貳零肆捌

組員：傅冠鈞 肖可依 陳柏豪

The 2048 Problem

- Data Structure

```
testGrid = [['',  4,  32,  128],    \
            ['', '', 128,  256],    \
            ['',  4,  64, 2048],    \
            [ 2, 16,   4, 4096]]
```

- Operations
  - Up,Down,Right,Left

- Settings
  - 2’s probability = 0.9
  - 4’s probability = 0.1

Evaluation Function

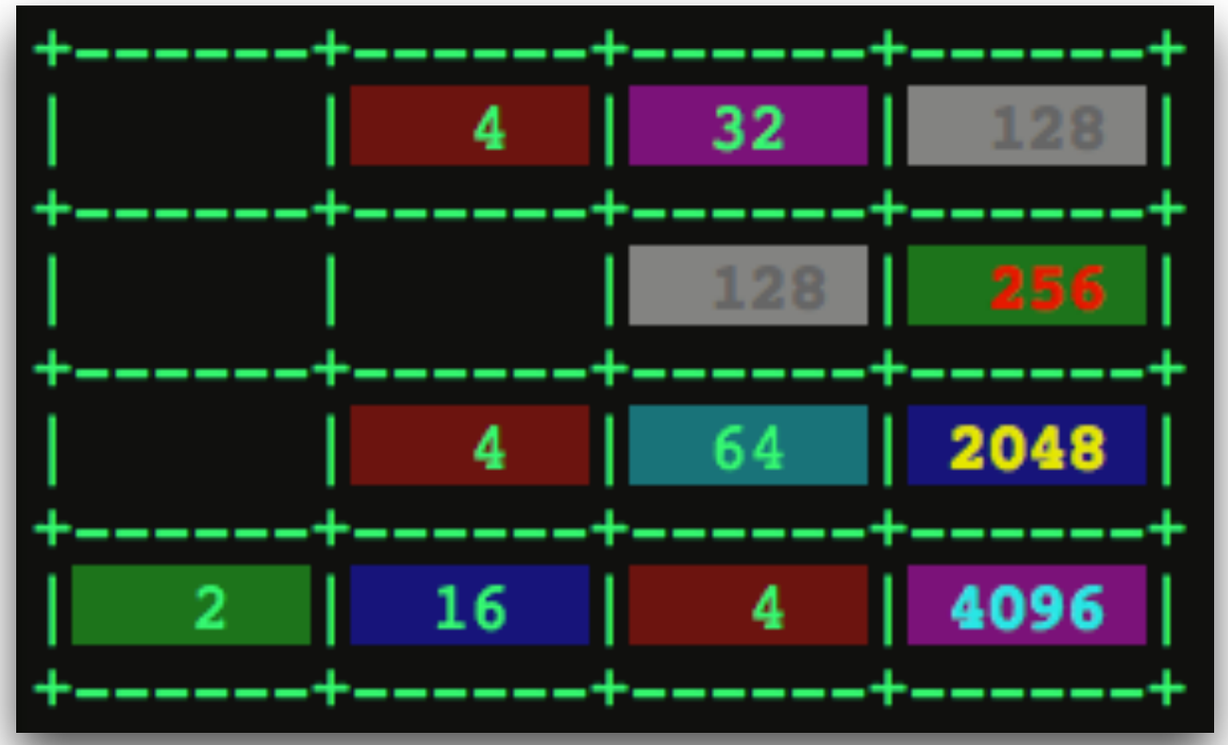
- emptyTiles()

- monotonicity() + smoothness() + emptyTiles()

- snake()

emptyTiles()

- emptyTiles() = 4



mono()+smooth()+emptyTiles()

