A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240030016 |
| **NAME** | BARATH S |
| **PROJECT TITLE** | A knowledge-based AI simulation of Wumpus World using Python and Tkinter |
| **DATE OF SUBMISSION** | 29/10/2025 |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

Artificial Intelligence (AI) is a field of computer science focused on creating systems that can think, learn, and act intelligently. AI systems mimic human cognitive processes such as reasoning, problem-solving, and decision-making. These capabilities are achieved through intelligent agents that perceive their environment and act to achieve specific goals. Game-playing has always been a benchmark for testing AI strategies. Among various games, chess represents a complex decision-making environment requiring strategic reasoning and planning. AI in chess involves searching through possible moves, predicting opponent responses, and selecting the most optimal move. Building a chess-playing AI demonstrates how intelligent agents can operate in adversarial conditions where each move impacts the opponent’s future choices. It helps understand core AI topics like problem-solving agents, heuristic search, and adversarial game trees. This project aims to design and implement a Python-based Chess AI that uses the Minimax algorithm with Alpha–Beta pruning to make intelligent, strategic moves against a human player.

**PROBLEM STATEMENT**

Design and implement a knowledge-based AI agent that explores and survives a Wumpus World environment. The agent should sense its surroundings, update an internal knowledge base, reason about safe/unsafe locations, plan actions, and interact with a graphical interface built in Tkinter.

**GOAL**

**Expected Result:**  
 A functional **knowledge-based Wumpus World AI agent** that intelligently navigates the environment, analyzes percepts, infers safe and unsafe locations, and successfully finds the gold and escapes the cave using logical reasoning and decision-making strategies.

.

**Possibilities:**

* Explore more advanced AI reasoning methods such as **probabilistic inference**, **Bayesian reasoning**, or **reinforcement learning** to enhance the agent’s decision-making accuracy.
* Introduce **adaptive difficulty levels** where the environment complexity (number of pits, grid size, or Wumpus behavior) changes based on the agent’s success rate.
* Implement **self-learning or memory-based mechanisms** enabling the agent to improve performance over multiple runs.
* Extend the simulation to include **multi-agent coordination** or **competitive exploration** between agents.

**THEORETICAL BACKGROUND**

**Wumpus World:**The Wumpus World is a classic problem in Artificial Intelligence that serves as a benchmark for testing knowledge-based agents and logical reasoning systems. It is a grid-based environment consisting of rooms that may contain hazards like pits and a Wumpus (a dangerous creature), along with a piece of gold that the agent must find and retrieve.

The agent perceives limited information about its surroundings through percepts such as:

* **Breeze –** indicates a pit in one of the adjacent cells.
* **Stench –** indicates the Wumpus is in a nearby cell.
* **Glitter** – indicates the presence of gold in the current cell.
* **Bump** – occurs when the agent hits a wall.
* **Scream** – occurs when the Wumpus is killed.

Using these percepts, the agent maintains and updates a Knowledge Base (KB) containing logical statements about the environment. It applies inference rules to deduce which cells are safe to move into and which may contain hazards.

This problem illustrates the concepts of:

* Knowledge Representation (storing information as facts and rules),
* Logical Inference (deriving new knowledge from existing facts), and
* Decision Making Under Uncertainty (acting with incomplete information).

By integrating these AI principles, the Wumpus World demonstrates how an intelligent agent can operate effectively even with partial perception and uncertainty—forming the foundation of knowledge-based AI systems.

**Algorithm:**  
 **Inference and Decision-Making:**  
The Wumpus World agent uses logical reasoning to make decisions based on percepts. It applies **knowledge-based inference** to identify safe and unsafe cells using predefined logical rules. The agent updates its **knowledge base** after each move and selects the next action—move, grab, shoot, or climb—based on inferred safety.  
To improve efficiency, the agent prunes unnecessary or unsafe paths, reducing computation while maintaining accuracy.

**Literature Survey:**

*  **Stanford AI Laboratory (1970s):** Introduced the Wumpus World as a benchmark problem for testing logic-based AI agents and knowledge representation systems.
*  **Russell and Norvig (Artificial Intelligence**: A Modern Approach, 2010): Described the Knowledge-Based Agent model and reasoning techniques for environments like the Wumpus World.
*  **GeeksforGeeks (2020)** and AI tutorials by Code Bullet and Sebastian Lague provide practical insights into implementing logical inference and environment simulation using Python.

