Project 2: Pddl for planning problems

School of Science and Engineering

Meriem Lmoubariki

Dr. Tajjeeddine Rachidi

Al Akhawayn University in Ifrane

October 29, 2025

Table of Contents

Introduction

In this project, we used a special tool called Planning Domain Definition Language (PDDL) to solve different types of planning problems (Dolejsi, n.d.; Kalaboud, n.d.). Our main goal was to see how well various planning methods work and to understand their results. We worked on several tasks, including arranging blocks, solving a puzzle, managing a robot, and exploring a game scenario as an extra challenge.

This report explains how we set up our planning tools, how we wrote and ran our plans, and what we learned from the outcomes. We aimed to find out which planning methods are best for different tasks and why.

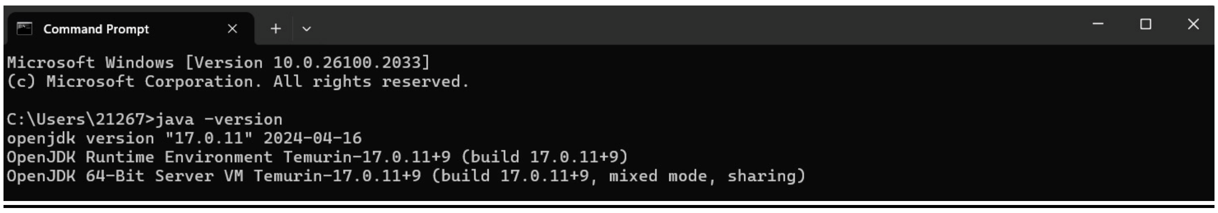
The setup

Before starting to define and solve planning problems with the Planning Domain Definition Language (PDDL), we made sure that our tools and environment were properly set up.

Process:

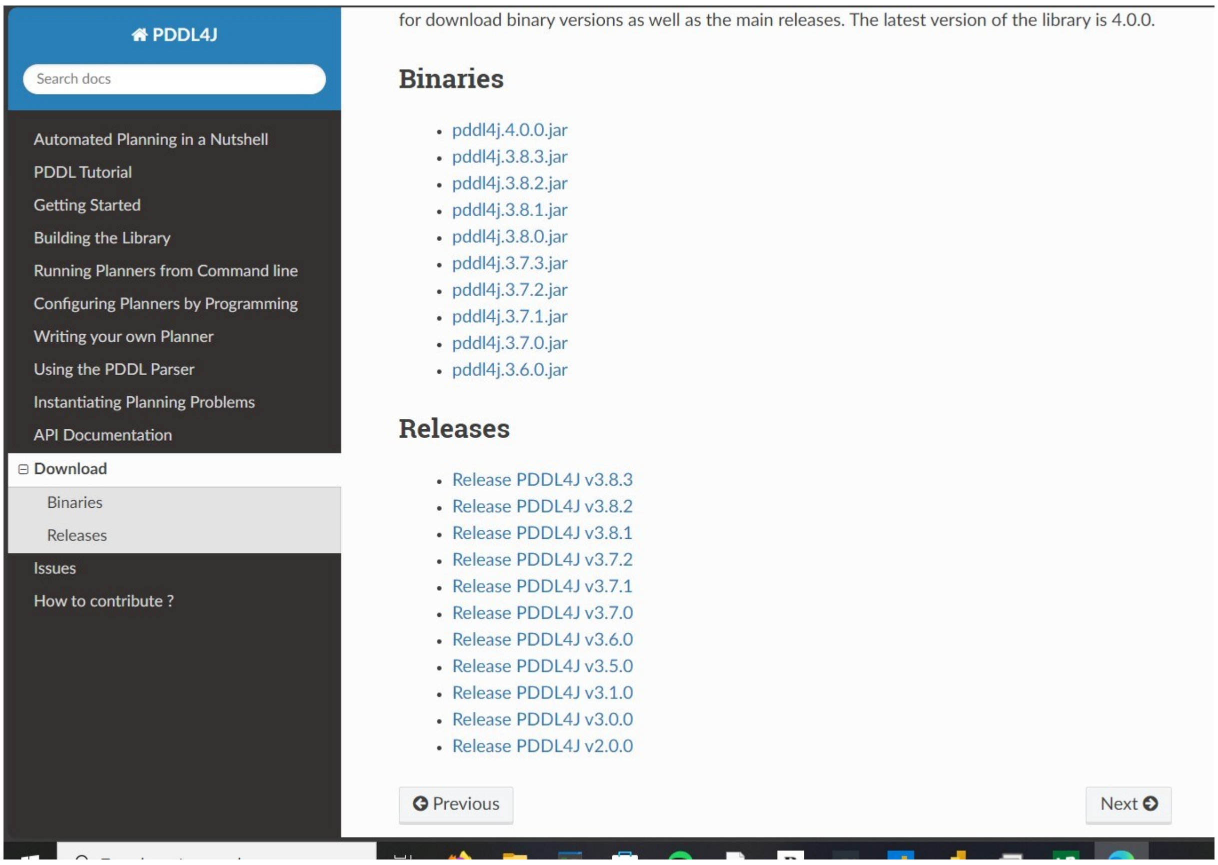
- Since Java is essential for running the PDDL4J tool, our first step was to check the existing installation. We used the command java -version in the command prompt to verify that Java was already installed and to confirm that the version met the requirements for PDDL4J. With Java confirmed, we proceeded without needing to install or update it, saving us time and effort.

Results:



```
Microsoft Windows [Version 10.0.26100.2033]
(c) Microsoft Corporation. All rights reserved.

C:\Users\21267>java -version
openjdk version "17.0.11" 2024-04-16
OpenJDK Runtime Environment Temurin-17.0.11+9 (build 17.0.11+9)
OpenJDK 64-Bit Server VM Temurin-17.0.11+9 (build 17.0.11+9, mixed mode, sharing)
```

Next, we focused on setting up PDDL4J, the tool we would use to parse and execute our PDDL files. We downloaded the latest version from its official website and placed the files in a specific directory dedicated to our project (Laborie, n.d.). Following the installation guide, we set up PDDL4J, making sure all components were ready and compatible with our Java setup.



for download binary versions as well as the main releases. The latest version of the library is 4.0.0.

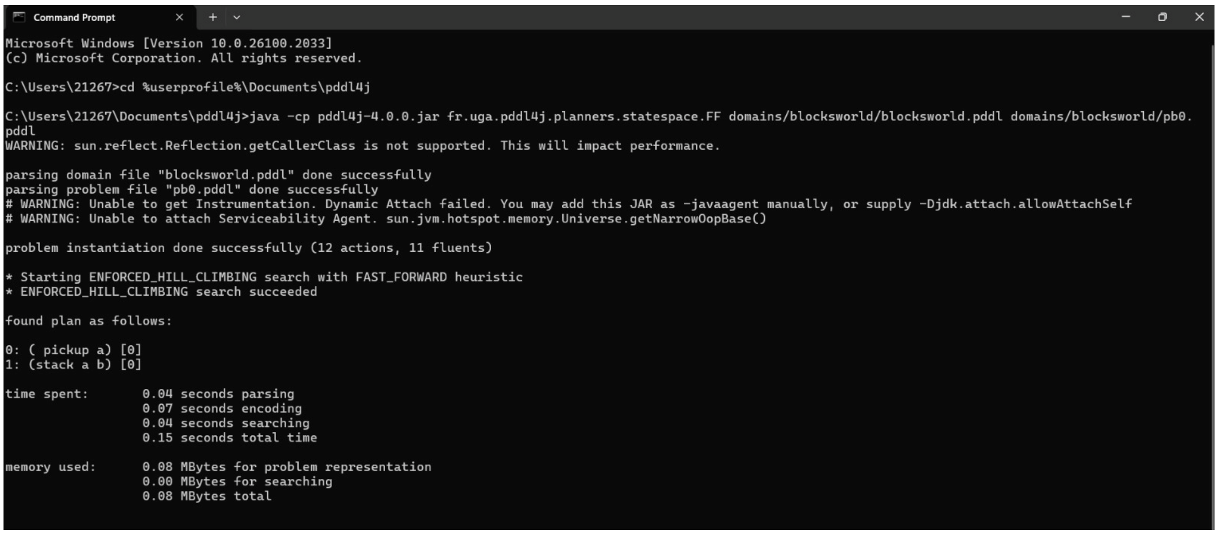
Binaries

- pddl4j.4.0.0.jar
- pddl4j.3.8.3.jar
- pddl4j.3.8.2.jar
- pddl4j.3.8.1.jar
- pddl4j.3.8.0.jar
- pddl4j.3.7.3.jar
- pddl4j.3.7.2.jar
- pddl4j.3.7.1.jar
- pddl4j.3.7.0.jar
- pddl4j.3.6.0.jar

Releases

- Release PDDL4J v3.8.3
- Release PDDL4J v3.8.2
- Release PDDL4J v3.8.1
- Release PDDL4J v3.7.2
- Release PDDL4J v3.7.1
- Release PDDL4J v3.7.0
- Release PDDL4J v3.6.0
- Release PDDL4J v3.5.0
- Release PDDL4J v3.1.0
- Release PDDL4J v3.0.0
- Release PDDL4J v2.0.0

In the initial testing phase, we executed the PDDL4J planner with a simple blocks world problem to be sure that the software was correctly interfacing with Java and that our environment was properly configured. Despite minor warnings related to Java API support and dynamic attachment failures, which suggest areas for further configuration refinement, the planner operated efficiently. It parsed the input files quickly, utilized minimal memory, and found the expected solution, demonstrating the readiness of our setup for more complex tasks.



```
Microsoft Windows [Version 10.0.26100.2033]
(c) Microsoft Corporation. All rights reserved.

C:\Users\21267>cd %userprofile%\Documents\pddl4j

C:\Users\21267\Documents\pddl4j>java -cp pddl4j-4.0.0.jar fr.uga.pddl4j.planners.statespace.FF domains/blocksworld/blocksworld.pddl domains/blocksworld/pb0.
pddl
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "blocksworld.pddl" done successfully
parsing problem file "pb0.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (12 actions, 11 fluents)

* Starting ENFORCED_HILL_CLIMBING search with FAST_FORWARD heuristic
* ENFORCED_HILL_CLIMBING search succeeded

found plan as follows:

0: ( pickup a) [0]
1: (stack a b) [0]

time spent:       0.04 seconds parsing
                  0.07 seconds encoding
                  0.04 seconds searching
                  0.15 seconds total time

memory used:      0.08 MBytes for problem representation
                  0.00 MBytes for searching
                  0.08 MBytes total
```

Task 1: Blocks words

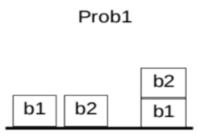
The Blocks World problem is often used as an introductory task to understand PDDL structures, and tutorials like “Modeling in PDDL - Episode 1: Blocksworld” were particularly useful here, as they detailed similar scenarios with stacking blocks (Modeling in PDDL, n.d.).