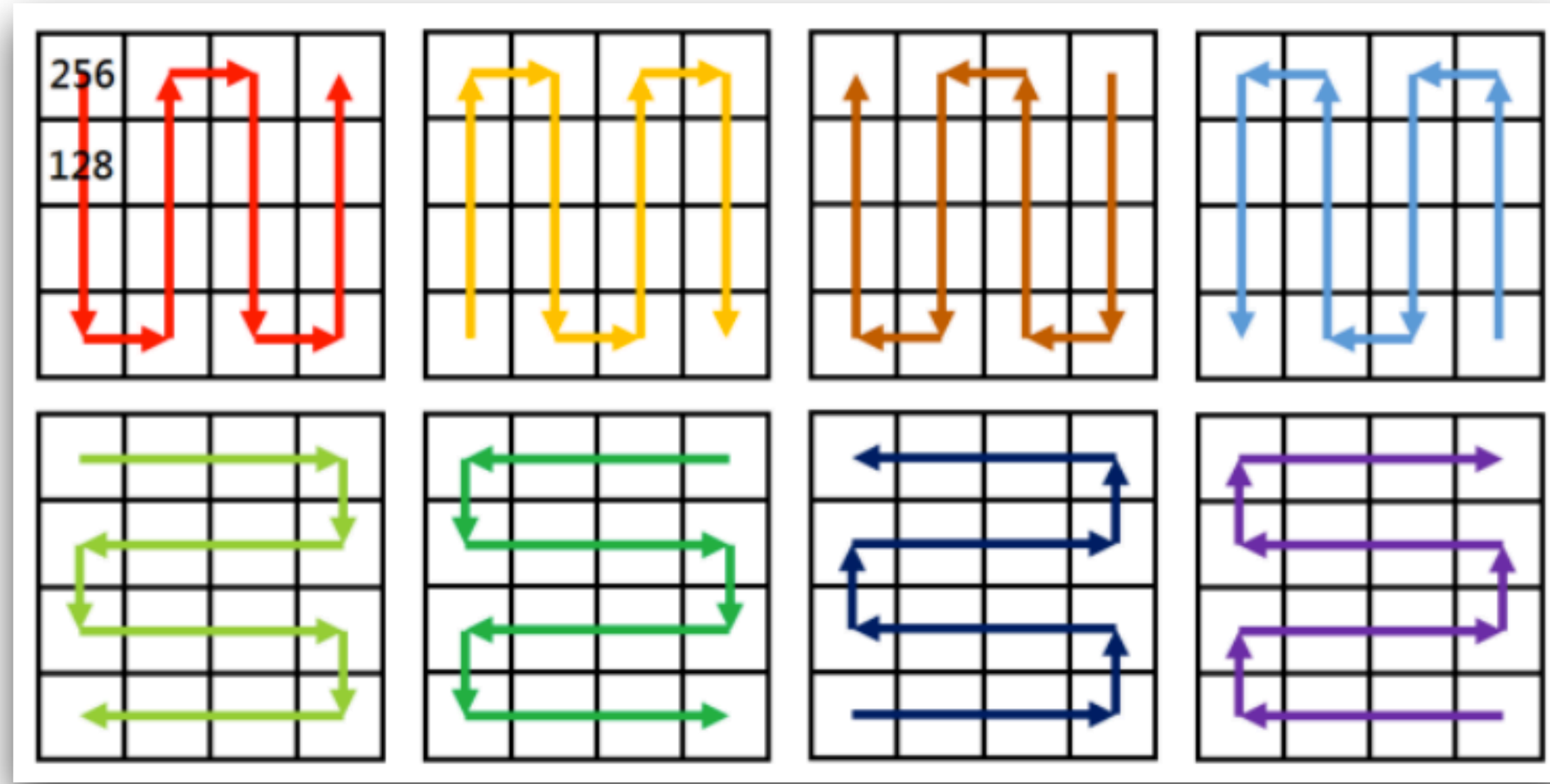
- mono() = 37

- smooth() = 88

- emptyTiles() = 4

- w1*emptyTiles() + w2*smooth() - w3*mono()

snake()

