Introduction to Automated Planning

Automated planning is a core segment of artificial intelligence (AI) which is used to create a set of action for an intelligent agent to reach its goal from the initial state (GeeksforGeeks, 2024). This task uses PDDL (Planning Domain Definition Language) to describe a world, specify the domains possible actions, and define the specific problem scenario.

We first have the Gripper domain, and in the gripper domain, the robot (Robby) operates in two rooms and can carry balls using the two grippers attached to itself. The planning task is to move all four balls from one room (room a) to another (room b). The domain includes the standard functions: ‘move’, ‘pick’, and ‘drop’.

This project utilizes STRIPS-based planning, which is goal definition through logical predicates, with the use of preconditions and effects to direct action selection (Wikipedia Contributors, 2024).

The Wumpus World project was developed using PDDL and STRIPS-based logic to model an agent operating in a 4×4 grid-based world with hazards (pits and a Wumpus), a goal item (gold), with movement constraints. The objective is for the agent to collect the gold and return to the starting square, and avoid death by falling into a pit or being eaten by Wumpus.

Implementation Overview

The domain file (gripper_domain.pddl) defines all relevant predicates and actions using STRIPS-based logic (Wikipedia Contributors, 2024):

```
(define (domain gripper-strips)
```

The predicates used represent important facts about the world: robot (Robby), the rooms, the balls, and the grippers:

```
(:predicates (room ?r)    ;; Defines valid room locations
     (ball ?b)            ;; Identifies movable objects
     (gripper ?g)         ;; Defines robot grippers
     (at-robby ?r)        ;; Indicates Robby's location
     (at ?b ?r)           ;; Indicates ball position
     (free ?g)            ;; Describes an available (empty) gripper
     (carry ?o ?g))       ;; Links a gripper to the object it is holding
```

In the STRIPS-based planning, each action are written with clear preconditions (what must be true for the action to occur) and effects (changes after the action, what becomes true/false after the action). The domain defines three actions 'move', 'pick', and 'drop', and each action schema contains the parameter declarations, logical preconditions, and precise effects (Wikipedia Contributors, 2024).:

```
(:action move
    :parameters  (?from ?to)
    :precondition (and
    (room ?from)          ;; ?from must be a valid room
    (room ?to)            ;; ?to must be a valid room
    (at-robby ?from)      ;; Robby must currently be in the ?from room
    )
    :effect (and  (at-robby ?to)      ;; Robby is now in the destination room
          (not (at-robby ?from))      ;; Robby is no longer in the original room
     )
)
```

```
 (:action pick
  :parameters (?obj ?room ?gripper)
  :precondition (and
    (ball ?obj)           ;; The object must be a ball
    (room ?room)          ;; Room must be valid
    (gripper ?gripper)    ;; Gripper must be valid
    (at ?obj ?room)       ;; Ball must be in the same room
    (at-robby ?room)      ;; Robby must be in the room
    (free ?gripper)       ;; Gripper must be free
  )
  :effect (and
    (carry ?obj ?gripper) ;; The gripper is now holding the ball
    (not (at ?obj ?room)) ;; The ball is no longer on the floor of the room
    (not (free ?gripper)) ;; The gripper is no longer free
      )
  )
```

```
(:action drop
  :parameters (?obj ?room ?gripper)
  :precondition (and
    (ball ?obj)           ;; Must be a ball
    (room ?room)          ;; Room must be valid
    (gripper ?gripper)    ;; Gripper must be valid
    (carry ?obj ?gripper) ;; The gripper must be carrying the ball
    (at-robby ?room)      ; Robby must be in that room
  )
  :effect (and
    (at ?obj ?room)               ;; The ball is now in the room
    (free ?gripper)               ;; The gripper becomes free
    (not (carry ?obj ?gripper))   ;; The gripper is no longer carrying the ball
  )
)
```

The problem file (gripper-problem.pddl) establishes an specific planning instance where:

We have a list of objects:

- Four balls
- Two grippers
- Two rooms

An initial state using grounded facts:

- All 4 balls (ball1-ball4) start off in ‘rooma’.
- Robby starts at ‘rooma’ with both grippers (left, right) free.

And the goal state which expresses a combination of required facts is to have all all four balls in ‘roomb’.

```
(define (problem strips-gripper-x-1)
   (:domain gripper-strips)
   (:objects rooma roomb ball4 ball3 ball2 ball1 left right)
   (:init (room rooma)
          (room roomb)
          (ball ball4)
          (ball ball3)
          (ball ball2)
          (ball ball1)
          (at-robby rooma)
          (free left)
          (free right)
          (at ball4 rooma)
          (at ball3 rooma)
          (at ball2 rooma)
          (at ball1 rooma)
          (gripper left)
          (gripper right))
   (:goal (and (at ball4 roomb)
               (at ball3 roomb)
               (at ball2 roomb)
               (at ball1 roomb))))
```

The initial and goal states are defined using grounded facts, which lets the planner compute a plan from the initial configuration to the desired end state (goal). Execution was performed using the Planning Domains Editor. After uploading both domain and problem files, the planner generated the sequence of actions taken.