**Justification for Choosing the Algorithm:**  
 The knowledge-based inference approach provides an effective balance between intelligence and simplicity, making it ideal for a mini project. It allows the agent to make logical decisions using percepts and reasoning rules while keeping computation manageable in Python. This method clearly demonstrates AI reasoning and decision-making without requiring complex learning algorithms.

**ALGORITHM EXPLANATION WITH EXAMPLE**

**Knowledge-Based Inference Algorithm Steps:**

1. The agent perceives inputs (Breeze, Stench, Glitter, etc.) from the environment.
2. It updates its Knowledge Base (KB) with these percepts.
3. Using logical rules, it infers which adjacent cells are safe, unsafe, or unknown.
4. The agent selects the next action — Move, Grab, Shoot, or Climb — based on inferred safety.
5. The process repeats until the agent finds the gold or exits safely**.**

**Example:**If the agent feels a Breeze in a cell, it infers that one of the nearby cells may contain a Pit.  
If another adjacent cell has no Breeze, it concludes that the common neighboring cell is safe, and moves there confidently.

**IMPLEMENTATION AND CODE**

**Language Used:** Python

**Libraries:**

* **Tkinter** – for creating the graphical user interface (GUI).
* **Random** – for generating random placement of the Wumpus, pits, and gold.
* **Time** – for controlling simulation speed and animation delays.

**SOURCE CODE:**

import tkinter as tk

import time

from environment import Environment, GRID\_SIZE

from agent import Agent

CELL\_SIZE = 100

class Game:

def \_\_init\_\_(self, master):

self.master = master

self.master.title("Wumpus World AI Simulation 🧠")

self.canvas = tk.Canvas(master, width=GRID\_SIZE\*CELL\_SIZE, height=GRID\_SIZE\*CELL\_SIZE, bg="#F8F6E7")

self.canvas.pack()

self.env = Environment()

self.agent = Agent(self.env)

self.draw\_grid()

self.draw\_world()

self.update\_gui()

self.master.after(1000, self.step)

def draw\_grid(self):

for i in range(GRID\_SIZE):

for j in range(GRID\_SIZE):

x1, y1 = j \* CELL\_SIZE, i \* CELL\_SIZE

x2, y2 = x1 + CELL\_SIZE, y1 + CELL\_SIZE

self.canvas.create\_rectangle(x1, y1, x2, y2, outline="black")

def draw\_world(self):

self.canvas.delete("all")

self.draw\_grid()

# Pits

for pit in self.env.pits:

self.draw\_emoji(pit, "🕳️")

# Wumpus

self.draw\_emoji(self.env.wumpus, "👹")

# Gold

self.draw\_emoji(self.env.gold, "💰")

# Agent

self.draw\_emoji(self.agent.pos, "🤖")

def draw\_emoji(self, pos, emoji):

x, y = pos[1]\*CELL\_SIZE + CELL\_SIZE//2, pos[0]\*CELL\_SIZE + CELL\_SIZE//2

self.canvas.create\_text(x, y, text=emoji, font=("Arial", 40))

def step(self):

if self.agent.found\_gold:

print("🎉 AI Agent completed the mission!")

return

percepts = self.agent.perceive\_and\_update()

print(f"🤖 At {self.agent.pos} perceives {percepts}")

next\_pos = self.agent.choose\_next\_move()

self.agent.visited.add(tuple(next\_pos))

self.agent.pos = next\_pos

if next\_pos == self.env.wumpus:

print("💀 The agent was eaten by the Wumpus!")

self.draw\_emoji(next\_pos, "💀")

return

if next\_pos in self.env.pits:

print("🕳️ The agent fell into a pit!")

self.draw\_emoji(next\_pos, "💀")

return

self.draw\_world()

self.master.after(1000, self.step)

def update\_gui(self):

self.draw\_world()

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = Game(root)

root.mainloop()

import random

GRID\_SIZE = 4

class Environment:

def \_\_init\_\_(self):

self.gold = self.random\_cell([])

self.wumpus = self.random\_cell([self.gold])

self.pits = [self.random\_cell([self.gold, self.wumpus]) for \_ in range(3)]

self.agent\_pos = [0, 0]

def random\_cell(self, exclude):

while True:

pos = [random.randint(0, GRID\_SIZE - 1), random.randint(0, GRID\_SIZE - 1)]

if pos not in exclude:

return pos

def get\_percepts(self, pos):

