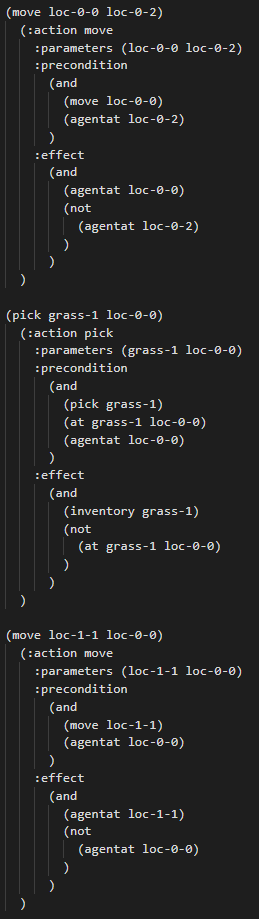
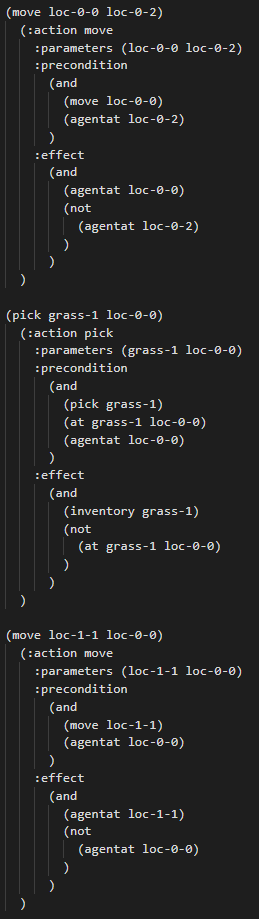
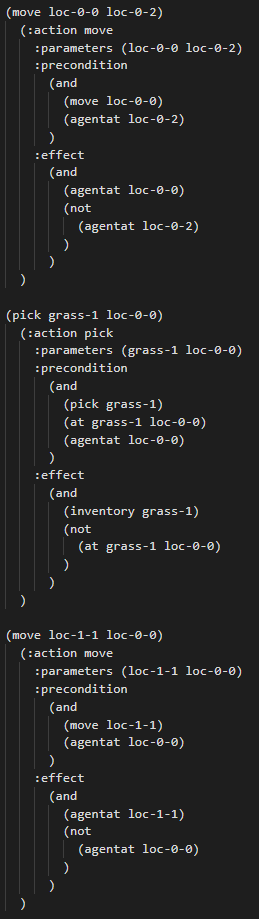
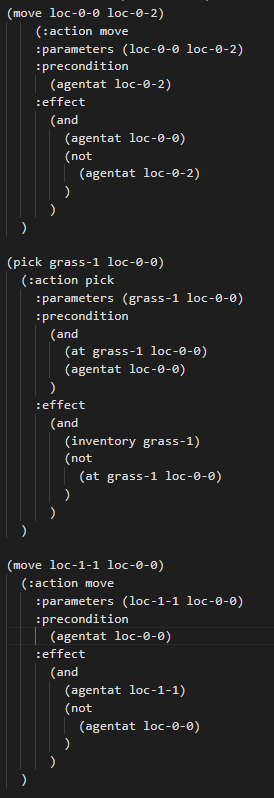
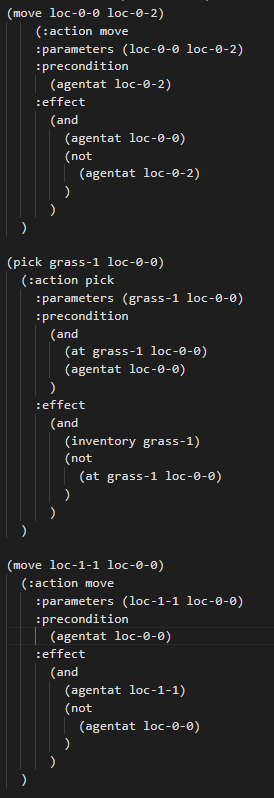
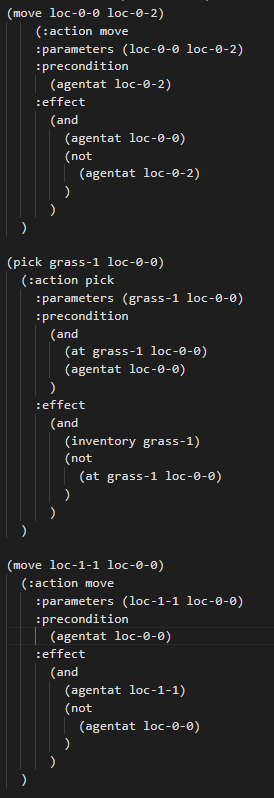
Automated planning is a core area of focus regarding AI technology. It is an integrated automated solution that determines how specific actions should be sequenced when an AI entity/system faces certain kinds of environmental problems, allowing it to act in a way that transitions its environment from an initial state to an intended goal state. By leveraging computational intelligence techniques like evolutionary algorithms, neural networks, and fuzzy logic, automated planning can be solved according to different problem scenarios with varying degrees of complexity. The plan outcomes are typically produced by a solver, two kinds of which will be analysed and compared to determine the implications of choosing one over the other. This involves a comprehensive understanding of how states and state space, actions, goals, and plans are interpreted, based on their logical representation and implications within the system’s knowledge base. It’s paramount that the way a dynamic environment is represented (in the case of this report, generally through capturing relevant variables and their interactions) accounts for any logical inconsistencies that may cause unintended outcomes. Of course, to reason about planning within a dynamic environment requires a careful balance between resource and time usage/acquisition, where planning algorithms are often employed to make this process as efficient as possible. For the focus of this report, which is on automated planning with PDDL, the optimisations and improvements will be mainly representational, using Boolean logic to clarify and refine goals and actions, ensuring the knowledge base represents the problem domain as accurately as possible. The implications of these adjustments will then be explored while unpacking the computational strengths and weaknesses of the solver types. By analysing, improving, discussing, and justifying the outcomes of two PDDL example problems and domains (a Minecraft-inspired problem, and the classic Wumpus problem) with different solvers, this report will focus on the implications of clarifying the knowledge base of an automated planning system for reflective accuracy, and the implications of choosing a specific planner in context of problem complexity.

The initial planning problem used for this report was a Minecraft-inspired PDDL implementation, where an agent interacts with different objects in a grid-like environment. To understand the permissible actions and requirements of how this plan should be reasoned about, the first step required developing a comprehensive domain definition. This shows the available options and possible resources for an arbitrary problem scenario in this domain, but the definition logic must be logically reflective of all essential requirements for the intended planning outcomes. The original problem domain uses a STRIPS model to formalize its