Automated Planning with Goal Reasoning in Minecraft

Mark Roberts[1] **Wiktor Piotrowski**[2] **Pryce Bevan**[3] **David Aha**[1] **Maria Fox**[2] **Derek Long**[2] **Daniele Magazzeni**[2]
[1]Naval Research Laboratory, Code 5514; Washington, DC, USA | {first.last}@nrl.navy.mil
[2]Department of Informatics, King's College London, United Kingdom | {first.last}@kcl.ac.uk
[3]Georgetown University, Washington, USA | pwb8@georgetown.edu

Abstract

We combine PDDL/PDDL+ planning with goal reasoning to leverage the strengths of both and succeed in a limited variant of Minecraft. Automated planners have long been able to reason about numeric fluents and exogenous events, but remain largely confined to closed worlds with full observability. Goal reasoning can respond to dynamic, open worlds and partial observability, but must rely on an effective planner. We demonstrate that combining goal reasoning with automated planning reduces the overall computational effort to achieve goals while succeeding at multiple domain specific metrics. We highlight important design decisions in PDDL1.2, PDDL2.1, and PDDL+, including the use of PDDL+ events to model opportunistic goals. We close with a discussion of trade-offs associated with choosing the modeling features and identify a number of challenges for the next generation of planning systems.

1 Alex's Quest

Consider an agent, Alex, at the end of a 300-meter hallway with entries to dozens of rooms along each side containing randomly placed resources (e.g., wood, diamonds, iron ore, and coal), necessary crafting equipment (e.g., a workbench for crafting and a furnace for smelting ore), and randomly spawned zombies which can harm Alex. Alex can only observe the world directly nearby and must reach the far end of the hallway with a crafted diamond sword in hand. After this hallway, Alex faces a dungeon filled with enemies, therefore it would be best to also have a full complement of armor, a stack of torches, some ladders, etc.

This quest is derived from Minecraft, an open-world sandbox game where players choose their own objectives such as building structures, collecting resources, crafting, fighting enemies, or exploring. The world consists of 1 meter voxels (i.e., cubes). Minecraft has many properties of an ideal testbed for designing planning and acting techniques. It includes such issues as multiple agents (both cooperative and adversarial), partial observability, complex tasks (e.g. crafting tools, building), to name a few. The quest, and similar Minecraft challenges, can be easily solved by automated planning systems using replanning with two additions: opportunistic goals provided by PDDL+ events and goal reasoning.

PDDL+ (Fox and Long 2006) is an extension of the standardized modeling language in automated planning, PDDL (IPC Committee 1998). In conjunction with previous versions, PDDL+ introduces independent processes and exogenous events. Processes have time-dependent continuous effects, whereas events bring about instantaneous discrete change. Both elements they are triggered *by the environment* as soon as their preconditions are satisfied; the planner does not control processes/events. Events are ideal to model the appearance of entities and resources and enable the modeling of opportunistic goals.

Goal reasoning allows an agent to deliberate about its own goals during execution. This allows an agent to be more adaptable to changes in the world by determining, for example, when it should replan or when it should adjust its sensing. A recent implementation of goal reasoning, called ACTORSIM (Roberts et al. 2016b), has been applied to Minecraft, but ACTORSIM lacked a connection to PDDL-based planners and some features needed to support observations.

We extend ACTORSIM to translate Minecraft into PDDL and examine trade-offs of this combination. The contributions of this paper include: (1) an extension of ACTORSIM to include mobs, resources, and the ability to observe them; (2) PDDL/PDDL+ models of Minecraft; to our knowledge, these are the first PDDL models of Minecraft that include randomly placed resources and zombies; (3) a discussion of how the domain modeling evolved, highlighting the strengths and shortcomings of each approach; (4) an effective mechanism for managing an open, dynamic world via opportunistic goals implemented as PDDL+ events; and, (5) an application of goal reasoning to reduce the cumulative search effort of the planner whilst maintaining domain-specific metrics such as resources collected. In the remainder of the paper, we describe the planning models we constructed, how we leverage them during goal reasoning, our evaluation of the system, and related work. We close with a discussion of future work.

2 Interacting with Minecraft

To interact with the Minecraft game, we use the publicly available tool called ACTORSIM (Roberts et al. 2016b). This connector exposes state information about the world and discrete motion primitives for controlling character. It also provides a way to construct challenge problems like the introductory quest. The previous version of ACTORSIM only supports actions to move forward, mine, or build a bridge and only reports the blocks directly around the player.

