MEC 2023 - Programing

Ctrl-Alt-Succeed

Mathieu Geoffroy
Theo Ghanem
Sehr Moosabhoy

Maze

Objective: Find a path to the end through a hexagonal coordinate system

Method:

- A* method is a targeted search algorithm
- G score - cost from start to node n
- F score - ideal cost from start to end if node n is used
- Heuristic - conversion of hexagonal coordinates to x-y coordinates and calculate the linear distance
- Node_set - All available nodes in a min heap

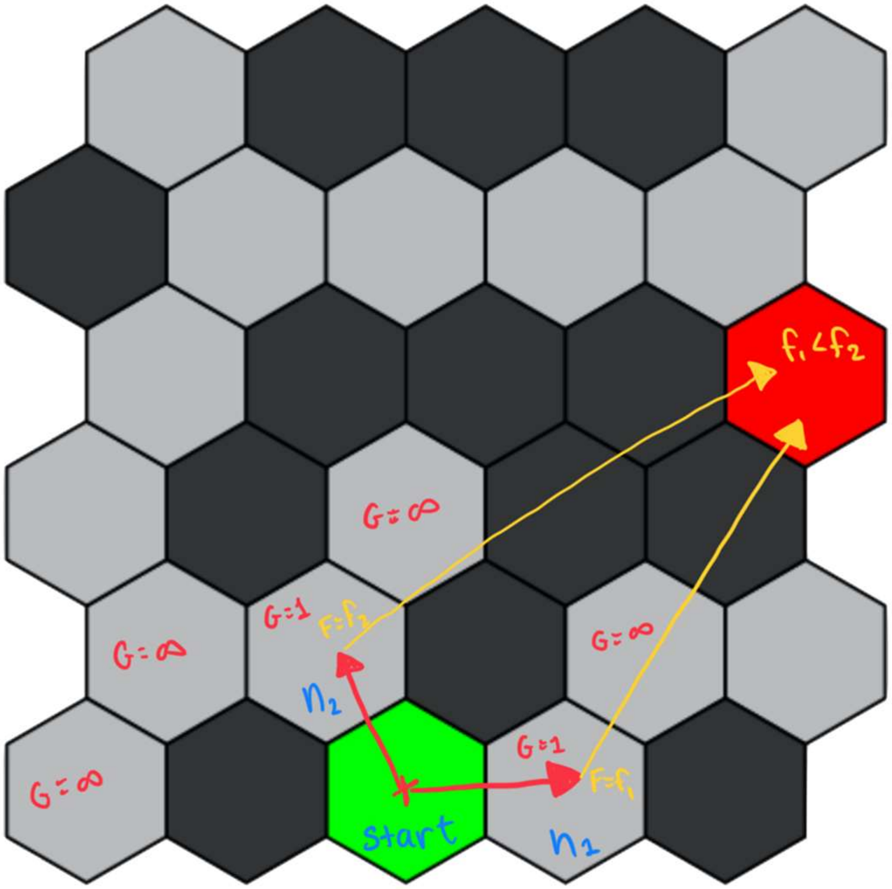
Initialization

- All scores infinity except for start (G = 0, F = distance)
- Node_set only contains start
- Calculate all possible neighbour sets (existing tiles that aren’t walls)

