# GROUP 17

Hoàng Trần Nhật Minh
20204883

Nguyễn Hoàng Phúc
20204923

Lý Nhật Nam
20204886

Lê Thảo Anh 20200054

Đỗ Xuân Phong 20219701

Contributors: Hoàng Trần Nhật Minh
Nguyễn Hoàng Phúc
Lý Nhật Nam
Lê Thảo Anh
Đỗ Xuân Phong

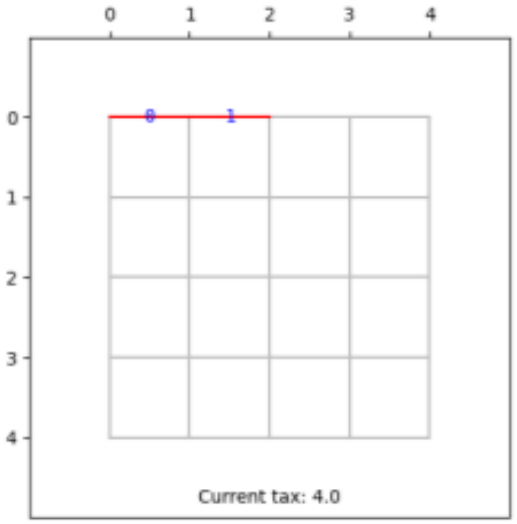
Group 17 presents

# THE PENNILESS PILGRIM RIDDLE

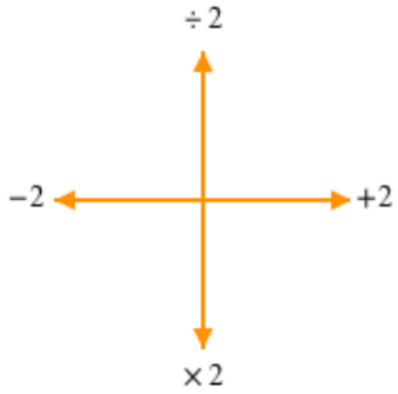
An Introduction to AI group project

# Problem description

- Inspired by this TED-ED video
- There's a strange tax in this city
- Can you find your way to goal without having to pay?



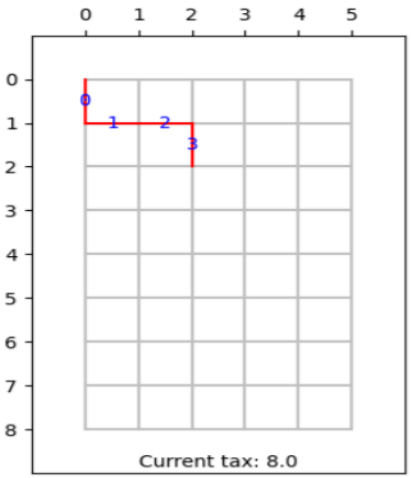

The initial state of the game in the video



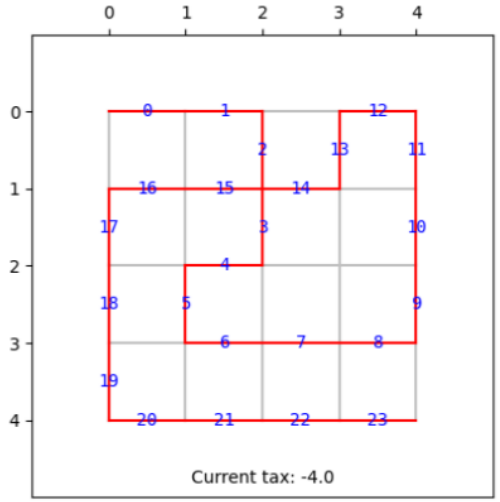

The tax fluctuate differently based on direction

# Problem Formulation

- Single-state problem, solving by searching
- Randomized initial board sizes: 4 to 8 each side
- 4 available actions: U, D, L, R
- Output: time complexity, space complexity




One of the randomized initial states




Example solution

# Depth-first Search (DFS)

- Depth-first Search vs. Breadth-first Search:

Averaging/assuming two branches per node, c = m*n nodes with depth d = c/2

| | DFS | BFS |
|---|---|---|
| Principle | Expand nodes of the same branch, until depth reached. | Expand all nodes of the same depth. |
| Space complexity (Theory) | $O(2 \times c)$ | $O(2^{d+1})$ |
| Space complexity (Practice) | $O(2 \times (m-2) \times (n-1))$ | $O(2^{d+1})$ |
| Time complexity (Theory) | $O(2^c)$ | $O(2^{d+1})$ |
| Time complexity (Practice) | $O(2^{(m-2)(n-1)})$ | $O(2^{d+1})$ |

- However, DFS never reaches worst-case for space and time complexity, no guarantee for BFS

⇒ DFS more suitable

# DFS Implementation

- t_limit: 10 seconds
- Simple recursive DFS implementation

```
#!/pseudo/code
func dfs(curr) -> None:
  if time_limit_reached(): exit();
  for move in curr.available_moves_list():
    temp = copy(curr)
    temp.move_to(move)

    if temp.current_pos == START.current_pos and \
       temp.current_tax >= START.current_tax:
      save_path();
      exit();

    temp.path = copy(curr.path);
    dfs(temp);

main:
  dfs(GOAL);
```

- Difficulties: No setbacks aside from needing decent CPU and plenty of RAM.