We extended ACTORSIM considerably to support our study. We added actions to move the character north, east, south, or west and actions to collect resources. We extended the sections available to include zombies, diamonds, bread, and wood as well as observations of entities and items around the player. Finally, we added a PlanManager to convert this state to PDDL and run a planner. Figure 1 shows the relationships between the core components for our study.

The PDDL planner accepts input files and produces a single plan; not shown is the PlanManager component that assists with this process by creating the PDDL files, running the planner process, and parsing the plan output. The 'Alex Controller' ensures that all actions taken are safe to execute – that is, that the character doesn't walk off a cliff or into lava. It also abstracts the game state (e.g., blocks, observations, inventory) into data structures that can be read by other components. Finally, the 'Goal Reasoner' sets up challenges, monitors experiments, and manages the character's goals.

3 Modeling Minecraft

All planning modeling languages have been designed for particular classes of domains. With a multitude of domain definition languages available, consideration of all features of the scenario and its future extensions is necessary to capture the model accurately. The choice of modeling language is crucial as it affects the size of the search space, branching factor, and generally how closely the domain represents the corresponding real world problem. A domain language not well-suited to a given problem can result in an inaccurate representation of the essence of the real-world scenario, as well as a convoluted and bloated domain. A more expressive model allows a closer resemblance to the Minecraft world but can increase search cost. We aim to arrive at a model that is expressive enough while still solvable within a few seconds. We modeled the Minecraft domain incrementally, adjusting our choice of modeling languages as the desired set of features expanded over time.

Observation and Plan Zones All the models share the concept of zones. Minecraft coordinates are given in X (east-west), Y (up-down), and Z (north-south) coordinates where north, west, and down are negative. In this paper, we focus on the X-Z grid of obstacles around the player and plan to add height in future work. Even without height, the challenges we present are 20 blocks wide by 150 blocks long. The full obstacle course takes 90-180 seconds (or more) to plan, far too slow for reactive execution. So we must consider how much of the world to reason with when encoding the local state around the player into a problem file for the planner. We define *observation and plan zones* around the player given by the distance (front,back,left,right) away from the player in each of these directions. An *observation zone* is the larger of the two, and defines the distance away from Alex that items are observed. While this reduces the level of detail, planning for the entire observation zone can still take tens of seconds or longer and is too slow for a tight execution loop. So we use a much smaller *plan zone* that the planner can solve quickly.

Consider Figure 2 which captures the abstract state from the image in Figure 1. Alex (shown as Y) is about to walk into

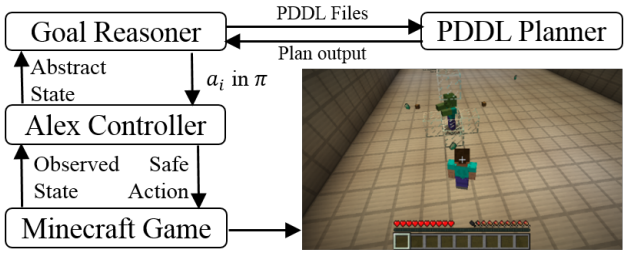

Figure 1: Overview of the planning and acting components.

```
      x3 X4 X5 X6 X7 X8 x9
z-9   D  W                W
z-8      .  .  T  .  .
Z-7      .  X  X  X  .
Z-6      .  X  X  X  .
Z-5      .  X  X  X  .
Z-4      .  .  .D .  .
Z-3      .  .  .  .  .
Z-2      .  .  .Y .  .
Z-1      .  .  .  .  .
            PDDL2.1
```

```
      x3 X4 X5 X6 X7 X8 x9
z-9   D  W                W
z-8      .  .  T  .  .
Z-7      .  .  =  .  .
Z-6      .  =  Z  =  .
Z-5      .  .  =  .  .
Z-4      .  .  .D .  .
Z-3      .  .  .  .  .
Z-2      .  .  .Y .  .
Z-1      .  .  .  .  .
            PDDL+
```

Figure 2: Sample problem and observation zones around Alex (denoted Y) containing walkable cells (.), unsafe cells (X), the intermediate target (T), glass blocks (=), diamonds (D), wood (W), and a zombie (Z). The left plot shows the PDDL2.1 representation, while the right plot shows the PDDL+ version.

a zombie (Z) noted in the PDDL 2.1 model as unsafe (X) or in the PDDL+ model as surrounded by glass (=). We cage the zombies in glass because we lack an effective way to defend against zombies; in future work, we discuss our plans for using reactive strategies for defense. Walking in any corner squares around the zombie, results in health damage, so Alex must avoid these positions if possible. The PDDL+ model encodes this knowledge but the PDDL2.1 model does not. Thus, the goal reasoner marks as unsafe all blocks around a zombie, effectively enforcing safety. The observation zone in this small example includes the entire area. The plan zone of (6,1,2,2) is six cells in front of Alex, one cell behind Alex, and two cells to the left and right. Walkable cells (.) in the plan zone are safe spots where Alex can stand. An intermediate target (T) creates a feasible subproblem for the planner. Just outside the plan zone are resources of wood (W) and diamond (D). As the end of the hallway is always north of the character, the intermediate target is the northernmost walkable cell closest to the hallway's end.

