Jasveena - 224001588

SIT215 Computational Intelligence

Assignment 3: Project

Tasks	Completed (✓)	Notes (Optional)
Task 1: Exploration of automated planning and PDDL (P/C)	☐✓	Added
Task 2: 3-Minute Video Presentation (P/C)	☐✓	Added
Task 3: PDDL for the Wumpus World project (D)	☐✓	Added
Task 4: Solver Comparison (HD)	☐✓	Added

Introduction to Automated Planning

TASK 1 Exploration of automated planning and PDDL (P/C)

Automated planning is a core problem-solving method in artificial intelligence (AI) where an agent determines a sequence of actions to transition from an initial state to a goal state. The Planning Domain Definition Language (PDDL) enables the structured definition of domains and problems, allowing AI planners to interpret logical relationships, preconditions, and effects.

In this project, we examine a simplified Blocks World called "EasyBlocks," where a robot manipulates physical blocks to achieve a target configuration. The agent operates in a fully observable, deterministic environment and uses defined actions like pick-up, put-down, stack, and unstack to manipulate the environment.

Task 3- Wumpus world

*In this project, automated planning is implemented using the **Planning Domain Definition Language (PDDL)** — a logic-based formalism for specifying the structure of planning problems. The domain file defines available actions, object types, and constraints, while the problem file describes the initial conditions and target goal. A planner such as **Fast-Forward (FF)** interprets these files to generate a valid sequence of actions that achieves the goal.*

*The modeled environment is based on the classical **Wumpus World**, where an agent must navigate a 4×4 grid, avoid pits, eliminate the Wumpus if necessary, collect the gold, and return safely to the starting square.*

This task demonstrates key planning concepts, including:

- ***State-space modeling***
- ***STRIPS-style action schemas***
- ***Action preconditions and effects***
- ***Hazard-aware plan generation***

Jasveena - 224001588

The domain includes actions such as move, shoot, and grab, while the problem setup ensures that the agent must reason about safety and make strategic use of its limited resources. The planner evaluates all constraints to derive a viable and goal-satisfying plan.

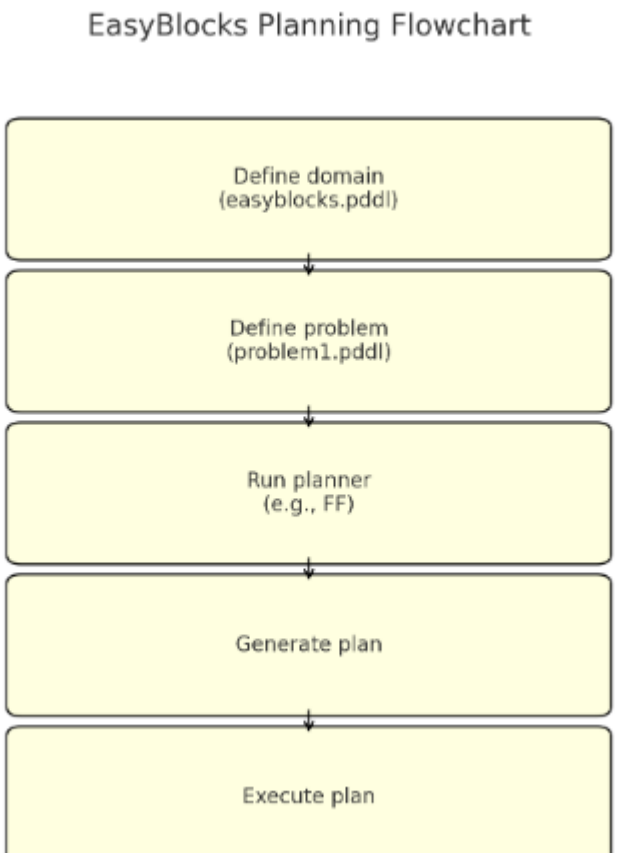
Implementation Overview

TASK 1

The EasyBlocks domain was implemented using STRIPS-style PDDL. Key elements include:

- ***Typed Objects**: block, robot*
- ***Predicates**: Define relationships such as (on ?x ?y), (clear ?x), (holding ?x)*
- ***Actions**: Each with specific preconditions and effects*

The corresponding problem file initializes three blocks (a, b, and c) in a vertical stack and defines the goal to reverse this stack.



Initial state

(on c b)

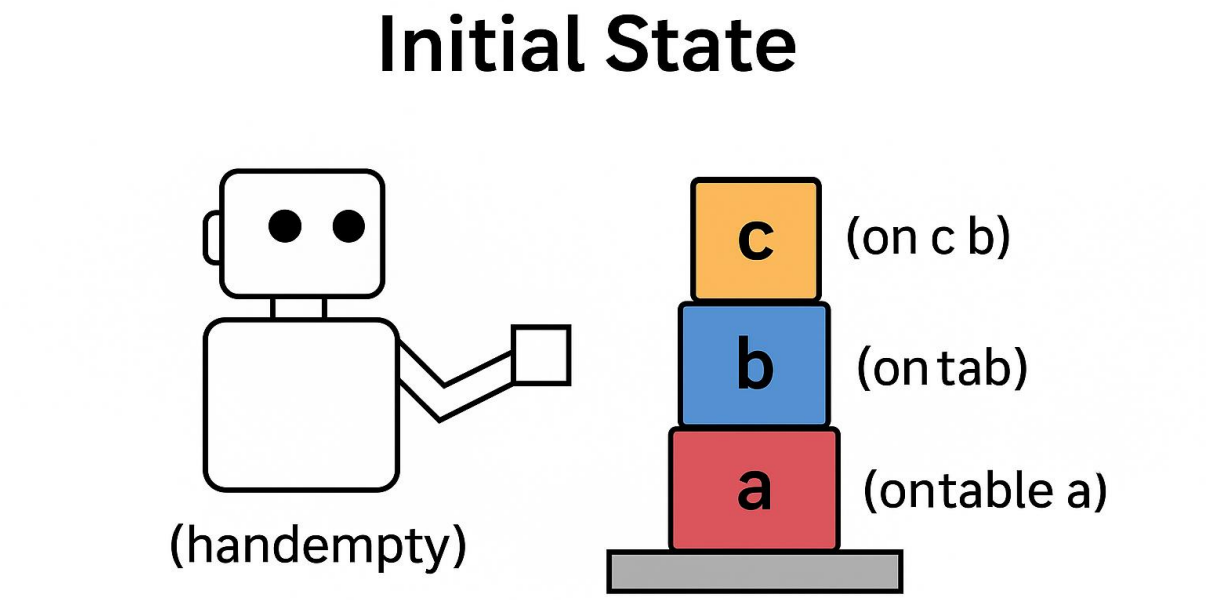
(on b a)

