**MINISTRY OF EDUCATION AND TRAINING**

**UNIVERSITY OF SCIENCE**

**INFORMATION TECHNOLOGY**


**HO CHI MINH UNIVERSITY OF SCIENCE**

**CSC14003 – Introduction to Artificial Intelligence**

**Project 2. Logic**

**Topic: Wumpus World Agent**

**Class: 23CLC01 - Group 9**


| | |
|---|---|
| **Members:** | Trần Thọ - 23127264 |
| | Trần Văn Huy - 23127380 |
| | Phan Trung Hiếu - 23127529 |
| | Nguyễn Tuấn An - 23127316 |
| **Instructor:** | Nguyen Ngoc Thao, PhD |
| | Nguyen Hai Dang, MSc |
| | Nguyen Thanh Tinh |
| | Nguyen Tran Duy Minh, MSc |
| | Ho Thi Thanh Tuyen, MSc |


Ho Chi Minh City, August 13rd, 2025

## CONTENT

## LIST OF TABLES

## LIST OF FIGURES

## CONTENT

Installation and instruction: README.md

## QUICK LOOK:

```
KB Facts: {('safe', 0, 1), ('safe', 1, 0), ('safe', 0, 0), ('no_wumpus', 1, 0), ('no_stench', 0, 0), ('no_pit', 0, 1),
('no_pit', 1, 0), ('no_wumpus', 0, 1), ('no_breeze', 0, 0), ('visited', 0, 0)}
Moving to adjacent unvisited safe cell: (0, 1)
[Step 0] Action: turn_left, Position: (0, 0), Direction: E, Has Gold: False, Scream: False, Bump: False
Map View:
# # # # # #
# . W . . #
# G . W . #
# . . . . #
# A . . . #
# # # # # #
KB Facts: {('safe', 0, 1), ('safe', 1, 0), ('safe', 0, 0), ('no_wumpus', 1, 0), ('no_stench', 0, 0), ('no_pit', 0, 1),
('no_pit', 1, 0), ('no_wumpus', 0, 1), ('no_breeze', 0, 0), ('visited', 0, 0)}
Moving to adjacent unvisited safe cell: (0, 1)
[Step 1] Action: move, Position: (0, 0), Direction: N, Has Gold: False, Scream: False, Bump: False
Map View:
# # # # # #
# . W . . #
# G . W . #
# A . . . #
# . . . . #
# # # # # #
```