A propositional model Our first model employed PDDL1.2 (IPC Committee 1998) and relied on propositional variables to represent Alex and the surrounding area. Each cell in the grid was defined in relation to its neighboring cells (i.e. cell_i "is north of" cell_j). Discrete change and purely propositional set of variables served well as a proof of concept and the character could achieve the end goal. Research in propositional encodings is the most mature, and every

planner we tried succeeded at the model for small problem sizes. The character was able to move through hallway and overcome most obstacles, but this required replanning every 3-5 steps. The planner could sometimes stall if the planning zone wasn't big enough to get around an obstacle.

Though simple to construct, the model scaled poorly. This version only included the goal of reaching a specified target cell and extending the model to include additional tasks was particularly difficult. The key to reducing the number of replanning episodes and eliminating stalls was to increase the problem size, allowing the planner to plan further each episode. However, the problem could not scale or include height because each cell and its n-way connections must be explicitly stated in the problem file.

A Numeric PDDL2.1 model To increase the problem zone, we turned to PDDL2.1 (Fox and Long 2003) which adds numeric fluents, durative actions and continuous action effects. Expanding the model to account for entities and resources required major changes in the domain.

Instead of predicates between cells, we represented each cell with a numeric function for each dimension (e.g., x_cell, z_cell) and a type (e.g., walkable, unsafe). Each cell in the grid only takes four statements: the declaration, the x and y values, and the cell type (walkable, unsafe, etc.). Queries such as "is-north-of" are easily calculated and transitive. This formulation provided information about distances between cells, reduced the problem size, and eased scaling.

An added benefit of moving to PDDL2.1 was the ease in representing inventory and resource collection. Resource location, type, and quantity could easily be specified using numbers without adding new predicates or functions.

The PDDL 2.1 model heavily relies on the goal reasoner to add the goal conditions forcing Alex to visit cells with resources and avoid cells with zombies. Extending the domain to consider collecting resources is complicated by the fact that these are randomized. A simple approach is to add a goal condition to "collect" all resources, but this makes the problem unsolvable if no resources exist in the current plan zone. To solve this, the PDDL2.1 model uses a "visited" predicate to force the planner to collect resources. Similarly, avoiding zombies requires an "unsafe" annotation for cells around a zombie. The goal reasoner and PlanManager include these predicates for resources/zombies within the plan zone.

A Numeric PDDL2.1 model with a plan metric To facilitate collecting resources without relying on "visited" predicates, our next model employed plan metrics that maximized wood and diamond resources in the inventory, through `collect` actions. However, the metric alone does not encourage the planner to collect the drop, and instead the planner focuses on a direct trajectory to the target cell at the end of the corridor. We believe this behavior results from the bias of a planner toward the shortest plan. This approach completely ignores the resources, a focal point of this work, so we exclude this model from further discussion and evaluation. Instead, we employ PDDL+ events with what we call opportunistic goals.

A Numeric PDDL+ model with opportunistic goals PDDL+ (Fox and Long 2006) extends PDDL2.1 with independent processes and exogenous events. It has mostly been used to define hybrid systems, i.e. models exhibiting both discrete and continuous behavior, often with non-linear system dynamics. Indeed, some of the continuous behavior in Minecraft can only be represented by independent processes (e.g., health regeneration cannot be directly manipulated by Alex because Alex needs to eat, which in turn activates a continuous process increasing his health level at a steady rate.) On the other hand, events are best suited to model some of the innate features of Minecraft, such as resources or hostile entities entering Alex's field of view. In addition, PDDL+ events can also be compiled to act as opportunistic goals/plan preferences which allow us to maximize the collected resources, as described in Section 4.

Opportunistic goals enabled by PDDL+ events are triggered when such resources are observed and need to be collected. Unlike soft goals, which can be ignored by the planner, opportunistic goals must be satisfied when encountered (e.g. collect resources when seen along Alex's path or avoid zombies). Moreover, opportunistic goals are only ever considered when required (e.g. if Alex has not collected a sufficient amount of a given resource), otherwise they are ignored to avoid unnecessary computational effort.

