Artificial Intelligence for Human-Robot Interaction
Papers from the AAAI 2015 Fall Symposium

Minecraft as an Experimental World for AI in Robotics

Krishna Aluru and **Stefanie Tellex** and **John Oberlin** and **James MacGlashan**
Humans to Robots Laboratory
Brown University
Providence, RI 02912

Abstract

Performing experimental research on robotic platforms involves numerous practical complications, while studying collaborative interactions and efficiently collecting data from humans benefit from real time response. Roboticists can circumvent some complications by using simulators like Gazebo (Koenig and Howard 2004) to test algorithms and building games like the Mars Escape game to collect data (Chernova, DePalma, and Breazeal 2011). Making use of existing resources for simulation and game creation requires the development of assets and algorithms along with the recruitment and training of users. We have created a Minecraft mod called BurlapCraft which enables the use of the reinforcement learning and planning library BURLAP (MacGlashan 2015) to model and solve different tasks within Minecraft. BurlapCraft makes AI-HRI development easier in three core ways: the underlying Minecraft environment makes the construction of experiments simple for the developer and so allows the rapid prototyping of experimental setup; BURLAP contributes a wide variety of extensible algorithms for learning and planning, allowing easy iteration and development of task models and algorithms; and the familiarity and ubiquity of Minecraft trivializes the recruitment and training of users. To validate BurlapCraft as a platform for AI development, we demonstrate the execution of A* (Hart, Nilsson, and Raphael 1968), BFS, RMax (Brafman and Tennenholtz 2003), language understanding, and learning language groundings from user demonstrations in five Minecraft "dungeons."

Introduction

Robots can help humans execute simple tasks in controlled environments for applications such as manufacturing and assistance of the disabled and elderly. Increasing the capability of robots to handle diverse, complex tasks and environments requires performing experiments and gathering data to develop and train new algorithms.

Performing experimental research on robotic platforms is expensive because of the cost of robotic hardware, it consumes lots of time and labor in order to control the environment, and involves problems in planning and perception which are difficult and slow to solve due to large state and input spaces. Using a platform where we can model and study tasks without such hurdles can provide significant time gains for both developers and users.

Figure 1: A Simulated World in Minecraft

Two virtual platforms which have been used to develop AI for robots are simulators and games. The robotics community uses the simulator Gazebo (Koenig and Howard 2004) to develop algorithms for grasping and movement, among other tasks. Developing a game from scratch requires significant overhead. Using a simulator such as Gazebo removes some of the overhead by providing a realistic world, but requires the developer to provide assets and algorithms. Even if a convenient game engine such as Unity (Uni 2015) provides assets and physics, algorithms must be sourced and user bases established and trained.

To address these limitations, we combined Minecraft, a game consisting of a rich interactive 3D environment, with the Brown-UMBC Reinforcement Learning and Planning (BURLAP) Java code library (MacGlashan 2015), which is powerful and extensible.

Minecraft is an open-ended game where players gather resources and build structures by destroying and stacking 3-D blocks in a virtual world; an example world is shown in Figure 1. At over a 100 million registered users, it is among the most popular video games of all time (Makuch 2014). Minecraft's state and action space allow users to create complex systems, including logic gates and functional scientific graphing calculators. It can serve as a platform to model a

wide range of robotics tasks such as cooking assistance, assembling items in a factory, object retrieval and complex terrain traversal.

We created an open source Minecraft mod called BurlapCraft[1] to model different tasks within the game. BurlapCraft uses the Minecraft Forge API, which contains hooks into Minecraft, to shuttle information between BURLAP and the Minecraft world. BURLAP's rich state and domain representation framework, based on the Object-Oriented Markov Decision Process (OO-MDP) (Diuk, Cohen, and Littman 2008) paradigm, can model tasks within Minecraft, making it possible to transfer object-based knowledge for planning across environments.