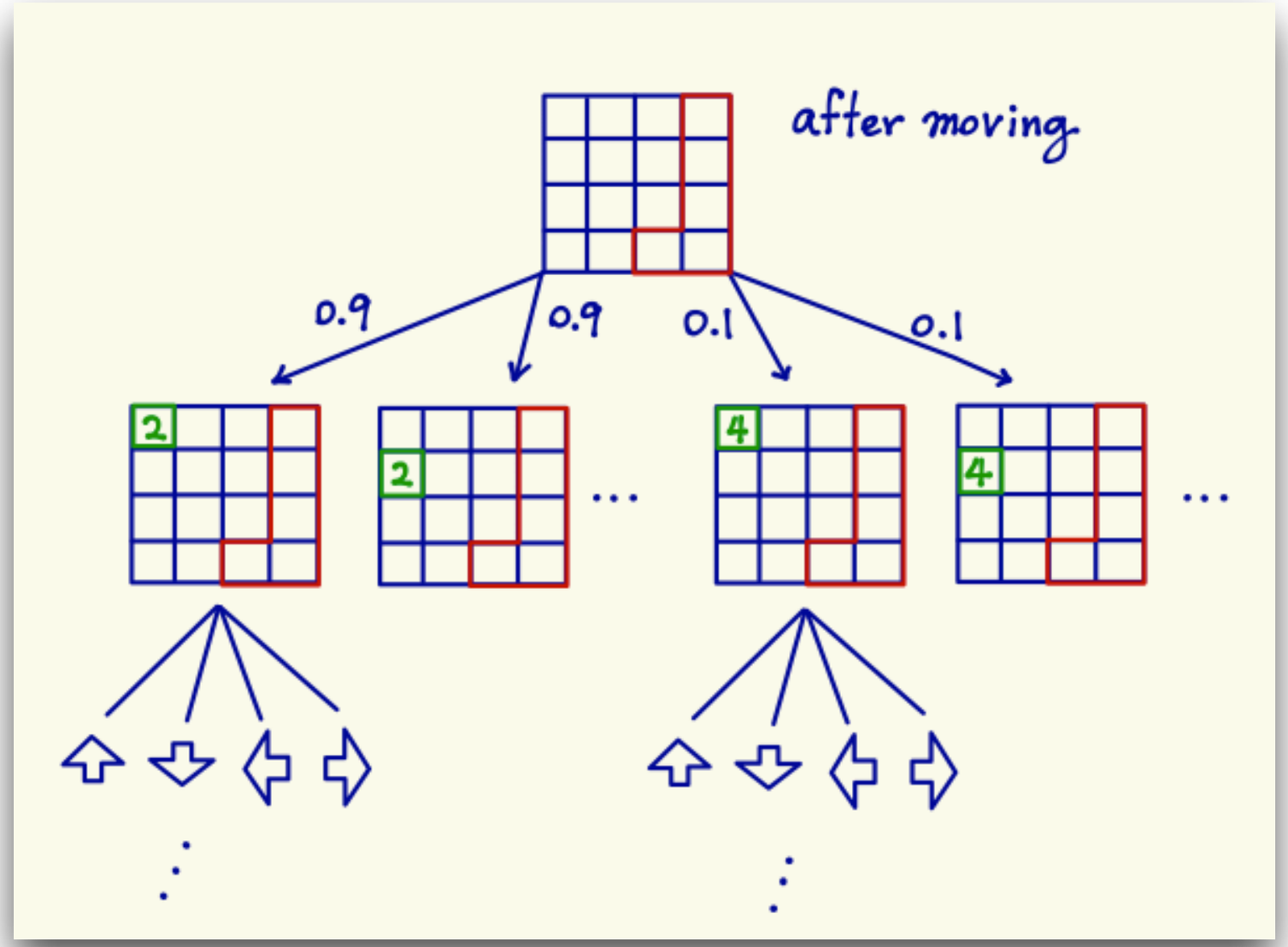


score = ∑ value(tile(n))*(r^n)

snake()

score = 4096 + 2048*r + 256*(r^2) + 128*(r^3)
+ 32*(r^4) + 128*(r^5) + ….. = 6782

Search Algorithm



expectiminimax(grid,player,depth)

Search Algorithm

```
def get_next_action(grid):
    maxScore = 0
    action = ''

    if number_of_empty_cells > 3:
        depth = 2
    else:
        depth = 4

    for move in ['up', 'down', 'left', 'right']:
        functions[move](grid)
        score = expectiminimax(grid, 0, depth)

        if score > maxScore:
            maxScore = score
            action = move

    return action
```

get_next_action(grid)

Search Algorithm

```python
def expectiminimax(grid, player, depth):

    # check terminal condition
    if depth == 0:
        return evaluation_function(grid)

    # computer's turn (consider all random conditions)
    if player == 0:
        alpha = 0

        for x, y in get_empty_cells(grid):
            grid[x][y] = 2
            alpha = alpha + prob_2 * (1 / number_of_empty_cells) * expectiminimax(grid, 1, depth - 1)

        for x, y in get_empty_cells(grid):
            grid[x][y] = 4
            alpha = alpha + prob_4 * (1 / number_of_empty_cells) * expectiminimax(grid, 1, depth - 1)

    # AI's turn
    elif player == 1:
        alpha = 0

        for move in ['up', 'down', 'left', 'right']:
            functions[move](grid)
            alpha = max(alpha, expectiminimax(grid, 0, depth - 1))

    return alpha
```

expectiminimax(grid,player,depth)