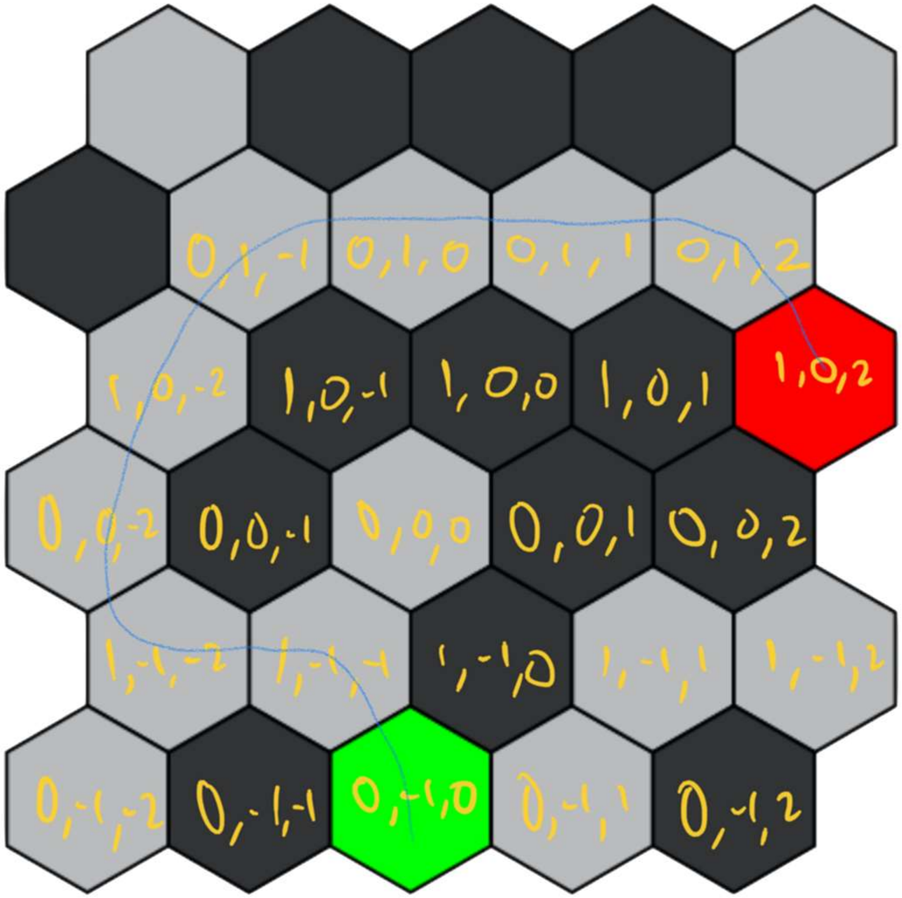
Maze Continuation

Ongoing:

- While there are nodes, take the most favourable one (lowest f score)
- If at the end, finish path!
- For all neighbours of current node, update the g scores
- If g score is lower than it was previously, add this neighbour to the list of available nodes with f score as its priority



Path Example



Objective: Find the starting budget of the 4 players (500-2500)

Given data:

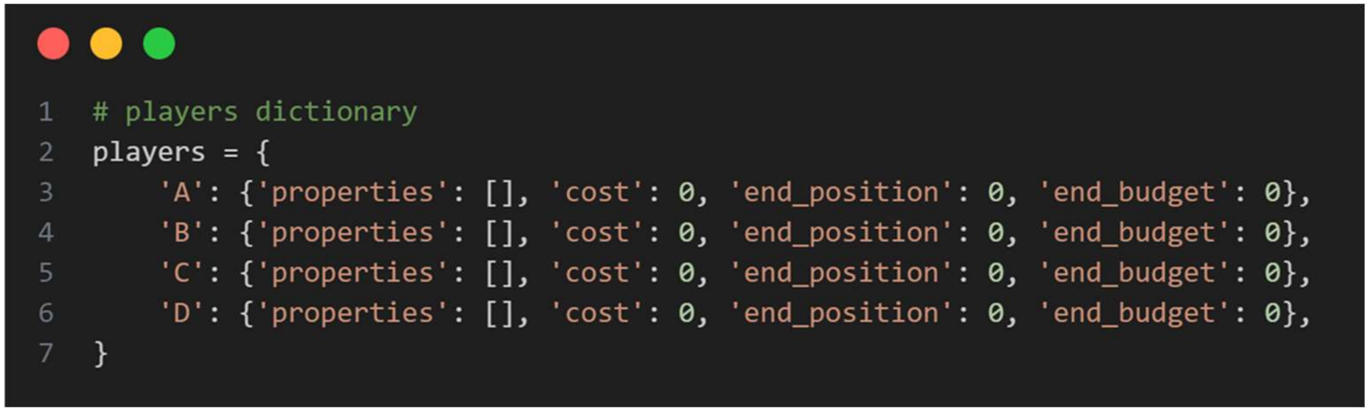
- Properties purchased in last 5 rounds and by which player
- Remaining budget and location of each player
- 1000 different games

Technique: Use only first 32 lines (1 game)

Monopoly - Methodology

Method:

