Gold Mine Game Design

Project Report

Submitted by

K.INDU MEGHANA - AP22110010540

P.VINEETHA - AP22110010580

A.MAHITHA - AP22110010576

Under the Supervision of

Mr. Sarvani Anandarao

Assistant Professor

Department of Computer Science and Engineering
SRM University-AP

In partial fulfilment for the requirements of the project

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY-AP

NEERUKONDA

MANAGALAGIRI - 522503

ANDHRA PRADESH, INDIA

APRIL - 2025

Problem statement

Given a gold mine of n*m dimensions. Each field in this mine contains a positive integer 0 to 10, which is the amount of gold in tons. Initially the miner is at the first column but can be at any row. He can move only

- right
- right up
- right down

that is from a given cell, the miner can move to the cell diagonally up towards the right or right or diagonally down towards the right. Find out maximum amount of gold he can collect.

Methodology

1. **Grid Representation**

The mine is represented as a 2D array gold[n][m].

2. **Dynamic Programming for Logic Validation**

To simulate and validate the logic of movement and gold accumulation, a DP approach is commonly used:

Let dp[i][j] represent the maximum gold that can be collected from cell (i, j) to the end.

Recurrence Relation:

$$
\begin{gathered}
dp[i][j] = gold[i][j] + \max(\\
dp[i-1][j+1] \text{ if } i-1 >= 0, \\
dp[i][j+1], \\
dp[i+1][j+1] \text{ if } i+1 < n \\
)
\end{gathered}
$$

3. **Propositional Logic Encoding**

Each cell can be represented as a propositional variable, like P_ij meaning the miner is at cell (i,j).

Logical Rules:
The miner can move only to one of the allowed next positions:

P_ij → (P_i(j+1) ∨ P_(i-1)(j+1) ∨ P_(i+1)(j+1))

The miner must be in exactly one cell in each column:

∀j, ∃!i: P_ij is True

Transitions between cells (movement constraint):

P_ij → (¬P_i(j-1) ∧ ¬P_(i-1)(j-1) ∧ ¬P_(i+1)(j-1))