*Figure 1. Console step by step*

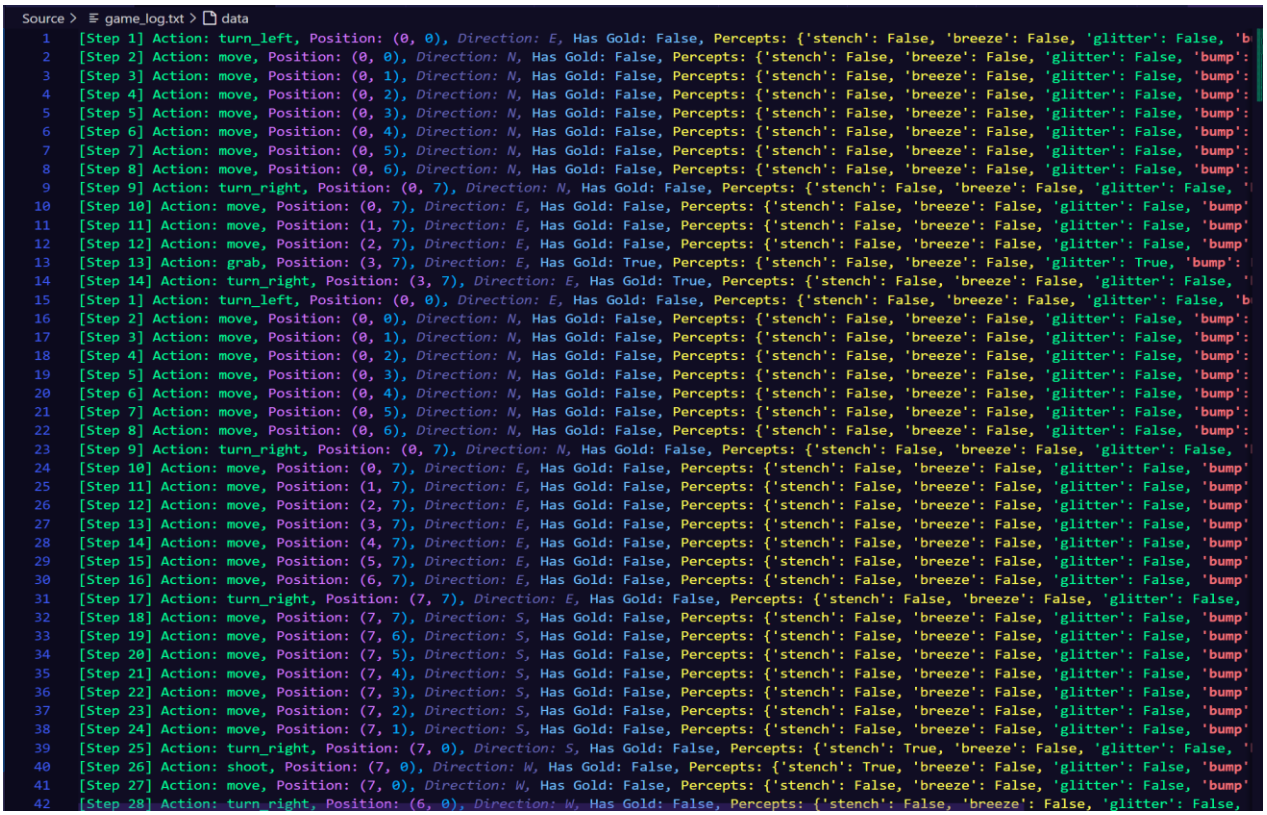


*Figure 2. Game log*


***Figure 3. Graphic visualization***

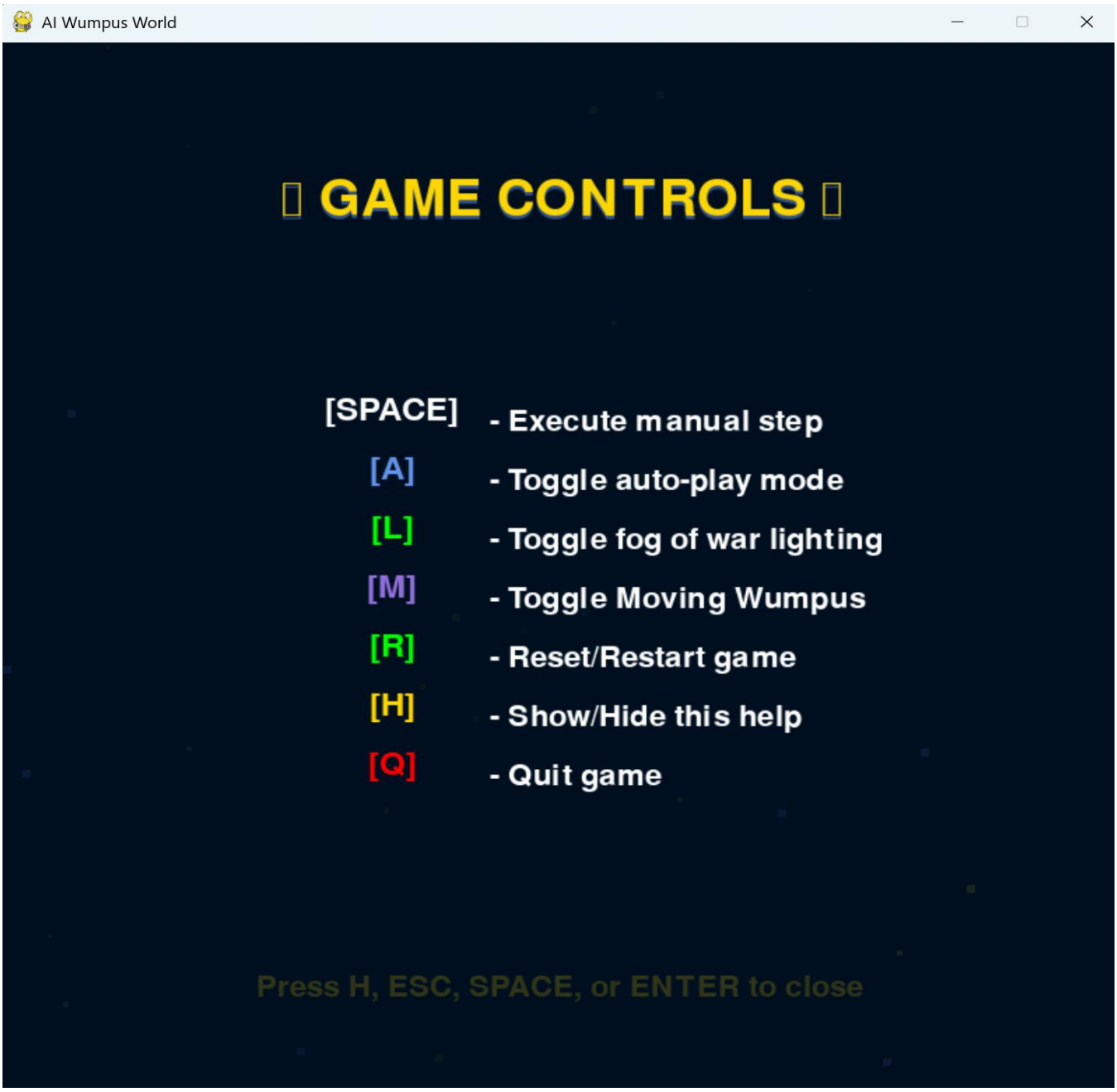


*Figure 4. Game control help center*

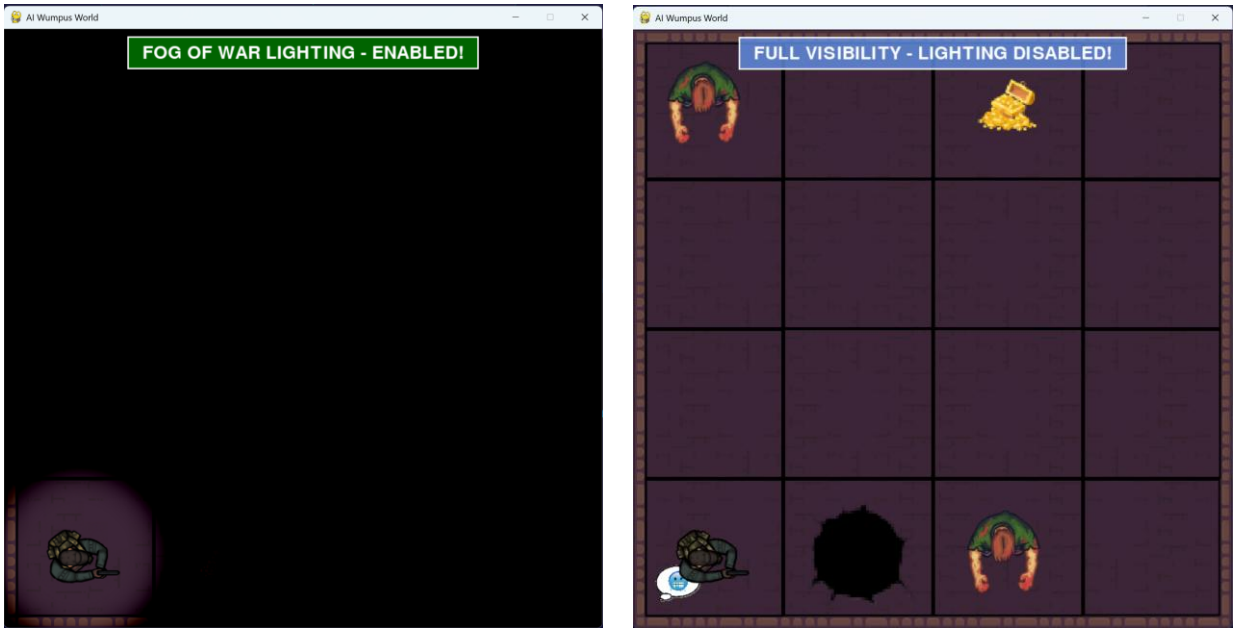


*Figure 5. "Fog" effect represents currrent percepts*

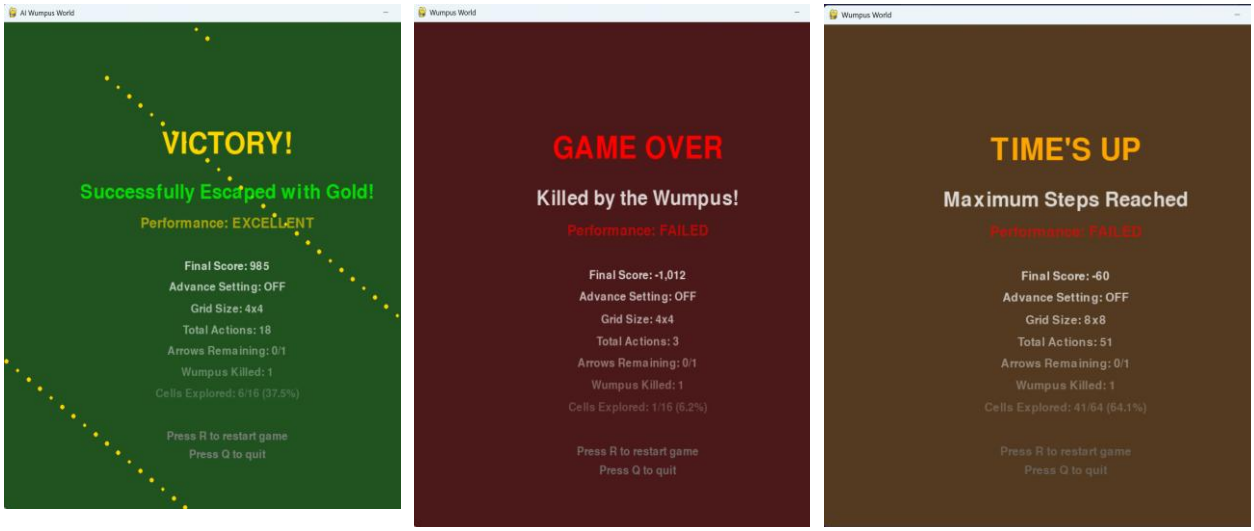


*Figure 6. Result screen*

**CONTENT**

# 1. PROBLEM INTRODUCTION AND OBJECTIVES

## 1.1. Problem introduction:

Wumpus World is a classic AI problem that demonstrates concepts of logic, reasoning, and decision-making for intelligent agents. It serves as an ideal simulation environment for testing AI algorithms. In this game, a hunter is placed inside a grid-like cave system where the goal is to find the gold and escape safely while avoiding deadly pits and a dangerous creature called the Wumpus.

There are 3 clues that hunter can perceives:

- **Breeze**: Indicates the presence of a pit in an adjacent cell.
- **Stench**: Indicates the presence of the Wumpus in an adjacent cell.
- **Glitter**: Indicates gold in the current cell.

**We need to** design an intelligent agent that can explore the environment, infer the locations of hazards using propositional logic, plan safe paths to retrieve the gold, and return to the start without dying. In the advanced setting, the Wumpus can move, adding an additional layer of complexity that requires dynamic reasoning and frequent knowledge base updates.