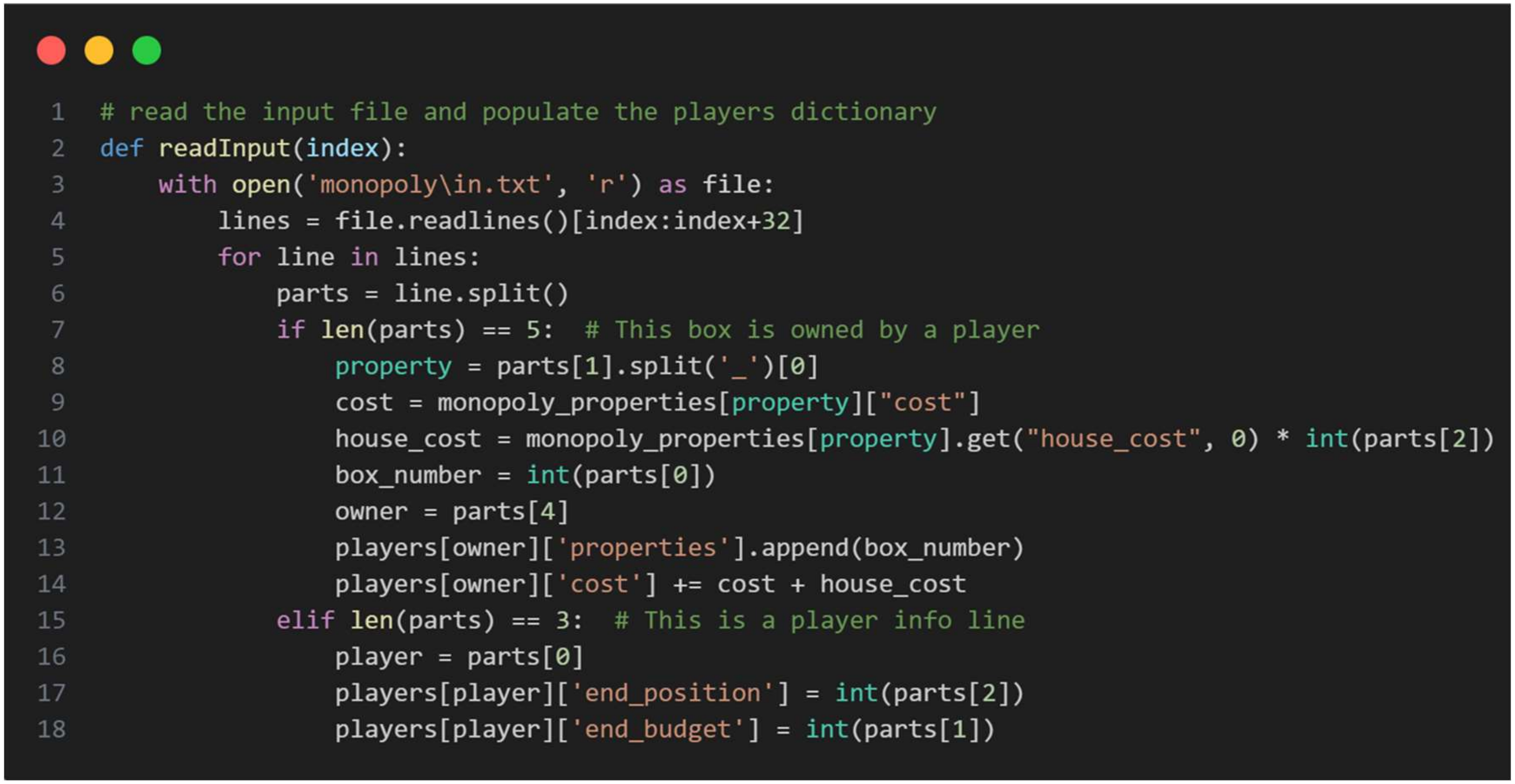
1) Make a dictionary to store all player information



```
# players dictionary
players = {
    'A': {'properties': [], 'cost': 0, 'end_position': 0, 'end_budget': 0},
    'B': {'properties': [], 'cost': 0, 'end_position': 0, 'end_budget': 0},
    'C': {'properties': [], 'cost': 0, 'end_position': 0, 'end_budget': 0},
    'D': {'properties': [], 'cost': 0, 'end_position': 0, 'end_budget': 0},
}
```

