Automated Planning is a profound area of emphasis in Artificial Intelligence, providing a distinct and purpose-focused layer that provides a logical representation of a problem-solving environment, where an agent would be tasked with transitioning that environment from an initial state to an intended goal state. PDDL, an industry-wide definition language for problem domain defining and planning, allows developers to categorise and distinguish a domain problem’s different objects, state, properties, and relationships, governing how problems must be interpreted by the agent as per the corresponding domain. In the grand scheme of an AI system, this layer provides a critical foundation which defines how a solver, or some class of algorithms, may go about producing plans for how the agent may transition their environment towards its intended goal state. This relationship between dynamic problem/domain representation and computationally robust calculations aimed at achieving the desired outcomes as efficiently as possible, captures the essence of dynamic problem-solving by an AI system. These plans can be adopted in a variety of different ways, from being collected as training data for a model, to serving as plausible plans to be considered by any higher-order decision-maker. This report will focus on the importance of clarity and refinement in the scope of automated domain planning, providing different examples of how similar problem configurations can be navigated slightly differently by employing different solvers to parse the PDDL application. Furthermore, the implications of domain complexity refinement, logical clarity and redundancies will be addressed to explain how these can be improved for clarity, applicability, and accuracy regarding the problem space.

This report will focus on two different domains, a Minecraft-inspired problem space and a Wumpus world problem space (both grid-like environments), leveraging existing unrefined problems to establish more robust definitions and environmental representations. The initial Minecraft problem consisted of an intended goal state where the agent possesses a grass block in their inventory, as well as being at location 1-1. The initial state was cluttered with a long list of action literals, and a single log object. When considered with the multiple `craftplank` action literals, it was safe to assume that an additional intention was for the agent to craft the log into planks, bringing it with them to the goal location, implying that it should perhaps be part of their inventory. The domain also consisted of redundant predicates like `isgrass` and `hypothetical`, signaling the potential for refining the complexity of the domain by consolidating the two properties that a log may be identified with, and ignoring explicit clarification for considerations that do not serve the transition of the log into planks. This would streamline the state space, since a log would now be represented in conjunction with `islog` or `isplanks`, while still maintaining its `moveable` predicate, and ignoring irrelevant properties to the crafting process. Furthermore, the action-specific predicates could be removed to further enforce the separation of action definitions, object state and properties, and their relationships. Additionally, the action definitions, each used in conjunction with a corresponding predicate, would require refinement to better aligned with automated planning and PDDL structuring principles. After the domain had been more closely aligned with accurate environment representation, the problem could be addressed in a simplified and refined domain, leading to less unintended consequences in the results moving forward. To capture the dynamic applicability of the intended problem, the established rules needed to be considered in greater context – a Minecraft player agent has multiple methods of moving across its environment. To simplify the problem and allow for a wider variety of location transition methods (in theory), adjacencies were omitted to focus purely on the essential logic required to obtain the desired resources and move them all back to an initial location (to be assumed as inventory cells). This was supported by the concept of `craftplanks` being an arbitrary action resulting in the transition of logs to planks, which would otherwise require additional actions and resources in the problem space. With the assumption of these abstractions being covered by another part of the AI system, the essential considerations for operating in this domain implied:

* The agent can change its own location
* The agent has access to an inventory
* The inventory is always located at the agent
* Log blocks cannot change their own location
* Grass blocks cannot change their own location
* Log Blocks and Grass Blocks can be inserted in the agent inventory
* The agent can recall different cells in its inventory
* The agent has access to a corresponding action to craft planks

At its most basic, the problem was concerned with a goal-state transition that depended on one specific crafting task, the transition of objects that otherwise cannot action toward their intended goal location (the agent’s inventory, located at the player), and moving to a goal location. While these things may seem implicit, the definition of the problem domain must be reflective of all these basic considerations as a matter of preconditions and transitional effects. The solution was therefore approached from the angle of refinement and reducing unintended consequences from adjusting state in an irrelevant capacity, regardless of whether the same outcomes were achieved. Doing so also made the problem domain easier to manage.