## 1.2. Objectives:

1. Formulate a knowledge base using propositional logic to represent environmental rules and percepts.
2. Implement an inference engine capable of deducing safe and unsafe cells, including handling moving Wumpus scenarios.
3. Design and integrate a planning algorithm that allows the agent to choose optimal actions.
4. Evaluate the agent's performance against a random baseline in terms of:
    - Success rate (percentage of games won)
    - Average score
    - Decision-making efficiency

## 2. FORMULATION OF KNOWLEDGE BASE (KB) RULES USING PROPOSITIONAL LOGIC

### 2.1. Core Knowledge Representation

The system uses a fact-based knowledge representation where each fact is represented as a tuple:

- Format: (predicate, x, y) where x,y are grid coordinates
- Example: ("pit", 2, 3) means there is a pit at position (2,3)

### 2.2. Propositional Logic Rules Implementation

#### 2.2.1. Breeze rules (Pit Detection Logic)

Location: knowledge_base.py - breeze_rule()

Logical Formulation:

- If Breeze(x,y) then ∃(adjacent cell) Pit(adj_x, adj_y)
- If ¬Breeze(x,y) then ∀(adjacent cells) ¬Pit(adj_x, adj_y)

Implementation Details:

##### 2.2.1.1. Positive breeze detection:

When breeze detected at (x,y):

- Mark all unknown adjacent cells as "possible_pit"
- If only one adjacent cell is unknown → definite pit location
- Facts generated: ("possible_pit", nx, ny) for each adjacent unknown cell

##### 2.2.1.2. Negative breeze detection:

When no breeze at (x,y):

- Mark all adjacent cells as "no_pit"
- Mark all adjacent cells as "safe" (if also no wumpus)
- Facts generated: ("no_pit", nx, ny), ("safe", nx, ny)

Encoding for Multiple Pits:

- Each cell can independently contain a pit
- Breeze indicates $\geq 1$ pit in adjacent cells
- System tracks "possible_pit" candidates across multiple breeze sources
- Intersection logic: If multiple breeze sources point to same cell → higher confidence

### 2.2.2. Stench rules (Wumpus Detection Logic)

Location: knowledge_base.py - stench_rule()

Logical Formulation:

- If Stench(x,y) then ∃(adjacent cell) Wumpus(adj_x, adj_y)
- If ¬Stench(x,y) then ∀(adjacent cells) ¬Wumpus(adj_x, adj_y)

Implementation Details:

#### 2.2.2.1. Positive stench detection:

When stench detected at (x,y):

- Mark unknown adjacent cells as "possible_wumpus"
- Apply constraint: only mark if not already known as pit-free
- If only one possible location → definite wumpus location

#### 2.2.2.2. Negative stench detection:

When no stench at (x,y):

