Designing an Adaptive Agent for the Wumpus World

Cesar Augusto Pulido Cuervo - 20222020048

Cristian David Romero Gil - 20222020138

Universidad Distrital Francisco José de Caldas

Designing an Adaptive Agent for the Wumpus World

Introduction

This document outlines the design of an autonomous adaptive agent operating in a simulated environment. The system is based on cybernetic principles and reinforcement learning, where the agent interacts with a Wumpus-like environment that features pits, a Wumpus enemy, gold, and other elements. The design process starts with a basic Q-learning approach and evolves towards more advanced methods such as Deep Q-Networks (DQN). For example, Mnih et al. Mnih, Kavukcuoglu, Silver, et al. (2015) provide a foundational description of how deep reinforcement learning can achieve human-level control in complex environments.

1. Functional Specifications

The system's functional specifications are divided into several key components:

Sensors:

- **Position and Orientation:** The agent retrieves its current grid position and orientation (e.g., North, East, South, West). Grid coordinates (x, y), where $0 \leq x, y \leq 3$ (discrete 4x4 space).
- **Perceptual Inputs:** The agent processes three perceptual signals:
  - *Breeze:* Binary signal (1 if adjacent to pits; 0 otherwise).
  - *Stench:* Binary signal (1 if adjacent to the Wumpus; 0 otherwise).
  - *Glitter:* Binary signal (1 if gold is in the current cell; 0 otherwise).

Actuators:

- **Movement:** The agent can move in the four cardinal directions, with movement constrained by the boundaries of the environment.

- **Shooting:** The agent can shoot an arrow in its current orientation (limited to one arrow per episode). If the arrow trajectory intersects the Wumpus, a significant reward is given and the Wumpus is removed from the environment.

- **Gold Collection:** The agent automatically collects gold when located in the same cell, triggering a reward.

Reward Function:

- **Step Cost:** Every action incurs a cost of -1 to promote efficiency.

- **Positive Rewards:**
  - Collecting gold yields a reward of +1000.
  - Successfully winning the episode (returning to the start with the gold) adds an extra +2000.

- **Penalties:**
  - Falling into a pit results in a -1000 penalty and terminates the episode.
  - Being caught by the Wumpus similarly incurs a -1000 penalty and ends the episode.
  - An unsuccessful shooting action incurs a penalty of -50.

1.2 Use Cases

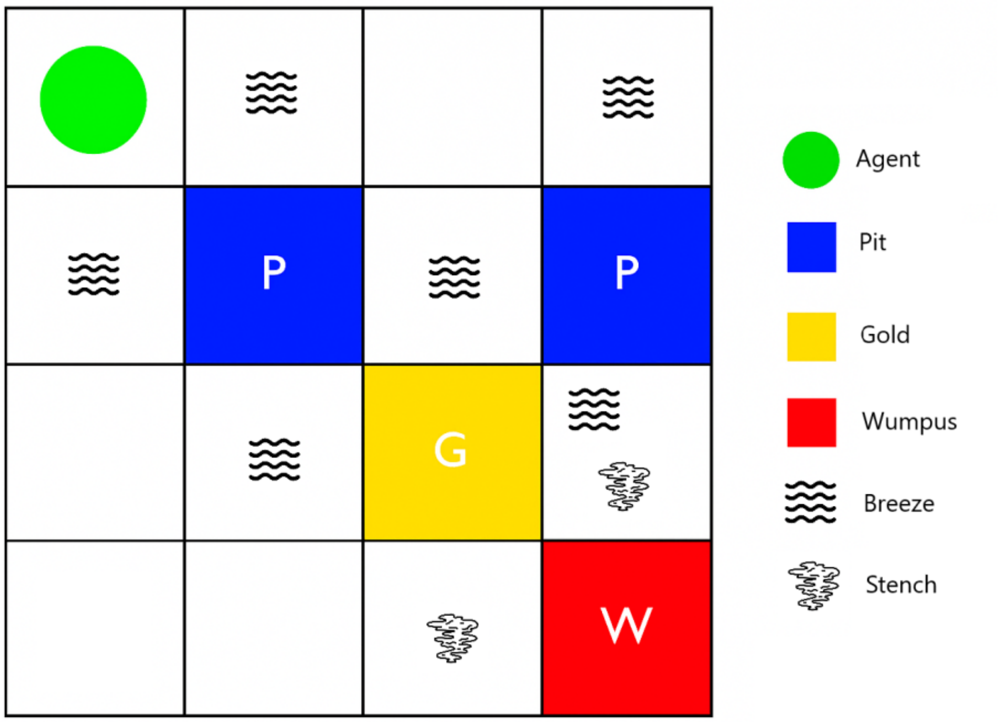

Figure 1.

