
‘

Three features of our goal representation complement prior research in action selection (e.g., reinforcement learning or automated planning). First, we model the subgoal choice at a descriptive level, assuming that committing to a subgoal results in an effective operational sequence (i.e., a plan) to achieve the goal. We rely on feedback from the Controller running the plan to resolve the subgoal. Second, the entire state space from start to finish is inaccessible to the GRPROCESS so it cannot simply perform offline planning or interleave full planning with online execution. Each obstacle course is distinct and there must be an interleaving of perception, goal reasoning, and acting. Third, the operational semantics of committing to a subgoal are left to the Controller. Thus, the GRPROCESS must learn to rank the subgoals based on the current state using prior experience.

Prior work by Roberts et al. (2016) examined how making effective choices at the GTN level can be done by learning from traces (i.e., examples) that lead to more efficient behavior, where improved efficiency was measured as reaching the goal in fewer steps or failing less frequently. In this paper, we focus on learning a subgoal selection policy from an expert. We next describe how ACTORSIM connects to Minecraft and how we collect that expert experience.

3.2 The ACTORSIM Connector for MineCraft

The ACTORSIM Connector integrates ACTORSIM abstractions with a reverse-engineered game plugin called the Minecraft Forge API (Forge), which provides methods for manipulating Minecraft. We implemented basic motion primitives such as looking, moving, jumping, and placing or destroying blocks. These motion primitives compose the operational plans for the four sub-goals: walking forward, creating stairs, removing obstacles, and bridging obstacles. Although some of this functionality was present in BURLAPCraft [Abel *et al.*, 2015], our implementation better matches the abstractions provided by the ACTORSIM Core and ACTORSIM Coordinator.

We have simplified Steve's motions to be axis aligned. Steve always faces North and the course is constructed such that the gold block is North of Steve in a straight line. Steve is 1.8 meters high; voxels in Minecraft are 1 meter square. So, Steve occupies roughly a 1x2 meter space. Steve interacts with a limited set of world objects: cobblestone, emerald, air, lava, water, and gold.

The ACTORSIM Connector for MineCraft constructs the obstacle courses for our study. Figure 4 (top) shows six of the nine sections the GRPROCESS may encounter:lava, pond, short wall, tall wall, obstacle, empty, stairs, arch, and comb. Figure 4 (bottom) displays a course composed of three sections.

Each obstacle has an appropriate subgoal choice. For lava or pond, the best choice is a bridge; alternatively the GRPROCESS may also move closer and go around the pond. For the short wall, the best subgoal is to create a single stair and step up. For the tall wall or pillar, which are both three blocks high, the best subgoal is to mine through the wall; alternatively, the GRPROCESS may also move closer and go around the pillar.

Observations Figure 5 shows the set of states around Steve that the GRPROCESS can observe. These include the eight blocks directly around Steve's feet, the two blocks directly behind and in front of Steve, one block behind and below Steve, the block just above Steve's head to the front, and the block three down and in front of Steve as shown in Figure 5. A state is labeled with a unique string using the relative position left/right (l), front/back (f), and height (h) with either a positive (p) or negative (n) offset, where zero is denoted as a positive number. For example, the block immediately in front of Steve's feet would be left positive 0, front positive 1, height negative 1 creating the string designation