For each type of resource, a propositional fact is added to the goal condition and falsified by events. For example, to trigger wood collection we falsify `(wood_resource_collected)`:

```
(:event wood_resource_appears
  :parameters (?start_cell - cell ?resource - cell)
  :precondition (and (alex_at ?start_cell)
    (wood_resource_collected)
    (< (wood_in_inventory) (wood_goal))
    (< (- (z_coord ?resource) (z_coord ?start_cell)) 5)
    (= (resource_type ?resource) 3))
  :effect (and (not (wood_resource_collected))))
```

This event triggers when wood is close to Alex.

There are cells that Alex should avoid at all costs. Zombies are hostile entities which attack Alex when nearby. Each strike from a zombie decreases Alex's health. To trigger zombie avoidance we falsify `(alive)`:

```
(:event zombie_damage
  :parameters ( ?start - cell ?zombie_cell - cell )
  :precondition (and (alive) (alex_at ?start)
    (= (entity_type ?zombie_cell) 100)
    (>= (- (z_cell ?zombie_cell) (z_cell ?start)) -1)
    (<= (z_cell ?zombie_cell) (z_cell ?start))
    (<= (- (x_cell ?start) (x_cell ?zombie_cell)) 1)
    (<= (- (x_cell ?zombie_cell) (x_cell ?start)) 1) )
  :effect (and (not (alive)) ))
```

This event triggers when a zombie appears in the plan zone and when Alex is in striking distance from the zombie.

These goal conditions are satisfied in the initial state (i.e. set to true by default) and falsified by the event, forcing Alex to collect the resource or avoid the zombie before continuing to the target cell. In the absence of resources or zombies, the resource-related goal conditions remain satisfied throughout.

Overall, the PDDL+ domain allows finding plans which prioritize immediate acting in the local space, while assuming

the subsequent sections of the search space are restricted to movement only. As mentioned before, the triggered events also drive the replanning strategy, specifying when to replan, to efficiently catch changes in the local plan zone without redundant effort. However, problems with large numbers of resources and zombies, even in a restricted subproblem, can quickly become intractable. So we turn to goal reasoning.

4 Goal Reasoning

Goal reasoning enables an actor to deliberate online about its goals. Roberts et al. (Roberts et al. 2016b) formalize goal reasoning as progression of goals according to a goal lifecycle. A goal is *formulated* and *selected* before planning; these two stages can filter an open world by only formulating or selecting relevant goals. Let a plan $\pi = \langle a_i..a_n \rangle$ be a sequence of actions $a_i..a_n$. A goal is then *expanded* into one or more plans Π and a single plan $\pi \in \Pi$ is *committed* for execution; goal reasoning naturally extends to approaches that find more than a single plan. When π is sent for execution the goal is *dispatched*; the simplest system sequentially executes each action $a_i \in \pi$. A goal impacted during execution is *evaluated* for the best course of action and *resolved* appropriately. Example *resolve* strategies include repairing π, replanning to create a new π, regoalling, etc. We focus on replanning in this work.

Goal Reasoning Example Our study focuses on three goals from the introductory paragraph: get to the end of the hallway (rooms excluded for the moment), avoid zombies, and collect resources. These goals are formulated in a combined 'Complete Hallway' goal that is selected once the hallway is constructed. The PlanManager expands a plan for this goal by converting the goal to PDDL and calling an automated planner; the zombie and collect goals are translated to the correct PDDL model, as described earlier. The planner automatically commits to the first plan it finds.

The 'Complete Hallway' goal is dispatched by preparing the full plan for execution. But because the Controller only executes one command at a time, the goal reasoner creates subgoals of 'Complete Hallway' so each action of the plan can be tracked independently. For example, a move-north action is converted to a subgoal to be at the location north of the player. The goal reasoning system automatically selects the subgoal, skips planning since it is not needed, and dispatches the single action to the controller. The controller completes the action and reports back to the subgoal. When the subgoal completes, 'Complete Hallway' is notified and can check to see if any of its goals are impacted. Changes during execution may result in the need to replan. Further, Alex will clearly need to replan when all actions of the current plan are completed. Alex might also replan when zombies, wood, or diamonds appear. In all cases, the goal reasoner determines when replanning is needed.