In this first exerice, the concept is to stack blocks in a specified configuration. We selected different planners within this tool to evaluate their performance across several scenarios involving different starting and goal states of block configurations. These configurations were defined in problem files pb0.pddl, pb1.pddl, pb2.pddl, and pb3.pddl. Our goal was to understand how various planners manage these problems and to identify the most efficient planning methods based on the complexity of each task.

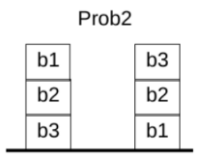
The Fast Forward (FF) heuristic was applied in combination with two planning methods, Enforced Hill Climbing (EHC) and A*, to optimize the solution path, inspired by resources on planning techniques (Kalaboud, n.d.; University of Toronto, n.d.).

Problem Configuration of each prob

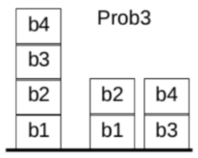
Prob1: Two blocks needed to be stacked one on top of the other



Prob2: Three blocks had to be rearranged into a tall stack.



Prob3: Four blocks needed to be organized in a specific order.



What we did :

Each block can sit on the table or on top of another block, and the goal is to stack them in a specific order. We tried to solve three variations of this problem, each increasing in difficulty from just two blocks to four blocks. So we made sure to use right and adapted code to model our problem.

Planners Used and results:

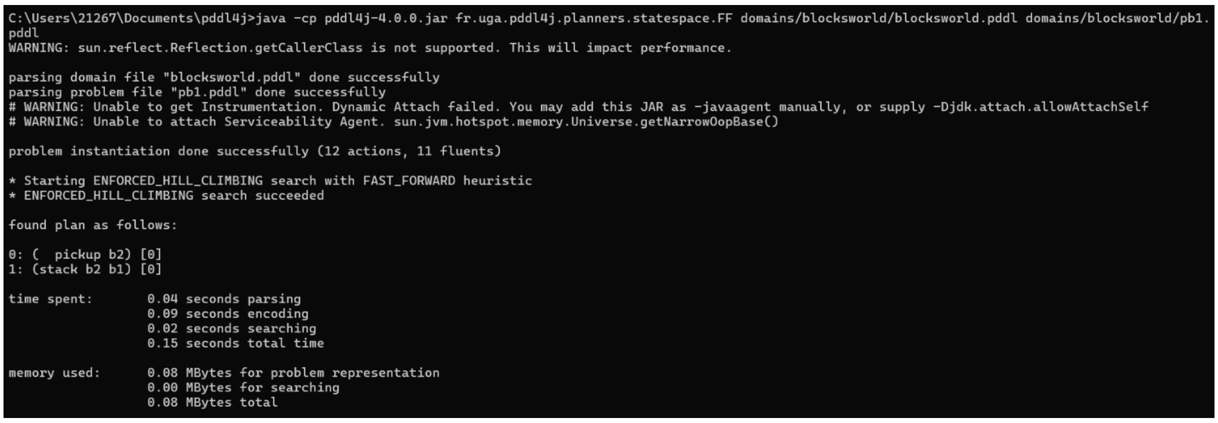
We used the FF heuristic with two different planning methods:

Enforced Hill Climbing (EHC) and A* Search (Astar). We picked these methods because they work differently, and we wanted to compare how each one performs with the help of FF.

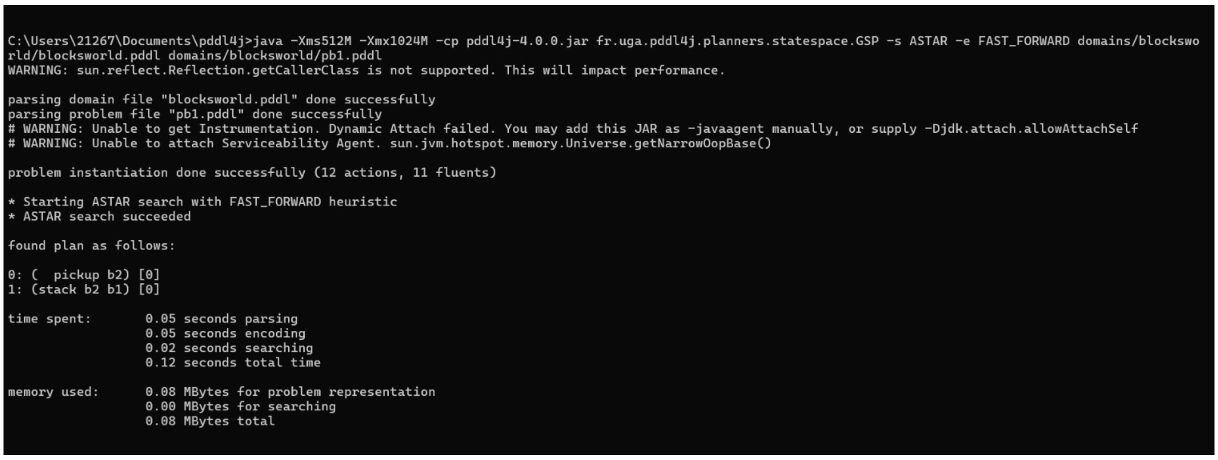
While there are various other planners like Greedy Best-First Search, Depth-First Search, Breadth-First Search, and others that could potentially offer different advantages, we did not have access to these alternatives within our toolset. (we tried to run them but they too ktoo much time to be extucted and we exited . This limited our ability to employ and compare a broader range of planning strategies.

Enforced Hill Climbing and Astar for problem 1:

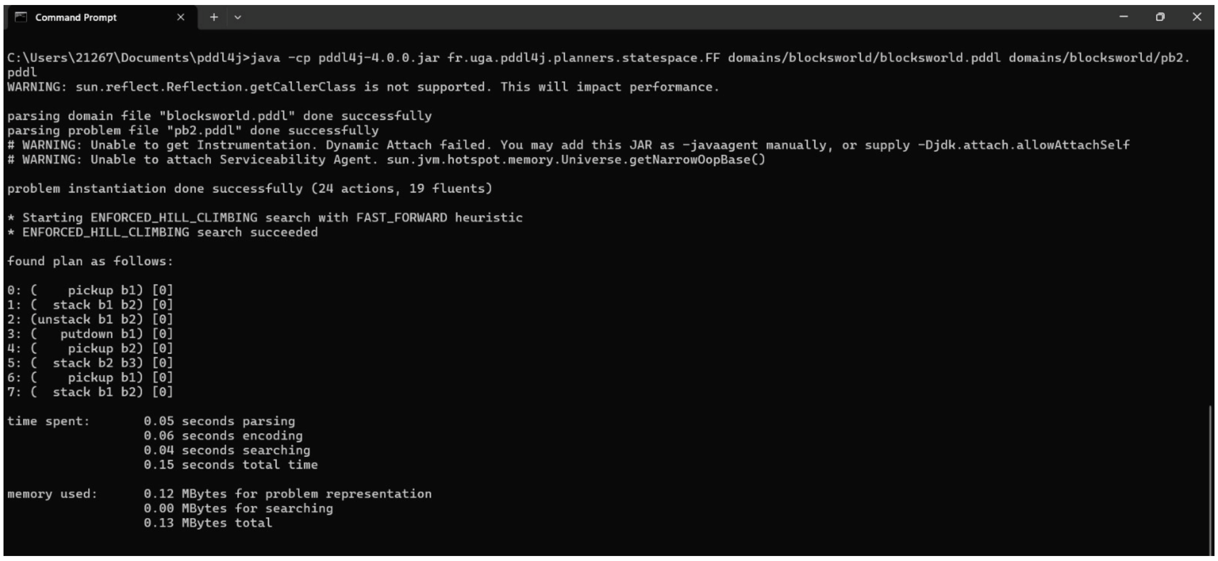
Enforced Hill Climbing and Astar for problem 2 :



```
C:\Users\21267\Documents\pddl4j>java -cp pddl4j-4.0.0.jar fr.uga.pddl4j.planners.statespace.FF domains/blocksworld/blocksworld.pddl domains/blocksworld/pb1.
pddl
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "blocksworld.pddl" done successfully
parsing problem file "pb1.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (12 actions, 11 fluents)

* Starting ENFORCED_HILL_CLIMBING search with FAST_FORWARD heuristic
* ENFORCED_HILL_CLIMBING search succeeded

found plan as follows:

0: (  pickup b2) [0]
1: (stack b2 b1) [0]

time spent:       0.04 seconds parsing
                  0.09 seconds encoding
                  0.02 seconds searching
                  0.15 seconds total time

memory used:      0.08 MBytes for problem representation
                  0.00 MBytes for searching
                  0.08 MBytes total
```