BurlapCraft makes AI development easier in three core ways: (1) the underlying Minecraft environment makes the construction of experiments simple for the developer and so allows the rapid prototyping of experimental setup; (2) BURLAP contributes a wide variety of extensible algorithms, allowing easy iteration and development of task models and algorithms; and (3) the familiarity and ubiquity of Minecraft trivializes the recruitment and training of users.

To validate BurlapCraft as a platform for AI development, we solved navigation and block placement tasks in five dungeons. We also used these dungeons to train language groundings that enabled a user to give natural language commands that were carried out by the agent.

System Design

An overview of the BurlapCraft system design is shown in Figure 2. The system begins with the standard Minecraft game server. The first component of BurlapCraft is a function that examines the world around the player at any given time and turns it into an Object-Oriented MDP (OO-MDP) state representation. This state is passed to an agent that uses one of BURLAP's planning or learning algorithms to choose an action. The selected action is then passed to BurlapCraft's action controller for execution in the environment, after which the process repeats. Reward functions or goals necessary to motivate the agent are typically specified through some other user-defined channel since there are no intrinsic rewards or goals in Minecraft. We talk about each piece of this system in more detail next.

State Representation

For the state representation, we adopt the Object-oriented MDP (OO-MDP) formalism, which is supported in BURLAP and aligns well with the needs of Minecraft. An OO-MDP extends the classic MDP formalism by including a set of object classes, each defined by a set of attributes, and a set of propositional functions whose parameters are typed to object classes. The propositional functions provide high-level information about the state that can useful for planning and learning. For example, consider an OO-MDP that includes room and block objects defined by their position in the world. Such an OO-MDP can include a "blockInRoom" propositional function that operates on block and

[1]Code and tutorials are available online at: https://github.com/h2r/burlapcraft

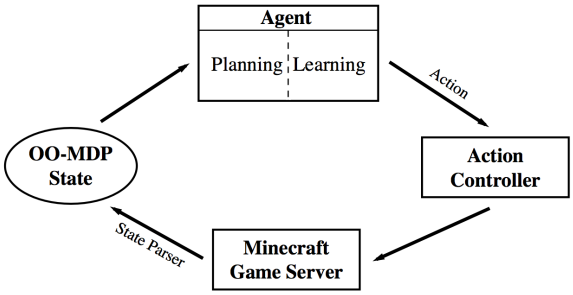

Figure 2: An overview of the BurlapCraft system design. At each time step, an OO-MDP state representation of the current state of the Minecraft server is created; this created state is passed to an agent employing a BURLAP reinforcement learning or planning algorithm to select actions; the selected action is actuated by BurlapCraft's low-level action controller; then the cycle repeats.

room objects and returns true if the block is within the room. In Minecraft, we represent states by an agent object (defining information about the player), block objects (which correspond to interactive blocks in Minecraft), special room objects (which define contiguous areas in the world), and inventory objects (objects the agent has in their player inventory). A visual representation of the object classes and their attributes are shown in Figure 3. This representation can be trivially extended to encompass more of Minecraft by adding additional object class definitions (for example, object classes for farm animals could be included). The text description for a simple example OO-MDP state for Minecraft using this representation is shown in Figure 4.

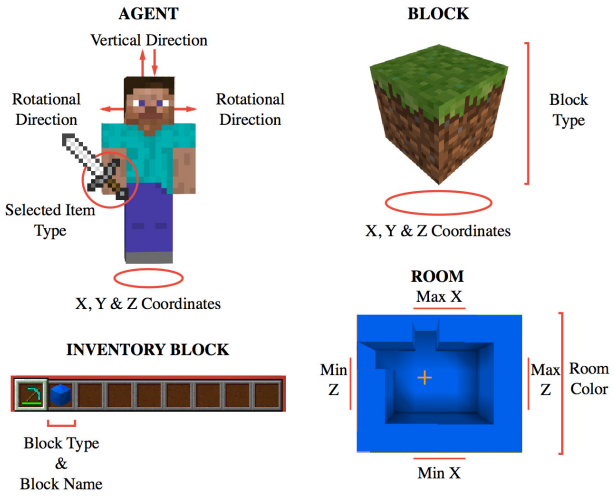

Figure 3: Object Classes and their respective Attributes. The rotational direction for south, west, north, and east corresponds to the numbers zero to three, respectively.