Illustration of Basic Exploration Use Case

Use Case 1: Basic Exploration.

- Input: Agent starts at $(0, 0)$ with initial percepts (breeze=0, stench=0, glitter=0).
- Process:
  - The agent moves east ($\rightarrow$) using an ϵ-greedy policy (high ϵ initially).
  - The environment updates the agent's position to $(1, 0)$ and returns breeze=1 (indicating adjacent pits).
- Output: Reward -1 (step cost), new percepts logged, Q-values updated.

Use Case 2: Adaptive Decision-Making.

- Input: Agent detects stench=1 at $(2, 3)$ (adjacent to Wumpus at $(3, 3)$).
- Process:

  - The agent shoots east ($\rightarrow$) based on Q-network predictions.
  - The arrow trajectory intersects the Wumpus, removing it and granting a +500 reward.

- Output: Updated environment state (no Wumpus), reward +500, policy adjusted via experience replay.

3. Frameworks

This section outlines the potential frameworks and tools considered for implementing the Wumpus World environment and training the cybernetic agent.

Gymnasium. *Gymnasium* is a modern reinforcement learning (RL) environment toolkit, successor of the original OpenAI Gym. It offers a standard API for building custom environments and provides compatibility with popular RL libraries such as Stable-Baselines3 Terry et al. (2023). Its modular design makes it ideal for representing the grid-based Wumpus World environment with custom observation and action spaces.

- Allows integration with RL pipelines and rendering through custom methods.
- Supports vectorized environments for batch learning in future expansions.

PyTorch. *PyTorch* is a widely used deep learning library that provides flexibility and performance. In this project, PyTorch is used to build custom deep Q-networks (DQN), allowing full control over the architecture, activation functions, and training dynamics Paszke et al. (2019).

- Enables implementation of custom neural networks for value function approximation.
- Supports GPU acceleration for training efficiency.
- Integrated seamlessly with Gymnasium-style environments.

Pygame. *Pygame* is used for visual rendering of the grid environment Pygame Community (2023). It enables the creation of a custom 2D view of the agent, pits, gold, and Wumpus, making it easier to debug and observe learning behavior.

- Lightweight 2D graphics library.
- Ideal for educational and simulation environments.

Timeline

The next timeline defines how the learning approach can evolve to a DQN RL.

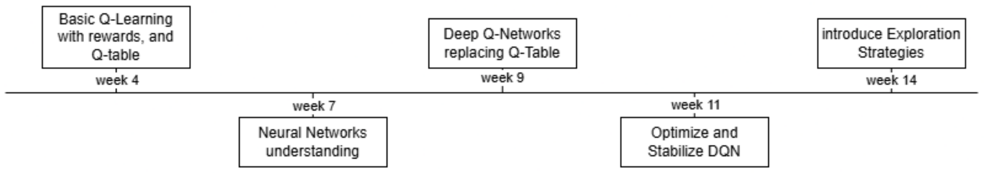
Basic Q-Learning with rewards, and Q-table
week 4
week 7
Neural Networks understanding
Deep Q-Networks replacing Q-Table
week 9
week 11
Optimize and Stabilize DQN
introduce Exploration Strategies
week 14

Figure 2.

Timeline of RL evolution

The progression from basic Q-learning to advanced DQN methods unfolds over a flexible timeline that supports the order of implementations of algorithms of reinforcement learning. As limitations of tabular methods emerge, week 7 is dedicated to developing a solid understanding of neural networks, preparing for their integration into reinforcement learning systems. Then, this foundation enables the transition to DQN, where neural networks replace the Q-table, allowing for scalable learning in more complex environments. Ultimately, optimizing and stabilizing DQNs using techniques such as experience replay and target networks, which are critical to preventing divergence and improving learning efficiency, and introducing advanced exploration strategies.

References

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. Retrieved from `https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`

Pygame Community. (2023). *Pygame: Python game development library.* `https://www.pygame.org`. (Version 2.5.0)

Terry, J., Chan, K., Tucker, G., Zholus, A., Gao, X. B., Agarwal, R., et al. (2023). *Gymnasium: A standard api for reinforcement learning environments.* `https://gymnasium.farama.org`. (Farama Foundation)