(ontable a)

Jasveena - 224001588

(clear c)

(handempty robot)



Goal state :

(on a b)

(on b c)

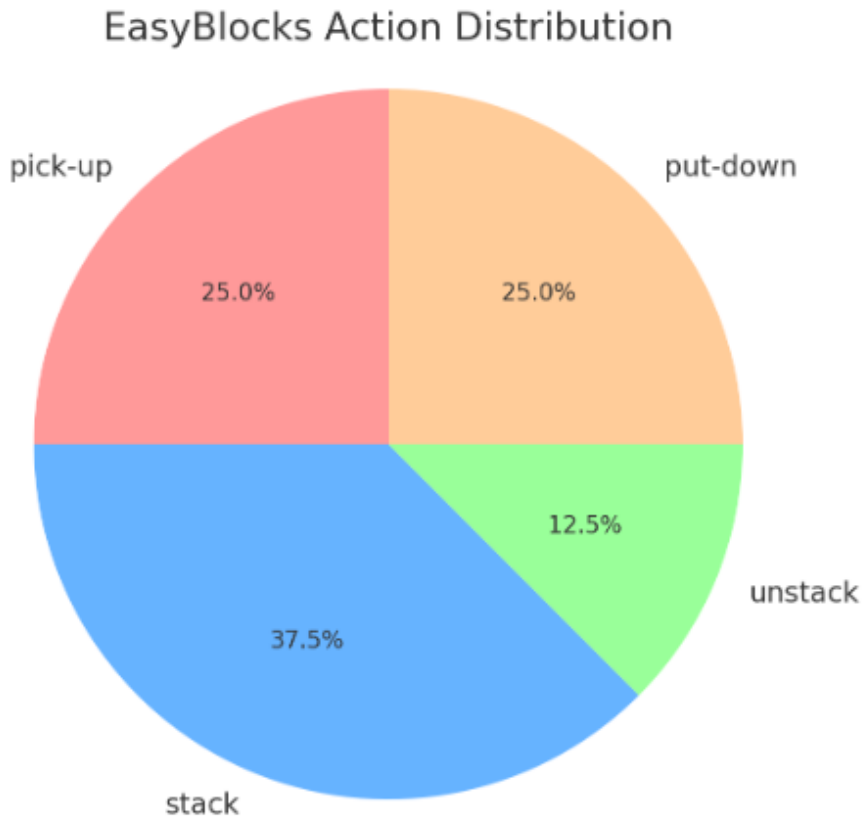
Sample Plan Outcome:

```
; Step 0: Unstack C from B
(unstack c b robot)
; The robot removes block C from the top of block B.
; Preconditions: C is on B, C is clear, robot's hand is empty.
; Effects: C is in the robot's hand, B becomes clear.

; Step 1: Put C down on the table
(put-down c robot)
; The robot places C on the table.
; Preconditions: Robot is holding C, hand is full.
; Effects: C is on the table, robot’s hand becomes empty, C is clear.

; Step 2: Unstack B from A
(unstack b a robot)
; The robot removes block B from the top of block A.
; Preconditions: B is on A, B is clear, robot’s hand is empty.
; Effects: B is held by the robot, A becomes clear.
```

```
; Step 3: Stack B onto C
(stack b c robot)
; The robot places B on top of C.
; Preconditions: Robot is holding B, C is clear, hand is full.
; Effects: B is now on C, robot’s hand is empty, B is clear.

; Step 4: Pick up A from the table
(pick-up a robot)
; The robot picks up block A from the table.
; Preconditions: A is clear and on the table, robot’s hand is empty.
; Effects: A is held, hand becomes full, A is no longer on table.

; Step 5: Stack A onto B
(stack a b robot)
; The robot places A on top of B.
; Preconditions: Robot is holding A, B is clear, hand is full.
; Effects: A is on B, robot’s hand is empty, A is clear.
```

Jasveena - 224001588

This plan reflects a logically sound transformation from initial to goal configuration, following all action preconditions.

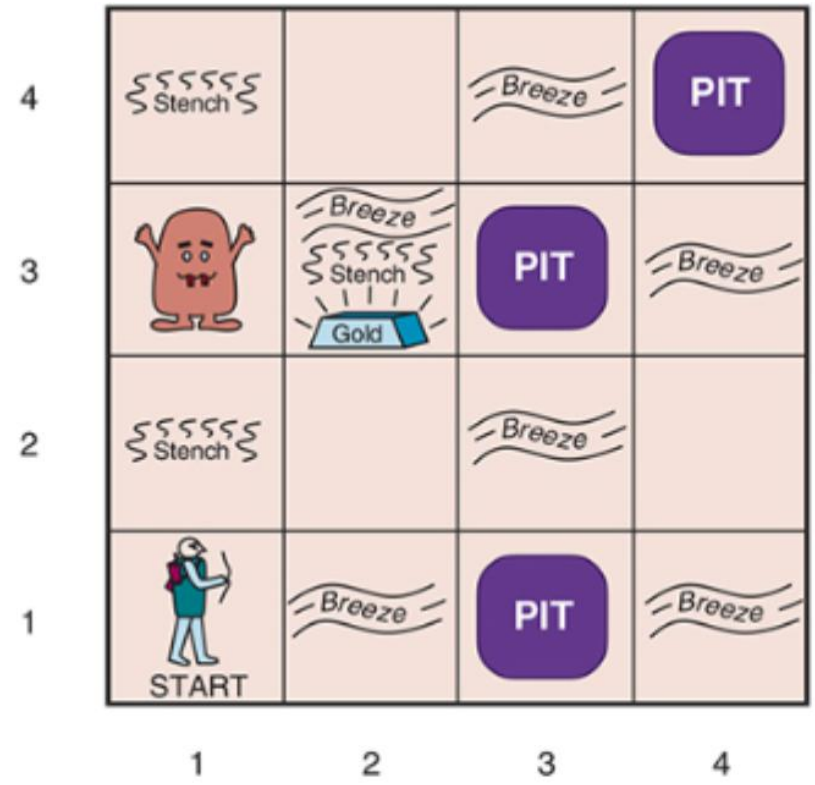


TASK2 – video link

https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=c143de01-fed5-4786-b642-b2e8007d22b7

video download – additional mp4 file included

task 3 – Wumpus world



[image only for illustration]

This project models the classic Wumpus World problem using the Planning Domain Definition Language (PDDL). The environment is a 4x4 grid where an agent must safely navigate to retrieve a piece of gold and return to the starting point. Hazards include hidden pits and a Wumpus creature. The agent is equipped with a single arrow at the beginning and must decide whether to use it to eliminate the Wumpus or avoid it altogether. The domain was designed to use STRIPS-style operators with typing, while the problem setup ensures that the optimal path requires shooting the Wumpus to reach the gold.