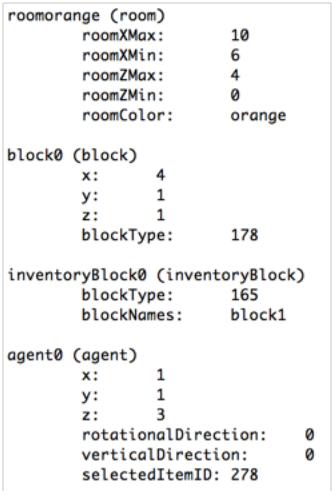

Figure 4: An example state from a Minecraft dungeon

Action	Effect
Move Forward	Agent's X +/- 1 or agent's Z +/- 1 based on the agent's rotational direction.
Rotate Left	Agent's rotational direction cycles backwards from 3 to 0.
Rotate Right	Agent's rotational direction cycles forwards from 0 to 3.
Look Down	Agent's vertical direction is set to 1.
Look Ahead	Agent's vertical direction is set to 0.
Change Item	Agent's selected item type changes to the type of the item in the next slot. It is set to -1 if there is no item in the next slot.
Destroy Block	The instance of the block is removed from the state and a new inventory block instance is added to it.
Place Block	The instance of the inventory block is removed from the state and a new block instance is added to it.

Table 1: Possible actions and their effects.

Agent Control

When the Minecraft game is parsed into an OO-MDP state, it is passed to an agent that must decide how to act. BurlapCraft currently has 8 possible actions that can be selected by the agent and then passed to a low-level action controller that actuates the action in the Minecraft game server. The list of all currently supported actions and their effects are listed in Table 1. The actions are implemented by overriding mouse and keyboard inputs for a fixed period of time or by calling Minecraft Forge's API methods. For example - for the "Move Forward" action, we override keyboard input by simulating a click on the "w" key for 1.5 seconds whereas for the "Change Item" action, we call an inbuilt method to set the agent's current item to the next item in the agent's inventory. This makes its possible to define other actions like crafting or smelting, which we plan to implement in the future.

Agents implementing learning algorithms will either learn their own model of the world and choose actions accordingly (as in model-based reinforcement learning), or directly learn the utilities of the actions to determine action selection (as in model-free reinforcement learning). To facilitate planning algorithms, BurlapCraft includes a model of Minecraft's dynamics for all included actions. This model allows a planning algorithm to take as input the initial state of the problem, generate a policy for it, and the follow the computed policy in a closed loop fashion until task termination. Given the state representation and (in the case of planning) the model, any of BURLAP's included reinforcement learning or planning algorithms can be used to control the agent in Minecraft.

Learning from Demonstration

Another area of active research is learning from demonstration (LfD) (Argall et al. 2009), in which a human provides example demonstrations of a behavior from which an agent learns. Minecraft is an especially appealing space to perform this kind of research since it includes a large user base that is familiar with playing the game. Therefore, BurlapCraft includes a demonstration extractor in which the human plays the game with standard keyboard and mouse input and an OO-MDP state and action sequence usable by LfD algorithms is produced. The demonstration extractor operates by frequently polling the state parser every 100ms. If the state is a duplicate of the most recently observed state, then it is discarded from the demonstration sequence. Because the action inputs the human uses (keyboard and mouse) are different than the MDP action set used by the BurlapCraft agent, the actions applied between the observed states are inferred by finding the action in the BurlapCraft action set that would lead to the observed state transition.

Case Studies

In this section we explore multiple learning and planning applications within Minecraft and highlight how Minecraft presents a challenging environment that motivates future research. For each of these case studies we created small "dungeons" in which to test the algorithm. We used five different test dungeons which we will describe in the relevant section where they were used: grid, bridge, easy maze, tough maze, and four rooms.