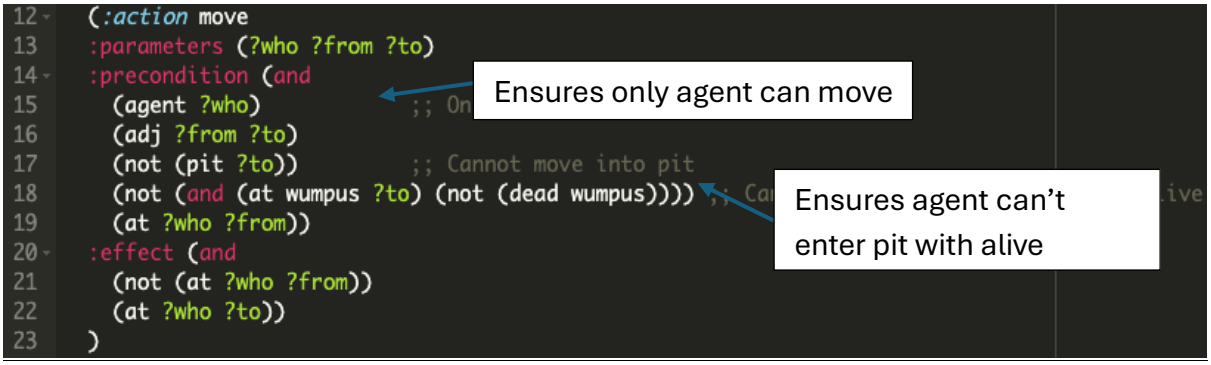
Implementation Overview for Wumpus World

The predicates used represent important facts about the world, with added ‘agent’ and ‘gold’ predicates to ensure only certain objects can perform certain actions:

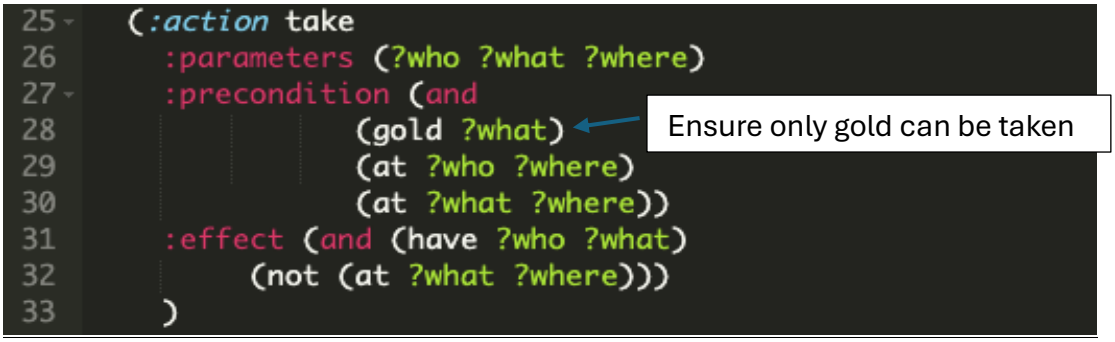
```
(:predicates
 (adj ?square-1 ?square-2)     ;; Adjacency between squares
 (pit ?square)                 ;; Pit is present
 (at ?what ?square)            ;; Location of agent/items/Wumpus
 (have ?who ?what)             ;; Inventory: agent has item
 (agent ?who)                  ;; Declares what is an agent
 (gold ?item)                  ;; Marks an item as gold
 (dead ?who))                  ;; Marks the Wumpus as dead
```

The original Wumpus World domain (wumpus_domain_a.pddl) defined basic movement, shooting, and item pickup actions. It was designed to simulate safe navigation and action execution by the agent to achieve the goal. However, it included multiple logical oversights, which were identified and corrected in this project:

- Agent type was not defined so any object could move or act, the predicate (agent ?who) ensures only agents can move. Additionally in the 'move' action, agents could move into the pits and the square with Wumpus even if it was alive, a precondition is added to ensure that agent can't enter the pit with Wumpus still alive. This makes the agent use 'shoot' if I wants to go the square where Wumpus was as shoot kills the Wumpus making the square accesible:



- The 'take' action previously allowed agent to pick up any item which includes arrow or even Wumpus. Added predicate (gold ?item) to restrict 'take' action to only gold:



The problem file (wumpus.pddl) establishes an specific planning instance where:

We have a list of objects:

- 16 squares which represents the 4x4 grid
- 1 agent which explores the world
- 1 gold object
- 1 arrow
- 1 Wumpus

An initial state using grounded facts:

- The agent starts at (square) sq-1-1
- The gold is at sq-2-3
- The Wumpus is in sq-1-3
- Agent starts with arrow in its possession
- 3 pits are placed in sq-3-1, sq-4-4, and sq-2-2
- All squares are connected through the adjacency predicates (adj) to form the navigation grid