Goal Reasoning with Planning So far, we have explained how opportunistic goals in PDDL+ allow a planner to synthesize a plan under partial observability and how a goal reasoning system adjusts goals based on Alex's context. Consider again Figure 2. The resulting plan for either plot moves Alex north two blocks to collect the diamond, east and north around the zombie, and west toward the intermediate target. But there is a problem with the resources in row z-9 that lie outside the plan zone: the planner is blind to them because they are not reported in the PDDL model. It could replan at every step but this may waste computational effort or needlessly slow down Alex. There are several more reasonable choices for the goal reasoner that we explore: (1) It could use a Small plan zone to identify opportunities close to Alex and force replanning at fixed intervals. (2) It could use a Large plan zone, which could result in increased planning cost when there are no opportunities available. (3) It could use a Dynamic plan zone by starting with a Small plan zone when there are no nearby opportunities but increase the zone size when opportunities surface in the observation window. We next explore the tradeoffs of these three choices.

5 Evaluation

In this section, we assess three research hypotheses concerning the use of an expressive planning model and a goal reasoning system: (1) The use of a Dynamic planning zone by the goal reasoner will significantly reduce planning effort *when compared with always using a large planning zone*; (2) the use of PDDL+ to collect resources via events will significantly reduce planning effort over the numeric PDDL model for any zone size; and (3) the use of the Dynamic plan zone or PDDL+ will not significantly reduce resource collection.

Similar to previous ACTORSIM experiments for Minecraft, we randomly generate hallways consisting of 10 sections. Each section is 20 blocks wide and 15 blocks long, and contains one randomly placed element: a wood drop, a diamond drop, or a caged zombie.

The goal reasoning system selects for the planner a planning zone, defined as the number of cells away from the player in each of the following directions: (front, back, left, right). The *Small plan zone* of (5,1,3,3) has the benefit of a fast planning time at the expense of frequent replanning and possibly missing opportunities to collect resources. The *Large plan zone* of (8,3,8,8) reduces the number of replanning episodes at the cost of increasing the overall computational effort and planning time; however, it is less likely to miss opportunities. The *Dynamic plan zone* defaults to the Small zone until a resource is within 8 blocks, at which point it enlarges the plan zone in the direction needed to reach that drop. In effect, the goal reasoner is varying the replanning strategy as imposed by the plan zone. We leave more sophisticated zone-selection strategies for future work.

For this study, we limit the models and planning systems to focus on our research questions. For planning models, we focus on the numeric PDDL2.1 model that uses "visited" and "unsafe" predicates as well as the PDDL+ model that uses opportunistic goals. For a planner, we used the POPF planner extended for reasoning with PDDL+ processes and events (Coles and Coles 2014). This choice was motivated by the capabilities of the planner which considers all features of our models. Using the same planner for all models guarantees fair comparison between the different variants.

Table 1 summarizes 180 runs: 30 runs for each of the six zone/model combinations. The *S(N)*, *L(N)*, and *D(N)*

rows show the numeric PDDL2.1 runs, whereas rows denoted *S(+)*, *L(+)*, and *D(+)* summarize the PDDL+ runs for *Small*, *Large*, and *Dynamic* planning areas, respectively. The columns summarize the mean and standard deviation for each of the response variables: cumulative CPU time, number of nodes evaluated, and memory as well as the ratio of items collected versus those available in the hallway. A two-factor, paired-sample ANOVA for zone and course across each metric reveals that zone size significantly effects the results of each metric ($p \approx 0$ for all five metrics), justifying pairwise comparisons between the zone results. We now make a number of pairwise comparisons to examine how strongly the evidence supports our three research hypotheses. We report p-values for the Tukey Honest Significant Difference test ($\alpha = 0.05$) but we verified these results using Scheffe's method, which is more robust to potential effects of heteroscedasticity.

A dynamic planning zone reduces planning effort. The evidence suggests that the Dynamic plan zone can significantly reduce planning time while not using more memory. It is evident that in either the numeric or PDDL+ models there is 70-90 second difference in the CPU time between the Small and Large planning zones. As expected, the Dynamic plan zone is significantly different from the Large planning zone ($p < 0.0001$) and significantly similar to the Small planning zone ($p \approx 0.84$ for the numeric PDDL and $p \approx 0.99$ for PDDL+). In terms of nodes evaluated and memory, the Dynamic uses more nodes but less (or similar) memory than Small zone. However, the Dynamic zone never uses significantly more nodes or memory than the Large zone.

A PDDL+ model is less computationally intensive than a numeric PDDL2.1 model. The evidence is mixed. A PDDL+ model uses a statistically similar number of nodes as a PDDL2.1 model. It uses significantly more CPU time for the large zone but is similar for the other two zones. Finally, it uses significantly less memory for the Small zones and similar time for the Large and Dynamic zones. This is seen by comparing the Time, Nodes, or Memory usage for S(N) with S(+), L(N) with L(+), or D(N) with D(+).