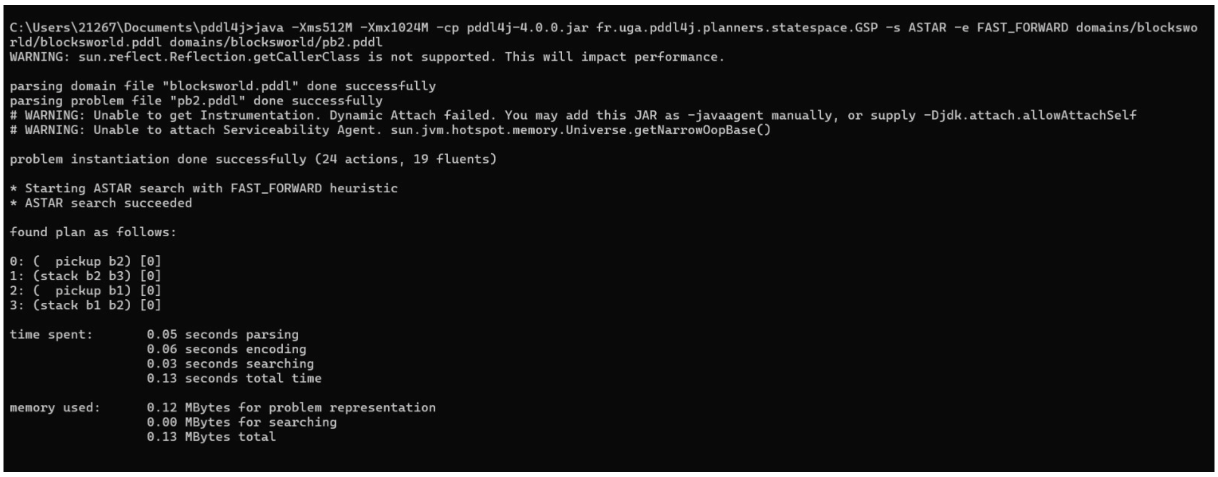
```
C:\Users\21267\Documents\pddl4j>java -Xms512M -Xmx1024M -cp pddl4j-4.0.0.jar fr.uga.pddl4j.planners.statespace.GSP -s ASTAR -e FAST_FORWARD domains/blockswo
rld/blocksworld.pddl domains/blocksworld/pb1.pddl
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "blocksworld.pddl" done successfully
parsing problem file "pb1.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (12 actions, 11 fluents)

* Starting ASTAR search with FAST_FORWARD heuristic
* ASTAR search succeeded

found plan as follows:

0: (  pickup b2) [0]
1: (stack b2 b1) [0]

time spent:       0.05 seconds parsing
                  0.05 seconds encoding
                  0.02 seconds searching
                  0.12 seconds total time

memory used:      0.08 MBytes for problem representation
                  0.00 MBytes for searching
                  0.08 MBytes total
```



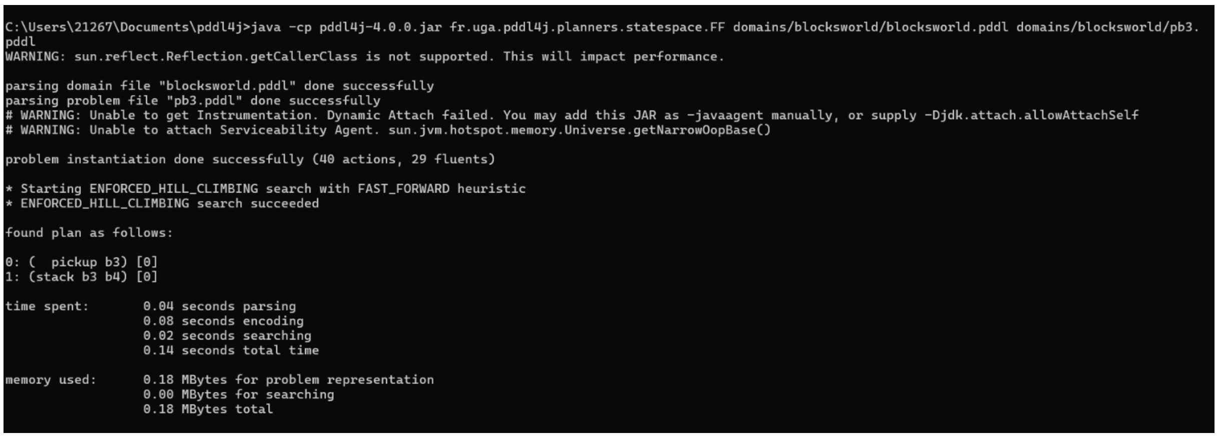
```
C:\Users\21267\Documents\pddl4j>java -cp pddl4j-4.0.0.jar fr.uga.pddl4j.planners.statespace.FF domains/blocksworld/blocksworld.pddl domains/blocksworld/pb2.
pddl
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "blocksworld.pddl" done successfully
parsing problem file "pb2.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (24 actions, 19 fluents)

* Starting ENFORCED_HILL_CLIMBING search with FAST_FORWARD heuristic
* ENFORCED_HILL_CLIMBING search succeeded

found plan as follows:

0: (    pickup b1) [0]
1: (  stack b1 b2) [0]
2: (unstack b1 b2) [0]
3: (   putdown b1) [0]
4: (    pickup b2) [0]
5: (  stack b2 b3) [0]
6: (    pickup b1) [0]
7: (  stack b1 b2) [0]

time spent:       0.05 seconds parsing
                  0.06 seconds encoding
                  0.04 seconds searching
                  0.15 seconds total time

memory used:      0.12 MBytes for problem representation
                  0.00 MBytes for searching
                  0.13 MBytes total
```

For problem 2 , ASTAR is faster than EHC and both consumed same amount of memory .

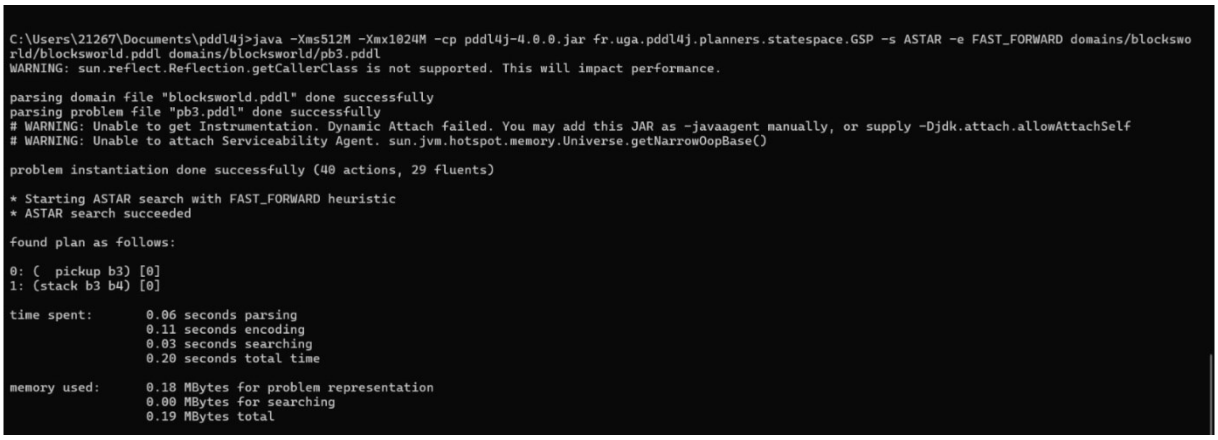
Enforced Hill Climbing and Astar for problem 3:



```
C:\Users\21267\Documents\pddl4j>java -Xms512M -Xmx1024M -cp pddl4j-4.0.0.jar fr.uga.pddl4j.planners.statespace.GSP -s ASTAR -e FAST_FORWARD domains/blockswo
rld/blocksworld.pddl domains/blocksworld/pb2.pddl
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "blocksworld.pddl" done successfully
parsing problem file "pb2.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (24 actions, 19 fluents)

* Starting ASTAR search with FAST_FORWARD heuristic
* ASTAR search succeeded

found plan as follows:

0: (  pickup b2) [0]
1: (stack b2 b3) [0]
2: (  pickup b1) [0]
3: (stack b1 b2) [0]

time spent:       0.05 seconds parsing
                  0.06 seconds encoding
                  0.03 seconds searching
                  0.13 seconds total time

memory used:      0.12 MBytes for problem representation
                  0.00 MBytes for searching
                  0.13 MBytes total
```



```
C:\Users\21267\Documents\pddl4j>java -cp pddl4j-4.0.0.jar fr.uga.pddl4j.planners.statespace.FF domains/blocksworld/blocksworld.pddl domains/blocksworld/pb3.
pddl
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "blocksworld.pddl" done successfully
parsing problem file "pb3.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (40 actions, 29 fluents)

* Starting ENFORCED_HILL_CLIMBING search with FAST_FORWARD heuristic
* ENFORCED_HILL_CLIMBING search succeeded

found plan as follows:

0: (  pickup b3) [0]
1: (stack b3 b4) [0]

time spent:       0.04 seconds parsing
                  0.08 seconds encoding
                  0.02 seconds searching
                  0.14 seconds total time

memory used:      0.18 MBytes for problem representation
                  0.00 MBytes for searching
                  0.18 MBytes total
```



```
C:\Users\21267\Documents\pddl4j>java -Xms512M -Xmx1024M -cp pddl4j-4.0.0.jar fr.uga.pddl4j.planners.statespace.GSP -s ASTAR -e FAST_FORWARD domains/blockswo
rld/blocksworld.pddl domains/blocksworld/pb3.pddl
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "blocksworld.pddl" done successfully
parsing problem file "pb3.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (40 actions, 29 fluents)

* Starting ASTAR search with FAST_FORWARD heuristic
* ASTAR search succeeded

found plan as follows:

0: (  pickup b3) [0]
1: (stack b3 b4) [0]

time spent:       0.06 seconds parsing
                  0.11 seconds encoding
                  0.03 seconds searching
                  0.20 seconds total time

memory used:      0.18 MBytes for problem representation
                  0.00 MBytes for searching
                  0.19 MBytes total
```

Interpretation and comparison of results:

For problem 1 , EHC completed the task in 0.15 seconds, while Astar was slightly faster at 0.12 seconds. Both used minimal memory, making them highly effective for simple tasks. This comparison shows that for straightforward problems, either planner can be chosen based on slight differences in performance. For problem 3 , EHC was faster than ASTAR and consumed less memory .