Planning

For applying planning algorithms in Minecraft we used BURLAP's implementation of breadth-first search (BFS) and A* to solve the bridge dungeon, easy maze, and tough maze. In the case of A*, we also provided a Manhattan distance heuristic to the goal gold block.

The bridge dungeon (Figure 5) is an enclosed area with dimensions 10x10x5 and it contains a mineable block, a gold block and lava separating the two halves of the dungeon. The goal in this dungeon is to reach the gold block, but to do so, the agent will need to mine the block and place in the lava so that it can cross unharmed to the gold block. We use this

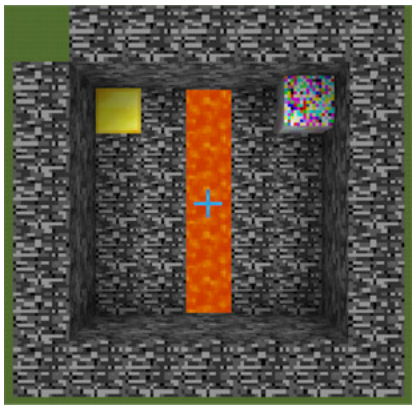

Figure 5: 7x7 Bridge Dungeon. Actions used by the agent to solve the dungeon include 'Rotate Left', 'Rotate Right', 'Move Forward', 'Place Block', 'Change Item', 'Look Down' and 'Destroy Block.'

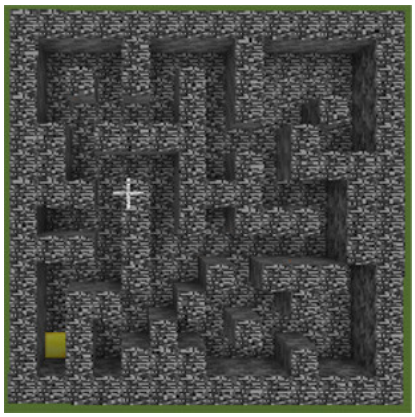

Figure 6: 14x14 Easy Maze. Actions used by the agent to solve the dungeon include 'Rotate Left', 'Rotate Right' and 'Move Forward.'

dungeon as a simple example of a reach goal location problem that requires more complex mining and building behavior.

The two maze dungeons: easy maze (Figure 6) and tough maze (Figure 7) are identical 14x14x4 mazes with the goal of reaching a target gold block from the furthest point away from it. However, the tough maze also includes 6 blocks that need to be destroyed to reach the goal gold block's location.

For the bridge dungeon and the easy maze, both BFS and A* were able to solve the problem in less than 1ms (executing the plan takes longer since the BurlapCraft's action controller takes 1.5 seconds per action). However, attempting to solve the tough maze resulted in both A* and BFS running out of memory before solving the problem. Although the differences between the easy and hard maze seem trivial, the ability for the agent to destroy and replace the blocks anywhere else in the maze greatly increases the size of the state space resulting in a difficult planning problem. Specifically, since the maze has 90 traversable cells, including 6 replaceable blocks multiplies the size of the state space by $\binom{90}{6} = 622,614,630 \approx 10^8$ states. Tackling a more full version of Minecraft with even more blocks that can be mined and placed will present even greater challenges and we believe will help motivate the development of better planning algorithms that can handle these kinds of complex spaces.

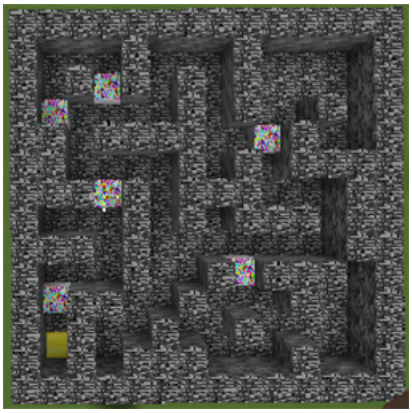

Figure 7: 14x14 Tough Maze. Actions used by the agent to solve the dungeon include 'Rotate Left', 'Rotate Right', 'Move Forward', 'Destroy Block' and 'Look Down'.