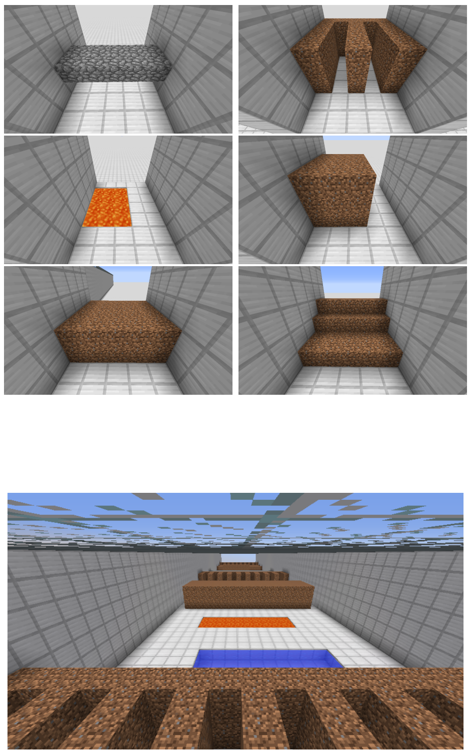
Figure 4: Six example section types (top left to bottom right) include arch, comb, lava, pillar, short wall, and steps. The bottom image shows a portion of an obstacle course where the GrProcess must traverse from the emerald block behind it at the start (not shown) to a gold block at the opposite end in front of it (not shown). The course is covered with a clear top to prevent Steve from walking along the wall.

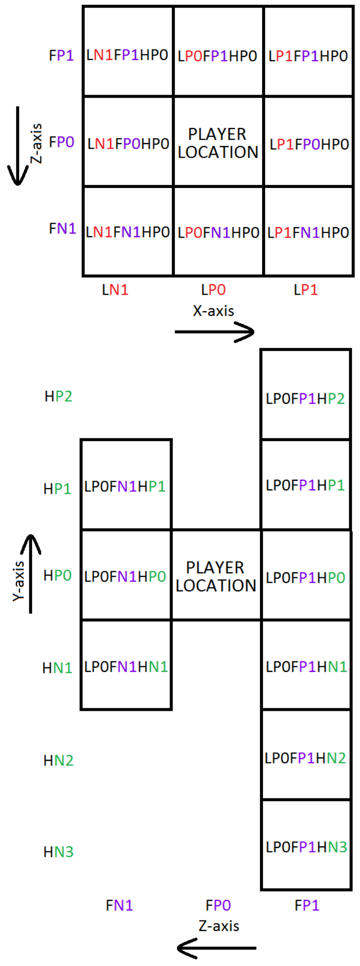

Figure 5: Observable blocks around Steve from the top view (top), where the player is facing "up" and the side view (bottom), where the player is facing to the right.

"LP0FP1HN1". Each state is assigned a unique string (shown in each box) to denote the world object in that position.

Collecting Traces of Experience The original study collected various kinds of experience. However, in this study we use traces from the **expert** training procedure, which is hand-coded (by an author of the previous study) and examines detailed state information to select the best subgoal. The expert procedure never fails to reach the gold block but also represents extremely biased knowledge about which subgoal is appropriate.

The subgoal selected by the expert trace is used to train a convolutionoal neural network which is given both imagery and the corresponding subgoal. We then study how effective the convolutional neural network can predict the subgoal given imagery alone.

4 Approach

The camera view we chose for this study contains the obstacles and the agent (see Figure 6). Because of this it is possible that the blocks in front of Steve are occluded by Steve himself, hindering navigation.

The subgoal selection procedure that ActorSim uses is informed by state information that is read directly from the MineCraft environment. However, the expert procedure is blind to the imagery that would be directly used by a human player. In this section we describe the technique we use to train a DL network to select the optimal subgoal as shown in Figure 7. During training, the network uses information provided by ActorSim as well as imagery. During testing, the DL network uses imagery *only* to predict which subgoal should be used to navigate the course.

4.1 Data Generation

Obstacle courses are generated using the ActorSim Connector. These obstacle courses are then run by Steve using the expert training procedure as described in Section 3. We modified ActorSim to generate an image prior to every subgoal selection. Many subgoals may be required to overcome a single obstacle. For example, when encountering an arch, Steve must walk to the arch, build stairs, and then walk up the stairs and across the arch. Because of this, an obstacle course of a few hundred obstacles may generate thousands of image/subgoal pairs.