And the goal state which expresses a combination of required facts which is for agent to retrieve gold and return to starting square safely.

```
(define (problem wumpus)
  (:domain wumpus_domain_a)
  (:objects
   sq-1-1 sq-1-2 sq-1-3 sq-1-4
   sq-2-1 sq-2-2 sq-2-3 sq-2-4
   sq-3-1 sq-3-2 sq-3-3 sq-3-4
   sq-4-1 sq-4-2 sq-4-3 sq-4-4
   the-gold
   the-arrow
   agent
   wumpus)

  (:init
   ;; Vertical adjacencies notation: sq-{X-axis-coordinate}-{Y-axis-coordinate}
   (adj sq-1-1 sq-1-2) (adj sq-1-2 sq-1-1)
   (adj sq-1-2 sq-1-3) (adj sq-1-3 sq-1-2)
   (adj sq-1-3 sq-1-4) (adj sq-1-4 sq-1-3)
   (adj sq-2-1 sq-2-2) (adj sq-2-2 sq-2-1)
   (adj sq-2-2 sq-2-3) (adj sq-2-3 sq-2-2)
   (adj sq-2-3 sq-2-4) (adj sq-2-4 sq-2-3)
   (adj sq-3-1 sq-3-2) (adj sq-3-2 sq-3-1)
   (adj sq-3-2 sq-3-3) (adj sq-3-3 sq-3-2)
   (adj sq-3-3 sq-3-4) (adj sq-3-4 sq-3-3)
   (adj sq-4-1 sq-4-2) (adj sq-4-2 sq-4-1)
   (adj sq-4-2 sq-4-3) (adj sq-4-3 sq-4-2)
   (adj sq-4-3 sq-4-4) (adj sq-4-4 sq-4-3)

   ;; Horizontal adjacencies notation: sq-{X-axis-coordinate}-{Y-axis-coordinate}
   (adj sq-1-1 sq-2-1) (adj sq-2-1 sq-1-1)
   (adj sq-2-1 sq-3-1) (adj sq-3-1 sq-2-1)
   (adj sq-3-1 sq-4-1) (adj sq-4-1 sq-3-1)
   (adj sq-1-2 sq-2-2) (adj sq-2-2 sq-1-2)
   (adj sq-2-2 sq-3-2) (adj sq-3-2 sq-2-2)
   (adj sq-3-2 sq-4-2) (adj sq-4-2 sq-3-2)
   (adj sq-1-3 sq-2-3) (adj sq-2-3 sq-1-3)
   (adj sq-2-3 sq-3-3) (adj sq-3-3 sq-2-3)
   (adj sq-3-3 sq-4-3) (adj sq-4-3 sq-3-3)
   (adj sq-1-4 sq-2-4) (adj sq-2-4 sq-1-4)
   (adj sq-2-4 sq-3-4) (adj sq-3-4 sq-2-4)
   (adj sq-3-4 sq-4-4) (adj sq-4-4 sq-3-4)

   (gold the-gold)
   (at the-gold sq-2-3) ;; Gold's location

   (pit sq-3-1) ;; Example of a pit
   (pit sq-4-4) ;; Example of a pit
   (pit sq-2-2) ;; Additional pit
   ;; You have to complete the location for all PITs

   (agent agent)
   (at agent sq-1-1) ;; Agent starting position
   (have agent the-arrow) ;; Agent starts with an arrow
   (at wumpus sq-1-3)) ;; Wumpus's location

  (:goal (and (have agent the-gold) (at agent sq-1-1)))
  )
```

Knowledge Base and Representation

The knowledge base in this PDDL planning project was constructed using a structured symbolic representation defining the agent's environment, available actions, and goal conditions. It encodes all of the domain's knowledge which the planner uses to reason about the world to create a valid sequence of actions to get from the initial state to goal state.

In the gripper domain, the agent (Robby) operates in the world consisting of: two rooms (rooma, roomb), four balls (ball1-ball4), and two grippers (left,right). They are declared in the problem file as types (room, ball, gripper). The logical predicates are the knowledge representation which describes the state of the world, which the planner then uses to determine valid next actions. For example:

- (at ball1 rooma) states that ball1 is initially in rooma as it it initialized in the problem:

```
(at ball1 rooma)
```

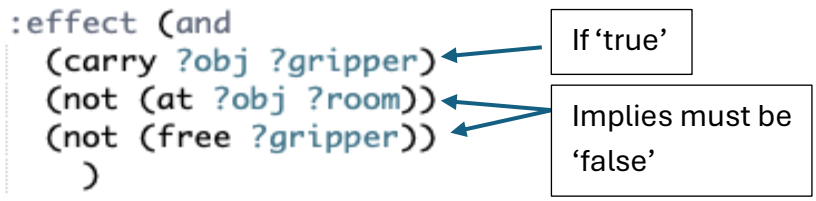
- (at ball1 roomb) as goal state specifies that ball1 must be moved to roomb.

```
(:goal (and (at ball4 roomb)
            (at ball3 roomb)
            (at ball2 roomb)
            (at ball1 roomb))))
```

These predicates represents the agent's current belief state, and they evolve according to the applied actions.

The planner infer knowledge using logical rules, for example:

- If (carry ball1 left) is true => free(left) must be false and at(ball1 rooma) must no longer hold, because the left gripper is now occupied by ball1 and ball1 is no longer in the room as it is now on the gripper.



- If (at ball2 rooma) ∧ (free right) ∧ (at-robby rooma) => pick(ball2, rooma, right) is applicable.
- If (at ball1 roomb) ∧ … ∧ (at ball4 roomb) => goal is achieved.

As a progression planner, the intelligent agent moves forward from its initial state, selecting actions whose preconditions meets the current state, then applying the effects to update the world model. For example:

- Starting state: ball1-ball4 and Robby in rooma, with both its grippers free.
- Planner picks 2 balls using 'pick' action.
- Planner moves to roomb using 'move'.
- Planner drops both balls using 'drop'.
- Planner returns to repeat.

At each step, the planner reasons using the preconditions and effect. This shows a forward state-space search strategy, as the successor states are created from applying valid actions.

The domain follows the STRIPS model where:

- States are sets of predicates.
- Actions contains clear preconditions and effects.
- Goals are a set of desired conditions.

This allows it to be compatible with planners. Showing that this knowledge base encodes state transition logic, allows action validation, and enables planner to form goal-directed plan for the agent.

Knowledge Base and Representation for Wumpus World

The Wumpus World domain is a good representation of structured symbolic AI, where the agent operates with declarative knowledge. The knowledge base includes:

- The state information: predicates such as (at agent sq-1-1), (dead Wumpus), (pit sq-4-4) tells us the environment's state
- Action preconditions and effects: describes the transitions between states, such as (have agent the-gold) only happens after the action 'take' is executed in the gold square.

```
(:action take
  :parameters (?who ?what ?where)
  :precondition (and
                (gold ?what)      ;; Only gold can be taken
                (at ?who ?where)
                (at ?what ?where))
  :effect (and (have ?who ?what)
       (not (at ?what ?where)))
  )
```

Agent has the gold after 'take'

