Automated Planning Project
Emergency Autonomous Evacuation System

Halim Djerroud

revision: 1.0

Project Context

An office building is on fire and autonomous robots must evacuate all people present to the emergency exit. The system must handle several realistic constraints:

- **Fire spread**: Some rooms become smoky and dangerous
- **Limited capacity**: Robots can only carry one person at a time
- **Limited resources**: Limited number of available robots
- **Critical time**: Evacuation must be fast and efficient
- **Safety**: Avoid dangerous zones

Educational Objectives

This project will allow you to:

1. Analyze a complex real-world planning problem
2. Choose and justify a modeling approach (classical PDDL, HTN, temporal PDDL)
3. Implement a complete system: model + simulation + visualization
4. Evaluate and critique your solution's performance
5. Work in pairs on a large-scale project

Detailed Problem Description

Building Structure

The building has 3 floors with several rooms:

```
Floor 2:  [Office-A] -- [Office-B] -- [Corridor-2] -- [Office-C]
              |                             |
Floor 1:  [Room-1] -- [Room-2] ---- [Corridor-1] -- [Room-3]
              |                             |
Ground:   [Hall] ---------------------- [Reception] -- [EXIT]
              |                             |
          [Stairs] ------------------ [Stairs]
```

Important characteristics:

- Stairs connect all floors
- Hall and reception lead to emergency exit
- Corridors connect multiple rooms
- Some doors may be closed

Entities and Constraints

Robots

- **Number**: 2 available robots
- **Capacity**: 1 person at a time (or empty)
- **Actions**: move, pick up a person, drop off a person
- **Constraints**: Cannot cross smoky rooms while carrying someone

People to Evacuate

- **Number**: 10 people distributed in rooms
- **Initial state**:
  - 3 people in Office-A (Floor 2)
  - 2 people in Office-C (Floor 2)
  - 2 people in Room-1 (Floor 1)
  - 2 people in Room-3 (Floor 1)
  - 1 person in Hall (Ground floor)

Smoky Zones (Danger)

- Office-B (Floor 2) - Smoky from the start
- Room-2 (Floor 1) - Smoky from the start
- Robots can cross these zones **only if they are empty** (without a person)
- We assume fire does not spread (simplification)

Final Objective

Goal: All 10 people must be evacuated (predicate `evacuated` or present at the `exit`)
Optimization metric (bonus):

- Minimize total number of actions
- Or minimize evacuation time (if temporal PDDL)
- Or maximize safety (avoid smoky zones as much as possible)

Required Work

The project takes place over **4 hours** and is divided into 4 phases:

Phase 1: Analysis and Modeling Choice

Task 1.1: Problem Analysis

Write a structured analysis (2 pages max) including:

1. **Entity identification**:
   - What are the object types?
   - What relationships exist between them?
   - What are the possible states of each entity?
2. **Action identification**:
   - What actions can robots perform?
   - What are the preconditions of each action?
   - What are the effects of each action?
3. **Constraints and challenges**:
   - What are the safety constraints?
   - What are the resource limitations?
   - What are potential blocking points?

Task 1.2: Planning Approach Choice

Compare at least 2 approaches among:

- **Classical PDDL** (STRIPS)
- **Hierarchical Planning** (HTN with PyHOP)
- **Temporal PDDL** (durative-actions) (not covered in class, optional)
- **Probabilistic PDDL** (PPDDL)

For each approach, discuss:

- **Advantages** for this specific problem
- **Disadvantages** and limitations
- **Expressiveness**: can it model all constraints?
- **Complexity**: implementation difficulty

Conclusion: Justify your final choice of approach (you can also propose a hybrid approach).

Tip: There is no single "correct" answer. What matters is justifying your choice with solid technical arguments.

Phase 2: Modeling (2h sessions)

Task 2.1: Domain Modeling

Create modeling files according to your chosen approach:

If Classical PDDL:

```
(define (domain evacuation)
  (:requirements :strips :typing)

  (:types
    robot person room floor - object
    ;; TODO: Add other types if necessary
  )

  (:predicates
    (robot-in ?r - robot ?s - room)
    (person-in ?p - person ?s - room)
    ;; TODO: Add other predicates (smoky, door-between, etc.)
  )

  (:action move-robot
    :parameters (?r - robot ?from ?to - room)
    :precondition (and
      ;; TODO: Define preconditions
    )
    :effect (and
      ;; TODO: Define effects
    )
  )

  ;; TODO: Define other actions (pick-up, drop-off, etc.)
)
```