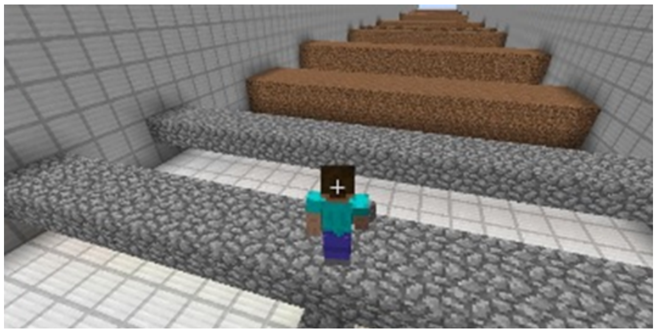

Figure 6: An example of imagery that was used to train the CNN. The image contains Steve as well as multiple obstacles. The imagery is dependent on the viewing angle of Steve.

The generated images is representative of what a human player would see while running the course. This perspective is from behind the player and was chosen as it maximizes the amount of state information for surrounding blocks present in an imagry. The image is not only dependent on the content of the scene but also the viewing angles. Because of this, we generate training data with variable viewing angles (both azimuth and elevation).

The expert procedure is biased: it chooses the 'walk' subgoal 78% of the time. For training purposes we removed this bias by undersampling the data set removing the between-class imbalance [He and Garcia, 2009]. This yields a total of 348 frames per subgoal with 278 for training and 70 for validation. The assignment of which frames to keep, as well as whether a frame is used for validation or training, was chosen randomly.

Finally, we generated a set of test data by running ACTORSIM on an independent obstacle course with varing viewing angles resulting in 892 image/subgoal pairs. We use all 892 images for testing maintaining the unbalanced distribution of subgoals inherent in the problem. The images (with labels removed) are run through the CNN which generates a subgoal which can be compared to the subgoal "truth" generated by the expert procedure.

4.2 CNN Architecture

Convolutional Neural Networks (CNNs), a specific DL architecture, have frequently been used to process imagery for classification problems. We used the trained DL architecture defined in [Krizhevsky *et al.*, 2012] (i.e., AlexNet) as the basis for our DL network.

This architecture is implemented in Caffe [Jia *et al.*, 2014], which defines the architecture and learning procedure for the CNN. In our study experimentation we used the weights from the origonal AlexNet model for all but the final inner product layer.

We applied a fine-tuning procedure to train this network [Karayev *et al.*, 2014]. We modified the final inner product layer, which has 1000 outputs, to instead output our four possible subgoals. During training, the weights of this layer were tuned using the standard learning rate and all other layers were trained using a reduced rate. The CNN model trained quickly, achieving high accuracy in a short amount of time as shown in Figure 8.

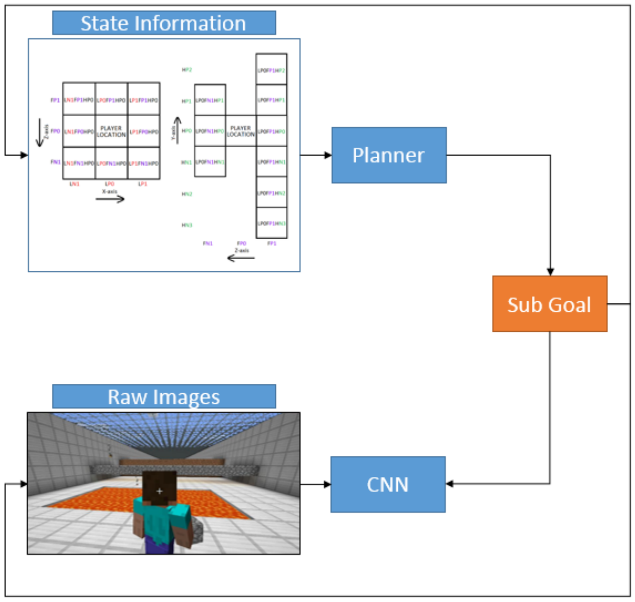

Figure 7: The approach for training a CNN. The ACTORSIM Connector produces a subgoal given state information of Minecraft. This subgoal, and the corresponding imagery, are used to train the network.