Learning

We next use BURLAP's RMax (Brafman and Tennenholtz 2003) implementation as a demonstration of using a reinforcement learning (RL) algorithm within BurlapCraft. RMax is a model-based RL algorithm that uses an "optimism in the face of uncertainty" heuristic to guide exploration until it has a confident model of how the world works. We applied RMax with a tabular learning model in a small 5x5x3 grid dungeon shown in Figure 8. It takes 3 to 4 episodes of RMax within the grid dungeon to learn the transition dynamics and generate an optimal plan on all consecutive runs, which took a total of 197 seconds with a 1.5 second execution time for actions. In larger dungeons with more blocks, this learning time would rapidly grow. RMax is typically accelerated by including compact model representations that can be quickly learned with little data. For example, SLF-RMax (Strehl, Diuk, and Littman 2007) is capable of accelerating learning by learning a dynamic Bayesian network to model the environment dynamics, but since Minecraft has an extremely large number of state variables that increase with the number of blocks and block types, it will not be able to scale well. Deterministic Object-oriented RMax (Diuk, Cohen, and Littman 2008) factors the model dynamics along the objects in an OO-MDP and can generalize well, but can not handle any uncertainty and is not capable of handling all of the possible complex transitions in Minecraft with providing a large amount of background knowledge. We believe these challenges will motivate the development of additional RL algorithms.

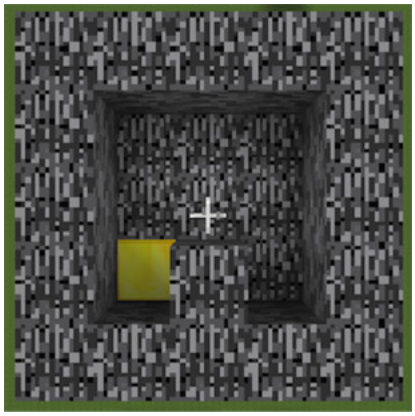

Figure 8: 5x5 Grid Dungeon

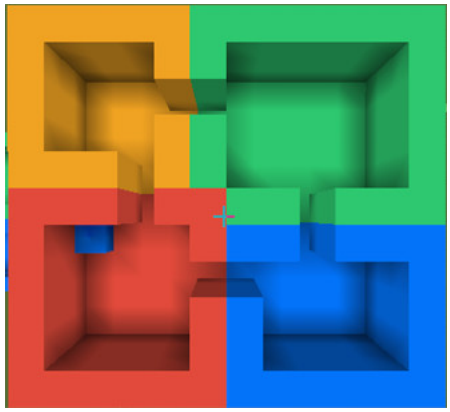

Figure 9: 12x12 Four Rooms Dungeon

Learning Language Groundings

An intuitive way for people to convey tasks that they want a robot to complete is to communicate the task through natural language. Minecraft players who use specialized bots to perform monotonous tasks, such as Minebot (Min 2014), may also benefit from being able to communicate the task they what carried through natural language. In this case study, we apply the methods described in MacGlashan et al. (2015) to learn natural language groundings from demonstrations which allows a user to iteratively train a language model that can be executed by the agent.

The goal of this language grounding work is to interpret natural language commands as high-level task reward functions that indicate a goal for the agent to complete. After interpreting the command as a reward function, a planning algorithm can be used to determine the appropriate behavior that solves the task. Grounding to reward functions allows tasks to be communicated in a variety of environments without requiring step-by-step solution information to be conveyed by the human user. To train the model, a user supplies example commands paired with demonstrations (state-action sequences) that carry out the example command. From the demonstrations, the agent infers a latent reward function that motivated the behavior. The reward function representation uses OO-MDP propositional functions to provide a logical structure that allows language to ground to it using the IBM Model 2 language model (Brown et al. 1990; 1993). Minecraft is a particularly appealing environment to test this kind of work because it has a very large user base that can be queried to provide expert demonstrations and commands for a variety of tasks.