The dynamic plan zone (as selected by goal reasoning) or PDDL+ do not reduce resource collection The evidence suggests that neither goal reasoning nor PDDL+ significantly reduces Alex's ability to collect resources. This question is answered by examining the ratios of wood and diamonds collected. At first glance, it may appear that these ratios are very different because the means vary so much. Examining the pairwise comparisons reveals a significant difference for the Small zone. Because there exists no significant difference between collection results for most of the zone/model combinations, it is clear that adding goal reasoning or PDDL+ to the system allowed it to maintain resource collection at the Large zone level.

Discussion The results show that goal reasoning can significantly reduce the planning time by a factor of 15-20. The results did not show that PDDL+ provided a further reduction in planning effort, but rather both models exhibited similar performance. Further, neither approach fared worse at resource collection than the Large zone, which was a baseline upper bound. However, the PDDL+ provides a much easier and more encompassing model for resource collection, which reduces the computational effort exerted by the goal reasoning system.

	Time		**Nodes**		**Memory**		**Wood**		**Diamond**	
	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
S(N)	2.0	0.1	304	50	134	2	0.47	0.47	0.27	0.32
S(+)	3.4	0.5	312	44	110	9	0.25	0.28	0.28	0.29
L(N)	74.2	12.3	12439	21870	120	40	0.96	1.01	0.94	0.87
L(+)	93.4	13.0	7508	16175	104	31	0.65	0.33	0.64	0.38
D(N)	4.3	2.2	1804	3339	124	5	0.66	0.64	0.75	0.78
D(+)	3.5	0.8	446	204	124	9	0.68	0.77	0.64	0.74

Table 1: Summary of statistics for the hallway study.

6 Related Work

Perhaps the closest planning work related to our use of opportunistic/soft goals in an open world is the Open World Quantified Goals of Talamadupala et al. (2010), where quantified goals allow the planner to expand on goals that may appear during execution. An earlier work by Etzioni et al. (1997) used Local Closed-World statements to integrate an open world with a closed-world planner. In contrast to employing quantification or locality, a PDDL+ opportunistic goal is always in the problem's goal conjunction and a conditional event enables the goal.

A variety of research systems have been built to study Minecraft. The first, called BurlapCraft, by Abel et al. (2015) integrated the BURLAP machine learning platform[1] and examined how to use knowledge to select actions. More recently, Microsoft has released an open-source platform called Malmo[2] that provides extensive support for multiple programming languages, the ability to set up experiments, as well as support for reinforcement learning. ACTORSIM is primarily distinguished from these other systems in its use of goal reasoning, existing experiments using deep learning (Roberts et al. 2016c), and integration with other simulators including robotics platforms. Each system has merits depending on the task at hand, but none of these systems supported observations when we first examined them.

Recently, AI research in game-playing concentrated on exploiting Deep Learning techniques, particularly deep Q-networks (DQN) which beat expert human players on a range of ATARI games (Mnih et al. 2015). ATARI games have also been tacked using classical planning (Lipovetzky, Ramirez, and Geffner 2015). More advanced games such as Starcraft and Minecraft require a non-trivial transition to a much more complex environment. Recent work (e.g., (Bonanno et al. 2016; Tessler et al. 2016; Abel et al. 2015; Usunier et al. 2016)) shows promise in this area, though these studies examine simplified models or restricted subtasks. Massive training data sets, sparse rewards, vast state-action spaces, and difficult-to-define evaluation functions significantly limit the scaling potential and efficiency of DQNs.

[1] `http://burlap.cs.brown.edu/`

[2] `https://github.com/Microsoft/malmo`

In fact, Starcraft and Minecraft games are both prime examples of domains from automated planning (e.g. the Settlers domain (Long and Fox 2003)) where long-term, high-level goals need to be achieved.

Prior work has explored the use of PDDL to represent Minecraft (Branavan et al. 2012). This paper presents the first PDDL+ domains of Minecraft. Similar domains were previously defined in either purely propositional PDDL (IPC Committee 1998) or non-temporal PDDL2.1 with numeric fluents (Fox and Long 2003). PDDL+ is designed to compactly represent hybrid systems with mixed discrete/continuous behavior through processes and events. In recent years, PDDL+ planning has become a rising trend in AI and multiple approaches have been proposed to deal with PDDL+ domains (Shin and Davis 2005; Cashmore et al. 2016; Coles and Coles 2014; Della Penna et al. 2009; Piotrowski et al. 2016).

