Multi-Agent Intelligent Systems Project

Implementation and Analysis of Agent Architectures

Course: Artificial Intelligence
Instructor: Dr. Mahdi Eftekhari

Due Date: 27 June 2025
Duration: 4 Weeks

1 Project Overview

1.1 Introduction

This project provides hands-on experience with fundamental concepts in artificial intelligence, specifically focusing on intelligent agent architectures and multi-agent systems. You will implement three distinct agent architectures based on Russell and Norvig's taxonomy, analyze their performance characteristics, and gain practical understanding of the trade-offs between reactive and deliberative approaches to artificial intelligence.

The project bridges classical AI theory with modern multi-agent systems, providing insights into how individual intelligent behavior scales to collective intelligence in distributed environments. Through systematic implementation and experimentation, you will develop deep understanding of agent design principles and their practical implications.

1.2 Learning Objectives

Upon completion of this project, you will be able to:

- **Understand Agent Architectures**: Implement and compare simple reflex, model-based reflex, and goal-based agent architectures from Russell & Norvig's framework
- **Apply Planning Algorithms**: Implement path planning using A* algorithm and utility-based goal selection for deliberative agents
- **Analyze Performance Trade-offs**: Evaluate computational efficiency versus problem-solving effectiveness across different agent types
- **Design Behavioral Rules**: Create condition-action rules and priority systems for reactive agent behavior
- **Implement Internal Models**: Develop world state representation and maintenance systems for model-based agents
- **Conduct Empirical Analysis**: Design experiments, collect performance data, and draw insights from comparative analysis
- **Connect Theory to Practice**: Relate abstract AI concepts to concrete implementation challenges and design decisions

1.3 Project Context

This assignment directly implements concepts from Chapter 2 of *Artificial Intelligence: A Modern Approach* by Russell and Norvig, specifically:

- Agent architectures and the PEAS framework
- Environment characteristics (partially observable, stochastic, dynamic, multi-agent)
- Rational agent behavior and performance measures
- Simple reflex agents with condition-action rules
- Model-based reflex agents with internal state
- Goal-based agents with planning capabilities

The gridworld environment provides a controlled testbed that exhibits key characteristics of real-world multi-agent scenarios while remaining computationally tractable for educational purposes.

2 Technical Specifications

2.1 Environment Description

The multi-agent gridworld environment consists of:

Grid Structure Discrete 2D grid with configurable dimensions (typically 8×8 to 12×12)

Cell Types Empty spaces, walls (obstacles), goals (task completion zones), resources (collectible items), hazards (energy-depleting areas)

Agent Representation Triangular markers with unique colors and energy indicators

Partial Observability Agents perceive only local 5×5 neighborhoods centered on their position

Energy System Agents start with 100 energy points, consume energy for actions, become inactive at zero energy

Task Objective Collect resources and deliver them to goal zones while avoiding hazards and managing energy

2.2 Agent Perception Model

Each agent receives structured perception data including:

- Current position coordinates
- Visible cell types within perception range
- Positions of other visible agents
- Current energy level (0-100)
- Resource carrying status
- Inter-agent communication messages

2.3 Action Space

Agents can perform eight distinct actions:

- **Movement**: MOVE_NORTH, MOVE_SOUTH, MOVE_EAST, MOVE_WEST
- **Object Manipulation**: PICKUP (collect resources), DROP (release resources)
- **Utility**: WAIT (conserve energy), COMMUNICATE (send messages)

3 Implementation Requirements

3.1 Agent Architecture 1: Simple Reflex Agent

3.1.1 Theoretical Foundation

Simple reflex agents represent the most basic form of intelligent behavior, operating through direct condition-action mappings without internal state maintenance. These agents respond immediately to perceptual inputs using predefined behavioral rules, making them computationally efficient but limited in handling complex scenarios requiring memory or planning.

3.1.2 Implementation Requirements

Your SimpleReflexAgent must implement the following behavioral rules in priority order:

1. **Hazard Avoidance** (Priority 1): If current position contains hazard, move to adjacent safe cell
2. **Resource Collection** (Priority 2): If adjacent cell contains resource, execute PICKUP action
3. **Goal Seeking** (Priority 3): If carrying resource and goal visible, move toward goal
4. **Resource Pursuit** (Priority 4): If resource visible and not carrying, move toward resource
5. **Random Exploration** (Priority 5): If no specific stimuli present, move randomly to valid adjacent cell

3.1.3 Key Implementation Details

- Use condition-action rules: `if (condition) then (action)`
- Implement rule priority system to handle multiple applicable rules
- Track rule activation frequencies for analysis purposes
- Handle edge cases (no valid moves, boundary conditions)
- Provide descriptive reason strings for each action decision

3.1.4 Expected Behavior Characteristics

- Fast decision-making with minimal computational overhead
- Reactive responses to immediate environmental stimuli
- Lack of strategic planning or long-term goal pursuit
- Susceptibility to local optima and repetitive behavior patterns
- Effective performance in simple, well-structured environments

3.2 Agent Architecture 2: Model-Based Reflex Agent

3.2.1 Theoretical Foundation

Model-based reflex agents extend simple reflex behavior by maintaining internal representations of environment state. This internal model enables more sophisticated decision-making based on accumulated knowledge rather than purely immediate perceptions, allowing for better handling of partially observable environments and temporal dependencies.