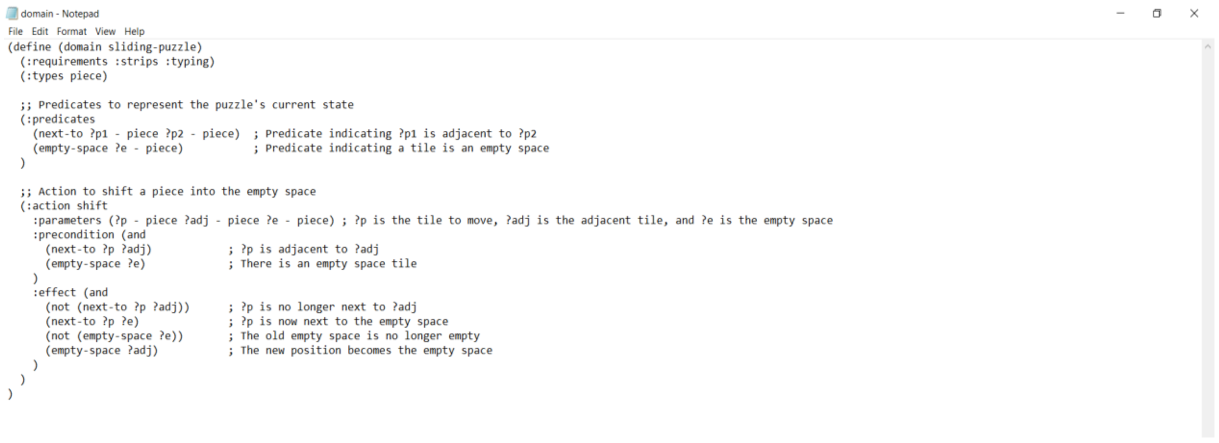
Comparison table :

Problem	Planner used	Plan to take	Time it took	Memory used	Interpretation
Problem 1	EHC	Pickup block B2, stack on block B1	0.15	0.08	Slightly slower than Astar, executed the same plan
Problem 1	ASTAR	Pickup block B2, stack on block B1	0.12	0.08	Slightly faster than EHC, identical in plan and memory efficiency

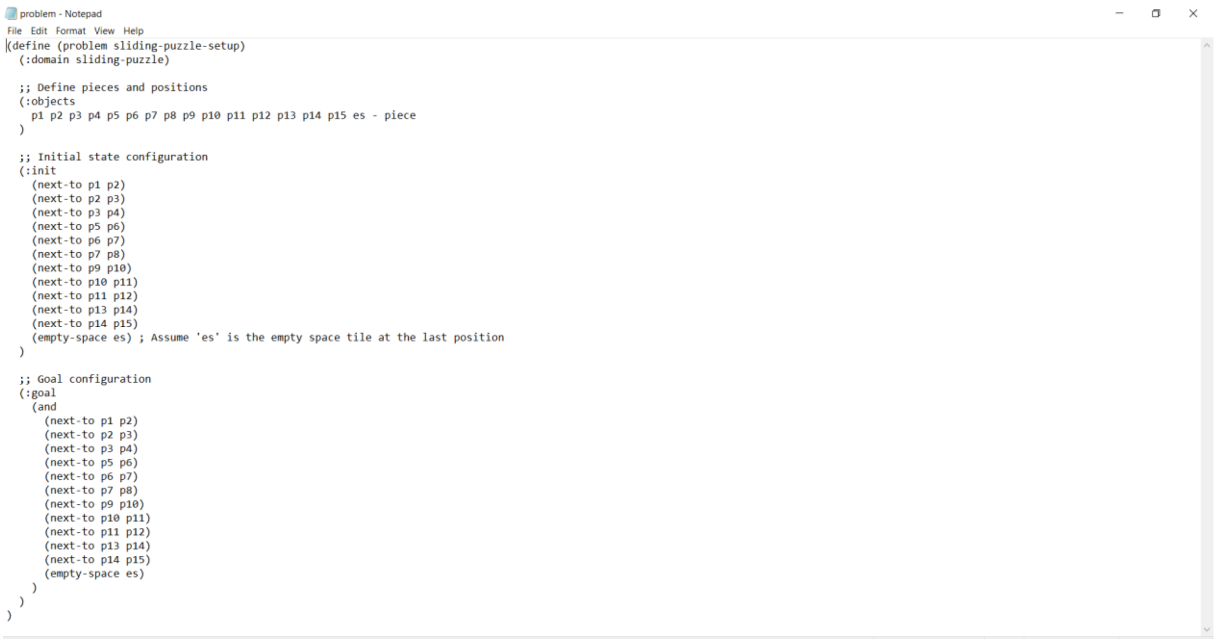
Problem 2	EHC	longer sequence of pickup, stack, and unstack actions	0.15	0.13	Executed a longer plan compared to Astar, yet managed similar memory usage
Problem 2	ASTAR	Optimized sequence with fewer actions than EHC	0.13	0.13	More efficient in terms of action sequence, slightly faster with the same memory usage as EHC
Problem 3	EHC	Pickup block B3, stack on block B4	0.14	0.18	Faster than Astar, showing slightly better memory efficiency.
Problem 3	ASTAR	Pickup block B3, stack on block B4	0.20	0.19	Slightly slower, using more memory compared to EHC, though executing the same actions.

Task 2: 4x4 puzzle

Our code snippets for domain and problem:

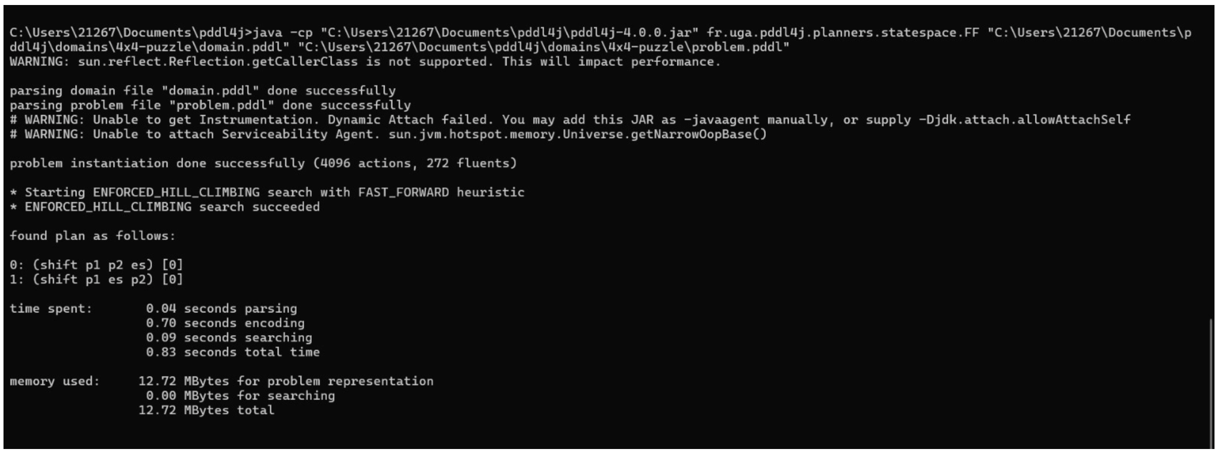


```
(define (domain sliding-puzzle)
  (:requirements :strips :typing)
  (:types piece)

  ;; Predicates to represent the puzzle's current state
  (:predicates
    (next-to ?p1 - piece ?p2 - piece)  ; Predicate indicating ?p1 is adjacent to ?p2
    (empty-space ?e - piece)           ; Predicate indicating a tile is an empty space
  )

  ;; Action to shift a piece into the empty space
  (:action shift
    :parameters (?p - piece ?adj - piece ?e - piece) ; ?p is the tile to move, ?adj is the adjacent tile, and ?e is the empty space
    :precondition (and
      (next-to ?p ?adj)            ; ?p is adjacent to ?adj
      (empty-space ?e)             ; There is an empty space tile
    )
    :effect (and
      (not (next-to ?p ?adj))      ; ?p is no longer next to ?adj
      (next-to ?p ?e)              ; ?p is now next to the empty space
      (not (empty-space ?e))       ; The old empty space is no longer empty
      (empty-space ?adj)           ; The new position becomes the empty space
    )
  )
)
```



```
(define (problem sliding-puzzle-setup)
  (:domain sliding-puzzle)

  ;; Define pieces and positions
  (:objects
    p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 es - piece
  )

  ;; Initial state configuration
  (:init
    (next-to p1 p2)
    (next-to p2 p3)
    (next-to p3 p4)
    (next-to p5 p6)
    (next-to p6 p7)
    (next-to p7 p8)
    (next-to p9 p10)
    (next-to p10 p11)
    (next-to p11 p12)
    (next-to p13 p14)
    (next-to p14 p15)
    (empty-space es) ; Assume 'es' is the empty space tile at the last position
  )

  ;; Goal configuration
  (:goal
    (and
      (next-to p1 p2)
      (next-to p2 p3)
      (next-to p3 p4)
      (next-to p5 p6)
      (next-to p6 p7)
      (next-to p7 p8)
      (next-to p9 p10)
      (next-to p10 p11)
      (next-to p11 p12)
      (next-to p13 p14)
      (next-to p14 p15)
      (empty-space es)
    )
  )
)
```

Our results:



```
C:\Users\21267\Documents\pddl4j>java -cp "C:\Users\21267\Documents\pddl4j\pddl4j-4.0.0.jar" fr.uga.pddl4j.planners.statespace.FF "C:\Users\21267\Documents\p
ddl4j\domains\4x4-puzzle\domain.pddl" "C:\Users\21267\Documents\pddl4j\domains\4x4-puzzle\problem.pddl"
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "domain.pddl" done successfully
parsing problem file "problem.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (4096 actions, 272 fluents)

* Starting ENFORCED_HILL_CLIMBING search with FAST_FORWARD heuristic
* ENFORCED_HILL_CLIMBING search succeeded

found plan as follows:

0: (shift p1 p2 es) [0]
1: (shift p1 es p2) [0]

time spent:       0.04 seconds parsing
                  0.70 seconds encoding
                  0.09 seconds searching
                  0.83 seconds total time

memory used:     12.72 MBytes for problem representation
                  0.00 MBytes for searching
                 12.72 MBytes total
```

Explanation and steps:

As we know now PDDL stands for Planning Domain Definition Language. It's a language used to tell a computer how to solve puzzles and problems by planning out steps. It's really important because it helps in designing solutions that computers can understand and carry out on their own, making it a great tool for automating complex tasks.

