جامعة الملك سعود
King Saud University
College of Computer and Information Sciences

Project Title: GameHub

Course: CSC 361: Artificial Intelligence

Instructor: Mohamed Maher Ben Ismail

Members:

1. Mohammed Alwanis 444100734
2. Alwaleed Almutairi 444101489
3. Majed AlDajani 444101223

1. Project Summary

We have developed GameHub a unified Python application that integrates three distinct game environments to demonstrate various Artificial Intelligence problem-solving techniques. The project includes:

1. **Sudoku:** A Constraint Satisfaction Problem (CSP) solver comparing and Simulated Annealing.
2. **Connect 4:** An adversarial game agent utilizing Minimax with Alpha-Beta Pruning and heuristic evaluation.
3. **8-Puzzle**: A pathfinding solver comparing Greedy Best-First Search and A* Search. The application features a centralized GUI (GameHubApp) that allows users to launch each module, select difficulty levels, and visualize the AI's decision-making process in real-time.

2. Project Description

The primary motivation for this project was to explore how different AI paradigms approach problem-solving and to practically apply what we learned in the CSC 361 course. We specifically wanted to demonstrate how **CSP** makes solvers strong in puzzles like Sudoku, and how the **Minimax algorithm** can create an effectively unbeatable agent in games like Connect 4. , Solve The 8 Puzzel Problem Using Two Algorithm And Compare Between Them And Show The Step of Solution (Path) Can go Next Step And Return Preveious Step .Ultimately, this project highlights how these approaches differ and why it is critical to use the right algorithm in the right place to achieve optimal performance.

(b) Background

Games serve as ideal AI testbeds due to their defined rules and vast combinatorial state spaces. This project explores the following classical AI paradigms:

- **Constraint Satisfaction (CSP): Modeling Sudoku cells as variables constrained by uniqueness rules. We solve this using Backtracking (depth-first search) optimized by the MRV Heuristic (Minimum Remaining Values), which prioritizes the most restricted variables to prune the search tree.**

- **Simulated Annealing: A stochastic local search algorithm that avoids getting stuck in local optima by probabilistically accepting worse states during the cooling process.**

- **Greedy Search: That Take Only Huristic(Manhaten distance) As A Cost Then Will Added To Priority Queue To Know Which Path Have Lowest Cost ,, That Take More Time And More Step Number Than A* ,, This is Our Reason To Compare This Two Algorithm.**

- **A* Search: That Take Huristic(Manhaten distance) + Cost(Depth) As A Cost Then Will Added To Priority Queue To Know Which Path Have Lowest Cost ,, That Take Lowest Time And Lowest Step Number Than Greedy ,, This is Our Reason To Compare This Two Algorithm.**

(c) Proposed Approach

We implemented the system using **Python** and **Tkinter** for the Graphical User Interface. The system is modular, controlled by a main hub (GameHubApp.py).

1. Sudoku Module

- **Architecture:** We implemented a SudokuGame class that functions as a generator to visualize steps.
- **Algorithms:**
    - *Smart CSP (MRV):* We implemented the Minimum Remaining Values heuristic, which selects the cell with the fewest legal moves first to prune the search tree early.
    - *Simulated Annealing:* A local search probabilistic algorithm using a cooling schedule and Metropolis acceptance criterion to escape local optima.

2. Connect 4 Module

- **Architecture:** The MinimaxAgent determines moves for the AI player.
- **Algorithms:**
    - ***Minimax with Alpha-Beta Pruning**:* We implemented recursive functions that prune branches to speed up the search significantly.
    - Heuristic Evaluation: We created a custom evaluation function that scores board states based on "windows" of 4 slots. It prioritizes center control and consecutive pieces while penalizing opponent threats.
    - Difficulty Levels: We controlled difficulty by adjusting the search depth (Easy=2, Medium=4, Hard=6).

    - **Adaptive Minimax:** We implemented the adaptive Minimax to play with depth = 6 but with more complex heuristic and adapting ability after each game it analyze the critical positions and other features and enhance the heuristic value.

- Also it will gently pull back to default to protect the values from being overreacted for some moves with decay = 0.02
- Features of the heuristic are 18: Center column control, adjacent center control, bottom row control, height penalty, edge penalty, horizontal threats, vertical threats, diagonal threats, threat count, open twos, double threats, mobility.
- It does have 42% winning rate against the Hard level after 100 games of training
- **Time complexty:** O(b^d/2) for **BOTH** with a more time for the adaptive since it will record the game and train on it
- **Space complexity:** O(d) for **BOTH** with the record space for the adaptive

3. 8-Puzzle Module

- **Architecture:** The Solver Will Provide Two Solution With Statistic And Paths A*,Greedy And User Can See Two Algorithm in One GUI And Can Compare Between Them.
- **Algorithms:**
  - *A* Search:* Minimizes cost [f(n) = g(n) + h(n)]. Where g(n) : Cost (Depth) , h(n) : Manhateen Distance .
  - *Greedy Best-First Search:* Minimizes heuristic (h(n)) only where h(n) : Manhateen Distance.

(d) Results and Discussion

Sudoku Analysis:

We compared the three algorithms on "Hard" difficulty puzzles.

- *Smart CSP:* Was the fastest, consistently solving puzzles with zero or very few backtracks due to the MRV heuristic.
- *Simulated Annealing:* Was Bad because it like a bruteforce trying and trying until it find the souletion/

Connect 4 Analysis:

- We tested the AI against a Human player.
- *Performance:* At Depth 6 (Hard), the AI is very difficult to beat. It successfully blocks opponent wins (score -= 120 in our code) and sets up its own traps.
- *Latency:* [UR INPUT NEEDED: Did the game lag at Depth 6? If it was slow, mention that this shows the trade-off between intelligence (depth) and speed.]

8-Puzzle Analysis:

- We ran the "Compare" mode in the GUI.
- *A* vs Greedy:* [A* is faster than Greedy When We Have A lot Of Steps (HighDepth) , Find Solution Faster Than Greedy ,,, When We Have Complex Arrange Of 8 puzzel The A* Will Be Better OtherWise A* And Greedy Will Be Approxmitlly Simillar]

- ***Link Of Demo Video : Click Here***
- ***Source Code Github: Click Here***