In the past PDDL+ was combined with Hierarchical Task Networks (HTNs) for goal-driven autonomy, implemented in the $\text{SHOP2}_{\text{PDDL+}}$ planner (Klenk et al. 2013; Molineaux et al. 2010). In contrast to hierarchical approaches, our work focuses on the first PDDL representation. We plan to incorporate more recent developments in hierarchical planning (e.g., (Ghallab, Nau, and Traverso 2016; Alford et al. 2016; Dvorak et al. 2014; Shivashankar et al. 2012)).

Soft goals are hard to express in PDDL. PDDL2.1 (Fox and Long 2003) introduced the notion of plan metrics enabling specification of soft goals and enhancing the quality of solutions. They could be useful in the context of assessing alternative plans. However, few planners actually employ plan metrics and they are limited to one specified optimizable function. For example, suppose each action modified the resources used or acquired, the character's change in health or food, and the number of steps to completion. Then, a planner could focus on producing diverse plans that span the trade offs. But plan metrics can only minimize or maximize the quality function which can exert unnecessary computational effort by collecting resources well beyond Alex's needs (and overloading Alex's inventory in the process).

On the other hand, PDDL3.0 (Gerevini and Long 2005) also incorporated plan metrics and combined it with the concept of *planning with preferences* which enable better reasoning with multiple objects and a more accurate method for specifying the desired plan characteristics. Planning with preferences is largely based on Linear Temporal Logic (LTL) (Pnueli 1977). PDDL3.0 provides a strong feature base for representing the Minecraft scenario. Numerical variables, plan metrics, and preferences are well-suited to build a concise and expressive Minecraft planning domain including maximizing collected resources when available. However, defining preferences in this manner can inflate the size of the domain and state variables sets, and significantly increase planning time.

These advanced features – plan metrics from PDDL2.1 and preferences from PDDL3.0 – could provide interesting and successful variants extending the modeling in this paper. The numeric PDDL model in this paper provides a solid foundation for assessing these features in Minecraft.

7 Closing Remarks

We have presented the first formulation of the Minecraft domain in PDDL+ extended with resource collection and zombie avoidance tasks. We also showed how events, a native feature of PDDL+, can be used to model opportunistic goals. Finally, we presented a goal reasoning approach to reducing the computational effort for finding a viable plan by selecting the zone size and partitioning the original problem into planning and observation zones. To the best of our knowledge this is the most extended model of Minecraft in AI, though it is only a preliminary stage of a larger project.

Future work will focus on modeling and solving the motivating example. It will include a revision of the plan zone and visibility, not only adjusting the size of the zone but also the shape which prunes areas of no interest while including areas with desirable resources and entities. We will also aim to expand the PDDL+ model to account for the continuous behavior in Minecraft, such as health management. Health decreases when attacked by hostile mobs, but regenerates slowly when resting. We will expand the list of happenings, actions, and goals to reason with and manage the agent's health. PDDL+ events are a natural fit for this modeling.

Our approach grew from a desire to accurately model health, food, resources, and entities in Minecraft, for which PDDL+ events and processes are best suited. It also provides a foundation for incorporating mixed discrete/continuous dynamics should this be desired in future domains. External happenings often modify the state of the environment without interference from the agent; we believe there is a great deal to learn from extending the model in this way.

A long-term focus of future work will enable the goal reasoning for long-duration autonomy. While it may be possible to manage short-term goals such as those in the motivating quest, we are interested in leveraging goal reasoning and automated planning for an agent that *perpetually learns* (Roberts et al. 2016a). Such an agent will need to manage its own learning agenda to master new tasks, revise previously learned tasks, and halt learning for already mastered tasks.

A final area of future work is in extending the model to incorporate Deep Learning. Minecraft was previously attempted using Deep Reinforcement Learning. We plan to compare the two approaches to identify their relative strengths. Deep Learning could manage short-term reactive behavior (i.e. self-defense) while planning with goal reasoning could manage long-term deliberative behavior.

Authors are sometimes circumspect about the actual development, design choices, and computational requirements of using a particular 'brand' of planning. In contrast, we have set out in this work to identify exactly our representational and design choices to the greatest extent possible. While it is unreasonable to expect a laymen to understand the nuances of automated planning, PDDL, or of effective domain modeling, we can at the least point to the active goal of the goal reasoning system, examine the domain or problem files produced, and examine the search trace of the planner to explain its decision. The transparency of the approach we have outlined is especially noteworthy in an era where AI systems are being called to arrive at sensible and correct output while also making transparent their decision-making process.