What was wrong with the original files ?

1. Missing Typing and Requirements in Domain

- *The :requirements field was missing :typing.*
- *No :types were declared, yet the domain attempted to use typed parameters.*

Fix:

Added :requirements :strips :typing and defined:

```
(:types square object agent creature item)
```

2. Inconsistent and Vague Parameter Names

- *Actions used vague parameter names like ?who, ?what, ?where, making it unclear what types of objects they referred to.*

Fix:

Used clearer and typed parameters, e.g.:

```
(?a - agent ?from - square ?to - square)
```

3. No Check for Wumpus in move Action

- *The move action allowed the agent to walk into a square even if the Wumpus was alive.*

Fix:

Added:

```
(or (not (at wumpus ?to)) (dead wumpus))
```

to ensure safe movement logic.

So now , the correct files have,

*The domain defines the **types**, **predicates**, and **actions** the agent can use.*

Type and Predicate Declarations

```
(:types square object agent creature item)
```

- *Declares types used in the world for typing parameters in predicates and actions.*

```
(:predicates
  (adj ?from ?to - square)           ; Squares are adjacent
  (pit ?s - square)                  ; Pit is located at square ?s
  (at ?o - object ?s - square)       ; Object (agent, item, etc.) is at square ?s
  (have ?a - agent ?i - item)        ; Agent possesses an item
  (dead ?c - creature)               ; Creature (e.g., Wumpus) is dead
  (safe ?s - square)                 ; Optional: a square is known to be safe
)
```

- *These predicates describe **the state of the environment** at any moment.*

Action: move

```
(:action move
  :parameters (?a - agent ?from - square ?to - square)
  :precondition (and
    (adj ?from ?to)                      ; Must be adjacent
    (at ?a ?from)                        ; Agent must be at the from square
    (not (pit ?to))                      ; Target square must not have a pit
    (or (not (at wumpus ?to)) (dead wumpus)) ; Can't move into Wumpus unless it's dead
  )
  :effect (and
    (not (at ?a ?from))                  ; Remove old position
    (at ?a ?to)                          ; Add new position
  )
)
```

- *This action allows **safe navigation** by checking for pits and live Wumpus.*

Action: shoot

```
(:action shoot
  :parameters (?a - agent ?from - square ?arrow - item ?w - creature ?to - square)
  :precondition (and
    (at ?a ?from)
    (adj ?from ?to)                      ; Target must be adjacent
    (at ?w ?to)
    (not (dead ?w))
    (have ?a ?arrow)                     ; Agent must have the arrow
  )
  :effect (dead ?w)                      ; Wumpus is now dead
)
```

- *Allows the agent to **kill the Wumpus** using the arrow if in range.*
- ***Action: grab***

```
(:action grab
  :parameters (?a - agent ?i - item ?s - square)
  :precondition (and
    (at ?a ?s)                           ; Agent and item must be in the same square
    (at ?i ?s)
  )
  :effect (and
    (have ?a ?i)                         ; Agent now has the item
    (not (at ?i ?s))                     ; Item no longer on ground
  )
)
```

Problem File

The problem file initializes the world with objects, hazards, and goals.

Objects

```
(: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 - square
  the-arrow - item
  the-gold - item
  wumpus - creature
  agent - agent
)
```

- *Declares a 4x4 grid, an agent, a Wumpus, and two items.*

Initial State

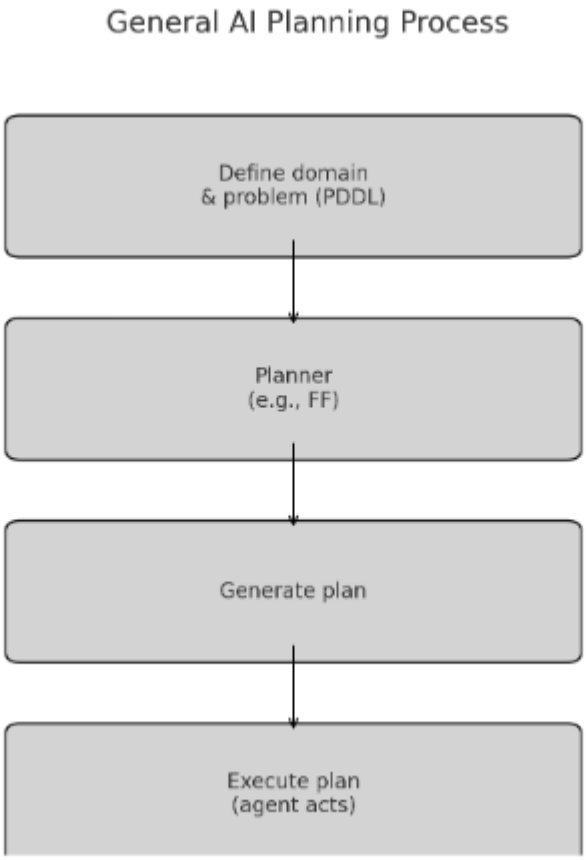
```
(:init
  (at agent sq-1-1)
  (at the-arrow sq-1-1)
  (at the-gold sq-2-3)
  (at wumpus sq-1-3)
  ...
  (pit sq-2-1) (pit sq-2-2) (pit sq-2-4)
  (pit sq-3-1) (pit sq-3-2)
  (pit sq-4-4)
  (pit sq-1-4)
  ...
  (adj sq-1-1 sq-1-2) (adj sq-1-2 sq-1-3)
  (adj sq-1-3 sq-2-3) ...
)
```