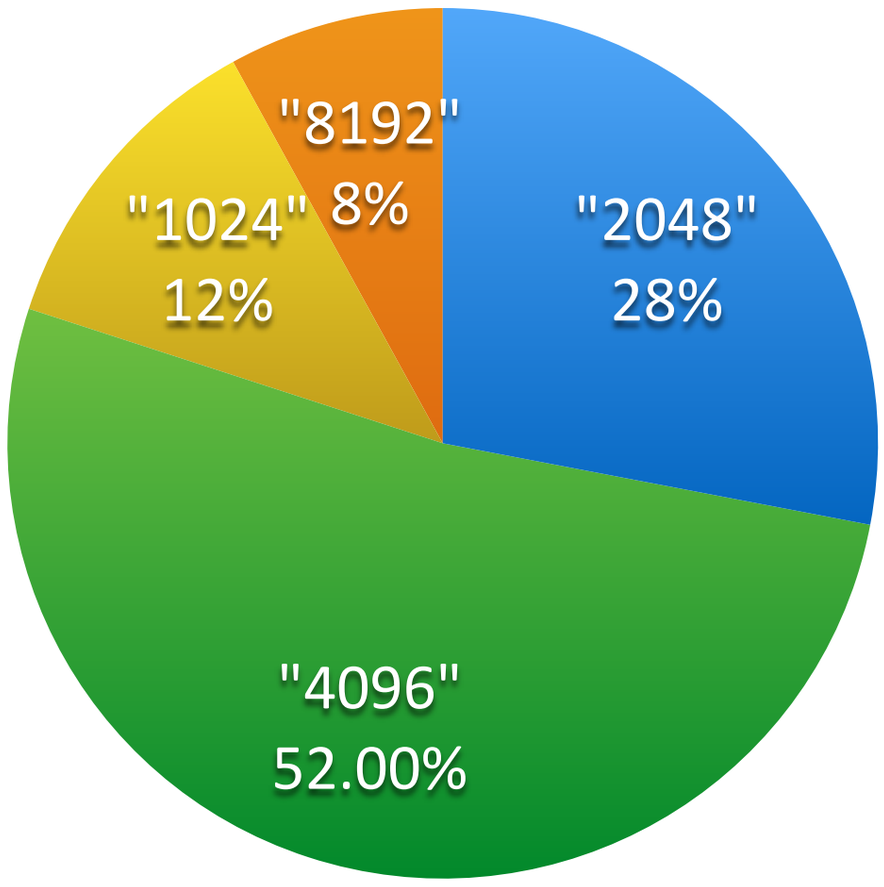
Demo

- emptyTiles() + smooth() with adaptive depth
- snake() with depth = 2
- snake() with adaptive depth (video)

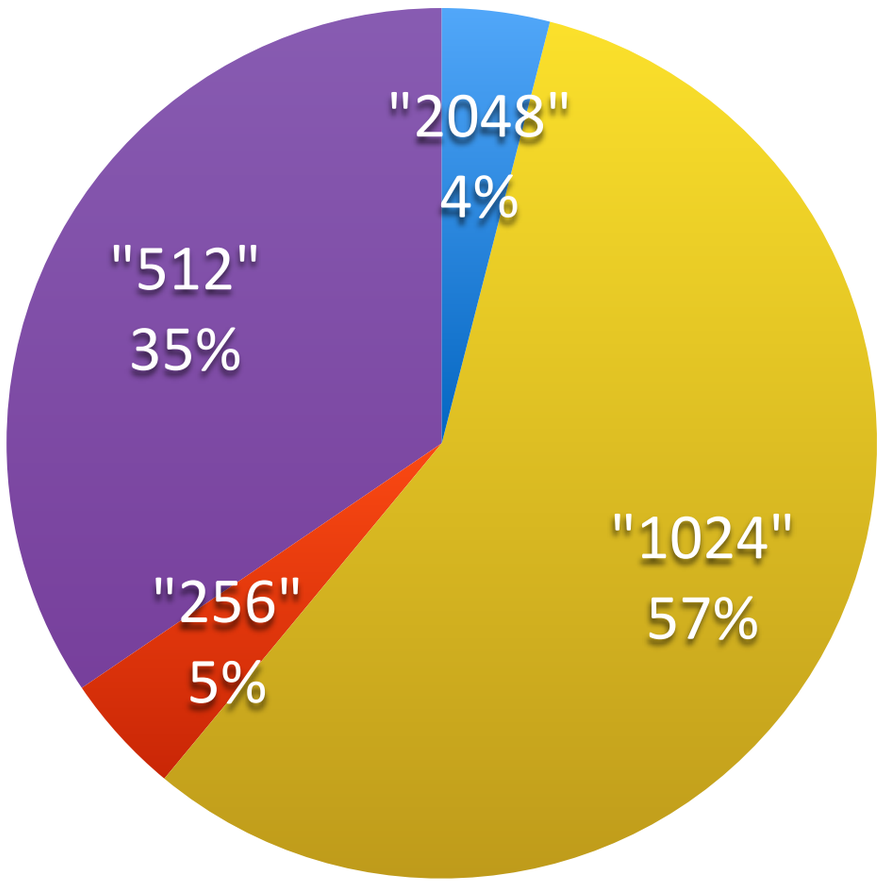
Results

- The max grid probability

snake()



emptyTiles() + smooth()



What else can be done?

- Speed up and deepen the depth by
  - encoding the grid
  - pruning

Q&A

Thanks for your attention!