Monopoly - Methodology

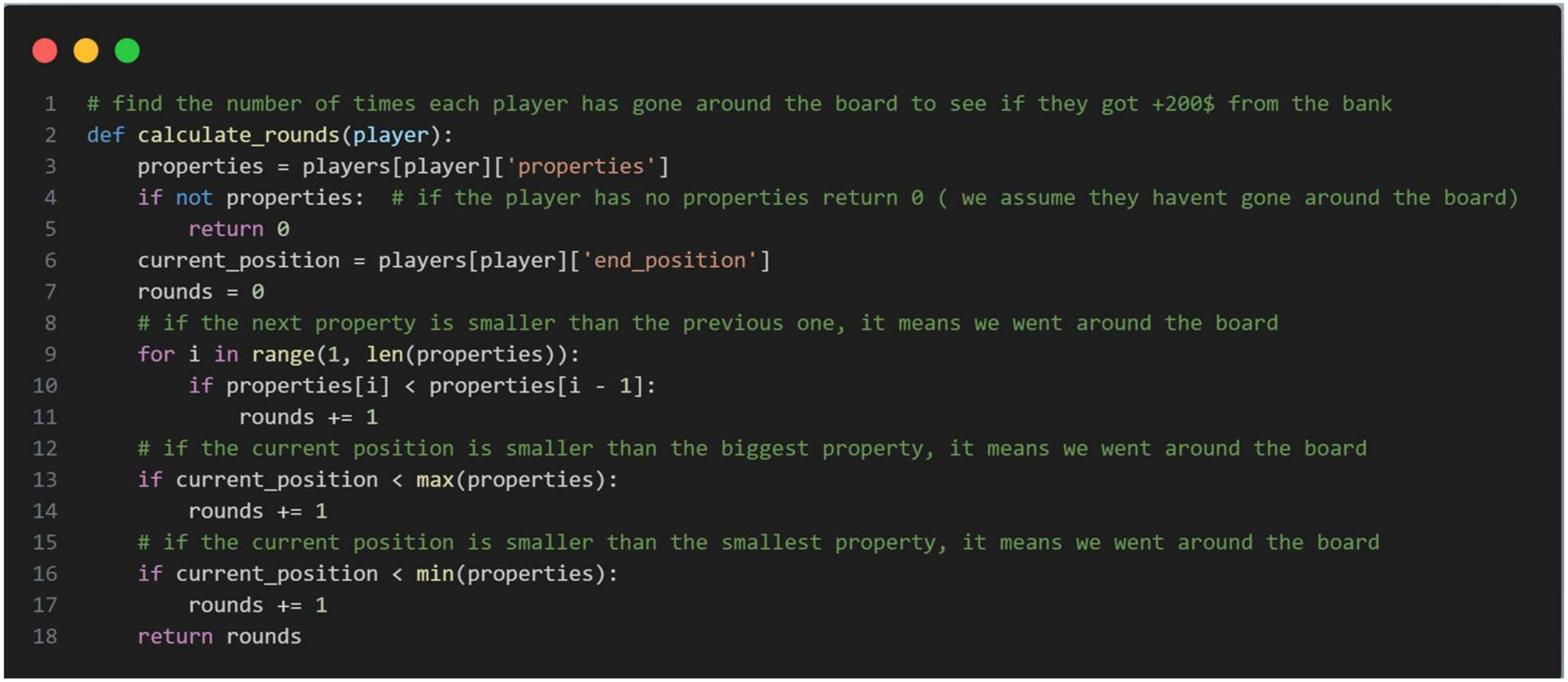
- Open file
- Populate player dictionary
- Ignore properties that are not owned
- Get player details after 5 rounds



```python
# read the input file and populate the players dictionary
def readInput(index):
    with open('monopoly\in.txt', 'r') as file:
        lines = file.readlines()[index:index+32]
        for line in lines:
            parts = line.split()
            if len(parts) == 5:  # This box is owned by a player
                property = parts[1].split('_')[0]
                cost = monopoly_properties[property]["cost"]
                house_cost = monopoly_properties[property].get("house_cost", 0) * int(parts[2])
                box_number = int(parts[0])
                owner = parts[4]
                players[owner]['properties'].append(box_number)
                players[owner]['cost'] += cost + house_cost
            elif len(parts) == 3:  # This is a player info line
                player = parts[0]
                players[player]['end_position'] = int(parts[2])
                players[player]['end_budget'] = int(parts[1])
```

Monopoly - Methodology

How do we know how many times each player has gone around the board?