If HTN (PyHOP):

```
# Define operators (primitive actions)
def move_robot(state, robot, from_loc, to_loc):
    # TODO: Implement logic
    pass

# Define decomposition methods
def evacuate_person(state, person):
    # TODO: Decompose into subtasks
    # Example: find_robot -> go_to_person -> pick_up ->
    #           go_to_exit -> drop_off
    pass
```

Task 2.2: Problem Modeling

Create a problem instance with initial state and goal:

```
(define (problem emergency-evacuation)
  (:domain evacuation)

  (:objects
    robot1 robot2 - robot
    p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 - person
    office-a office-b office-c corridor-2 - room
    room-1 room-2 room-3 corridor-1 - room
    hall reception exit stairs - room
    ;; TODO: Complete
  )

  (:init
    ;; Initial robot positions
    (robot-in robot1 hall)
    (robot-in robot2 reception)

    ;; Initial person positions
    (person-in p1 office-a)
    ;; TODO: Complete for all 10 people
```

```
    ;; Smoky zones
    (smoky office-b)
    (smoky room-2)

    ;; Connections between rooms
    ;; TODO: Define all doors/passages
  )

  (:goal (and
    (evacuated p1)
    (evacuated p2)
    ;; TODO: All people evacuated
  ))
)
```

Task 2.3: Testing and Validation

1. **Syntax test**: Verify that your PDDL files are valid
2. **Test with planner**:
   - If PDDL: Use Fast Downward or another planner
   - If HTN: Run PyHOP on your domain
3. **Plan validation**:
   - Does the plan reach the goal?
   - Does it respect all constraints?
   - Is it realistic?

```
# Example with Fast Downward
./fast-downward.py evacuation-domain.pddl evacuation-problem.pddl \
  --search "astar(lmcut())"
```

Phase 3: Implementation and Visualization (1h)

Task 3.1: Python Interface with Planner

Create a Python script that:

1. Loads domain and problem
2. Calls the planner
3. Retrieves and parses generated plan
4. Executes plan step by step

```
import subprocess
import re

class EvacuationPlanner:
    def __init__(self, domain_file, problem_file):
        self.domain_file = domain_file
        self.problem_file = problem_file

    def run_planner(self):
        """Call Fast Downward and retrieve plan"""
        # TODO: Implement planner call
        pass

    def parse_plan(self, plan_output):
        """Parse generated plan"""
        # TODO: Extract actions from plan
        pass

    def execute_plan(self, plan):
        """Execute plan in simulation"""
        for action in plan:
            # TODO: Update simulation state
            pass
```

Task 3.2: Visual Simulation

Create a graphical visualization of the evacuation with Pygame:

```
import pygame

class EvacuationSimulation:
    def __init__(self, width=1200, height=800):
        pygame.init()
        self.screen = pygame.display.set_mode((width, height))
        self.clock = pygame.time.Clock()

    def draw_building(self):
        """Draw building structure"""
        # TODO: Draw rooms, corridors, stairs
        pass

    def draw_entities(self, state):
        """Draw robots and people"""
        # TODO: Display robot and person positions
        # Use colors:
        # - Green for safe zones
        # - Red for smoky zones
        # - Blue for robots
        # - Yellow for people
        pass

    def animate_action(self, action):
        """Animate a plan action"""
        # TODO: Animate robot movement
        pass

    def run(self, plan):
        """Execute complete simulation"""
        running = True
```

```python
    action_index = 0

    while running and action_index < len(plan):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        self.screen.fill((255, 255, 255))  # White background
        self.draw_building()
        self.draw_entities(current_state)

        # Execute next action
        self.animate_action(plan[action_index])
        action_index += 1

        pygame.display.flip()
        self.clock.tick(2)  # 2 actions per second

    pygame.quit()
```

Minimum required features:

- Display of building structure (rooms, corridors)
- Visualization of smoky zones (in red)
- Robot positions (icons or blue squares)
- Person positions (icons or yellow squares)
- Animation of robot movement
- Counter of evacuated people
- Display of current action

Bonus features:

- Controls (pause, speed up, slow down, restart)
- Real-time statistics (elapsed time, actions performed)
- Multi-floor view with switching
- Export simulation to video

Task 3.3: Tests and Scenarios

Test your system on multiple scenarios:

1. **Base scenario**: The one described in the assignment
2. **Extended scenario**: 15 people, 3 robots
3. **Difficult scenario**: More smoky zones, fewer robots
4. **Stress-test scenario**: Larger building, 20+ people

For each scenario, measure:

- Plan computation time
- Number of actions in plan
- Simulated evacuation time
- Success or failure

Phase 4: Analysis and Report (Included in all sessions)

Task 4.1: Performance Evaluation

Create a comparative table:

Scenario	Comp. time	Nb actions	Evac. time	Success
Base				
Extended				
Difficult				
Stress-test				

Task 4.2: Critical Analysis

Write a critical analysis (1-2 pages max) including:

1. **Strengths of your approach**:
   - What works well?
   - Which constraints are well handled?
   - What are the advantages of your modeling?
2. **Identified limitations**:
   - Which constraints are not modeled?
   - What realistic aspects are missing?
   - What problematic situations did you encounter?
3. **Possible improvements**:
   - How to improve the model?
   - What extensions would be useful?
   - What other approaches could you try?

Task 4.3: Theoretical Comparison

If you tested multiple approaches, compare them:

Criterion	Approach 1	Approach 2
Expressiveness		
Computation time		
Plan quality		
Implementation ease		
Maintainability		

Deliverables and Evaluation

Final Deliverables

You must submit a **ZIP archive** containing:

1. **Folder** `modeling/`:
   - Modeling files (PDDL or Python HTN)
   - Multiple problem instances (scenarios)
   - README explaining your modeling
2. **Folder** `src/`:
   - Python simulation code
   - Interface script with planner
   - `requirements.txt` file for dependencies
   - README with execution instructions
3. **Folder** `results/`:
   - Plans generated for each scenario
   - Simulation screenshots
   - Execution logs
   - Result tables

Filename: `lastname1_lastname2_evacuation_project.zip`

Tips and Resources

Methodology Tips

1. **Start simple**:
   - First model a minimal scenario (2 people, 1 robot, 2 rooms)
   - Validate it works before adding complexity
2. **Iterate progressively**:
   - Session 1: Minimal scenario
   - Session 2: Add constraints (smoky zones)
   - Session 3: Complete scenario
   - Sessions 4-5: Simulation and testing
3. **Test regularly**:
   - Check syntax after each modification
   - Test planner on small problems first
   - Validate each action independently
4. **Document as you go**:
   - Take notes on your design choices
   - Capture intermediate results
   - Don't leave report writing for the end

Useful Resources

PDDL Planners

- **Fast Downward**: `https://www.fast-downward.org/`
- **Online PDDL Editor**: `http://editor.planning.domains/`
- **PDDL Manual**: `https://planning.wiki/`

HTN Planning

- **PyHOP**: Simple HTN planner in Python
- **SHOP2/JSHOP2**: More advanced HTN planners

Python and Visualization

- **Pygame**: `https://www.pygame.org/`
- **Matplotlib**: For graphs and statistics
- **NetworkX**: To visualize plan graphs

Common Mistakes to Avoid

- **Not testing early enough**: Don't wait until you've coded everything to test
- **Too complex model from start**: Simplify first, complexify later
- **Inconsistent preconditions**: Verify your actions are applicable
- **Forgetting negative effects**: Remember to remove old predicates
- **No plan validation**: Check that plan actually reaches goal

Frequently Asked Questions

Q1: Can we use advanced PDDL extensions?

Answer: Yes, but with moderation. You can use:

- `:durative-actions` for temporal PDDL
- `:fluents` for numeric resources
- `:conditional-effects` if necessary

But justify your choices and ensure your planner supports them.

Q2: Must we model fire spread?

Answer: No, you can simplify by considering smoky zones are fixed. If you want to model spread (bonus), use temporal PDDL with delayed effects.

Q3: Can we change the scenario?

Answer: You must solve the base scenario, but you can propose variants for your tests and analysis. Document differences well.

Q4: Must simulation be real-time?

Answer: No, you can do step-by-step animation. Real-time is a bonus.

Q5: How many lines of code are expected?

Answer: There is no minimum or maximum. A complete project typically has:

- 50-100 lines of PDDL (domain + problem)
- 300-500 lines of Python (simulation + interface)

But quality matters more than quantity.

Good luck with your project!

Don't hesitate to ask questions during sessions.
This project is an opportunity to put all course concepts into practice.