The guide from LearnPDDL (Kalaboud, n.d.) was instrumental in structuring predicates like empty and tile, which we used to represent the puzzle's positions. This tutorial also helped clarify how to set up initial and goal states to solve the puzzle efficiently.

Components of PDDL Used in Our Solution:

- **Predicates:** These are statements that help describe the situation, like telling if a spot is empty or if a specific tile is at a certain place.
- **Actions:** These are things that can be done to change the situation, like moving a tile from one spot to another.
- **Parameters:** These are the details of the actions, like which tile to move and where.
- **Preconditions:** These are conditions that must be true before an action can happen, like making sure there is an empty spot to move a tile into.
- **Effects:** These are the results of taking an action, like having a tile in a new spot and making the previous spot empty.

Domain and Problem File Structure in our prob 2

Domain File:

- **Predicates Detail:** We defined properties like whether a spot is empty (empty ?pos) or where a tile is located (tile ?tile ?pos).
- **Actions Explanation:** We set up actions to move tiles. For instance, to move a tile, the spot it's moving to must be empty, and the tile moves to that new spot, leaving its old spot empty.

Problem File:

- **Setting Up Initial and Goal States:** We described how the tiles are laid out at the start and what the puzzle should look like when it's solved.
- **Tile Configuration:** We explained where each tile starts and where it needs to end up, setting the stage for the puzzle-solving process.

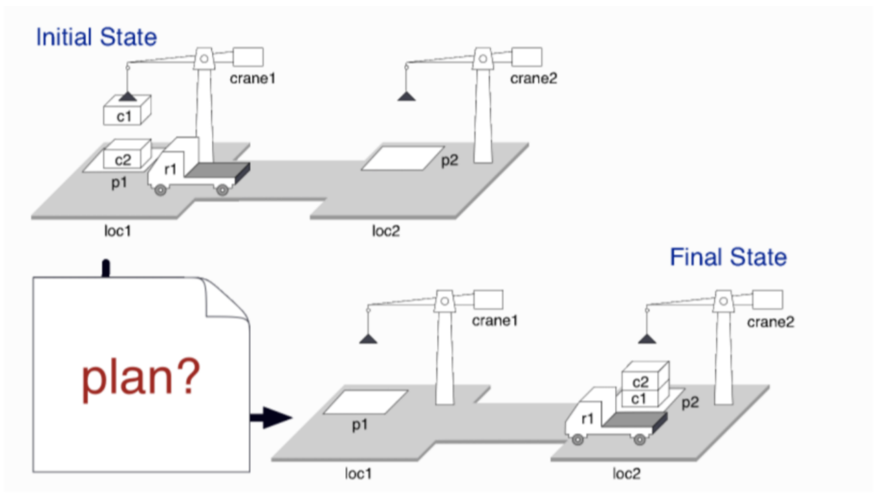
Choice of Planners:

- We used the Enforced Hill Climbing (EHC) planner with a Fast Forward (FF) heuristic. This means we chose a method that helps the planner figure out the best moves to solve the puzzle quickly, avoiding unnecessary steps.

Planner's Performance:

- **Execution Time and Memory Usage:** The planner was very quick and didn't use much computer memory, which shows our puzzle setup was efficient.
- **Contribution of PDDL Files:** The way we wrote our PDDL files made it easy for the planner to understand the puzzle and find the solution fast. Our clear definitions in the domain and problem files helped the computer figure out the best path to take without wasting time or resources.

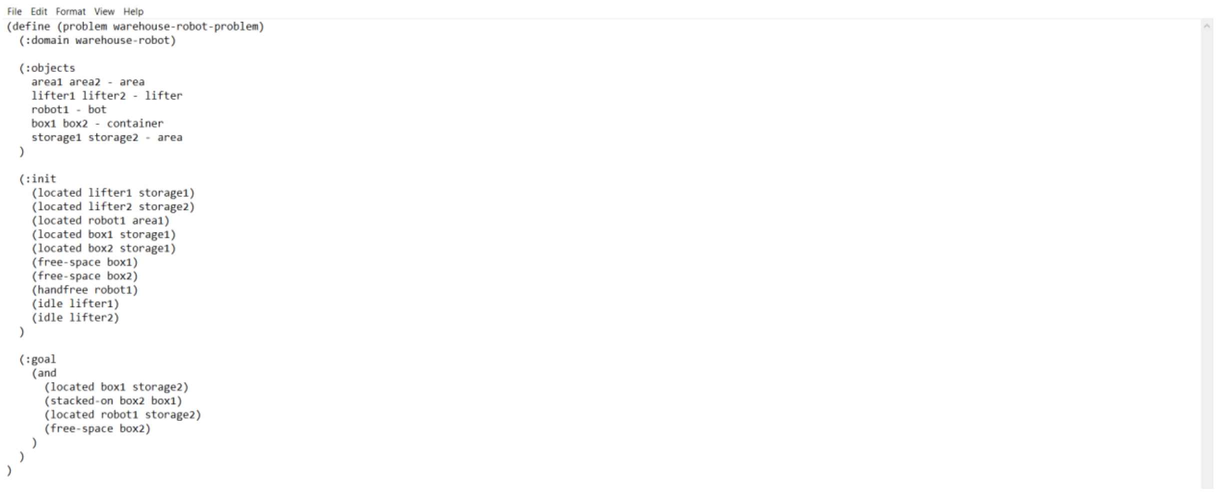
Task 3: 8 puzzles



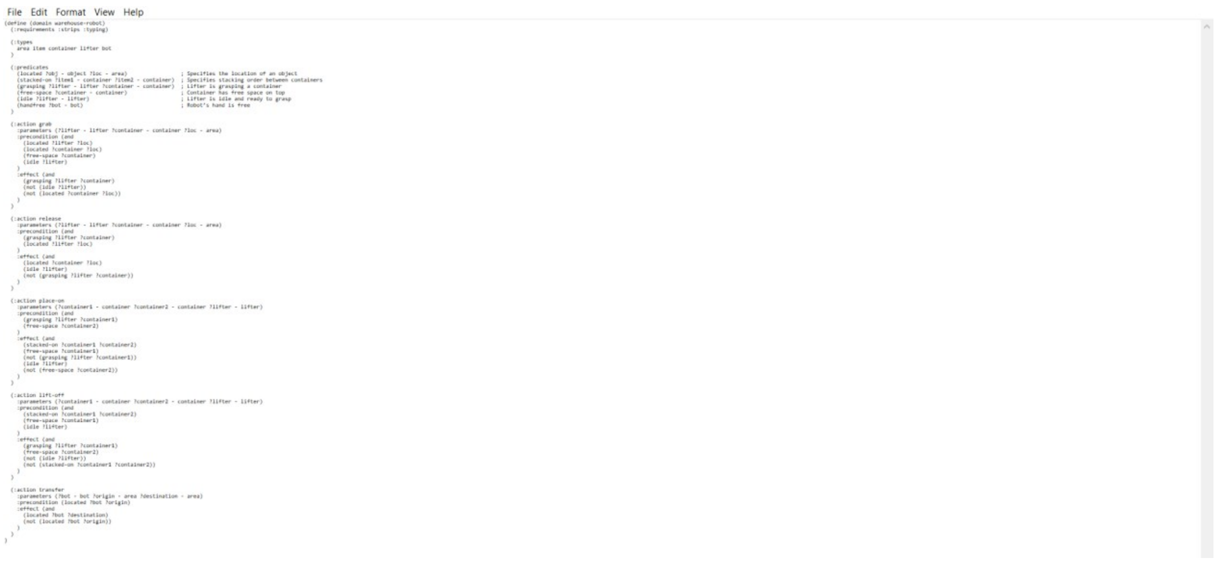
The goal is to stack container c2 on container c1 at location p2.

In Task 3, our objective was to implement a robust automated planning solution for a logistics problem involving the organization and stacking of containers using cranes at a port. The task required us to use PDDL to create domain and problem files that would efficiently solve the challenge of stacking container c2 on container c1 at a designated location using two different cranes.

The robot is responsible for transferring containers between areas, while lifters handle stacking and unstacking containers.



```
(define (problem warehouse-robot-problem)
  (:domain warehouse-robot)

  (:objects
    area1 area2 - area
    lifter1 lifter2 - lifter
    robot1 - bot
    box1 box2 - container
    storage1 storage2 - area
  )

  (:init
    (located lifter1 storage1)
    (located lifter2 storage2)
    (located robot1 area1)
    (located box1 storage1)
    (located box2 storage1)
    (free-space box1)
    (free-space box2)
    (handfree robot1)
    (idle lifter1)
    (idle lifter2)
  )

  (:goal
    (and
      (located box1 storage2)
      (stacked-on box2 box1)
      (located robot1 storage2)
      (free-space box2)
    )
  )
)
```



```
(define (domain warehouse-robot)
  (:requirements :strips :typing)

  (:types
    area item container lifter bot
  )

  (:predicates
    (located ?obj - object ?loc - area)                    ; Specifies the location of an object
    (stacked-on ?item1 - container ?item2 - container)     ; Specifies stacking order between containers
    (grasping ?lifter - lifter ?container - container)     ; Lifter is grasping a container
    (free-space ?container - container)                    ; Container has free space on top
    (idle ?lifter - lifter)                                ; Lifter is idle and ready to grasp
    (handfree ?bot - bot)                                  ; Robot's hand is free
  )

  (:action grab
    :parameters (?lifter - lifter ?container - container ?loc - area)
    :precondition (and
      (located ?lifter ?loc)
      (located ?container ?loc)
      (free-space ?container)
      (idle ?lifter)
    )
    :effect (and
      (grasping ?lifter ?container)
      (not (idle ?lifter))
      (not (located ?container ?loc))
    )
  )

  (:action release
    :parameters (?lifter - lifter ?container - container ?loc - area)
    :precondition (and
      (grasping ?lifter ?container)
      (located ?lifter ?loc)
    )
    :effect (and
      (located ?container ?loc)
      (idle ?lifter)
      (not (grasping ?lifter ?container))
    )
  )

  (:action place-on
    :parameters (?container1 - container ?container2 - container ?lifter - lifter)
    :precondition (and
      (grasping ?lifter ?container1)
      (free-space ?container2)
    )
    :effect (and
      (stacked-on ?container1 ?container2)
      (free-space ?container1)
      (not (grasping ?lifter ?container1))
      (idle ?lifter)
      (not (free-space ?container2))
    )
  )

  (:action lift-off
    :parameters (?container1 - container ?container2 - container ?lifter - lifter)
    :precondition (and
      (stacked-on ?container1 ?container2)
      (free-space ?container1)
      (idle ?lifter)
    )
    :effect (and
      (grasping ?lifter ?container1)
      (free-space ?container2)
      (not (idle ?lifter))
      (not (stacked-on ?container1 ?container2))
    )
  )

  (:action transfer
    :parameters (?bot - bot ?origin - area ?destination - area)
    :precondition (located ?bot ?origin)
    :effect (and
      (located ?bot ?destination)
      (not (located ?bot ?origin))
    )
  )
)
```