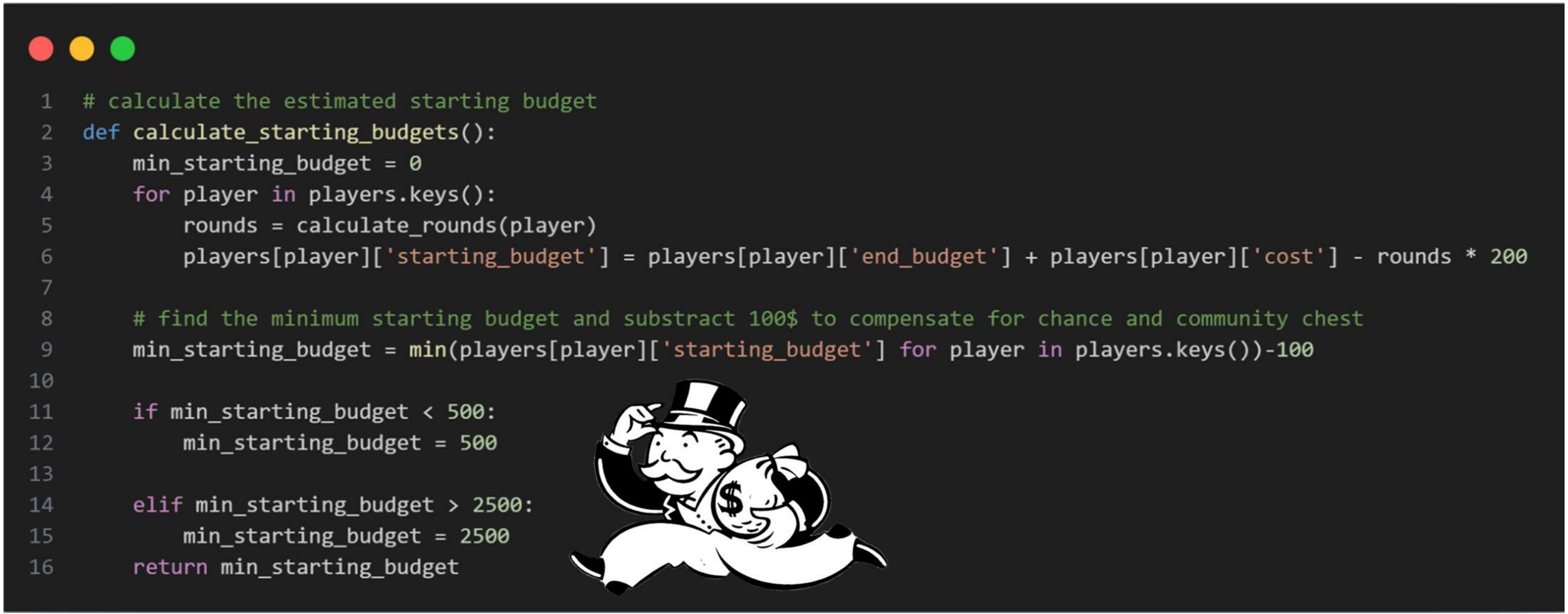


```python
# find the number of times each player has gone around the board to see if they got +200$ from the bank
def calculate_rounds(player):
    properties = players[player]['properties']
    if not properties:  # if the player has no properties return 0 ( we assume they havent gone around the board)
        return 0
    current_position = players[player]['end_position']
    rounds = 0
    # if the next property is smaller than the previous one, it means we went around the board
    for i in range(1, len(properties)):
        if properties[i] < properties[i - 1]:
            rounds += 1
    # if the current position is smaller than the biggest property, it means we went around the board
    if current_position < max(properties):
        rounds += 1
    # if the current position is smaller than the smallest property, it means we went around the board
    if current_position < min(properties):
        rounds += 1
    return rounds
```