In our first application of language grounding in BurlapCraft, we took the Amazon Mechanical Turk dataset used by MacGlashan et al., trained the language model on it, and built Minecraft environments and propositional functions that allowed the commands from that dataset to be interpreted and executed in Minecraft. In this dataset, users provided commands (that were paired with demonstrations) for taking toy objects like stars to different colored rooms or for navigating to different rooms. An example environment that we constructed is shown in Figure 9. Using this reconstruction of the dataset environments, we were able to have a user playing Minecraft enter a command taken from the dataset through the Minecraft command prompt and have it successfully interpreted and executed by the agent. A video demonstrating one of the commands being entered, interpreted, and executed has been made available online.[2]

We then extended this dataset transfer application to allow more Minecraft specific language and tasks to be learned and demonstrated in an online fashion. First, we added propositional functions used to describe tasks to include properties about Minecraft blocks and whether the agent was standing on blocks. Next, we set up a framework where the user can issue the 'learn' command in Minecraft's command prompt followed by the language command that they wish to demonstrate. After entering the command, the user demonstrates the behavior with the keyboard and mouse and we extract a state-action sequence from the behavior using BurlapCraft's demonstration extractor. This command-action pair is then added to an existing dataset and the language model is retrained with an additional Minecraft command prompt command.

As a demonstration of this application, we taught an agent about the "go to the gold block" task that we used in the previously discussed planning and learning application. In this case, we gave the agent a single training instance that consisted of the command "go to the gold block", with a user-provided demonstration of walking to the gold block from two steps away, as shown in Figure 10. After training the language model on this one demonstration, we gave the same command to the agent from a state that was further away from the gold block with a stream of lava separating the agent from it. As shown in Figure 11, the agent correctly interpreted the goal and used planning to determine that it would need to mine the free block to build a bridge, and then executed the plan. We are also then able to have this command successfully interpreted and executed in other dungeon environments, such as the easy maze dungeon.

Although this is an effective application of existing language grounding work, the Minecraft environment helps to

[2] Video available at https://vid.me/H7B1

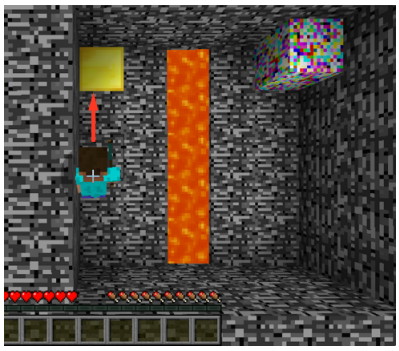

Figure 10: Demonstration provided in the Bridge Dungeon for the language command ‘go to the gold block’

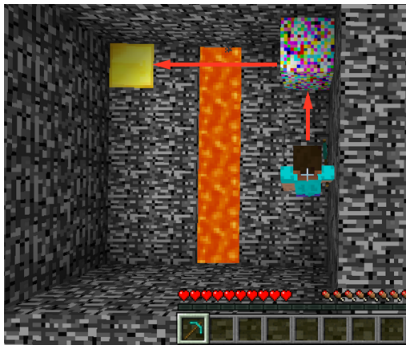

Figure 11: When issued the language command ‘go to the gold block’, the agent mines the multi-colored block and places it on the lava to cross over and the reach the gold block.

motivate future research. For example, we imagine users would want to be able to train an agent to follow commands like “build a house,” which describes a complex abstract goal that cannot currently be captured in MacGlashan et al.’s model or others to our knowledge.

Related Work

In general there two classes of approaches people have used for training robotics AI systems in virtual worlds: applications in games and more open ended environment engines.