Domain File Explanation

- **Types and Objects**:
    - We have different types like area, item, container, lifter, and bot.
    - Each type represents a specific role in the environment, e.g., container for boxes, lifter for tools to pick up boxes, and bot for the robot.
- **Predicates**:
    - **located**: Shows where each object (like the robot or boxes) is.
    - **stacked-on**: Indicates if one box is stacked on top of another.
    - **grasping**: Means a lifter is holding a box.
    - **free-space**: Marks a box that has space available for stacking.
    - **handfree** and **idle**: Signify that the robot or lifter is not holding anything.
- **Actions**:
    - **Grab**: Allows a lifter to pick up a box if they are in the same location and the box is free to be picked.
    - **Release**: Lets a lifter put down a box they are holding, making the lifter idle again.
    - **Place-on**: Stacks one box on top of another, if both boxes are available and positioned correctly.
    - **Transfer**: Moves the robot from one area to another.

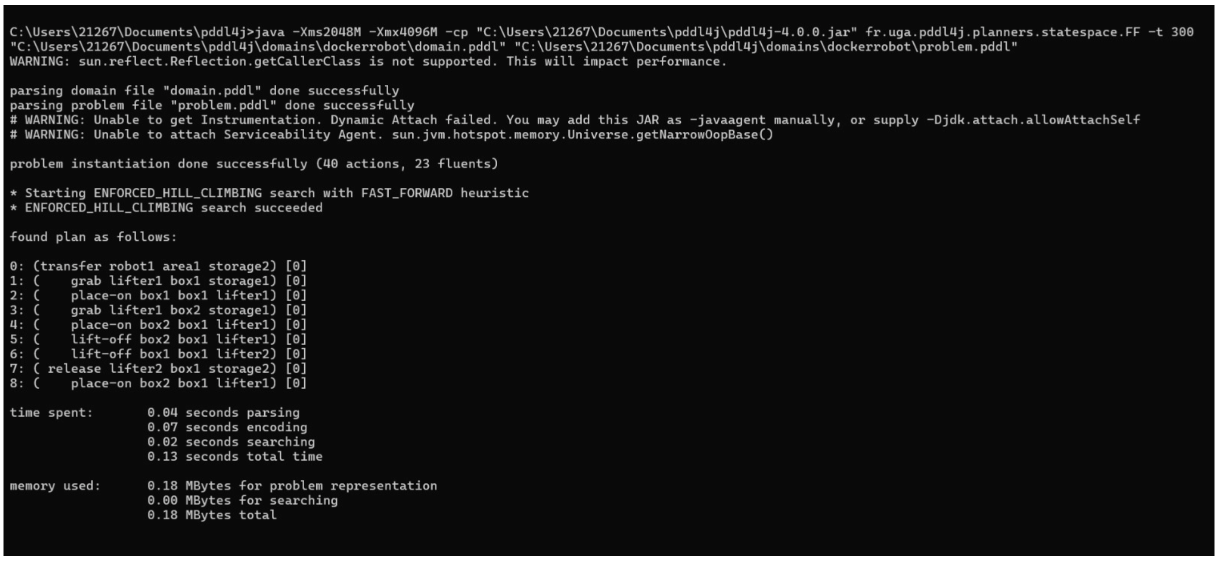
Problem File Explanation

- **Purpose**: The problem file sets up the starting positions and goal for the warehouse task. This includes where the robot, lifters, and boxes begin, as well as where they should end up.
- **Initial State**:
    - We specify the starting locations of each lifter, robot, and box.
    - Conditions like which boxes are free or held by a lift are also defined.
- **Goal State**:
    - The goal is to place each box in its correct final position.

- For example, in this task, box1 needs to be at storage2, box2 should be stacked on box1, and the robot should also be in storage2.

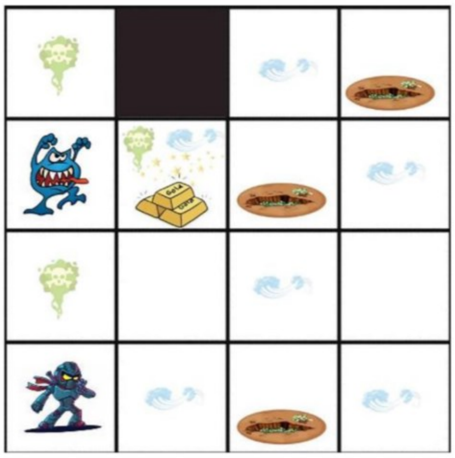
Execution and Interpretation

After defining the domain and problem files, we executed the plan. The actions in the result show how the robot and lifters achieved the goal efficiently by following the defined steps like moving, grabbing, stacking, and releasing items.



```
C:\Users\21267\Documents\pddl4j>java -Xms2048M -Xmx4096M -cp "C:\Users\21267\Documents\pddl4j\pddl4j-4.0.0.jar" fr.uga.pddl4j.planners.statespace.FF -t 300
"C:\Users\21267\Documents\pddl4j\domains\dockerrobot\domain.pddl" "C:\Users\21267\Documents\pddl4j\domains\dockerrobot\problem.pddl"
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "domain.pddl" done successfully
parsing problem file "problem.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (40 actions, 23 fluents)

* Starting ENFORCED_HILL_CLIMBING search with FAST_FORWARD heuristic
* ENFORCED_HILL_CLIMBING search succeeded

found plan as follows:

0: (transfer robot1 area1 storage2) [0]
1: (    grab lifter1 box1 storage1) [0]
2: (    place-on box1 box1 lifter1) [0]
3: (    grab lifter1 box2 storage1) [0]
4: (    place-on box2 box1 lifter1) [0]
5: (    lift-off box2 box1 lifter1) [0]
6: (    lift-off box1 box1 lifter2) [0]
7: ( release lifter2 box1 storage2) [0]
8: (    place-on box2 box1 lifter1) [0]

time spent:       0.04 seconds parsing
                  0.07 seconds encoding
                  0.02 seconds searching
                  0.13 seconds total time

memory used:      0.18 MBytes for problem representation
                  0.00 MBytes for searching
                  0.18 MBytes total
```

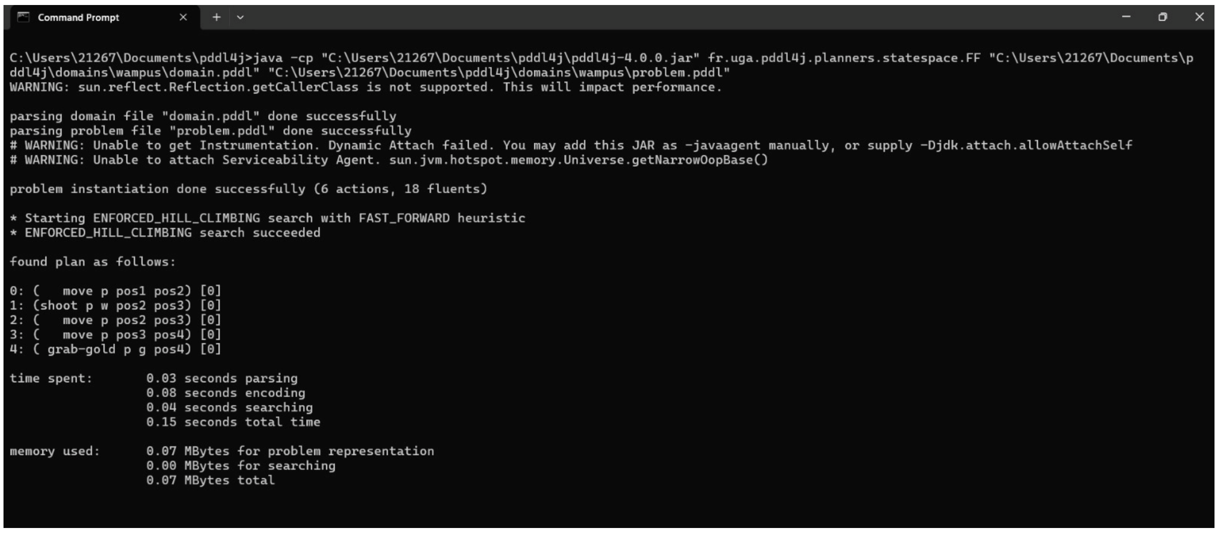
Task 4: The Wumpus word (bonus)



This game is about helping a character navigate through a cave to collect gold while avoiding dangerous traps and a monster called the Wumpus.