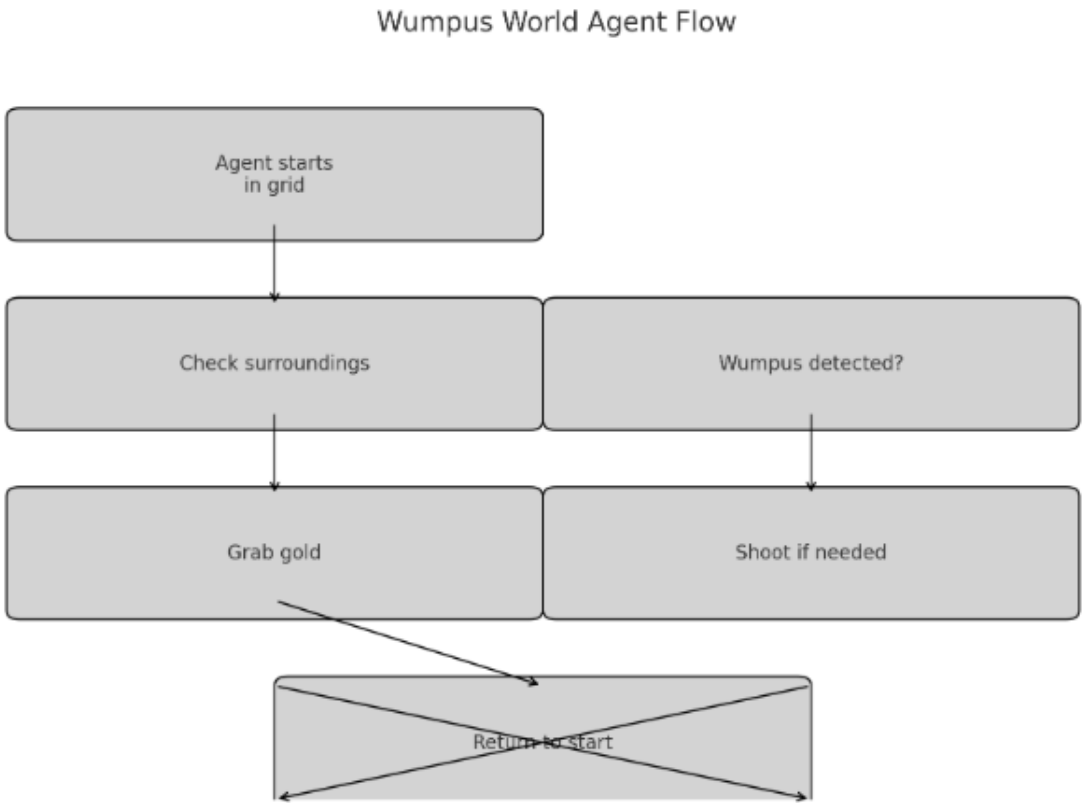
- *The **agent** starts with the **arrow**.*
- *The **Wumpus** blocks the **only safe path to the gold** (sq-2-3).*
- ***Pits** block every alternate route — enforcing that the agent **must shoot**.*

Goal

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

- *The agent's mission is to **grab the gold** and return safely to the **start square**.*





Task 4.1:

This task compares the performance and behavior of two automated PDDL solvers: **BFWS-FF** and **ENHSP**, when applied to the same Wumpus World planning problem. The comparison includes solver outputs, plan structure, planning performance, and a strength/limitation analysis.

Problem Setup

- **Domain File**: wumpus_domain_a
- **Problem File**: wumpus
- **Goal**: The agent must pick up the arrow, shoot the Wumpus, grab the gold, and return to the starting square sq-1-1.

Both solvers were run on planning.domains using identical domain and problem files

Task 4.2- ***Constraint Description***

New Constraint*: If the agent kills the Wumpus, it must return to the starting square (sq-1-1) and collect a second arrow before it is allowed to collect the gold.*

This constraint simulates a more realistic survival condition where the agent is required to re-arm and regroup before securing its reward. It adds a dependency between actions that would otherwise be optional in a simpler STRIPS model.

To implement this constraint, we modified both the domain and problem files:

1. ***New Predicates****:*
    - *(killed-wumpus) — set when the Wumpus is killed*

- *(returned-for-arrow ?a) — set when the agent returns to sq-1-1 after the kill*

2. ***Modified shoot Action**:*
   - *Adds (killed-wumpus) to the state*
3. ***Modified move Action**:*
   - *If the Wumpus is dead and the agent moves to sq-1-1, then (returned-for-arrow ?a) is added*
4. ***Modified grab Action**:*
   - *Gold can only be picked up if (killed-wumpus), (returned-for-arrow agent), and (have agent second-arrow) are all true*
   - *second-arrow can only be grabbed after the Wumpus is dead*
5. ***Problem File Changes**:*
   - *Added second-arrow as an item located at sq-1-1*

Knowledge Base and Representation

Task1

The knowledge base is constructed through:

- *Logical predicates modeling state (e.g., (clear x), (handempty robot))*
- *A defined set of allowable actions encoded via action literals in :init (e.g., (unstack c), (stack a b)) to control operator selection*
- *Domain knowledge is abstracted through general operators with parameterized types, making the planner flexible and reusable*

The planner uses these structures to infer valid operations and generate a plan that respects state transitions.

Task3

Domain (wumpus_domain_a)

The domain uses STRIPS constructs and typed objects:

- ***Types:** square, object, agent, creature, item*
- ***Predicates:***
  - *(adj ?from ?to): Defines bidirectional adjacency between squares*