Monopoly - Answer

Last step: calculate starting budget

Starting_budget = (remaining_budget + cost_properties - rounds*(200)) - 100



```python
# calculate the estimated starting budget
def calculate_starting_budgets():
    min_starting_budget = 0
    for player in players.keys():
        rounds = calculate_rounds(player)
        players[player]['starting_budget'] = players[player]['end_budget'] + players[player]['cost'] - rounds * 200

    # find the minimum starting budget and substract 100$ to compensate for chance and community chest
    min_starting_budget = min(players[player]['starting_budget'] for player in players.keys())-100

    if min_starting_budget < 500:
        min_starting_budget = 500

    elif min_starting_budget > 2500:
        min_starting_budget = 2500
    return min_starting_budget
```

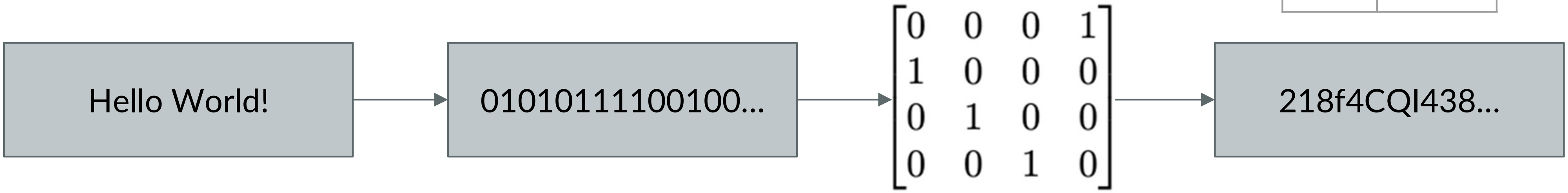
Monopoly - Problems

- Information not always clear
- Not given price of hotel (also not present in rules)
- Unsure if after 5 rounds, player needs to pay rent on the box they occupy
    - Has rent already been deducted from remaining budget or not yet?

Hamming

0	00000
1	00001
…	…
F	01111
CQI	1000
CQI0	1001
…	…
CQIf	11111

Encoding →



Hello World! → 01010111100100... →

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

→ 218f4CQI438...

ASCII

Binary

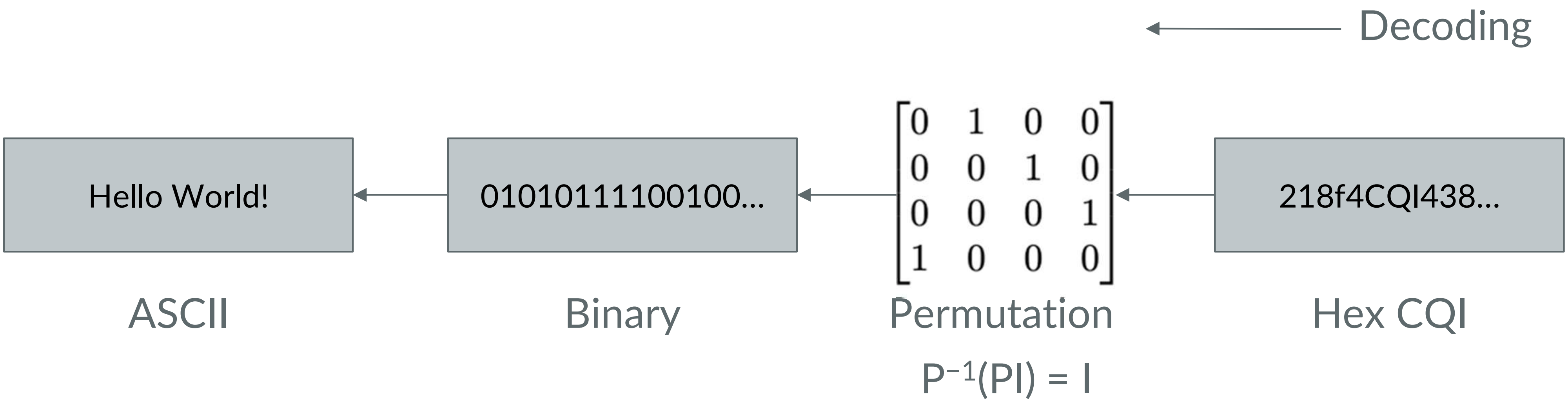
Array: I

Permutation

Array: PI

Hex CQI

Hamming



Decoding

Hello World!