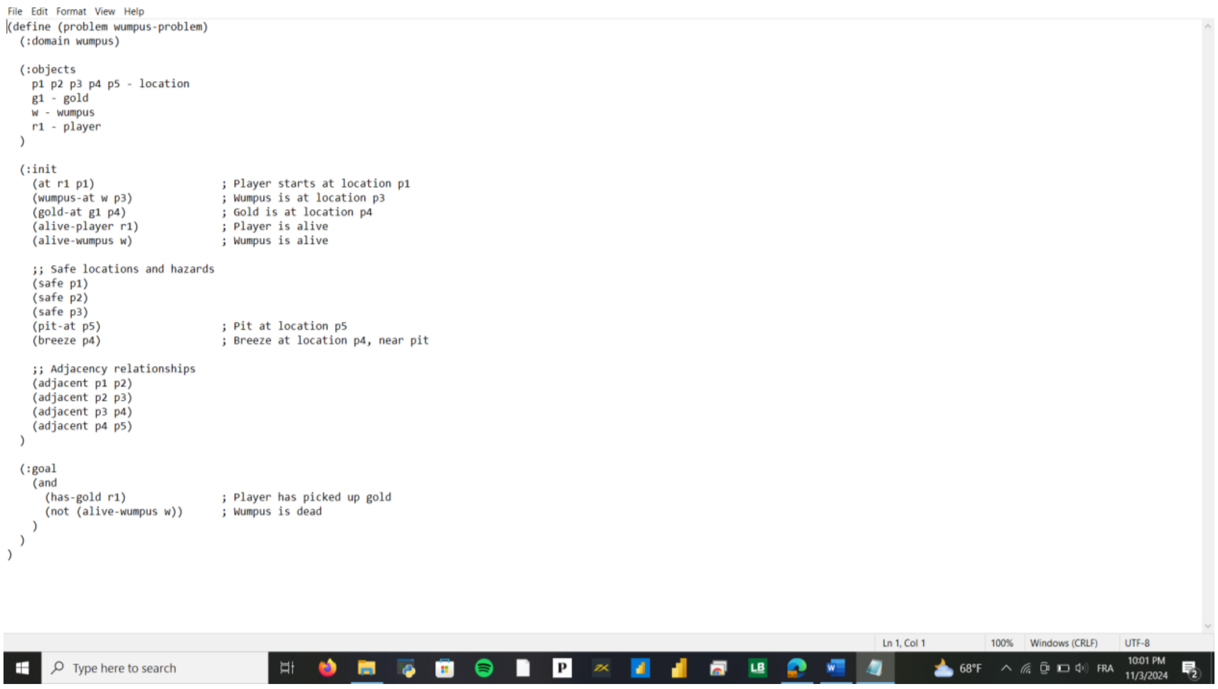
We consulted *'PDDL Tooling - Episode 4'* for additional guidance on structuring our PDDL code to handle complex logic in the Wumpus game.

Our results:

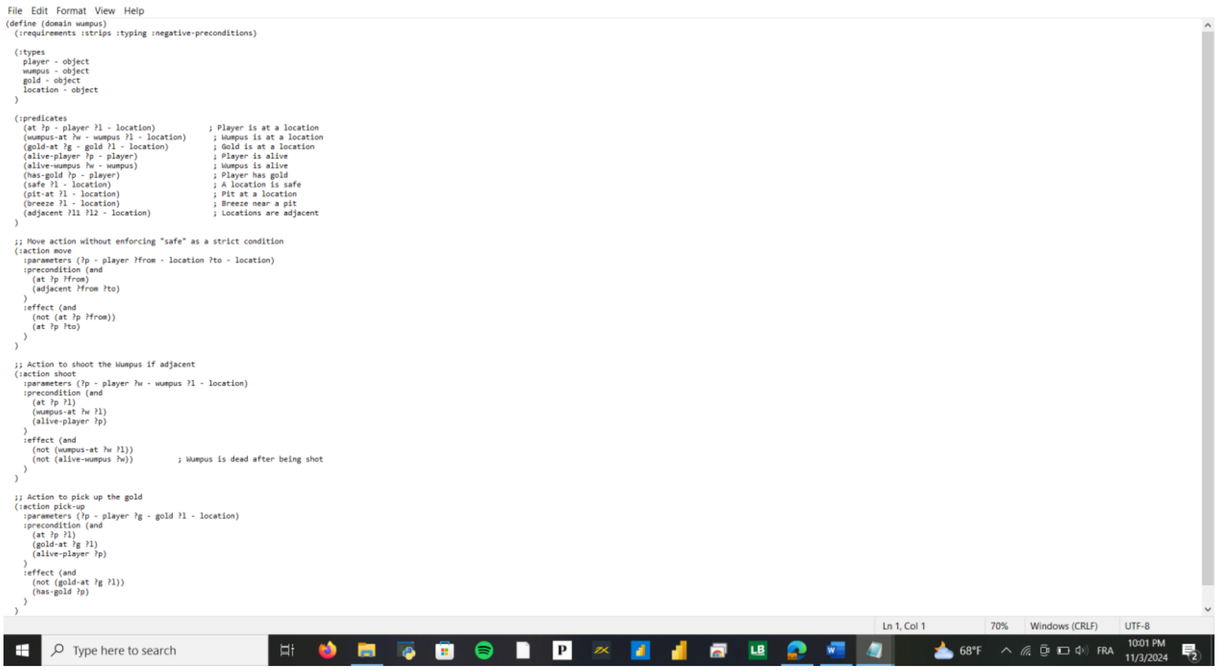


```
C:\Users\21267\Documents\pddl4j>java -cp "C:\Users\21267\Documents\pddl4j\pddl4j-4.0.0.jar" fr.uga.pddl4j.planners.statespace.FF "C:\Users\21267\Documents\p
ddl4j\domains\wampus\domain.pddl" "C:\Users\21267\Documents\pddl4j\domains\wampus\problem.pddl"
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.

parsing domain file "domain.pddl" done successfully
parsing problem file "problem.pddl" done successfully
# WARNING: Unable to get Instrumentation. Dynamic Attach failed. You may add this JAR as -javaagent manually, or supply -Djdk.attach.allowAttachSelf
# WARNING: Unable to attach Serviceability Agent. sun.jvm.hotspot.memory.Universe.getNarrowOopBase()

problem instantiation done successfully (6 actions, 18 fluents)

* Starting ENFORCED_HILL_CLIMBING search with FAST_FORWARD heuristic
* ENFORCED_HILL_CLIMBING search succeeded

found plan as follows:

0: (   move p pos1 pos2) [0]
1: (shoot p w pos2 pos3) [0]
2: (   move p pos2 pos3) [0]
3: (   move p pos3 pos4) [0]
4: ( grab-gold p g pos4) [0]

time spent:       0.03 seconds parsing
                  0.08 seconds encoding
                  0.04 seconds searching
                  0.15 seconds total time

memory used:      0.07 MBytes for problem representation
                  0.00 MBytes for searching
                  0.07 MBytes total
```

Our code:



```
(define (problem wumpus-problem)
  (:domain wumpus)

  (:objects
    p1 p2 p3 p4 p5 - location
    g1 - gold
    w - wumpus
    r1 - player
  )

  (:init
    (at r1 p1)                    ; Player starts at location p1
    (wumpus-at w p3)              ; Wumpus is at location p3
    (gold-at g1 p4)               ; Gold is at location p4
    (alive-player r1)             ; Player is alive
    (alive-wumpus w)              ; Wumpus is alive

    ;; Safe locations and hazards
    (safe p1)
    (safe p2)
    (safe p3)
    (pit-at p5)                   ; Pit at location p5
    (breeze p4)                   ; Breeze at location p4, near pit

    ;; Adjacency relationships
    (adjacent p1 p2)
    (adjacent p2 p3)
    (adjacent p3 p4)
    (adjacent p4 p5)
  )

  (:goal
    (and
      (has-gold r1)               ; Player has picked up gold
      (not (alive-wumpus w))      ; Wumpus is dead
    )
  )
)
```



```
(define (domain wumpus)
  (:requirements :strips :typing :negative-preconditions)

  (:types
    player - object
    wumpus - object
    gold - object
    location - object
  )

  (:predicates
    (at ?p - player ?l - location)            ; Player is at a location
    (wumpus-at ?w - wumpus ?l - location)      ; Wumpus is at a location
    (gold-at ?g - gold ?l - location)          ; Gold is at a location
    (alive-player ?p - player)                 ; Player is alive
    (alive-wumpus ?w - wumpus)                 ; Wumpus is alive
    (has-gold ?p - player)                     ; Player has gold
    (safe ?l - location)                       ; A location is safe
    (pit-at ?l - location)                     ; Pit at a location
    (breeze ?l - location)                     ; Breeze near a pit
    (adjacent ?l1 ?l2 - location)              ; Locations are adjacent
  )

  ;; Move action without enforcing "safe" as a strict condition
  (:action move
    :parameters (?p - player ?from - location ?to - location)
    :precondition (and
      (at ?p ?from)
      (adjacent ?from ?to)
    )
    :effect (and
      (not (at ?p ?from))
      (at ?p ?to)
    )
  )

  ;; Action to shoot the Wumpus if adjacent
  (:action shoot
    :parameters (?p - player ?w - wumpus ?l - location)
    :precondition (and
      (at ?p ?l)
      (wumpus-at ?w ?l)
      (alive-player ?p)
    )
    :effect (and
      (not (wumpus-at ?w ?l))
      (not (alive-wumpus ?w))          ; Wumpus is dead after being shot
    )
  )

  ;; Action to pick up the gold
  (:action pick-up
    :parameters (?p - player ?g - gold ?l - location)
    :precondition (and
      (at ?p ?l)
      (gold-at ?g ?l)
      (alive-player ?p)
    )
    :effect (and
      (not (gold-at ?g ?l))
      (has-gold ?p)
    )
  )
```

The results show the successful execution of a sequence of actions using the Enforced Hill Climbing (EHC) search with the Fast Forward (FF) heuristic.

The planner generated a plan that solved the problem as intended, with the following sequence:

1. **Move (p pos1 pos2):** The player moved from position pos1 to pos2.
2. **Shoot (p pos2 pos3):** At position pos2, the player used the "shoot" action to kill the Wumpus in pos3, neutralizing the threat.
3. **Move (p pos2 pos3):** The player moved into position pos3 (now safe after the Wumpus was eliminated).
4. **Move (p pos3 pos4):** The player continued moving to pos4, where the gold is located.
5. **Grab-Gold (p g pos4):** Finally, the player picked up the gold at position pos4.