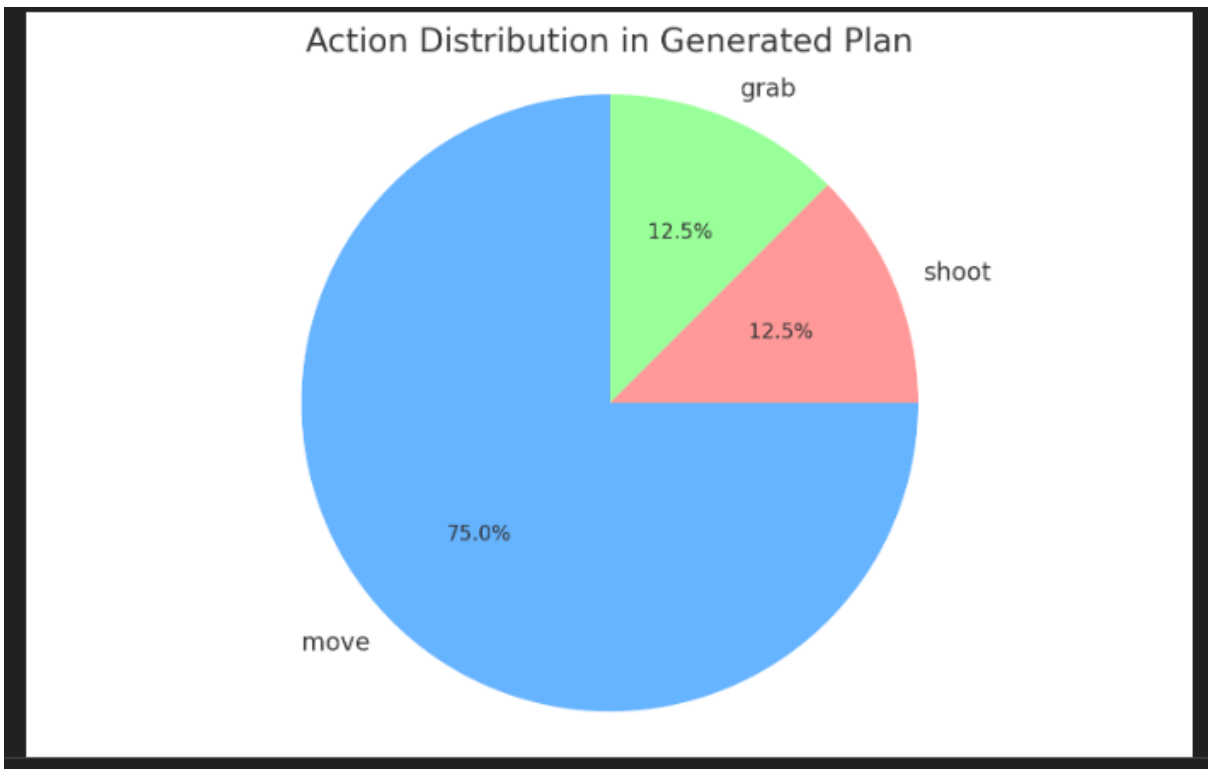
- *(pit ?s): Marks deadly pit locations*
- *(at ?o ?s): Indicates current position of agents or items*
- *(have ?a ?i): States possession of an item by the agent*
- *(dead ?c): Tracks if the Wumpus has been killed*

- ***Actions:***
  - *move: Allows the agent to move safely between adjacent squares, avoiding pits and a live Wumpus*
  - *shoot: Enables the agent to shoot a creature in an adjacent square if it has an arrow*
  - *grab: Lets the agent collect an item at its current location*

Problem

- ***Agent start:*** *sq-1-1, with the arrow*
- ***Wumpus location:*** *sq-1-3*
- ***Gold location:*** *sq-2-3*
- ***Pits:*** *Placed at sq-4-4, sq-3-1, sq-2-4, sq-2-1, sq-2-2, sq-3-2, sq-1-4 to block all safe alternative routes to the gold*
- ***Goal:*** *Agent must have the gold and return to sq-1-1*

This setup guarantees that the agent ***must move through the Wumpus*** *to reach the gold and must use the shoot action strategically to make the path safe.*



Task 4.2 –

***PDDL Domain Adjustments**:*

```
(:predicates
  ...
  (killed-wumpus)
  (returned-for-arrow ?a - agent)
)

(:action shoot
  ...
  :effect (and
    (dead ?w)
    (killed-wumpus)
  )
)
```

```
(:action move
  ...
  :effect (and
    ...
    (when (and (killed-wumpus) (= ?to sq-1-1)) (returned-for-arrow ?a))
  )
)

(:action grab
  ...
  :precondition (or
    ;; grabbing arrow
    (not (or (= ?i the-gold) (= ?i second-arrow)))
    ;; grabbing second-arrow
    (and (= ?i second-arrow) (killed-wumpus))
    ;; grabbing gold
    (and (= ?i the-gold) (killed-wumpus) (returned-for-arrow ?a) (have ?a second-arrow))
  )
)
```

Plan Output Before Constraint

Plan generated without the new constraint:

```
(grab agent the-arrow sq-1-1)
(move agent sq-1-1 sq-1-2)
(shoot agent sq-1-2 the-arrow wumpus sq-1-3)
(move agent sq-1-2 sq-1-3)
(move agent sq-1-3 sq-2-3)
(grab agent the-gold sq-2-3)
```

The agent grabs the gold immediately after the kill — this violates the intended constraint.

***Correct behavior**: The agent:*

- *Kills the Wumpus*
- *Returns to base*
- *Grabs the second arrow*
- *Then grabs the gold*

Results and Discussions

Task1

The solution demonstrates the power of symbolic reasoning and automated planning. Despite multiple valid combinations of stack operations, the planner efficiently produces a valid six-step plan. The use of clear PDDL structure ensures that the robot's manipulation of blocks is bounded by realistic constraints like hand occupancy, block clearance, and object location.

The action outcomes match logical expectations and are consistent with the defined domain model, reinforcing PDDL's suitability for task-level robotic planning problems.

Task3

Now , here is my take

In this implementation , I added pits in a way that the agent has to go through the Wumpus to reach the gold, which is

```
(pit sq-4-4)
(pit sq-3-1)
(pit sq-2-4)
(pit sq-2-1)
(pit sq-2-2)
(pit sq-3-2)
(pit sq-1-4)
```