The Mars Escape game (Chernova, DePalma, and Breazeal 2011) requires the players to work together to escape Mars before their oxygen supplies run out. The game is used to study human teamwork, coordination and interaction by gathering data from the players. This data is used to generate a behavior model that can be used by an autonomous robot to perform similar tasks. There are also games like Sophie’s Kitchen (Thomaz and Breazeal 2007), a virtual house where the player’s goal is to help Sophie bake a cake, and the ESP game (Von Ahn 2006), where players need to determine the contents of images by providing meaningful labels for them. All these games have a very narrow task scope and there is additional overhead associated with recruiting actual users to play and familiarize themselves with the games’ interfaces. On the other hand, a wide variety of tasks can be modeled quickly in Minecraft and it is popular enough to attract players who are already familiar with the dynamics of the game.

The Atari Learning Environment (ALE) (Bellemare et al. 2013) allows AI systems to interface with Atari 2600 games and is typically used in reinforcement learning research. For example, deep belief networks have recently had much success in outperforming human players in many of the games (Mnih et al. 2015). Although ALE provides a larger task scope than some other games used in research, it is still more narrow than Minecraft which is a sophisticated enough environment to support the construction of computers inside its world.

There are also simulation toolboxes like Gazebo (Koenig and Howard 2004), which have a much wider scope for modeling robotics tasks. Gazebo can be used to rapidly iterate algorithms, design robots, and perform tests using realistic scenarios. Nevertheless, it takes a considerable amount of time to create the simulation of a physical model and train users to use that simulation.

Our approach with BurlapCraft overcomes a lot of these problems in four ways: (1) there will not be a learning curve for most Minecraft users to use the dungeons; (2) BurlapCraft allows researchers to rapidly create worlds and experiments; (3) BURLAP has a lot of solvers and models built-in that can be easily used; and (4) the range of complexity for tasks that can be created in Minecraft is enormous.

There are also many systems that have exploited the compositional structure of language to generate symbolic representations of the natural language input (Kress-Gazit and Fainekos 2008; Dzifcak et al. 2009; Skubic et al. 2004; Tellex et al. 2011). Existing work in dialog systems (Doshi and Roy 2008; Roy, Pineau, and Thrun 2000; Williams and Young 2007a; 2007b; Young 2006) use MDP and POMDP models with a fixed, predefined state space to represent the user’s intentions. Many of these models require large amounts of training data to work effectively, data which is difficult to collect and evaluations that require real-world studies. We believe interfacing with Minecraft will facilitate the data gathering process since there is a large user base and a lot of material written about it online.

As far as using Minecraft itself as a research platform, Branavan et al. (2012) described a system for learning to play Minecraft by reading the Minecraft wiki. However they used a model of Minecraft defined in STRIPs rather than the actual game, making it difficult for people to interact with the agent and train it. We also believe that having an agent actually interact with the real game will introduce complexities that are not otherwise captured in STRIPS models.

Conclusion

This paper introduces a new Minecraft mod, BurlapCraft, that can be used to model and solve robotics tasks within a world that has varying complexity making it a useful test bed for AI in robotics. The ease of designing tasks in Minecraft and our integration with BURLAP’s range of extensible learning and planning algorithms allows rapid development of a variety of experiments. In this work, we

demonstrated how planning and learning algorithms can be quickly applied in five different dungeons. We have also built in the ability to collect demonstrations from users of the mod which was used for language learning, but could also be used for other learning from demonstration work.

Our results demonstrate how quickly Minecraft problems can become challenging for classic planning and learning algorithms and we believe these challenges will motivate the development of better planning and learning algorithm techniques that can be quickly validated and compared to other work. MacGlashan et al's language grounding work that we applied can only manage modeling simple tasks and cannot model more complex tasks that we believe users would want to teach, like building houses. High-level tasks like these may benefit from hierarchical task structures and can inspire additional work. In the future, we would like to be able to leverage the user base that comes with Minecraft to collect new and interesting data for such tasks and develop algorithms that can support it.