"""Return percepts based on adjacent cells"""

percepts = []

adj = self.adjacent\_cells(pos)

if any(p in adj for p in self.pits):

percepts.append("breeze")

if any(p == self.wumpus for p in adj):

percepts.append("stench")

if pos == self.gold:

percepts.append("glitter")

return percepts

def adjacent\_cells(self, pos):

x, y = pos

moves = [(0,1),(1,0),(-1,0),(0,-1)]

valid = []

for dx, dy in moves:

if 0 <= x+dx < GRID\_SIZE and 0 <= y+dy < GRID\_SIZE:

valid.append([x+dx, y+dy])

return valid

import random

from environment import GRID\_SIZE

class Agent:

def \_\_init\_\_(self, env):

self.env = env

self.pos = env.agent\_pos

self.safe = {tuple(self.pos)}

self.visited = {tuple(self.pos)}

self.knowledge = {}

self.found\_gold = False

def get\_possible\_moves(self):

x, y = self.pos

moves = [(0,1),(1,0),(-1,0),(0,-1)]

valid = []

for dx, dy in moves:

if 0 <= x+dx < GRID\_SIZE and 0 <= y+dy < GRID\_SIZE:

valid.append([x+dx, y+dy])

return valid

def perceive\_and\_update(self):

percepts = self.env.get\_percepts(self.pos)

self.knowledge[tuple(self.pos)] = percepts

if "glitter" in percepts:

self.found\_gold = True

print("🏆 Gold found at", self.pos)

adj = self.env.adjacent\_cells(self.pos)

if not percepts:

for cell in adj:

self.safe.add(tuple(cell))

else:

if "breeze" in percepts or "stench" in percepts:

pass # uncertain cells remain unknown

return percepts

def choose\_next\_move(self):

adj = self.env.adjacent\_cells(self.pos)

for cell in adj:

if tuple(cell) in self.safe and tuple(cell) not in self.visited:

return cell

for cell in adj:

if tuple(cell) not in self.visited:

return cell

return random.choice(adj)

from environment import Environment

from agent import Agent

def run\_simulation():

env = Environment()

agent = Agent(env)

print("🏁 Starting Wumpus World Simulation")

print(f"Wumpus at {env.wumpus\_pos}, Gold at {env.gold\_pos}, Pits at {env.pits}\n")

while agent.alive and not agent.has\_gold:

next\_move = agent.next\_move()

if next\_move is None:

print("😕 No more safe moves. Exiting...")

break

agent.move(\*next\_move)

if agent.has\_gold:

print("🎉 Agent won!")

elif not agent.alive:

print("💀 Agent lost!")

else:

print("🏳️ Agent stopped exploring.")

if \_\_name\_\_ == "\_\_main\_\_":

run\_simulation()

**A screenshot of a game

AI-generated content may be incorrect.OUTPUT**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a game

AI-generated content may be incorrect.**

**Explanation:**  
 The GUI displays a functional Wumpus World grid where each cell represents a part of the cave. After every agent move, the system updates percepts such as Breeze, Stench, or Glitter based on the agent’s position. The AI agent analyzes these percepts, updates its knowledge base, infers safe and unsafe cells, and chooses the next action logically — whether to move, grab the gold, shoot the Wumpus, or exit safely. The agent successfully avoids dangerous cells, makes reasoned decisions, and efficiently navigates the world to achieve its goal.RESULTS AND FUTURE ENHANCEMENT

**Results:**

* A fully functional knowledge-based Wumpus World AI agent capable of intelligent exploration and decision-making.
* Demonstrates logic-based reasoning, safe-path inference, and rational agent behavior in an uncertain environment.
* Correctly handles terminal conditions such as finding the gold, falling into a pit, being killed by the Wumpus, or exiting the cave safely.

**Future Enhancements:**

* Add adaptive difficulty by varying grid size, number of pits, or Wumpus behavior.
* Integrate machine learning or reinforcement learning to improve the agent’s reasoning and decision-making.
* Explore probabilistic inference or Bayesian networks for advanced hazard prediction.
* Enhance the GUI with features like step-by-step reasoning display, visual highlights for safe/unsafe cells, and a debug or hint mode for learning purposes.

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | **https://github.com/BARATH0202/Wumpus-World-AI-Project.git** |

**REFERENCES**

* Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th Ed.). Pearson Education.
* Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann.
* GeeksforGeeks. (2020). *Wumpus World in Artificial Intelligence.* Retrieved from <https://www.geeksforgeeks.org/wumpus-world-in-artificial-intelligence/>
* Sebastian Lague. (2018). *Coding Adventure: Wumpus World & AI Logic Systems* [YouTube Series].
* Wikipedia Contributors. (2024). *Wumpus World.* Retrieved from <https://en.wikipedia.org/wiki/Wumpus_World>