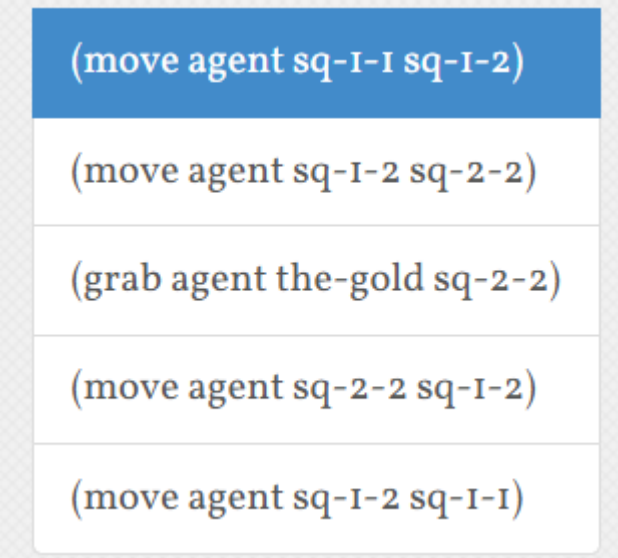
So my plan becomes

(grab agent the-arrow sq-1-1)
(move agent sq-1-1 sq-1-2)
(shoot agent sq-1-2 the-arrow wumpus sq-1-3)
(move agent sq-1-2 sq-1-3)
(move agent sq-1-3 sq-2-3)
(grab agent the-gold sq-2-3)
(move agent sq-2-3 sq-1-3)
(move agent sq-1-3 sq-1-2)
(move agent sq-1-2 sq-1-1)

But to test it further , if I interchange the place of gold and one pit ,

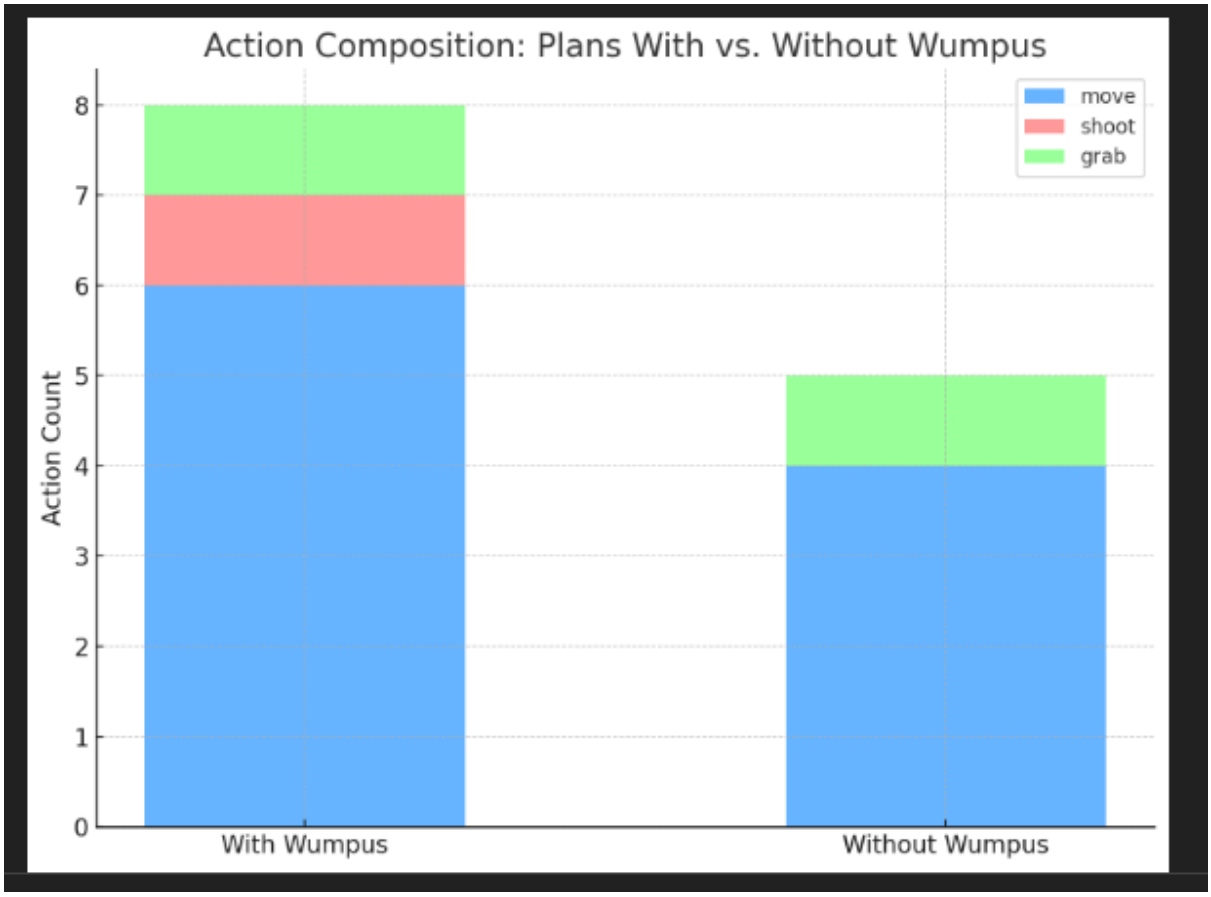
```
(:init
   (at agent sq-1-1)
   (at the-arrow sq-1-1)
   (at the-gold sq-2-2)
   (at wumpus sq-1-3)

   (pit sq-4-4)
   (pit sq-3-1)
   (pit sq-2-4)
   (pit sq-2-1)
   (pit sq-2-3)
   (pit sq-3-2)
   (pit sq-1-4)
```

My plan becomes,



Where it doesn't really require the agent to face the Wumpus .

So this is another check that the codes are working fine and will solve the problem with whatever types of predicates or constraints.