- Mark all adjacent cells as "no_wumpus"
- Combine with pit information for safety determination
- Facts generated: ("no_wumpus", nx, ny)

Encoding for Multiple Wumpuses:

- Each wumpus creates independent stench fields
- Multiple stenches can overlap in adjacent areas
- System maintains separate "possible_wumpus" facts for each suspected location
- Dynamic updating when wumpuses are eliminated or move (advanced mode)

### 2.2.3. Safety rules (Composite Logic)

Location: knowledge_base.py - infer() method

Logical Formulation: Safe(x,y) ↔ ¬Pit(x,y) ∧ ¬Wumpus(x,y)

Implementation:

- Automatic inference: If "no_pit" AND "no_wumpus" → "safe"
- Safety facts enable path planning and exploration decisions
- Critical for agent decision-making process

#### 2.2.4. Gold detection rules

Location: agent.py - perceive() method

Logical Formulation:

- If Glitter(x,y) then Gold(x,y)
- Glitter is direct evidence of gold presence

Implementation:

- Immediate fact assertion when glitter detected
- Triggers goal-oriented behavior (grab gold, return home)
- No uncertainty - glitter is definitive evidence

### 2.3 Knowledge Base Inference Engine

#### 2.3.1 Dynamic Inference Process

Location: knowledge_base.py - DynamicKB.infer()

Process:

1. Apply all rules until no new facts are generated (fixed-point)
2. Forward chaining inference
3. Automatic safety determination through rule combination
4. Maintains consistency across all fact assertions

Termination Condition:

- Loop continues until no new facts are discovered
- Guarantees complete inference given current knowledge

### 2.3.2. Multiple Threat Handling

#### 2.3.2.1.Multiple Pits Encoding:

- Independent pit probabilities for each cell
- Breeze sources create overlapping "possible_pit" regions
- Constraint satisfaction: pit locations must explain all observed breezes
- No limit on number of pits (configurable via pit_prob parameter)

#### 2.3.2.2. Multiple Wumpuses Encoding:

- Support for 0-N wumpuses (configurable via num_wumpus parameter)
- Each wumpus tracked independently in environment.wumpus_positions[]
- Stench fields can overlap without interference
- Dynamic knowledge updating when wumpuses move (advanced mode)
- Arrow shooting eliminates specific wumpus, updates KB accordingly

## 3. SYSTEM ARCHITECTURE AND MODULE DESIGN

### 3.1. Overall Architecture Pattern

Design Pattern: Model-View-Controller (MVC) with AI Agent Layer

Core Principle: Separation of concerns with modular, extensible design

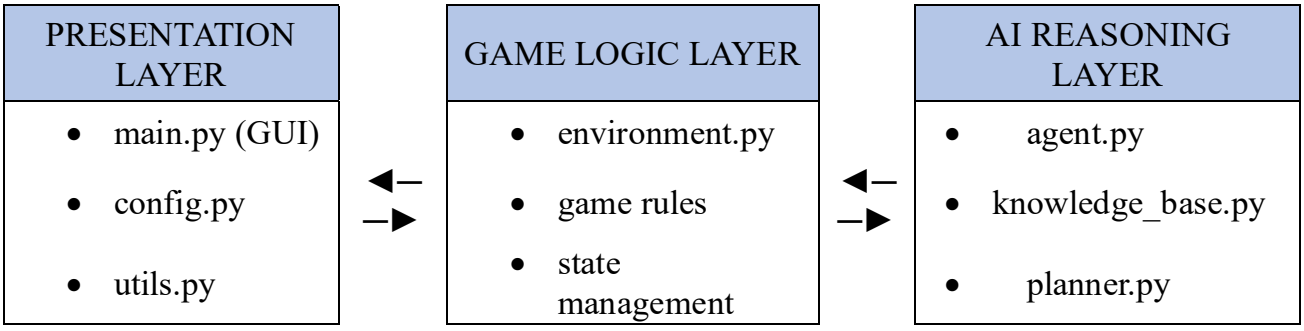


***Figure 7. System architecture and module design***

### 3.2. Module Detailed Design

#### 3.2.1. Environment Module (environment.py)

Purpose: Game World State Management and Physics

Key Classes:

- Environment: Main game world controller
- Cell: Individual grid cell state container

Responsibilities:

#### 3.2.1.1.World State Management:

- Grid initialization and management
- Pit and wumpus placement algorithms
- Gold placement logic
- Dynamic wumpus movement (advanced mode)

#### 3.2.1.2.Game physics:

- Action processing (move, turn, shoot, grab, climb)
- Collision detection (pit, wumpus, wall)
- Percept generation (breeze, stench, glitter, bump)
- Score calculation and tracking