Acknowledgements

This work was supported in part by DARPA under "Hierarchical, Probabilistic Planning and Learning for Collaboration" grant # W911NF-15-1-0503.

References

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robot. Auton. Syst.* 57(5):469–483.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Brafman, R. I., and Tennenholtz, M. 2003. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research* 3:213–231.

Branavan, S.; Kushman, N.; Lei, T.; and Barzilay, R. 2012. Learning high-level planning from text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, 126–135. Association for Computational Linguistics.

Brown, P. F.; Cocke, J.; Pietra, S. A. D.; Pietra, V. J. D.; Jelinek, F.; Lafferty, J. D.; Mercer, R. L.; and Roossin, P. S. 1990. A statistical approach to machine translation. *Comput. Linguist.* 16(2):79–85.

Brown, P. F.; Pietra, V. J. D.; Pietra, S. A. D.; and Mercer, R. L. 1993. The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.* 19(2):263–311.

Chernova, S.; DePalma, N.; and Breazeal, C. 2011. Crowdsourcing real world human-robot dialog and teamwork through online multiplayer games. *AI Magazine* 32(4):100–111.

Diuk, C.; Cohen, A.; and Littman, M. 2008. An object-oriented representation for efcient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08.

Doshi, F., and Roy, N. 2008. Spoken language interaction with model uncertainty: an adaptive human–robot interaction system. *Connection Science* 20(4):299–318.

Dzifcak, J.; Scheutz, M.; Baral, C.; and Schermerhorn, P. 2009. What to do and how to do it: translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, 3768–3773. Piscataway, NJ, USA: IEEE Press.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Koenig, N., and Howard, A. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, 2149–2154. IEEE.

Kress-Gazit, H., and Fainekos, G. E. 2008. Translating structured English to robot controllers. *Advanced Robotics* 22:1343–1359.

MacGlashan, J.; Babeş-Vroman, M.; desJardins, M.; Littman, M.; Muresan, S.; Squire, S.; Tellex, S.; Arumugam, D.; and Yang, L. 2015. Grounding English commands to reward functions. In *Robotics: Science and Systems*.

MacGlashan, J. 2015. Brown-umbc reinforcement learning and planning (burlap). http://burlap.cs.brown.edu/. Accessed: 2015-07-30.

Makuch, E. 2014. Minecraft passes 100 million registered users, 14.3 million sales on pc. http://www.gamespot.com/articles/minecraft-passes-100-million-registered-users-14-3-million-sales-on-pc/1100-6417972/. Accessed: 2015-08-06.

2014. Minebot. http://www.minecraftforum.net/forums/mapping-and-modding/minecraft-mods/wip-mods/2283099-minebot-0-3-released-can-now-store-in-chests-craft. Accessed: 2015-08-05.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Roy, N.; Pineau, J.; and Thrun, S. 2000. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*.

Skubic, M.; Perzanowski, D.; Blisard, S.; Schultz, A.; Adams, W.; Bugajska, M.; and Brock, D. 2004. Spatial language for human-robot dialogs. *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 34(2):154–167.

Strehl, A.; Diuk, C.; and Littman, M. 2007. Efficient structure learning in factored-state mdps. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, 645.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M.; Banerjee, A.; Teller, S.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proc. AAAI*.

Thomaz, A. L., and Breazeal, C. 2007. Asymmetric interpretations of positive and negative human feedback for a social learning agent. In *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*, 720–725. IEEE.

2015. Unity game engine. https://unity3d.com/. Accessed: 2015-08-06.

Von Ahn, L. 2006. Games with a purpose. *Computer* 39(6):92–94.

Williams, J. D., and Young, S. 2007a. Scaling POMDPs for spoken dialog management. *IEEE Transactions on Audio, Speech, and Language Processing* 15(7):2116–2129.

Williams, J. D., and Young, S. 2007b. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language* 21(2):393–422.

Young, S. 2006. Using POMDPs for dialog management. In *IEEE Spoken Language Technology Workshop*, 8–13.