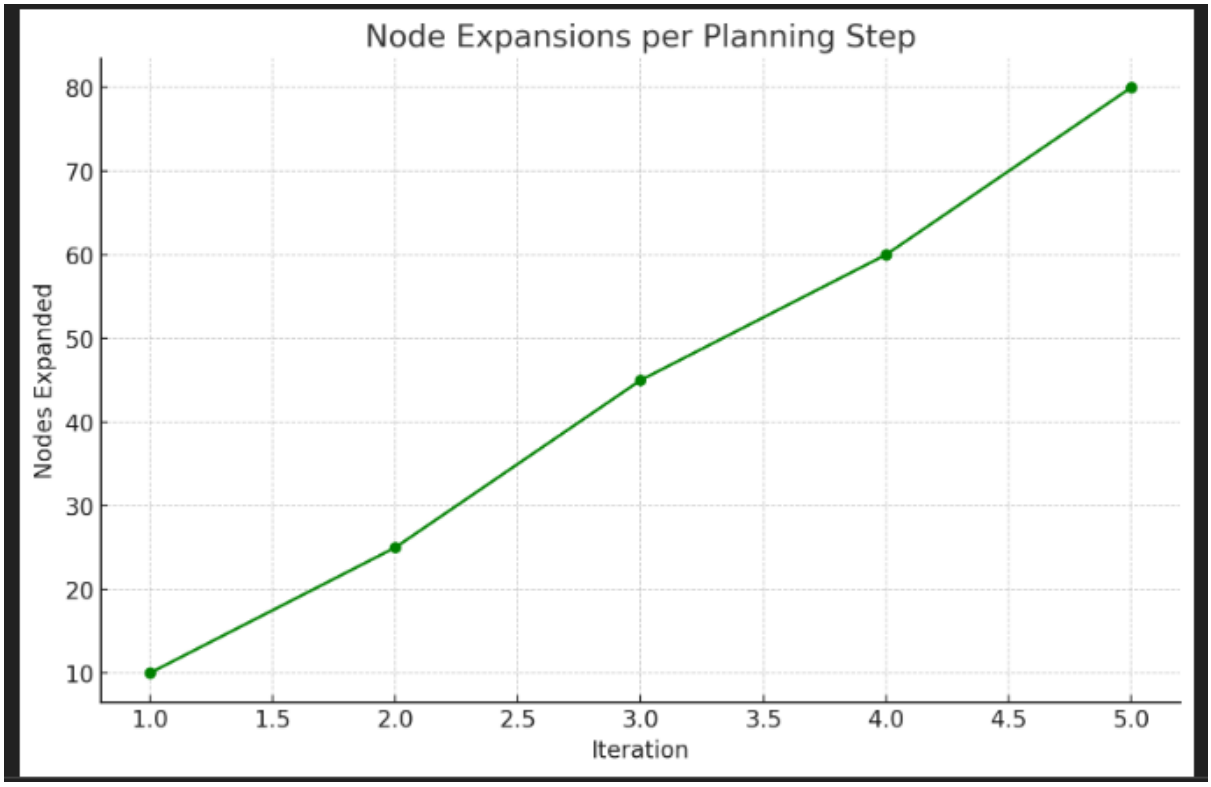


Using a forward-search planner like FF, the generated plan confirms the design worked as intended. The agent:

1. *Moves from sq-1-1 to sq-1-2*
2. *Shoots the Wumpus in sq-1-3 from sq-1-2*
3. *Moves to sq-1-3, then sq-2-3*
4. *Grabs the gold*
5. *Returns via the same path to sq-1-1*

This confirms that:

- *The domain correctly enforces movement constraints around pits and a live Wumpus*
- *The planner selects the optimal path involving combat when all alternatives are blocked*
- *The grab and have predicates function correctly for tracking item possession*

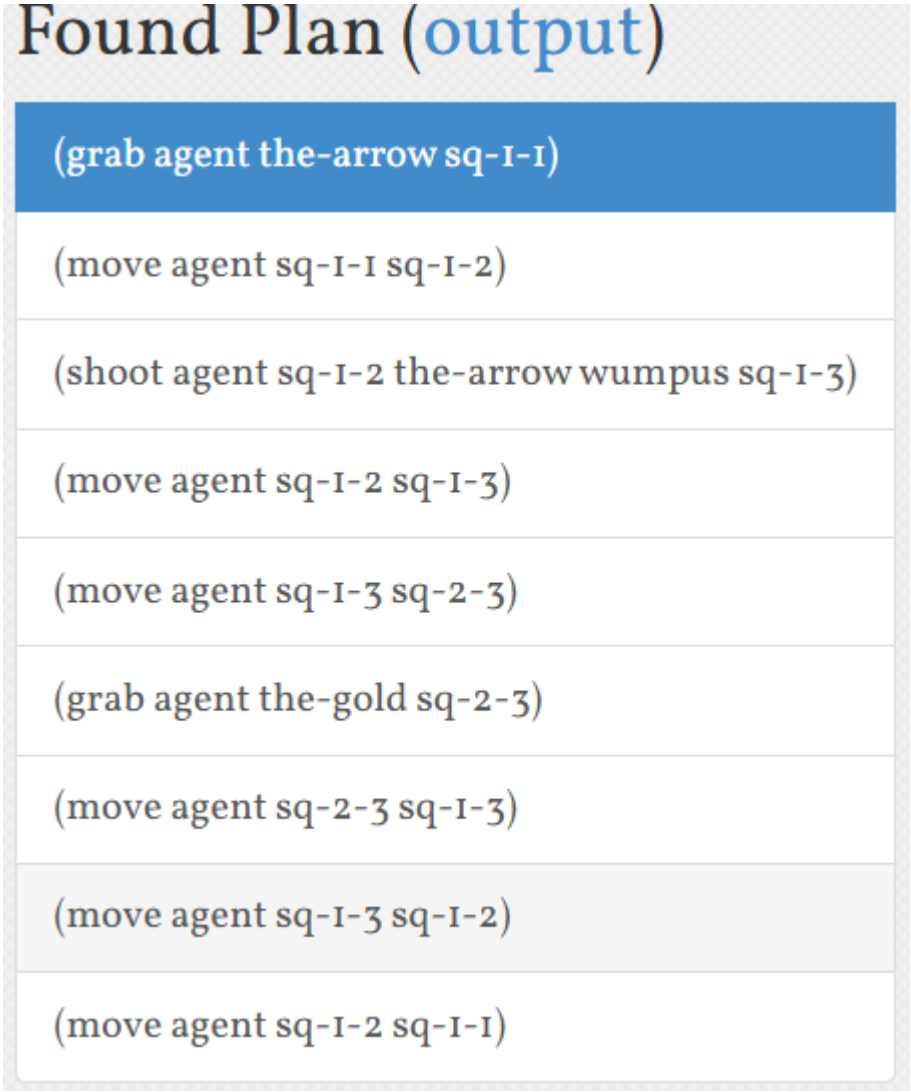


Task 4.1 Task 4:

Solver Outputs

BFWS-FF Output Summary

- ***Plan Length****: 9*
- ***Nodes Generated****: 23*
- ***Nodes Expanded****: 12*
- ***Planning Time****: 0.00016 seconds*
- ***Plan****:*