#### 3.2.1.3. Advance Features:

- **Moving wumpus system (every 5 actions)**
- Multiple wumpus position tracking
- Death cause analysis
- Score and performance metrics
- **Supports multiple game configurations**

### 3.2.2. Agent Module (agent.py)

Purpose: AI Reasoning and Decision Making

Key Classes:

- KBWumpusAgent: Intelligent agent using knowledge base
- RandomWumpusAgent: Baseline comparison agent

KBWumpusAgent Architecture:

#### 3.2.2.1.Knowledge Management:

- Maintains DynamicKB instance
- Processes percepts into logical facts
- Performs inference and reasoning
- Tracks visited locations and discoveries

#### 3.2.2.2. Decision making:

- Goal-oriented planning (find gold, return home))
- Safety-first exploration strategy
- Strategic arrow usage
- Dynamic plan adaptation

#### 3.2.2.3. Path Planning:

- Integration with A* pathfinding
- Safe path preference
- Unknown cell exploration with penalties
- Efficient direction changing (optimized turn logic)

#### 3.2.2.4.Learning and Adaptation:

- Knowledge base updates
- Wumpus movement adaptation (advanced mode)
- Experience-based decision refinement

### 3.2.3. Knowledge Base(knowledge_base.py)

Purpose: Logical Reasoning and Inference Engine

Key Classes:

- DynamicKB: Main knowledge base container
- Rule functions: breeze_rule, stench_rule

Architecture Components:

#### 3.2.3.1. Fact Storage:

- Set-based fact storage for O(1) lookup
- Tuple representation for logical facts
- Automatic duplicate prevention

#### 3.2.3.2.Rule Engine:

- Pluggable rule system
- Forward chaining inference – infer() method
- Fixed-point computation

- Consistency maintenance

**3.2.3.3. Inference Algorithm:**

- Constraint propagation
- Safety determination
- Threat localization
- Uncertainty handling

Extensibility:

- Easy addition of new rules
- Configurable inference depth
- Modular fact representation
- Clean separation from agent logic

### 3.2.4. Planner Module (planner.py)

Purpose: Pathfinding and Strategic Planning

Key Functions:

- astar(): A* pathfinding implementation
- heuristic(): Manhattan distance with penalties

Planning Architecture:

**3.2.4.1. Search Algorithm:**

- A* with configurable heuristics
- Priority queue implementation
- Path reconstruction
- Goal-oriented search

**3.2.4.2.Cost Modeling:**

- Base movement costs
- Danger penalties (pit/wumpus areas)
- Unknown cell penalties
- Safety bonuses

**3.2.4.3. Integration:**

- Knowledge base integration

- Dynamic cost updates
- Multiple goal support
- Replanning capabilities

Features:

- Optimal path finding
- Risk-aware navigation
- Configurable exploration behavior
- Efficient search algorithms

### 3.2.5. Presentation Layer (main.py, config.py, utils.py)

Purpose: User Interface and System Configuration

main.py - Game Controller + GUI:

#### 3.2.5.1. Game loop management:

- Real-time game execution
- Event handling (keyboard, mouse)
- State transitions
- Performance monitoring

#### 3.2.5.2. Visual Rendering:

- PyGame-based graphics
- Dynamic scaling for different grid sizes: convert from “grid coordinates” to “pixel coordinates” to display assets in place
- Lighting effects (fog of war): present the effect that hunter have no idea about the map information except percepts
- Animation systems: Wining, Game Over, Game configurations annimation

#### 3.2.5.3. User interaction:

- Control mapping
- Game mode switching (auto step, manual step)
- Configuration options (map size, pit probability, number of wumpus,….)
- Help systems

#### 3.2.5.4. config.py - Configuration Management:

Assets loading:

- Sprite loading and scaling
- Sound effect management
- Font and UI resources
- Performance optimization

Argument parsing:

- Command-line configuration
- Validation and error handling
- Default value management
- Help documentation

**3.2.5.5. utils.py - Utility Functions:**

Assets proccessing:

- Image scaling and rotation
- Grid position calculations
- Resource management
- Error handling

Helper function:

- Mathematical utilities
- File operations
- Debug functions
- Performance tools

## 3.3. Data Flow Architecture

### 3.3.1 Information Flow Diagram

User Input → main.py → Environment → Agent → Knowledge Base

↑ ↓ ↓ ↓

└── Visual Output ← Renderer ← State ← Decision ← Inference

### 3.3.2 Decision Making Pipeline

1. Percept Collection: Environment → Agent

2. Knowledge Update: Agent → Knowledge Base

3. Inference Process: Knowledge Base internal

4. Decision Logic: Agent using KB facts

5. Action Selection: Agent → Environment

6. State Update: Environment internal

7. Visual Update: Environment → Renderer → User

## 4. INFERENCE ENGINE APPROACH

The agent's reasoning is based on forward chaining in the propositional logic framework of the Wumpus World. At each step, the agent collects percepts from the environment — such as breeze, stench, glitter, bump, and scream — and immediately applies inference rules to update its knowledge base.