Gold no longer there

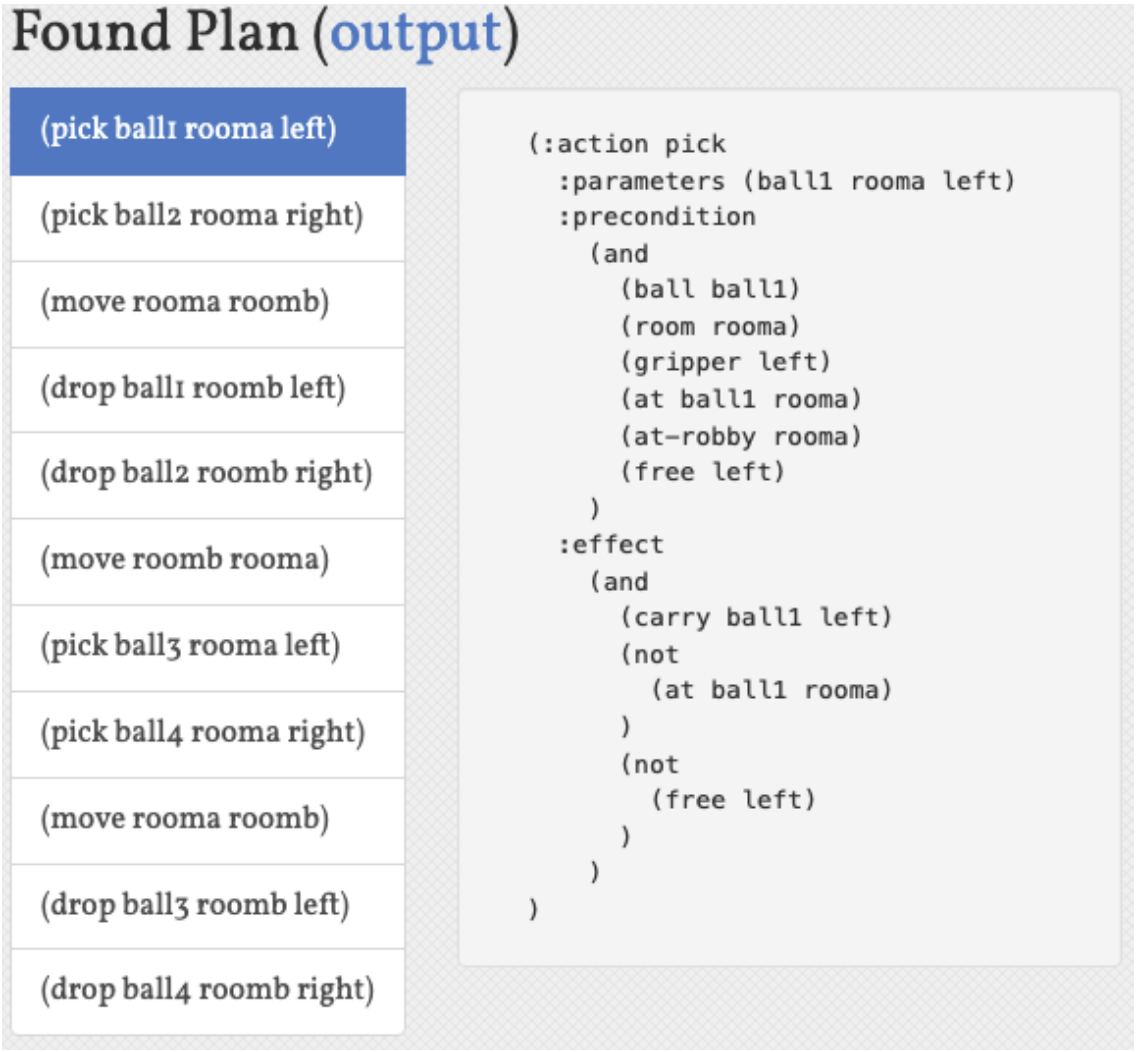
- Inferred knowledge: agent cannot move into (pit ?to) or (at wumpus ?to) unless (dead wumpus) is true, guiding safe planning behavior.

```
    (not (pit ?to))          ;; Cannot move into pit
    (not (and (at wumpus ?to) (not (dead wumpus)))) ;; Cannot move into Wumpus if it's alive
```

- Derived knowledge: planner may infer that to reach gold it has to shoot Wumpus first and agent deduces that if square is adjacent to Wumpus it has to use 'shoot' before moving through it.

This creates a goal-directed progression planner behaviour, using logical representations.

Results and Discussions



The result (output) proves that PDDL logic and action structure were correctly implemented. The predicates and action schemas allowed the planner to weight its available actions from its current world state and progress towards the goal. The plan output summary showed the plan generated by the solver is:

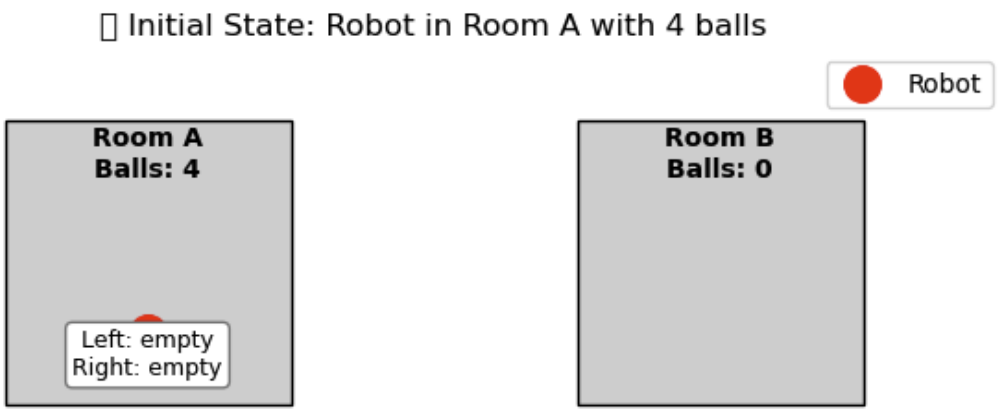
- Pick up two balls (one in each gripper)
- Move to roomb
- Drop both balls
- Return to rooma
- Repeat for the remaining balls