# A-star Search (A*)

- Heuristic problem: none actually worked for majority of boards => “bending” definition
- Implementing code
- Difficulties in implementing:

- Finding a solution to the heuristic problem:

For the sake of simple heuristic, understandable in a peek, without over-arching presentation time limit, we settled on a simple heuristic instead, accepting that solution is imperfect.

- Dropping optimal path searching:

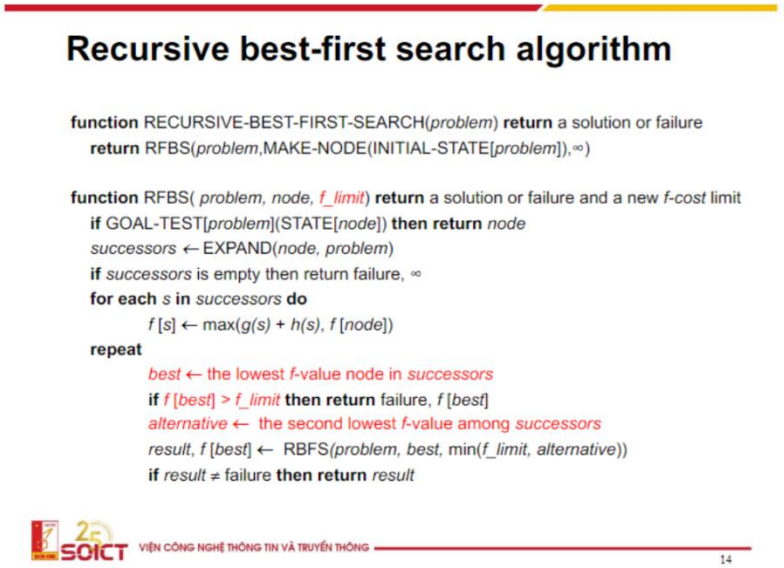
Cannot design true heuristic within the time available

- Simple formula, space-friendly, not optimal, can be pessimistic, guides A* to solution

# Recursive Best-first Search (RBFS)

- Applying RBFS algorithm
- Some difficulties:

- Estimating heuristic unlikely to be correct because non-uniformly changing path cost
- Can be stuck in infinite loop
- Not good enough heuristic leads to skipping some branches, not finding the ideal solution
- Difficult balancing time and space complexity
- Data samples may be unsolvable, not easy to prove



**Recursive best-first search algorithm**

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
  return RFBS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

function RFBS( problem, node, f_limit) return a solution or failure and a new f-cost limit
  if GOAL-TEST[problem](STATE[node]) then return node
  successors ← EXPAND(node, problem)
  if successors is empty then return failure, ∞
  for each s in successors do
      f [s] ← max(g(s) + h(s), f [node])
  repeat
      best ← the lowest f-value node in successors
      if f [best] > f_limit then return failure, f [best]
      alternative ← the second lowest f-value among successors
      result, f [best] ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```

# Comparing Algorithms' Results

| Algorithms | DFS | RBFS | A* Search |
|---|---|---|---|
| Board solved (out of 49) | 28 | 49 | 47 |
| Total Number of Iterations | 70808 | 150951 | 45049 |
| Total Running Time for solved Boards (s) | 111.3305 | 57.5106 | 252.1998 |
| Average Algorithm Memory Consumption (GB) | 0.2 | 0.3 | 0.1 |
| Peak Algorithm Memory Consumption (GB) | 0.4 | 0.4 | 0.2 |
| Peak CPU usage (%) | 67 | 100 | 45 |
| Average CPU usage (%) | 40 | 100 | 38 |
| Peak CPU Clock-speed (GHz) | 3.65 | 2.87 | 3.81 |

**Recap:**

- Did it find a solution? : RBFS > A* > DFS
- How fast?: RBFS > DFS > A*
- Space-efficient?: A* > DFS > RBFS

# Conclusion

- Search algorithms:
  - RBFS most suitable
  - A* not suitable due to having to take into account unpredictable cost up to the expanding node
  - DFS finds solution quickly if expanded from goal; however, it fails on all boards that start only has one way to reach goal, and adjacent node has less than 3 possible moves
- Possible extensions:

Road removal, path length limit, random modifiers, time constraint, etc.

# References and additional links

- TED-ED's video: https://youtu.be/6sBB-gRhfjE
- Demo program: https://github.com/PySimpleGUI/PySimpleGUI/blob/master/DemoPrograms/Demo_Matplotlib_Embedded_Toolbar.py
- GitHub repository: : https://github.com/htnminh/AI-intro-project
- Project website: https://htnminh.github.io/AI-intro-project/
- Report notion page: https://htnminh.notion.site/The-Penniless-Pilgrim-Riddle-e3bbfaf5d7b949fdadf5a898df1f8883