3.2.2 Implementation Requirements

Your ModelBasedReflexAgent must maintain internal models tracking:

- **Spatial Memory**: Set of visited positions to guide exploration
- **Resource Tracking**: Known locations of resources (discovered but not yet collected)
- **Goal Awareness**: Locations of goal zones for strategic navigation
- **Hazard Mapping**: Dangerous areas to avoid during navigation
- **Environment Layout**: Wall positions and navigable spaces

3.2.3 Decision-Making Priorities

Implement the following decision hierarchy:

1. **Emergency Response**: Immediate hazard avoidance using spatial memory
2. **Opportunistic Collection**: Resource pickup from adjacent cells
3. **Strategic Goal Completion**: Navigation to known goals when carrying resources
4. **Informed Resource Acquisition**: Movement toward known resource locations
5. **Intelligent Exploration**: Systematic exploration prioritizing unvisited areas

3.2.4 Key Implementation Details

- Update internal model with each perception using `_update_world_model()`
- Use set data structures for efficient location tracking
- Implement intelligent exploration strategy (avoid revisiting known areas)
- Apply internal knowledge to improve navigation efficiency
- Balance exploitation of known information with exploration of unknown areas

3.2.5 Expected Behavior Characteristics

- Superior performance compared to simple reflex agents
- Systematic exploration patterns avoiding redundant revisits
- Strategic resource collection based on accumulated knowledge
- Improved efficiency in partially observable environments
- Higher computational overhead due to model maintenance

3.3 Agent Architecture 3: Goal-Based Agent

3.3.1 Theoretical Foundation

Goal-based agents represent the most sophisticated architecture in this project, incorporating deliberative planning capabilities to achieve long-term objectives. These agents decompose complex goals into sequential action plans, evaluate alternative strategies using utility functions, and adapt dynamically when plans encounter obstacles or become invalid.

3.3.2 Implementation Requirements

Your GoalBasedAgent must implement:

- **Goal Selection**: Utility-based evaluation of candidate objectives
- **Path Planning**: A* algorithm for optimal navigation between positions
- **Plan Generation**: Decomposition of high-level goals into executable action sequences
- **Plan Execution**: Step-by-step execution with progress monitoring
- **Plan Adaptation**: Replanning when plans become invalid or suboptimal

3.3.3 Goal Types and Utilities

Implement utility functions for:

- **Resource Collection** (utility = 10.0): Acquire resources when not carrying any
- **Resource Delivery** (utility = 20.0): Transport resources to goal zones
- **Exploration** (utility = 1.0): Investigate unknown areas of environment
- **Hazard Avoidance** (utility = 50.0): Emergency response to immediate threats

Utility values should be distance-adjusted: `final_utility = base_utility / (distance + 1)`

3.3.4 A* Pathfinding Implementation

Your `_find_path()` method must implement A* algorithm with:

- Manhattan distance heuristic: `h(n) = |x_goal - x_n| + |y_goal - y_n|`
- Priority queue using Python's `heapq` module
- Path reconstruction from goal to start using came_from mapping
- Obstacle avoidance (walls) and hazard cost consideration
- Termination when goal reached or no path exists

3.3.5 Plan Representation

Use the provided `PlanStep` dataclass to represent plan elements:

```
@dataclass
class PlanStep:
    action: Action            # Specific action to execute
    target_position: Position # Expected position after action
    purpose: str              # Human-readable plan purpose
    estimated_cost: float     # Energy cost estimate
```

3.3.6 Key Implementation Details

- Plan validation: Check if preconditions still hold before execution
- Replanning triggers: Invalid plans, disappeared resources, blocked paths
- Reactive fallback: Simple behavior when planning fails completely
- Plan monitoring: Track execution progress and detect failures
- Utility maximization: Always select highest-utility achievable goal

3.3.7 Expected Behavior Characteristics

- Optimal or near-optimal task completion paths
- Strategic behavior with long-term planning horizon
- Adaptive response to changing environmental conditions
- Highest computational overhead due to planning algorithms
- Superior performance in complex, dynamic scenarios

4 Experimental Analysis

4.1 Experimental Design

Conduct systematic performance comparison across three scenarios:

Simple Collection 8×8 grid, 2 agents, 4 resources, 2 goals, no hazards, 100 steps

Maze Navigation 10×10 grid, 2 agents, 4 resources, 2 goals, 3 hazards, 150 steps

Competitive Collection 12×12 grid, 3 agents, 3 resources, 2 goals, 2 hazards, 200 steps

4.2 Performance Metrics

Collect the following quantitative measures:

- **Success Rate**: Percentage of resources successfully collected
- **Efficiency Score**: Composite metric incorporating resource collection, time usage, energy conservation, and collision avoidance
- **Task Completion Time**: Average steps required to complete objectives

- **Energy Utilization**: Average energy remaining at task completion
- **Collision Frequency**: Number of attempted invalid moves
- **Exploration Coverage**: Percentage of environment explored

4.3 Statistical Analysis

- Run minimum 5 trials per agent-scenario combination
- Calculate means and standard deviations for all metrics
- Create performance comparison tables and visualizations
- Identify statistically significant performance differences
- Analyze performance scaling with environment complexity

5 Implementation Guidelines

5.1 Development Methodology

Follow incremental development approach:

1. **Week 1**: Implement SimpleReflexAgent with basic rule system
2. **Week 2**: Develop ModelBasedReflexAgent with internal model maintenance
3. **Week 3**: Create GoalBasedAgent with planning and utility functions
4. **Week 4**: Conduct experiments, analyze results, write report

5.2 Code Quality Requirements

- **Error Handling**: Robust handling of edge cases and invalid states
- **Modularity**: Clear separation of concerns between different functionalities
- **Testing**: Use provided testing framework to validate implementations

5.3 Debugging Strategies

- Use provided `ProjectTester.test_single_agent()` for isolated testing
- Implement logging/print statements in decision-making methods
- Visualize agent behavior using environment visualization capabilities
- Test individual components (pathfinding, rule evaluation) separately
- Start with simple scenarios before testing complex environments

6 Deliverables and Assessment

6.1 Submission Requirements

1. **Source Code**: Complete Python implementation with all three agent types
2. **Experimental Data**: CSV files containing performance metrics from all trials
3. **Analysis Report**: 6-8 page technical report discussing implementation and results
4. **Documentation**: README file with installation and usage instructions

6.2 Report Structure

Your technical report should include:

1. **Introduction** (1 page): Project overview and objectives
2. **Implementation Details** (2-3 pages): Architecture-specific design decisions and key algorithms
3. **Experimental Results** (2-3 pages): Performance analysis with tables, graphs, and statistical comparisons
4. **Discussion** (1-2 pages): Insights, limitations, and connections to AI theory
5. **Conclusion** (0.5 page): Summary of key findings and lessons learned

6.3 Grading Rubric

Component	Weight	Criteria
Implementation Correctness	40%	Functional agents passing all test cases
Code Quality	20%	Documentation, style, error handling
Experimental Analysis	25%	Systematic testing and statistical analysis
Technical Report	15%	Clear writing, insights, theoretical connections

Table 1: Assessment Criteria and Weights

6.4 Excellence Indicators

To achieve top grades, demonstrate:

- All three agents implemented correctly with sophisticated behavioral patterns
- Comprehensive experimental analysis with statistical rigor
- Clear connections between implementation choices and theoretical concepts
- Well-documented, maintainable code following best practices
- Insightful discussion of results and their implications for AI system design

7 Resources and Support

7.1 Primary References

- Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Chapter 2: Intelligent Agents
- Course lecture materials on agent architectures and multi-agent systems
- Provided skeleton code and testing framework documentation

7.2 Technical Resources

- Python documentation for data structures (sets, dictionaries, heapq)
- A* pathfinding algorithm tutorials and implementations
- Matplotlib documentation for creating performance visualizations
- NumPy documentation for numerical computations and array operations

8 Academic Integrity

This is an individual assignment. You may discuss general concepts and approaches with classmates, but all code must be your own original work. Specifically:

- Do not share or copy implementation code
- You may discuss algorithm concepts and debugging strategies
- All sources must be properly cited in your report
- Use of AI coding assistants must be disclosed and explained
- Collaboration on experimental analysis is not permitted

Violations will result in assignment failure and may trigger broader academic misconduct procedures.

9 Timeline and Milestones

Week	Milestone	Deliverable
1	SimpleReflexAgent	Functional reactive agent with rule-based behavior
2	ModelBasedReflexAgent	Agent with internal world model and intelligent exploration
3	GoalBasedAgent	Planning agent with A* pathfinding and utility-based goals
4	Analysis & Report	Complete experimental analysis and technical report

Table 2: Project Timeline and Key Milestones

10 Conclusion

This project provides comprehensive hands-on experience with fundamental AI concepts, bridging theoretical understanding with practical implementation skills. Through systematic development of increasingly sophisticated agent architectures, you will gain deep insights into the trade-offs between computational efficiency and problem-solving capability that characterize modern AI systems.

The multi-agent gridworld environment serves as an accessible yet realistic testbed for exploring how individual intelligent behavior scales to collective intelligence in distributed systems. Your implementations will demonstrate the evolution from simple reactive systems to sophisticated deliberative agents capable of long-term planning and adaptation.

Success in this project requires careful attention to both theoretical foundations and practical implementation details. The combination of systematic coding, rigorous experimentation, and thoughtful analysis will prepare you for advanced study and professional work in artificial intelligence and related fields.

Good luck with your implementation!