5 Results

We tested the trained CNN by applying it to all 892 frames in the test set. The CNN achieved an average accuracy of 87.1% when compared to the subgoals generated by ACTORSIM. This accuracy is higher than a policy which picks the most common subgoal which would have accuracy of 78%. Table 1 displays the confusion matrix, which plots the CNN's predicted subgoal vs. the subgoal generated by ACTORSIM. Of the 115 frames that incorrectly predicted, 83 were labeled with the previous (correct) subgoal selection. This suggests that taking into account previous state information, in addition to current imagery, could greatly increase the overall accuracy of the DL network.

Many of the errors are due to occlusion, as not enough state information can be generated from the imagery due to either the viewing angle or the agent blocks the forward path, as shown in Figure 9. This lead to a problem where the agent would walk up to a location, correctly choose the subgoal, and then not realize that the subgoal had already been executed. We discuss plans for handling this problem in Section 6.

6 Summary and Future Work

We presented a pilot study on performing subgoal selection for a limited version of overcoming obstacles in Minecraft. We found that it is possible to train a Deep Learning (DL) network to perform well on this subgoal selection task. This network, which we trained using information from a goal reasoning

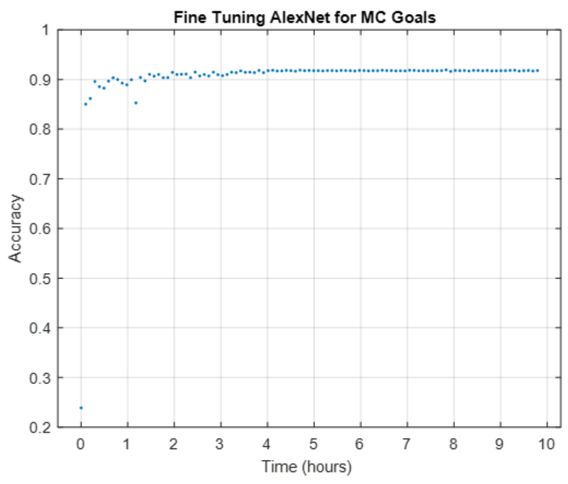

Figure 8: Training results of the finetuned CNN. The architecture is a modified version of AlexNet which has been repurposed for subgoal selection. Raw imagery is input into the network a suggested subgoal is returned bridging the gap between sensory information and appropriate policy.

		Predicted			
		Walk	**Stairs**	**Remove**	**Bridge**
Actual	Walk	632	10	6	51
	Stairs	0	26	0	0
	Remove	10	3	64	0
	Bridge	35	0	0	55

Table 1: The confusion matrix for the 892 image/subgoal pairs. This table shows a comparison between the actual subgoal as deteremined by ACTORSIM and the subgoal predicted by the CNN given imagery only.

simulator, can predict the proper subgoal using only imagery 87.1% of the time.

Our current implementation is limited by the inability to remember previous subgoal selections, which may be alleviated by adding memory to the network in the form of Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]. This technique has begun to be explored in video game tasks [A. Summerville, 2016] and would likely improve a network's ability to navigate complex environments. By allowing state information to be remembered it will likely enable a perspective shift from behind the character (third person) to a perspective from the characters perspective (first person).