Domain File

1. **Requirements**:
   - (requirements :strips :typing :negative-preconditions): Specifies the features we are using in PDDL. Here, :strips allows for standard action definitions, :typing helps us categorize objects, and :negative-preconditions lets us define actions that depend on something not being true (e.g., the Wumpus is not alive).
2. **Types**:
   - (types player object wumpus gold location): Defines different types of objects in the game, such as the player, Wumpus, gold, and location. This helps structure our domain by categorizing each object.
3. **Predicates**:

   - Predicates represent facts or conditions that describe the state of the game. They include:
      - (at ?p - player ?l - location): Indicates the player’s location.
      - (wumpus-at ?w - wumpus ?l - location): Shows if the Wumpus is at a specific location.
      - (gold-at ?g - gold ?l - location): Represents the location of the gold.
      - (alive-player ?p - player): Indicates if the player is alive.
      - (alive-wumpus ?w - wumpus): Shows if the Wumpus is alive.
      - (has-gold ?p - player ?g - gold): Specifies if the player has picked up the gold.
      - Other predicates, like (pit-at ?l - location) and (safe ?l - location), describe the presence of hazards and safe locations, helping to determine safe paths.

4. **Actions**:
   - Actions define what the player can do and under what conditions. Here’s a summary of each action:
      - **Move**: Allows the player to move from one location to another if they are adjacent. Preconditions ensure the player is at the starting location and that the destination is adjacent. After moving, the player is no longer at the original location but is now at the new one.

        - **Shoot**: Lets the player shoot the Wumpus if they are adjacent. The precondition is that the player is in a location adjacent to the Wumpus. The effect is that the Wumpus is marked as not alive if shot.
        - **Pick Up**: Allows the player to grab the gold if they are at the gold's location. The precondition is that the player and gold are at the same spot. The effect removes the gold from that location and updates the player's status to "has gold."

Problem File

1. **Objects**:
    - (define (problem wumpus-problem) (:domain wumpus)): Sets up the problem by associating it with the Wumpus domain.
    - (objects p1 p2 p3 p4 p5 - location g1 - gold w - wumpus r1 - player): Lists all objects involved in the game. Here, p1 to p5 are locations in the grid, g1 is the gold, w is the Wumpus, and r1 is the player.
2. **Initial State**:
    - The :init section describes the initial setup of the game:
        - (at r1 p1): The player starts at location p1.
        - (wumpus-at w p3): The Wumpus is initially positioned at p3.
        - (gold-at g1 p4): The gold is located at p4.

        - (alive-player r1) and (alive-wumpus w): Both the player and the Wumpus are alive at the beginning.
        - Other details like (safe p1), (safe p2), and (pit-at p5) define which locations are safe and where hazards like pits exist. The (breeze p4) indicates a breeze near the pit, giving the player a clue about the hazard.

3. **Goal State**:
    - The :goal section defines what needs to be true to solve the problem:
        - (has-gold r1 g1): The player must have the gold.
        - (not (alive-wumpus w)): The Wumpus must be dead.

Challenges and critical thinking

We faced some fun and tricky challenges while working on this project. First, understanding and writing the PDDL code took time because we had to define everything in a very precise way. We had to make sure each action—like moving a robot or picking up gold—had clear rules about what needs to happen before and after. This was like giving detailed instructions to a robot or character to make sure they didn't get lost or do something silly!

Picking the right planner, Enforced Hill Climbing (EHC) or A*, was also a bit of a puzzle. A* was supposed to be efficient, but sometimes it just couldn't find a solution and would say "plan not found." This happened when the setup was complicated, making it hard for A* to see a clear path. We had to adjust the way we wrote the PDDL files to make it easier for the planner to work through the problem.

Finally, debugging was like solving a mystery. If there was one tiny mistake in the code, the whole thing could go wrong. It was like following a recipe where every ingredient has to be exactly right, or the result wouldn't taste right! Figuring out these little bugs was challenging but also felt rewarding when everything finally worked smoothly.

In the end, these challenges helped us learn a lot about planning, patience, and paying attention to details. And while it was sometimes tricky, it was also pretty fun to see everything come together!

We, Meriem Lmoubariki and Mohamed Adam Sterheltou, shared the work equally by dividing the tasks fairly. We split up the project so that Meriem handled the setup and focused on the Blocks World and Robot Docker tasks, while Mohamed worked on the 4x4 Puzzle and Wumpus World tasks. At the end, we switched tasks to see if we could do a great job by trying both. We tested and compared the planners together, discussed the results, and wrote the report as a team. We also reviewed each other's work to make sure everything was done well. This way, we both contributed equally to every part of the project.

Bonus question: Heuristic

The default heuristic used in the Fast-Forward (FF) planner, implemented in the FastForward class within fr.uga.pddl4j.heuristics.state, is the **FF heuristic**, also known as the **Relaxed Planning Graph (RPG) heuristic**. This heuristic works by generating a relaxed plan that ignores negative effects (delete effects) of actions, simplifying the problem and helping to estimate the cost to reach the goal.

1. **Relaxed Planning**: The heuristic calculates a relaxed solution by treating actions as if they do not undo or delete any previously achieved goals, which simplifies the planning problem significantly and provides an optimistic cost estimate to reach the goal (Hoffmann & Nebel, 2001).

2. **Goal Cost Estimation**: It estimates the distance to the goal by counting the actions required to achieve each sub-goal in this relaxed setting, which provides a lower bound or optimistic (underestimated) cost (University of Basel, 2023).

In **PDDL4J's** GSP (Greedy Search Planner), this heuristic is applied by default if the heuristic type FAST_FORWARD is specified (-e option), using the relaxed planning graph approach. The FF heuristic's approach to ignoring delete effects makes it very effective in domains where these effects make the problem harder to solve, as it provides a fast and approximate solution path (PDDL4J API Documentation, n.d.).

Additionally, we consulted various resources to understand the FF heuristic's practical applications and implementation, including OpenAI's ChatGPT for explanations and clarifications on specific aspects of FF and RPG heuristics (OpenAI, 2024).

Another interpretation :

The FF heuristic, or Fast-Forward heuristic, used by default in the FF planner, simplifies the problem by ignoring delete effects, which are the parts of an action that "undo" or "remove" certain facts in the world state. This simplification allows the planner to estimate the cost of reaching a goal without worrying about some of the complexities in real-world actions. Here's a step-by-step breakdown:

Relaxed Planning Graph (RPG): The FF heuristic builds a relaxed planning graph, which is a series of layers or levels representing possible states and actions over time.

Each layer includes actions that can be taken in the current state to achieve new facts, and then it moves to the next layer, progressively adding more possible actions and facts.

Ignoring Delete Effects: In the real world, actions often have both positive effects (adding new facts) and negative effects (removing facts). For example, "pick up a block" might add "holding block" but remove "block on table." The FF heuristic ignores these negative effects or delete effects, meaning it assumes that once a fact (like "holding block") is achieved, it remains true forever in the relaxed plan. This makes it easier to achieve goals without "losing progress."

Cost Estimation:

The FF heuristic counts the minimum number of actions required to achieve each sub-goal in this simplified, relaxed plan. This count represents an underestimated cost of reaching the goal, as it doesn't account for any backtracking or actions that might undo previous progress in a real scenario.

Because it's optimistic (it assumes everything stays achieved), the FF heuristic provides a quick, approximate estimate of how "close" we are to the goal.

Goal Achievement through Enforced Hill Climbing: In FF planning, a common search method paired with the FF heuristic is Enforced Hill Climbing. This method tries to continually improve the state by only choosing actions that appear to bring the planner closer to the goal according to the heuristic. If a state improves the heuristic score (i.e., lowers the estimated cost to the goal), it's selected; otherwise, it backtracks or chooses an alternative path, but only when no further improvement can be made.

Application in Greedy Search:

In the PDDL4J library, with the GSP (Greedy Search Planner), the FF heuristic is used to guide the search towards the goal state by choosing actions that minimize this relaxed plan cost.

The heuristic is fast and efficient, making it ideal for large problems where calculating an exact cost would be too slow.

References

- *PDDL Tooling - Episode 4: Working with plans* :

  https://www.youtube.com/watch?v=XW0z8Oik6G8

- olejsi, J. (n.d.). *PDDL Reference Guide*. GitHub. Retrieved from https://github.com/jan-dolejsi/pddl-reference
- Kalaboud, F. (n.d.). *LearnPDDL: A Beginner's Guide to Planning Domain Definition Language*. Retrieved from https://fareskalaboud.github.io/LearnPDDL/
- University of Toronto. (n.d.). *An Introduction to PDDL*. Retrieved from https://www.cs.toronto.edu/~sheila/2542/w09/A1/introtopddl2.pdf
- Modeling in PDDL - Episode 1: Blocksworld. (n.d.). *YouTube*. Retrieved from https://www.youtube.com/watch?v=_NOVa4i7Us8
- Laborie, P. (n.d.). *PDDL4J: A Java Library for the Planning Domain Definition Language (PDDL)*. Retrieved from http://pddl4j.imag.fr/index.html
- Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research, 14*, 253-302. Retrieved from https://www.jair.org/index.php/jair/article/view/10302
- PDDL4J API Documentation. (n.d.). *Heuristic Class - Fast Forward (FF) heuristic*. Retrieved from

  http://pddl4j.imag.fr/repository/pddl4j/api/4.0.0/fr/uga/pddl4j/heuristics/Heuristic.html
- University of Basel. (2023). *Planning Heuristics* [PDF handout]. Retrieved from https://ai.dmi.unibas.ch/_files/teaching/hs23/po/slides/po-d08-handout4.pdf
- OpenAI. (2024). *ChatGPT (November 3, 2024), response generated on Fast Forward heuristic explanation and best heuristic* . Retrieved from OpenAI's ChatGPT platform.