### 4.1. Forward Chaining for Hazard tion

- Breeze Rules: If a breeze is perceived, adjacent cells are marked as possible pits. If no breeze is perceived, all adjacent cells are marked safe from pits.
- Stench Rules: If a stench is perceived, adjacent cells are marked as possible Wumpus locations. If no stench is perceived all adjacent cells are marked safe from Wumpus. When the hunter gets stench effect, it can also shoot an arrow to determine whether the wumpus is ther or not. If the wumpus is hit, we will have a "Scream" effect, telling the hunter that a Wumpus has hit the arrow and died, if not, the hunter will know that all of the cells where the arrow has gone through have no wumpus.
- Multiple Hazards: The reasoning engine supports multiple pits and multiple Wumpus creatures. Instead of assuming there is exactly one hazard, it maintains separate "possible" markings for each hazard type and removes these possibilities when contradicted by new percepts.
- Certainty Upgrade: When all but one adjacent cell is confirmed safe, any remaining unknown cell causing a breeze/stench is inferred to contain the hazard with certainty.
- Glitter effect: When the hunter has "Glitter", it will automatically know that there is gold where its is standing and do action "Grab" to grab the gold, then calculate the safest and shortest path to go to the initial cell (0,0).

### 4.2. Handling the Advanced Setting with Moving Wumpus

In the advanced setting, Wumpus creatures can move. This means a stench may appear or disappear in a location that was previously safe or dangerous. To handle this:

- The knowledge base does not permanently mark a cell as containing a Wumpus unless the inference is certain.
- When a stench disappears from a location that previously had it, the inference engine revises its hazard possibilities, removing outdated "possible Wumpus" facts.
- The reasoning is re-run after every percept change, ensuring that movement patterns of Wumpus are accommodated.

### 4.3. Decision-Making Integration

Once forward chaining updates the KB, the agent chooses actions based on the safest known cells. If no safe cell is available, the inference engine provides "possible hazard" cells, and the agent may choose to take a calculated risk depending on exploration needs.

## 5. KNOWLEDGE BASE UPDATE STRATEGY FOR MOVING WUMPUS

The knowledge base (KB) is designed to store and update logical facts about the environment based on percepts received at each step. Since the advanced setting allows the Wumpus to move, the update strategy must handle dynamic hazard locations without making irreversible incorrect assumptions.

### 5.1. Fact Storage

- Each fact is stored such as: safe, possible pit, possible wumpus, etc
- Facts are incrementally added when supported by percepts and removed when contradicted by new percepts.

### 5.2. Handling Moving Wumpus

When a stench is detected:

- All adjacent cells not known to be safe are marked as possible Wumpus locations.
- These "possible" markings are soft inferences — they remain in the KB only until contradictory evidence appears.

When a stench disappears:

- Any "possible Wumpus" facts for adjacent cells are retracted, because the Wumpus may have moved.
- The KB is immediately reprocessed with the updated percepts to infer the new set of possible Wumpus locations.

When a scream is heard: All Wumpus-related facts are removed from the KB, as it is known that the Wumpus in question has been eliminated.

### 5.3. Safe, Gold and Hazard Updates

- No Breeze: All adjacent cells are marked as safe from pits, and "possible_pit" facts for those cells are removed.
- No Stench: All adjacent cells are marked as safe from Wumpus, and "possible_wumpus" facts for those cells are removed.
- Certainty Inference: If only one unknown cell could explain a breeze or stench, it is upgraded to a confirmed hazard, replacing the "possible" marking.

### 5.4. Consistency Maintenance

At each turn:

- Percepts are received.
- The KB removes outdated facts (due to hazard movement or percept changes).
- Forward chaining is run to re-infer all possible and certain hazards.
- The updated KB is used for planning the next safe or strategic move.

This update strategy ensures that the agent can adapt to a changing world with moving hazards, keeping its beliefs consistent and avoiding outdated dangerous assumptions.

## 6. PLANNING ALGORITHM DESCRIPTION

The agent uses a goal-driven planning strategy that integrates its updated knowledge base with a pathfinding algorithm to decide the next sequence of actions.

### 6.1. Safe Exploration Priority

The planning process begins by identifying the set of safe target cells from the knowledge base. A cell is considered safe if:

- It is explicitly marked as safe by the inference engine.
- It is not marked as a possible or confirmed hazard (pit or Wumpus).