Minecraft presents worthwhile challenges for studying planning with respect to a simulated environment. As we moved from a propositional representation to a more detailed PDDL+ representation, the planners available to us diminished considerably – from close to a hundred to less than a few. Were we to move in the direction of PDDL3.0 features, a similar problem would occur. Similarly, few planners support advanced PDDL2.1 features such as metrics beyond action cost. Our findings underscore the need for the continued advance of planning systems – perhaps through the competitions – to better support the range of PDDL features used by applications.

Acknowledgments

We thank the anonymous reviewers whose comments improved this paper. MR, DA, and PB thank NRL for supporting this research.

References

Abel, D.; Hershkowitz, D. E.; Barth-Maron, G.; Brawner, S.; O'Farrell, K.; MacGlashan, J.; and Tellex, S. 2015. Goal-based action priors. In *ICAPS*, 306–314.

Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. W. 2016. Hierarchical planning: Relating task and goal decomposition with task sharing. In *IJCAI*. AAAI Press.

Bonanno, D.; Roberts, M.; Smith, L.; and Aha, D. W. 2016. Selecting Subgoals using Deep Learning in Minecraft: A Preliminary Report. In *IJCAI Workshop on Deep Learning for Artificial Intelligence*.

Branavan, S.; Kushman, N.; Lei, T.; and Barzilay, R. 2012. Learning high-level planning from text. In *Proc. of the Annual Meeting of the Assoc. for Comput. Linguistics*, 126–135.

Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *ICAPS*, 583–591.

Coles, A. J., and Coles, A. I. 2014. PDDL+ Planning with Events and Linear Processes. In *ICAPS*, 74–82.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *ICAPS*, 106–113. AAAI.

Dvorak, F.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. A flexible ANML actor and planner in robotics. In *Planning and Robotics (PlanRob) Workshop (ICAPS), Portsmouth, United States*.

Etzioni, O.; Golden, K.; and Weld, D. S. 1997. Sound and efficient closed-world reasoning for planning. *AIJ* 89(1):113 – 148.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR* 20:61–124.

Fox, M., and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *JAIR* 27:235–297.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. In *The Language of the 5th IPC. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, volume 75.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge Univ. Press.

IPC Committee. 1998. PDDL : the planning domain defnition language. Technical report, Yale Center for Computational Vision and Control. The committee consisted of: M. Ghallab and A. Howe and C. Knoblock and D. McDermott and A. Ram and M. Veloso and D. Weld and D. Wilkins.

Klenk, M.; Molineaux, M.; and Aha, D. 2013. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence* 29(2):187–206.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical planning with simulators: results on the atari video games. In *IJCAI*, 1610–1616.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *JAIR* 20:1–59.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; and Ostrovski, G. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Goal-driven autonomy in a navy strategy simulation. Technical report, DTIC Document.

Piotrowski, W.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. Heuristic Planning for PDDL+ Domains. In *IJCAI*, 3213–3219.

Pnueli, A. 1977. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, 46–57. IEEE.

Roberts, M.; Hiatt, L. M.; Coman, A.; Choi, D.; Johnson, B.; and Aha, D. 2016a. Actorsim, a toolkit for studying cross-disciplinary challenges in autonomy. In *Fall Symposium on Cross-Disciplinary Challenges in Autonomy*.

Roberts, M.; Shivashankar, V.; Alford, R.; Leece, M.; Gupta, S.; and Aha, D. 2016b. Goal reasoning, planning, and acting with ActorSim, the actor simulator. In *Proceedings of ACS*.

Roberts, M.; Alford, R.; Shivashankar, V.; Leece, M.; Gupta, S.; and Aha, D. W. 2016c. ACTORSIM: A toolkit for studying goal reasoning, planning, and acting. In *Working notes of the ICAPS PlanRob Workshop*.

Shin, J.-A., and Davis, E. 2005. Processes and Continuous Change in a SAT-based Planner. *AIJ* 166(1):194–253.

Shivashankar, V.; Kuter, U.; Nau, D.; and Alford, R. 2012. A hierarchical goal-based formalism and algorithm for single-agent planning. In *AAMAS*, volume 2, 981–988. Int. Found. for AAMAS.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for human-robot teaming in open worlds. *ACM TIST* 1(2):14:1–14:24.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2016. A Deep Hierarchical Approach to Lifelong Learning in Minecraft. *arXiv preprint arXiv:1604.07255*.

Usunier, N.; Synnaeve, G.; Lin, Z.; and Chintala, S. 2016. Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. *arXiv preprint arXiv:1609.02993*.