In the future, we plan to study more complex tasks such as agents that must protect themselves against mobs. A first step in this direction will be to encode our goal network using the goal lifecycle provided in ACTORSIM, since our current implementation applies goal reasoning without using much

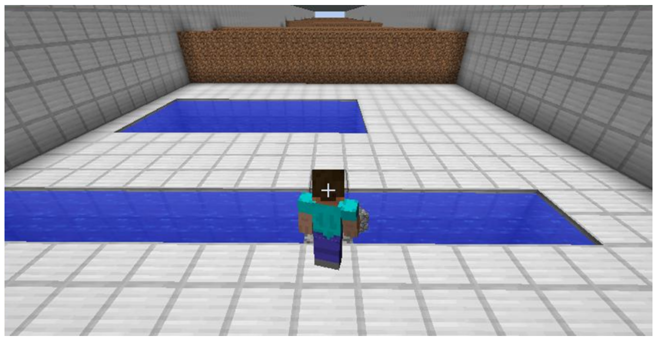

Figure 9: An example of occlusion. Here the proper action is to walk forward onto the previously constructed bridge. However the bridge is not easily seen in front of Steve resulting in the CNN predicting that a bridge needs to be constructed. This type of error, where the CNN produces a subgoal which was accomplished in the previous process, represents 83 of the 115 errors produced by the CNN.

of its functionality. This will allow us to build (or learn) more sophisticated goal networks and to leverage existing planning and scheduling techniques in ACTORSIM. Finally, we plan to include non-playing characters in Minecraft with our resulting goal networks. Each of these advancements requires an advanced understanding of imagery and high level goal reasoning.

Finally, we plan to couple ACTORSIM with the BURLAP reinforcement learning platform [MacGlashan, 2015], which would incorporate the DL and goal reasoning portions of the system and allow us to more easily integrate learning with GTN planning.

7 Acknowledgements

We wish to thank the NRL for sponsoring this research.

References

[A. Summerville, 2016] M. Mateas A. Summerville. Super mario as a string: Platformer level generation via LSTMs. arXiv prepring arxiv: 1603.00930, 2016.

[Abel *et al.*, 2015] David Abel, David Ellis Hershkowitz, Gabriel Barth-Maron, Stephen Brawner, Kevin OFarrell, James MacGlashan, and Stefanie Tellex. Goal-based action priors. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, 2015.

[Alford *et al.*, to appear] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proc. of the Int'l Joint Conf. on AI (IJCAI)*. AAAI Press, to appear.

[Aluru *et al.*, 2015] Krishna Aluru, Stefanie Tellex, John Oberlin, and James Macglashan. Minecraft as an experimental world for AI in robotics. In *AAAI Fall Symposium*, 2015.

[He and Garcia, 2009] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Trans. on Knowl. and Data Eng.*, 21(9):1263–1284, September 2009.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[Jia *et al.*, 2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[Karayev *et al.*, 2014] Sergey Karayev, Matthew Trentacoste, Helen Han, Aseem Agarwala, Trevor Darrell, Aaron Hertzmann, and Holger Winnemoeller. Recognizing image style. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[Lipovetsky *et al.*, 2015] N. Lipovetsky, M. Ramirez, and H. Geffner. Classical planning with simulators: Results of the atari video games. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*. AAAI Press, 2015.

[MacGlashan, 2015] James MacGlashan. The brown-umbc reinforcement learning and planning (burlap) library. Available at: http://burlap.cs.brown.edu/, 2015.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[Nau *et al.*, 2003] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *J. of Art. Intell. Res.*, 20:379–404, 2003.

[Nau, 2007] D. Nau. Current trends in automated planning. *Art. Intell. Mag.*, 28(40):43–58, 2007.

[Roberts *et al.*, 2016] Mark Roberts, Ron Alford, Vikas Shivashankar, Michael Leece, Shubham Gupta, and David W. Aha. Goal reasoning, planning, and acting with actorsim, the actor simulator. In *ICAPS Workshop on Planning and Robotics (PlanRob)*, 2016.

[Shivashankar *et al.*, 2013] Vikas Shivashankar, Ron Alford, Ugur Kuter, and Dana Nau. The GoDeL planning system: a more perfect union of domain-independent and hierarchical planning. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 2380–2386. AAAI Press, 2013.