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, a log crafted into planks, as well as returning to an intended location. This problem space emphasised the relationship between the agent’s actionable inventory mechanisms and the acquisition of blocks in the environment for inventory crafting/storage. This implied a layer of abstraction for sequencing more complex tasks like `move` and `craftplanks`. The inventory would need to be defined in conjunction with a `recall` mechanism so the agent could consider a specific item in their inventory on which to invoke `equip`, simultaneously toggling the `handsfree` predicate. The original domain alluded to the idea of the agent needing to be handsfree before picking or crafting, requiring an `equipped` state to be manipulated for each call of its corresponding action. The original domain consisted of redundant predicates like `isgrass` and `hypothetical`, signaling the potential for refining the complexity of the domain. 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, or blocks to the inventory, would make the specific domain less complex and more manageable. A log could now be represented with `islog` or `isplanks` predicates, ignoring irrelevant properties to the crafting process – the agent doesn’t need reclarification for blocks that are being implied to have a single state in that regard. Furthermore, the action-specific predicates could be removed to further enforce the separation of action definitions, object state and properties, and their relationships. After the domain had been more closely aligned with accurate environment representation, the problem could be addressed with less unintended consequences moving forward. To reinforce the applicability of the intended problem definition, the established rules needed to be tested in logical scenarios, where implicit relationships must be adhered to:

* Only the agent has access to the inventory and actions
* The inventory is always located at the agent
* Log blocks and grass blocks are both moveable but cannot change their own location
* The agent can `pick` a block from the environment and add it to their inventory
* The agent must be handsfree before invoking the `pick` action
* The agent can `equip` a block from its inventory
* The agent must `recall` a block in its inventory before invoking the `equip` action
* The agent must `equip` a log block before invoking the `craftplanks` action
* Using `equip` will toggle `!handsfree`

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 most difficult part of the implementation would be logically implying the intended transition of block states in conjunction with the mechanisms the agent could use in a specific order to invoke actions. By abstracting the process of determining movement and crafting sequences, this problem could focus on clearly articulating the relationships between agent inventory mechanisms and the blocks they acted upon. It’s important to understand that the solution can be made more applicable by providing an `unequip` action so that the agent doesn’t necessarily need to collect all necessary blocks before crafting, as was consistent across the different plan outputs. For the purposes of transitioning to the outlined goal states without further immediate concern, this was not strictly necessary.

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 potential hazardous encounters 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 dynamic nature of collecting knowledge of cue locations and inferring the potential location of hazards, the agent should only be initially privileged to cell adjacencies. The locations were logically obfuscated from the agent, with the agent only being able to infer potential locations from cues, avoiding these due to the implied critical contradiction that encountering hazards would pose to intended goal state fulfilment. The most immediate solution might be to enforce detection actions that are called on all squares to dynamically detect the cues and interpret them accordingly, but this would be very inefficient . The agent would therefore need to be able to remember squares that it’s visited, and interpret `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. This would allow the agent to perceive cues as they are encountered, before testing outcomes associated with immediate adjacencies. This approach was supported by the principle of dynamic environment representation, where an agent must navigate a problem environment 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 possessed 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 to fulfil the intended goals without passing through where the wumbus may be located. 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 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 typically guarantees an optimal solution by exhausting the options at each depth before proceeding. This method unfortunately requires significant memory to contain the complexity of the problem space in context of subsequent search depths, proposing a significant drawback in problem domains of vast complexity and increased 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 somewhat reflected in the computational requirements for each scenario. Where the BFWS solver consistently found the most optimal path quicker, the LAMA-first solver took slightly longer, and in one scenario required increased plan cost to achieve the desired outcomes. This suggested that for the scale of complexity in these small 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 may 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 and precision to complex searches.

The knowledgebase representation between the Minecraft and Wumpus world scenarios presented distinct problem planning challenges due to the unique focus and knowledge privileges. The Minecraft problem was largely concerned with the interpretation of inventory mechanisms (actions) and their relationships with objects that could be acquired in the problem space. The problem presented a layer of abstraction for the

The Wumpus world problem was slightly different, requiring thorough clarification of movement consequences based on the presence or absence of certain properties as each cell was explored. The considerations of executing `move` would therefore need to be quite comprehensive. The Minecraft example assumed the use of abstraction to delegate the explicit navigational sequence to another layer, concerning itself with interpreting a transitional relocation requirement, though not requiring a robust interpretation of the process with incremental structure. The Wumpus world required the inverse since the locations of hazards and goals could only be inferred by their adjacency cues, adhering to much stricter navigational constraints. This required a conditional approach to considering only the immediate location and iterating over potential adjacencies based on the absence or presence of these cues. Additionally, a `visited` predicate was employed to allow the knowledgebase to be updated dynamically, implying the interpretation of a problem space filled with hazards and safe spots of unknown variation. It’s important to note the dynamics of this specific domain, the developer can impart the presence of each hazard without making the locations explicitly available. It can however be defined when there were immediate adjacencies upon the arrival at cue cells, clarified and interpreted through conditional movement observations as the agent considers the outcome of the proposed cell transition – **where should the agent not go based only upon properties of where it immediately is, and is permitted to consider going to?** The implications of avoiding hazards of unknown locations required conditional consideration based only on what was immediately clarified on a cell-by-cell basis. The emphasis on survival and exploration was captured with thorough improvement of the knowledgebase, employing more robustly defined actions and comprehensive predicates that would infer the connections to be made based on immediate cues and known state. This enforced a stricter need-to-know privilege of potential hazard/goal locations where the agent must consider the implications of the immediately observable state at each depth of the solution search, enforcing a more incremental and comprehensive method of dynamically solving relocation sequences.