Safe cells are prioritized over uncertain cells to minimize risk.

### 6.2. Pathfinding with A*

When the agent needs to move to a target cell, it uses the A* search algorithm to compute the shortest path:

- Start Node: The agent's current position.

- Goal Node: The selected target cell (nearest unvisited safe cell, gold location, or return to start).
- Heuristic:  Manhattan distance to the goal.
- Cost: Number of moves required.

The result of A* is a sequence of waypoints. These are converted into low-level actions (turns and moves) based on the agent's current orientation.

### 6.3. Handling No Safe Adjacent Cells

If no safe cells are available:

- The agent considers cells marked with only "possible" hazards (i.e., not confirmed).
- It chooses the least risky cell based on surrounding KB facts.
- A* is used to plan a path toward that cell if exploration is deemed necessary.

### 6.4. Goal-Specific Planning

- Gold Collection: If glitter is perceived, the agent grabs the gold immediately. A* is then used to plan a safe route back to the starting position (0,0) to climb out.
- Wumpus Hunting: If the KB confirms a Wumpus location and the agent has an arrow, it plans a path to a cell from which it can safely shoot.
- Return Home: Once gold is collected or no further exploration is safe, A* plans the path back to (0,0).

### 6.5. Dynamic Replanning

Since the Wumpus may move and new percepts may change hazard inference:

- The agent replans after every action.
- If a new path becomes safer or shorter due to updated KB facts, the old plan is discarded and replaced.

This integration of logical inference for safety with A* search for movement efficiency ensures that the agent can operate both cautiously and effectively in a partially observable, changing environment.

## 7. ADVANCED IMPLEMENTATION FEATURES

### 7.1. Dynamic Wumpus Movement System

- Real-time knowledge base updates
- Adaptive agent behavior
- Complex multi-threat scenarios
- Performance impact analysis

### 7.2. Efficient Turn Optimization

- Minimal turn sequence calculation
- Direction efficiency algorithms
- Performance improvement metrics
- User experience enhancement

### 7.3. Scalable Grid Architecture

- Dynamic asset scaling
- Performance optimization for large grids
- Memory management
- Rendering efficiency

### 7.4. Audio-Visual Integration

- Immersive sound effects
- Dynamic lighting systems
- Responsive UI design

## 8. EXPERIMENTS

*Table 1. Random Agent baseline and KB Agent comparison*

| Comparison (30 games in 5x5 map) | | |
|---|---|---|
| Ceteria/Agent | Random Agent | KB Agent |
| Success rate | 3.33% (1/30) | 86.67% (26/30) |
| Avg score | -964.03 | 823.36 |
| Avg step | 31 | 36.23 |

**Evaluation:**

- Our agent significantly outperforms the random agent in success rate, proving the effectiveness of logical inference and planning.

- The random agent wastes many moves and often dies early due to lack of hazard awareness.

- The knowledge base and inference engine reduce unnecessary exploration, allowing our agent to complete objectives in fewer steps.

**Conclusion:**

Integrating propositional logic, inference rules, and dynamic knowledge base updates enables the agent to make safer and more efficient decisions, increases the chance of winning when compared to random exploration.

## 9. VIDEO DEMO

Demo video URL

## 10. PLANNING AND TASK DISTRIBUTION

*Table 2. Planning and members contribution*

| Tasks | Name | Completion Rate |
|---|---|---|
| Problem formulation and knowledge base design | Trần Thọ + Nguyễn Tuấn An | 100% |
| Environment simulator implementation | | |
| Inference engine implementation | Nguyễn Tuấn An | 100% |
| Planning module implementation | | |
| Hybrid agent integration | Trần Văn Huy + Phan Trung Hiếu | 100% |
| Advanced moving Wumpus module | | |
| Video demo | Phan Trung Hiếu | 100% |
| Graphic visualization | Trần Thọ + Phan Trung Hiếu | 100% |
| Console visualization | Trần Văn Huy + Nguyễn Tuấn An | 100% |
| Report quality and analysis | Group Contributed | 100% |

## REFERENCES


[1] P. Community, "Pygame Tutorial," [Online]. Available: https://www.pygame.org/.

[2] fabioo29, "Graphic Reference," [Online]. Available: https://github.com/fabioo29/ai-wumpus-world.

[3] GeeksforGeeks, "The Wumpus World Description - AI," [Online]. Available: https://www.geeksforgeeks.org/machine-learning/ai-the-wumpus-world-description/.

[4] U. o. California, "Logical Agent," [Online]. Available: https://aima.cs.berkeley.edu/newchap07.pdf.