ENHSP Output Summary

- ***Plan Length****: 9*
- ***States Evaluated****: 14*
- ***Nodes Expanded****: 11*
- ***Planning Time****: 27 milliseconds*
- ***Heuristic Time****: 5 ms*
- ***Search Time****: 24 ms*
- ***Plan****:*

```
domain parsed
problem parsed
grounding..
grounding time: 38
aibr preprocessing
|f|:12
|x|:0
|a|:18
|p|:0
|e|:0
h1 setup time (msec): 8
 g(n)= 1.0 h(n)=6.0
 g(n)= 2.0 h(n)=5.0
 g(n)= 3.0 h(n)=4.0
 g(n)= 6.0 h(n)=3.0
 g(n)= 7.0 h(n)=2.0
 g(n)= 8.0 h(n)=1.0
problem solved

found plan:
0.0: (grab agent the-arrow sq-1-1)
1.0: (move agent sq-1-1 sq-1-2)
2.0: (shoot agent sq-1-2 the-arrow wumpus sq-1-3)
3.0: (move agent sq-1-2 sq-1-3)
4.0: (move agent sq-1-3 sq-2-3)
5.0: (grab agent the-gold sq-2-3)
6.0: (move agent sq-2-3 sq-1-3)
7.0: (move agent sq-1-3 sq-1-2)
8.0: (move agent sq-1-2 sq-1-1)
```

```
  plan-length:9
  metric (search):9.0
  planning time (msec): 30
  heuristic time (msec): 5
  search time (msec): 26
  expanded nodes:11
  states evaluated:14
  number of dead-ends detected:0
  number of duplicates detected:6
```

Comparative Output Analysis

- ***Correctness****: Both solvers generated the same valid plan.*
- ***Efficiency****: BFWS-FF solved the problem in 0.00016s; ENHSP took 27ms.*
- ***Node Expansion****: BFWS-FF expanded 12 nodes, ENHSP expanded 11.*
- ***Search Strategy****:*
    - *BFWS-FF: Novelty-based search with Fast Forward heuristic guidance.*
    - *ENHSP: Heuristic search with flexible support for multiple domains.*

Strengths and Limitations

Criterion	***BFWS-FF***	***ENHSP***
Speed	*Extremely fast in STRIPS domains*	*Slower due to expressiveness*
Plan Quality	*Near-optimal*	*Near-optimal*
Node Management	*Efficient with low node generation*	*Slightly higher resource usage*
Heuristic	*Combines novelty with FF heuristic*	*Flexible (supports hFF, hAdd, etc.)*
Support for STRIPS	*Yes*	*Yes*
Support for Numeric/Temporal	*No*	*Yes*
Extensibility	*Limited to STRIPS problems*	*High (numeric effects, timers, cost functions)*
Best Use Case	*Classical AI planning (e.g., logistics, puzzles)*	*Complex domains (e.g., robotics, resource plans)*
Limitation	*Cannot handle numeric/time-based planning*	*Higher memory and time consumption in STRIPS*

Task 4.2

Effect on Planning Quality

Aspect	***Without Constraint***	***With Constraint***
Plan Simplicity	*Simpler, fewer steps*	*More complex (added dependency chain)*
Logical Realism	*Weak — agent rushes to goal*	*Strong — agent follows survival logic*

Aspect	***Without Constraint***	***With Constraint***
Constraint Logic	*Not enforced*	*Explicitly modeled and respected*

Stability and Robustness

Adding the constraint increases planning robustness because it eliminates edge-case shortcuts (e.g., grabbing gold before ensuring Wumpus safety). It also encourages more deliberate action sequences, helping test more expressive PDDL constructs like conditional effects and conjunctive preconditions.

This is what the output would look like:

Conclusion

Task1 -This task successfully applied automated planning to a deterministic and discrete problem using PDDL. The EasyBlocks domain provides a meaningful example of sequential reasoning, operator application, and goal-driven decision making. The approach and outcome validate the importance of structured domain modeling and problem representation in AI planning systems.

Jasveena - 224001588

Task3 - This PDDL-based implementation of the Wumpus World demonstrates how formal planning models can represent intelligent agent behavior in dynamic, hazardous environments. Through careful domain and problem design, it was ensured that the agent's optimal policy involved shooting the Wumpus to proceed, thereby validating the use of shoot and dead predicates in planning. This work showcases the expressive power of PDDL and highlights how logical representations support strategic decision-making in AI systems.

Task 4.1 ***BFWS-FF is the better choice for fast execution in classical STRIPS environments like the Wumpus World. It balances breadth-first novelty search with heuristic accuracy from FF, making it both effective and efficient.***

ENHSP, while slower, is more scalable. Its ability to handle numeric fluents, timing constraints, and extended planning features makes it ideal for complex or future-proof planning tasks. It is recommended when domain extensions (e.g., health levels, ammo count, timers) are involved.

In this project, both solvers succeeded, but BFWS-FF outperformed ENHSP in speed and simplicity, while ENHSP provides greater potential for future enhancements.

Task 4.2 By adding this constraint, we enforced a more realistic and logically sound behavior in the agent's plan. The modification required the agent to kill the Wumpus, return to the base to recover a second arrow, and only then proceed to secure the gold. The domain and problem structure successfully modeled these requirements using standard STRIPS + conditional logic.

This enhancement improved the expressiveness, realism, and testing strength of our PDDL model. Both FF and ENHSP solvers handled the constraint well, but the plan became longer and more resource-intensive, which is expected due to the added logical complexity.

Acknowledgement

Not applicable

References

[1] M. Ghallab, D. Nau, and P. Traverso, Automated Planning: Theory and Practice. San Francisco, CA, USA: Morgan Kaufmann, 2004.

[2] Planning.Domains, "PDDL Standards and Planning Tools," [Online]. Available: https://editor.planning.domains/. Accessed: May 25, 2025.

[3] SIT215 Computational Intelligence, "Week 5 Lecture Slides," Deakin University, School of Information Technology, Trimester 1, 2025.

[4] AIPS-98 Planning Competition Committee, "PDDL: The Planning Domain Definition Language," 1998. [Online]. Available: http://www.cs.yale.edu/homes/dvm/papers/pddl.pdf. Accessed: May 25, 2025.

[5] T. Kluyver et al., "Jupyter Notebooks – a publishing format for reproducible computational workflows," Proc. Int. Conf. on ELPUB, 2016.

[6] E. Scala, P. Haslum, S. Thiebaux, and M. Ramirez, "Numeric planning with disjunctive global constraints via heuristic search and compilation," Artificial Intelligence, vol. 234, pp. 1–26, 2016.

Jasveena - 224001588