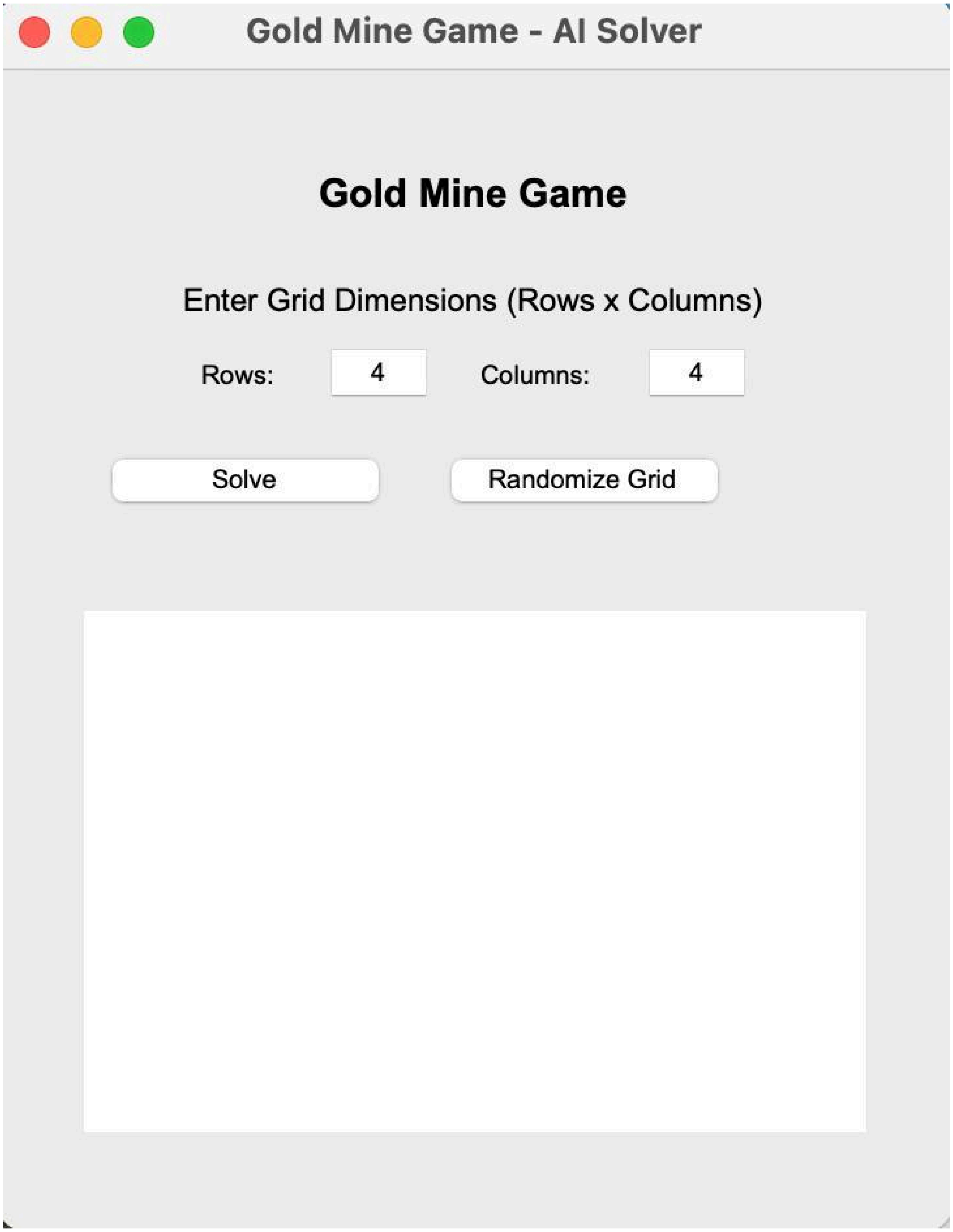
This can be reduced to CNF and solved using SAT solvers to simulate propositional reasoning over paths.

Code Logic

```python
def get_max_gold(gold):
    n = len(gold)
    m = len(gold[0])

    dp = [[0 for _ in range(m)] for _ in range(n)]
    path = [[[] for _ in range(m)] for _ in range(n)]

    for col in range(m-1, -1, -1):
        for row in range(n):
            if col == m - 1:
                dp[row][col] = gold[row][col]
                path[row][col] = [(row, col)]
            else:
                right = dp[row][col+1]
                right_up = dp[row-1][col+1] if row > 0 else 0
                right_down = dp[row+1][col+1] if row < n-1 else 0

                max_gold = max(right, right_up, right_down)

                if max_gold == right:
                    path[row][col] = [(row, col)] + path[row][col+1]
                elif max_gold == right_up:
                    path[row][col] = [(row, col)] + path[row-1][col+1]
                else:
                    path[row][col] = [(row, col)] + path[row+1][col+1]

                dp[row][col] = gold[row][col] + max_gold

    max_gold = 0
    best_path = []
    for i in range(n):
        if dp[i][0] > max_gold:
            max_gold = dp[i][0]
            best_path = path[i][0]

    return max_gold, best_path
```

Output Screenshots

Before Randomizing:



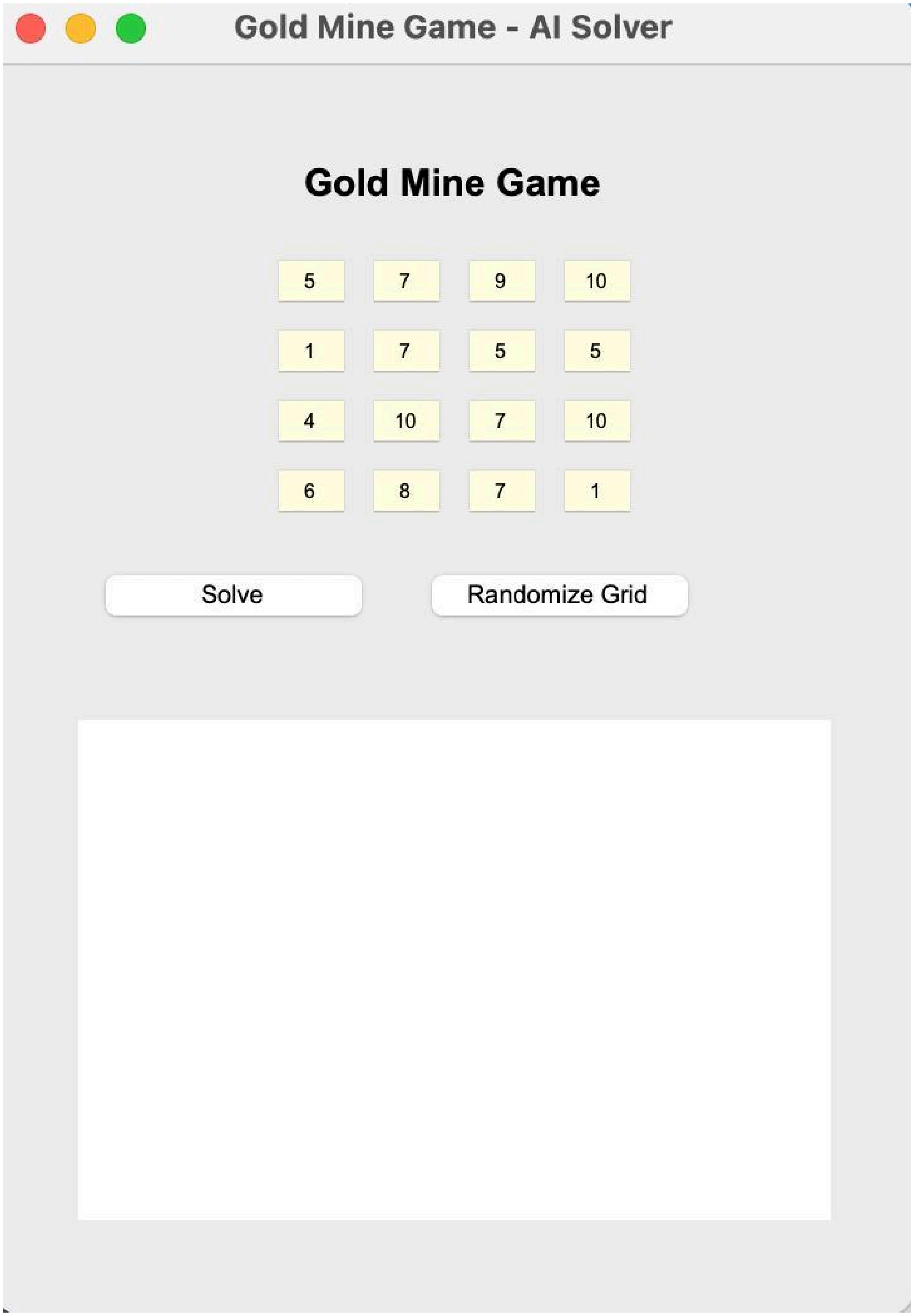
Gold Mine Game - AI Solver

Gold Mine Game

Enter Grid Dimensions (Rows x Columns)

Rows: 4 Columns: 4

After Randomizing:

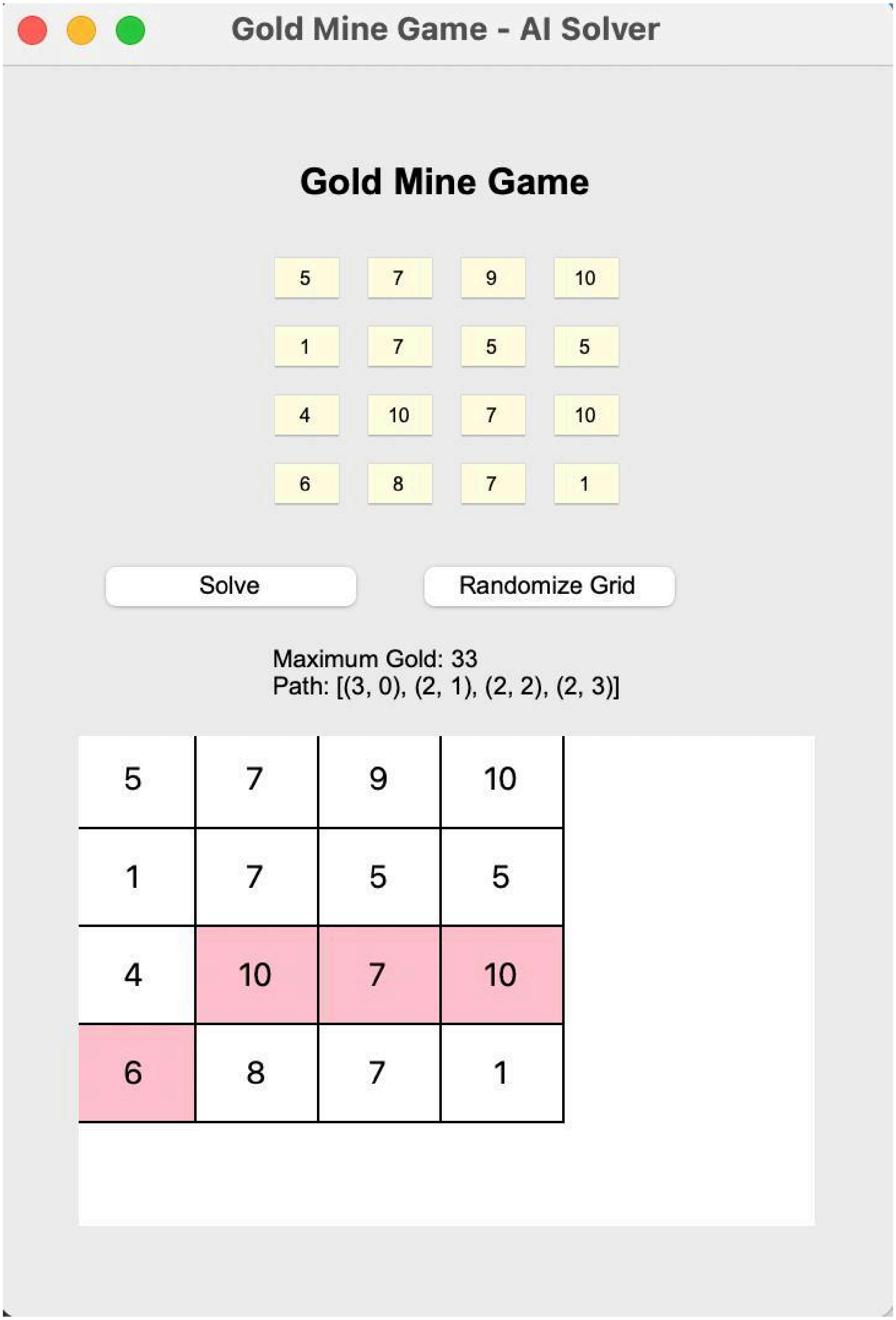


Gold Mine Game - AI Solver

Gold Mine Game

5	7	9	10
1	7	5	5
4	10	7	10
6	8	7	1

After Solving:



Gold Mine Game

5	7	9	10
1	7	5	5
4	10	7	10
6	8	7	1

```
Maximum Gold: 33
Path: [(3, 0), (2, 1), (2, 2), (2, 3)]
```

5	7	9	10
1	7	5	5
4	10	7	10
6	8	7	1