planning structure, paired with types to capture entity and object representations. Predicates are included as basic mechanisms that imply the absence or existence of conditions and properties, in the of predicate statements. Action definitions are also included to define the types of operations that can be carried out to change the state of the environment. The domain contains logical inconsistencies that skew the agent’s implied reasoning, going against some fundamental principles of automated planning. The presence of action literals within the definition of predicates goes against the distinction that is crucial to maintain between state and actions when planning for a problem scenario. This results in action applicability being defined through action-specific predicates, as opposed to the immediate state of objects and their relationships. The latter is the intended frame of reference at any given point in this problem space. These ****inconsistencies collectively affect the efficiency of resource usage and outcomes by the solving solution. Since these action-specific predicates are also passed as preconditions for their corresponding actions, the action definitions could also be refined, not having to account for additional predicates that are determined by the immediate state and object relationships. By addressing the domain’s predicate logic and action preconditions in this way, the problem file can be analysed to determine how the scenario is reflected in this implementation. The original variant of the problem defined the initial state and objects in line with the domain definition. The initial state, containing action literals, would need to be replaced by a more autonomous solution, allowing the agent to reason about the problem according to objects, state, properties, and relationships as they exist in the present environment. The goal definition is included to imply the outcome requirements based on this particular problem. The initial state definition therefore needs to only imply initial state, decoupling this concern from action sequencing. Initial state can therefore be reduced to the known initial problem properties with additional hypothetical inferred state (possible, though not yet actual). The result of refining this initial state definition sees the removal of unnecessary additional preconditions which would otherwise increase overhead. This should theoretically result in the original and refined plans reproducing similar successful steps, yet with each step’s preconditions reduced. By separating these areas of focus and allowing for the action sequence to follow a non-fixed action sequence, the goal state can be achieved with fewer precondition checks, and subsequently fewer required computations. This approach better aligned the agent’s perceived nature of the problem, while better fulfilling the intended outcomes according to automated planning principles. This hypothesis was backed by the results, where the solver achieved the intended goal state with less preconditions for each step, without sacrificing logical accuracy and outcome applicability.