The planner used the optimal actions as it considers that Robby has 2 free grippers, which it then uses both grippers before each ‘move’ to reduce the number of travel

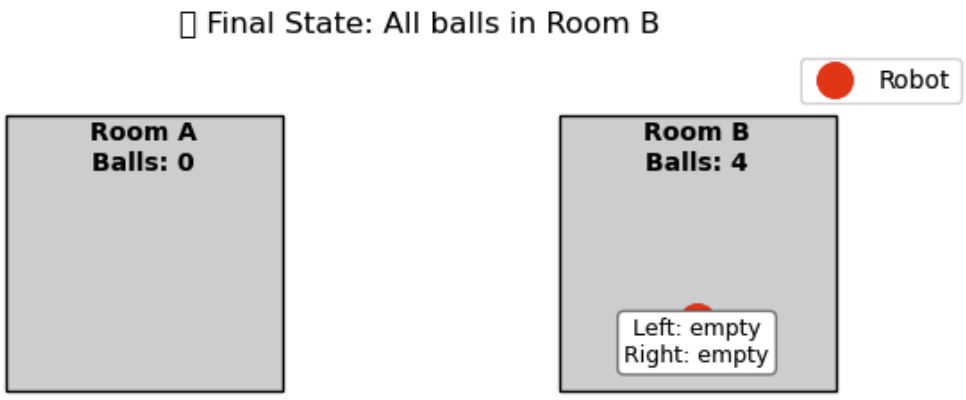
between rooms. It also displays how the action precondition and effect were logically structured, as Robby was always in the correct room with free grippers for picking and carrying a ball before dropping. Although the plan returned is in ordered sequence, some actions in the plan are independent and could occur parallelly such as:

- pick ball1 rooma left **and** pick ball2 rooma right
- drop ball1 rooma left **and** drop ball2 rooma right

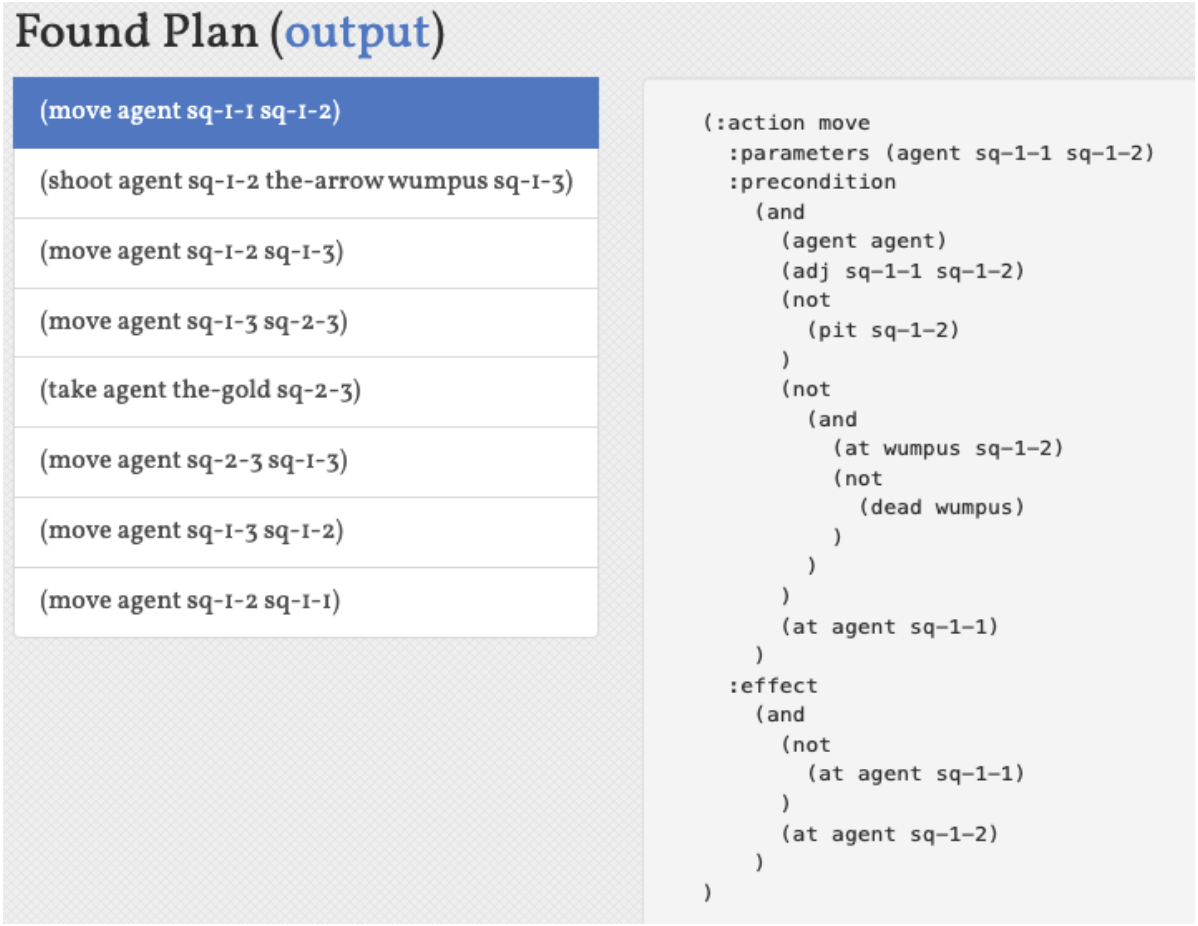
These show that the domain is able to use partial-order planning, where only the necessary ordering constraints are enforced, which improves execution flexibility.