The additional Wumpus world problem required a comprehensive analysis to define the intended transitional conditions the agent must adhere to. The problem environment, though similar in structure, contained additional objects that would be a hazardous encounter for the agent. The logic of implying the agent’s survivability according to observations made when adjacent to different hazards needed to be well-defined. Considering the queues for each, the agent should only seek about determining the danger when directly adjacent – when the agent encounters the perceptual queue of its adjacency. The location of the hazards and the gold required obfuscation from the agent, with the agent only being able to infer their location based on the encountered queues. The agent would also need to be defined also as the only entity that may invoke the `move` action, preventing the solver from allowing the gold to transition itself toward the intended goal state. All this considered, the goals would need to be clarified, allowing the solver to transition elements toward their goal states without unintended consequences. The agent must avoid the hazards, else it will die. The absence of the agent’s death would therefore need to be implied as a goal state. The most immediate solution might seem like detection actions that trigger at each square to infer the location of these objects based on the same steps run at each square, but considering the application, this is grossly inefficient. The agent would therefore need to be able to remember squares that it’s visited, and detect `breeze`, `stench`, and `glitter` adjacencies every time it moves to a new location. Conditional effects would need to be defined in the solution’s `move` action to allow the agent to perceive potential hazards/goals, determining whether the proposed path meets these critical requirement outcomes. This approach was supported by the principle of dynamic environment representation, where an agent must navigate about a problem without explicit knowledge of all critical states, updating its knowledgebase to develop its interpretation of the scenario across state transitions. Finally, the lethal dynamic between the agent and the Wumpus could be resolved so long as the agent possesses the arrow (default) and can shoot the Wumpus by inferring the correct location. This outcome should only be sought by the agent when there is no other way except over the Wumpus to avoid all pits. This problem required direct consideration of bidirectional adjacency, as the agent is permitted a strict grid-like movement to adjacent squares, which can simultaneously serve as cues for hazards and goals. The solution also included additional pits that could be comment-toggled to force a confrontation with the Wumpus, testing the domain’s logic for plan outcomes to omit Wumpus-killing if it was not a critical obstacle.

The results obtained from comparing multiple scenarios with alternating solvers provided insight which clarified the solution applicability in context of a wider AI system. Both domains and their variant scenarios were exposed to both a BFWS FF-parser solver, and a LAMA-first satisficing planner, both which produced slightly different outcomes with profound scaling implications. Interpreting these results required a comprehensive analysis of the mechanics and implications of each:

**BFWS FF-parser solver**

* Breadth-first width search algorithm in a solver tailored specifically to parsing PDDL files
* Systematically explores the search space, considering all nodes at the present depth before proceeding to nodes at the next layer
* Using a predefined heuristic, the branching factor of solving for intended outcomes is reduced by prioritising states that are most promising for its search
* The Fast-Forward heuristic is employed for estimating goal distance, guiding the algorithm’s path prioritisation method

This solver guarantees an optimal solution by comprehensively exhausting the options of each depth before proceeding. This method unfortunately requires significant memory to contain the complexity of the problem space in context of the immediate solving depth, proposing a significant drawback in problem domains of vast complexity and increased search depth. In smaller environments with more manageable memory constraints, this serves as an ideal solution due to its exhaustive nature, despite its scaling implications. In contrast, the same scenarios were exposed to a LAMA-first satisficing planner, presenting its own capacity to solve in automated planning scenarios:

**LAMA-first satisficing planner**

* Employing Landmarks, Action-Graphs, and Multi-Heuristic A\*, LAMA-first is a satisficing planner – it doesn’t guarantee the most optimal result
* Heuristic-based search, employing multiple cost-based heuristics to guide the search more efficiently
* A\* search algorithm is employed to balance exploration capacity with the potential to exploit lower-cost paths
* A Landmark approach, establishing sub-goals/intermediate states which stagger the transition to the final goal state, structuring the search process

LAMA-first doesn’t prioritise optimality, rather focusing on finding a favourable solution quickly. This bodes well for solving in large, complex domains when compared against BFWS. By addressing a variety of heuristics and guidance mechanisms, this algorithm is much better suited to satisficing in a domain where it would be impractical to address every choice at each search depth. This solver is therefore better suited to navigating time-sensitive situations in complex planning problems. The strengths and weaknesses of each solver were reflected in the computational requirements for each scenario. Where the BFWS solver consistently found the most optimal path as quickly as possible, the LAMA-first solver took slightly longer and required increased plan cost to achieve the desired outcomes. This suggested that for the scale of complexity in these grid-like problem spaces, BFWS was ideal, being best suited for solution optimality in these scenarios, where no strict memory constraints existed. If the complexity of these domains were to grow, requiring more in-depth considerations for how each goal can be fulfilled, the balance between solution applicability and execution time would quickly become imbalanced when using the BFWS solver. If the agent were required to make more complex decisions in time-constrained scenarios, the solver would likely benefit from the more robust guidance mechanisms of the LAMA-first planner, providing better structure to complex searches.

The knowledgebase representation between the Minecraft and Wumpus world scenarios presented distinct problem planning challenges due to the types of information the agent would logically be privileged to at the beginning of the problem. In each scenario, the agent’s interpretation of the scenario required slightly different clarification to accurately reflect the dynamics of the essential problem considerations.