Achieved goal of all four balls in room b:



Results and Discussions for Wumpus World

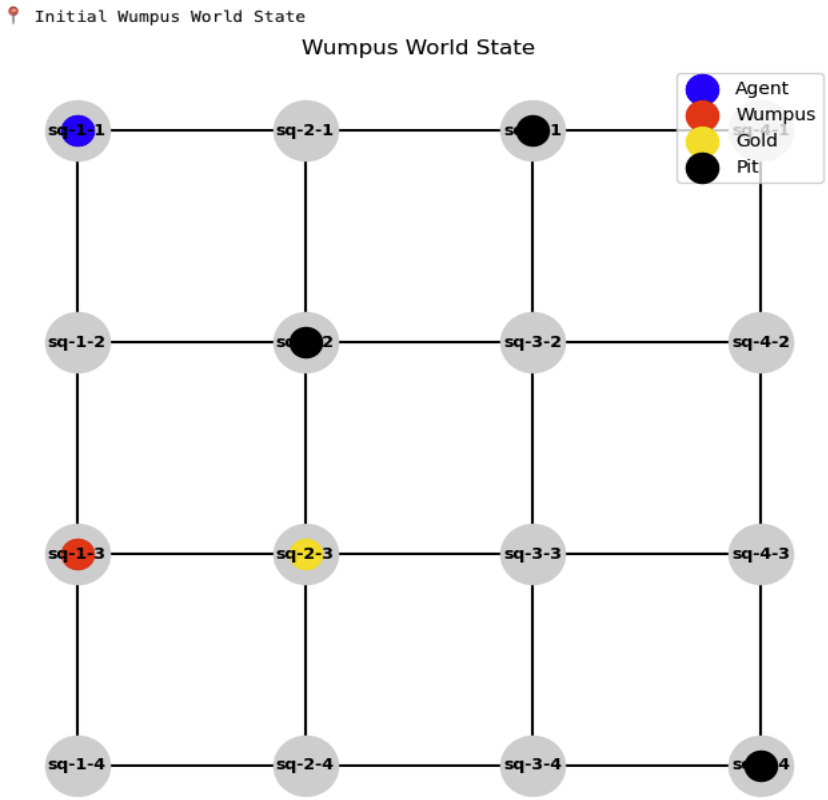


The plan output summary showed the plan generated by the solver is:

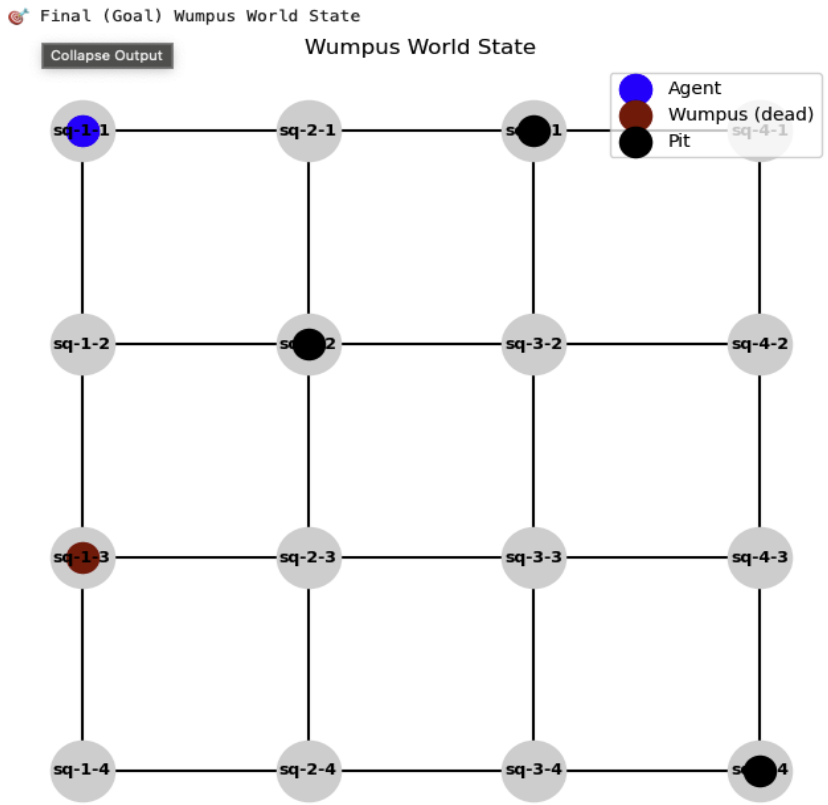
- Move down a square (from sq-1-1 to sq-1-2)
- Shoot Wumpus which is in adjacent square (from sq-1-2 to where Wumpus is sq-1-3)
- Move to the square where Wumpus previous was (sq-1-3)
- Move to the right square (from sq-1-3 to 2-3)
- Take gold which is at sq-2-3
- Return to starting grid

The planner has decided that to get to the square with gold, it had to cross the Wumpus which it then killed to cross its square. The planner also avoided pits to ensure safe navigation.

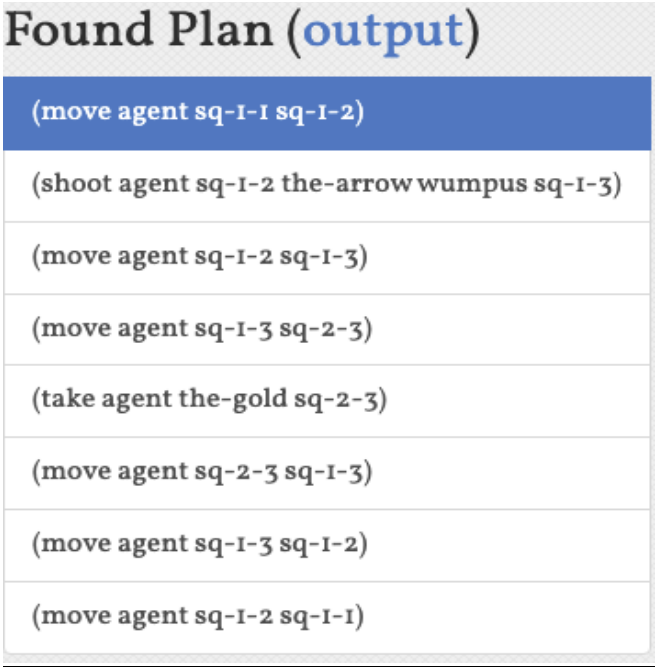
Initial Wumpus World State:



Final (Goal) Wumpus World State, achieved goal of taking gold and returning to starting square (sq-1-1):



Solver Comparison with Added Constraint to Wumpus (HD-task)



Both BFWS-ff parser and LAMA-first solver had the same steps, this is likely due to the environment forcing the planner to take similar steps. However, there are differences between these 2 solvers. Below is the comparison between the two solvers:

BFWS-ff parser	LAMA-first
Steps taken: 8	Steps taken: 8
`Fast-BFS search completed in 0.000243001 secs`	`[t=0.025509s, 12504 KB] Search time: 0.000411s`
`Nodes expanded during search: 23`	`Expanded 9 state(s).`
`Nodes generated during search: 45`	`Generated 47 state(s).`
Novelty-based search with FF heuristic	Alternating heuristic search (landmarks + FF)
Aggressive novelty exploration	Balanced heuristic search
Extremely fast on small domains, low overhead, good at discovering plans quickly.	Robust, more stable for larger or more complex problems
Limited plan diversity; novelty less impactful in small state spaces	Slightly more overhead; slower on trivial plans

Despite the identical output plan, the internal decision processes between solvers are different. BFWS prioritizes exploring novel states (unseen fact combinations), while LAMA alternates between multiple heuristics and prefers landmarks to guide its search more robustly. BFWS had a faster search time, likely due to the small size of the domain, where LAMA becomes slower due to the overhead (*Consolidating LAMA with Best-First Width Search*, 2023).

Added Constraint to Wumpus World

A constraint was added to the previous implementation of Wumpus World, the constraint is that the agent must collect a key before it can take the gold as the gold is locked.

I added the predicates for the new constraint of using key to unlock gold:

```
10    (key ?item)                    ;; Marks item as a key
11    (unlocked ?what)               ;; Marks if gold is unlocked
```

The 'take' action had added preconditions as it now checks if the gold is unlocked before taking the gold. The additional 'unlock' action is created for the agent to use the key to unlock the gold, so that 'take' can be executed after unlocking gold:

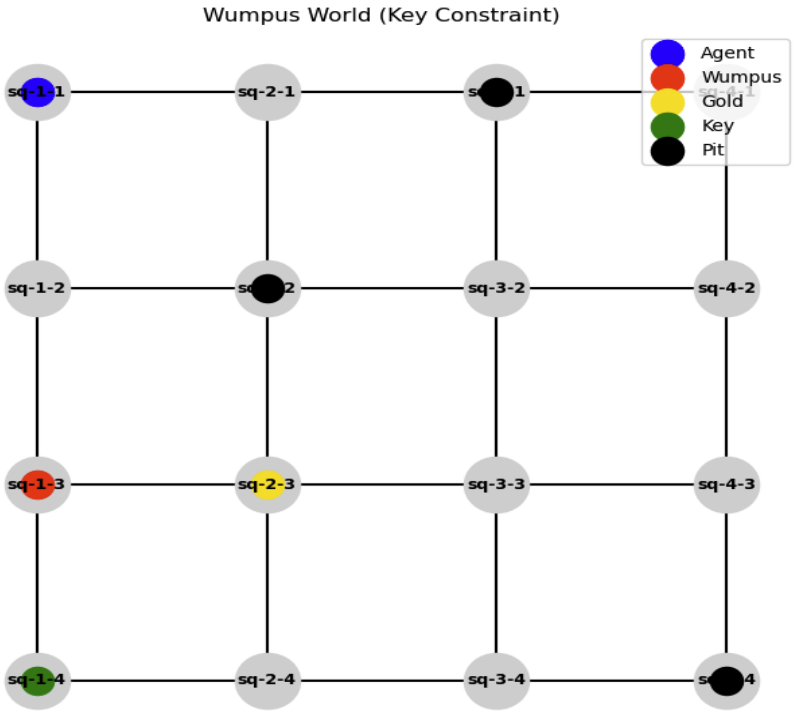
```
(:action take
  :parameters (?who ?what ?where)
  :precondition (and
          (at ?who ?where)
          (at ?what ?where)
          (or (key ?what)                          ;;agent is only allowed to take an object ?what if:
              (and (gold ?what) (unlocked ?what)))) ;;?what is a key or gold has been unlocked.
  :effect (and (have ?who ?what)
       (not (at ?what ?where)))
  )

  ;; additional 'unlock' action to use for added constraint, uses key to unlock gold
  (:action unlock
  :parameters (?who ?key ?gold ?where)
  :precondition (and
    (at ?who ?where)
    (at ?gold ?where)
    (have ?who ?key)
    (gold ?gold)
    (key ?key))
  :effect (unlocked ?gold) ;; Gold is now unlocked as effect of action
 )
```

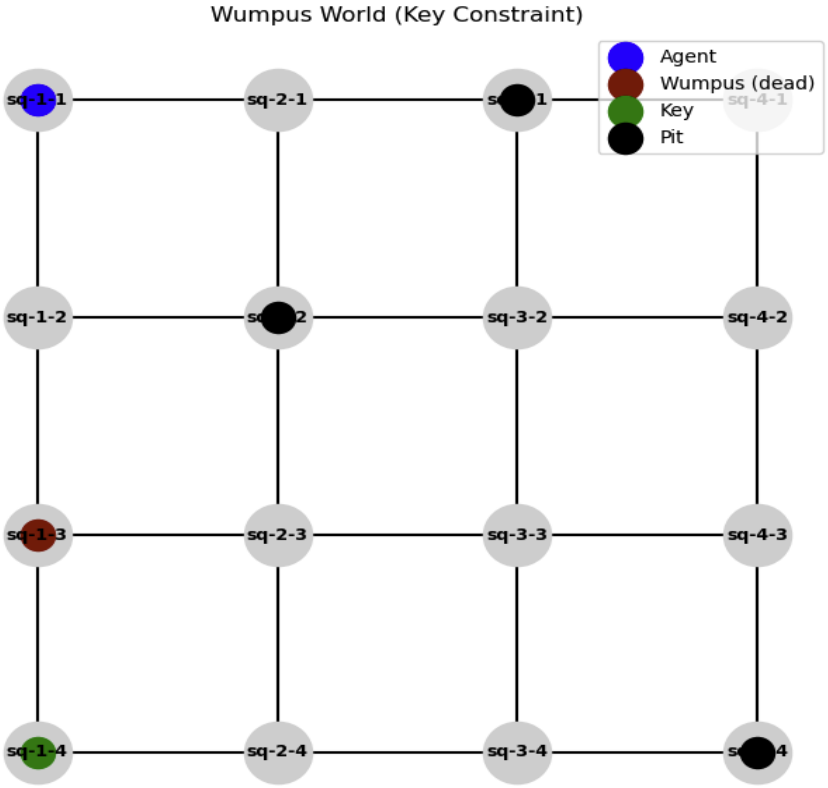
The problem file is updated to place the key in a square for agent to find:

```
45    (key the-key)
46    (at the-key sq-1-4)  ;; Key's location
```

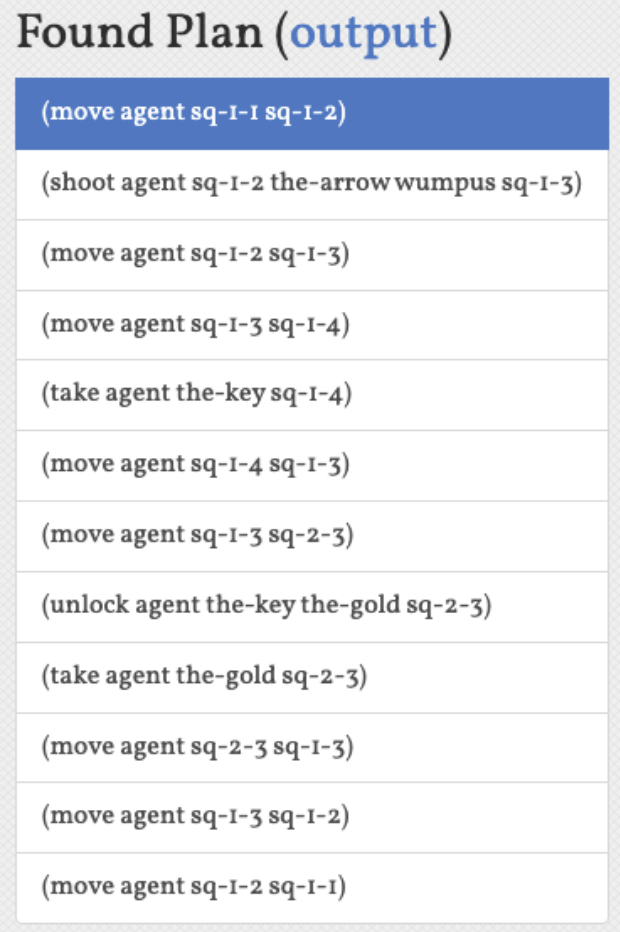
Initial Wumpus World State with added constraint:



Final (Goal) Wumpus World State with added constraint:



Planning Outcome After Added Constraints



The new constraint required the agent to pick up the key from the square its located (sq-1-4) and unlock the gold before taking it. This added both logical complexity of a new precondition in the 'take' action, and spatial complexity as additional movement was required. The previous implementation of Wumpus world without added constraint had a plan length of 8 steps. But with the addition of the new constraint, the plan length increased to 12 steps. This shows how a single dependency (needing the key) can significantly increase the reasoning depth. BFWS remained fast due to its aggressive novelty-driven exploration, while LAMA had slightly more overhead because of its reliance on multiple heuristic layers.

BFWS search time:

```
Fast-BFS search completed in 0.000353997 secs
```

LAMA search time:

```
Search time: 0.001122s
```

Conclusion

This project displays the effectiveness and efficiency of automated planning through PDDL and STRIPS-based logic. In the Gripper domain, the robot agent (Robby) planned and executed actions to transport all balls from one room to another. The planner utilised both of Robby's grippers to optimize the plan, which reduced redundant actions. This displays a goal-direction progression planning, where grounded predicates, preconditions, and effects allow for intelligent behaviour.

In the Wumpus World case, planning involved navigating a grid with hazards which the agent has to avoid and retrieve the gold and return to its starting square. Logical corrections, such as preventing unsafe movements (avoiding pits), makes the domain more realistic. The planner inferred that shooting the Wumpus was necessary in order to reach the gold.

Both domains showed the importance of structured knowledge representation and how planners use logical reasoning to obtain safe, goal-achieving sequences. This highlights key elements of cognitive intelligence, which are used in domains like robotics, logistics, and game AI.

The comparison of BFWS and LAMA solvers also demonstrated how planning performance and adaptability can vary under constraints, reinforcing the importance of selecting suitable solvers for different real-world scenarios.

References

Consolidating LAMA with Best-First Width Search. (2023). Arxiv.org. https://arxiv.org/html/2404.17648v1

GeeksforGeeks. (2024, May 27). *Automated Planning in AI*. GeeksforGeeks. https://www.geeksforgeeks.org/automated-planning-in-ai/

Wikipedia Contributors. (2024, October 31). *Stanford Research Institute Problem Solver*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Stanford_Research_Institute_Problem_Solver