ASCII

01010111100100...

Binary

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

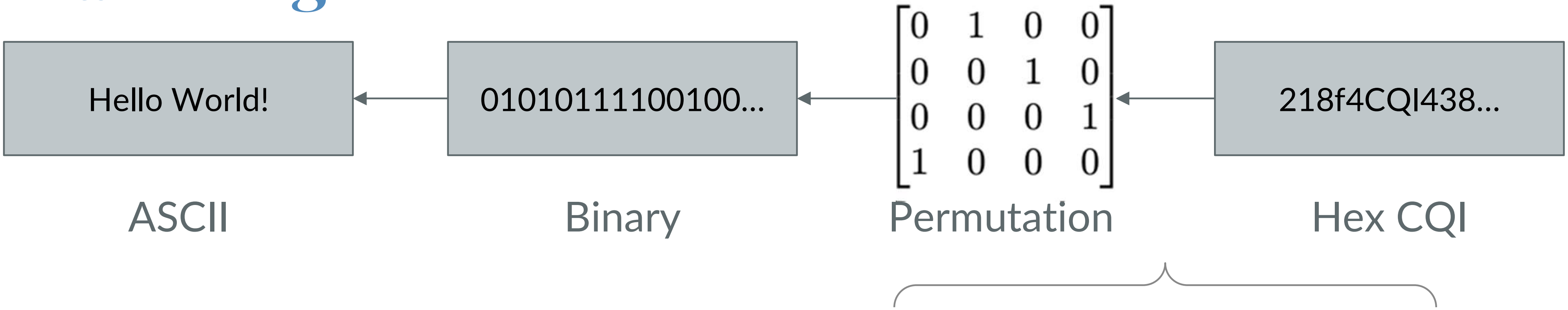
Permutation

$P^{-1}(PI) = I$

218f4CQI438...

Hex CQI

Hamming



Hello World!

01010111100100...

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

218f4CQI438...

ASCII

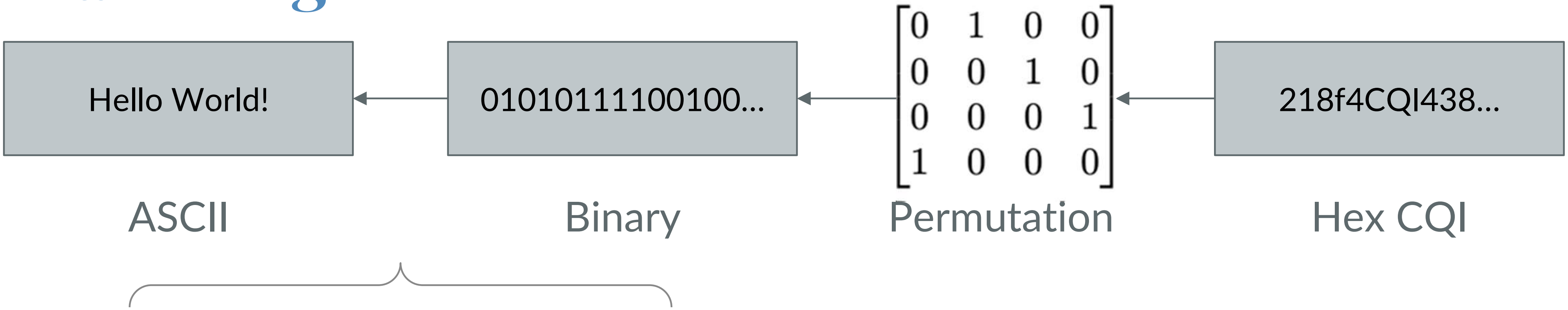
Binary

Permutation

Hex CQI

- Lookup Table
- Additional handling for CQI symbol

Hamming



Hello World!

01010111100100...

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

218f4CQI438...

ASCII

Binary

Permutation

Hex CQI

- ASCII parsing per 8 bits
- Built-in function

Shakespeare - Language Overview

Instructions are written as a play script, using characters as variables.
Most variable manipulation is done through dialogue.

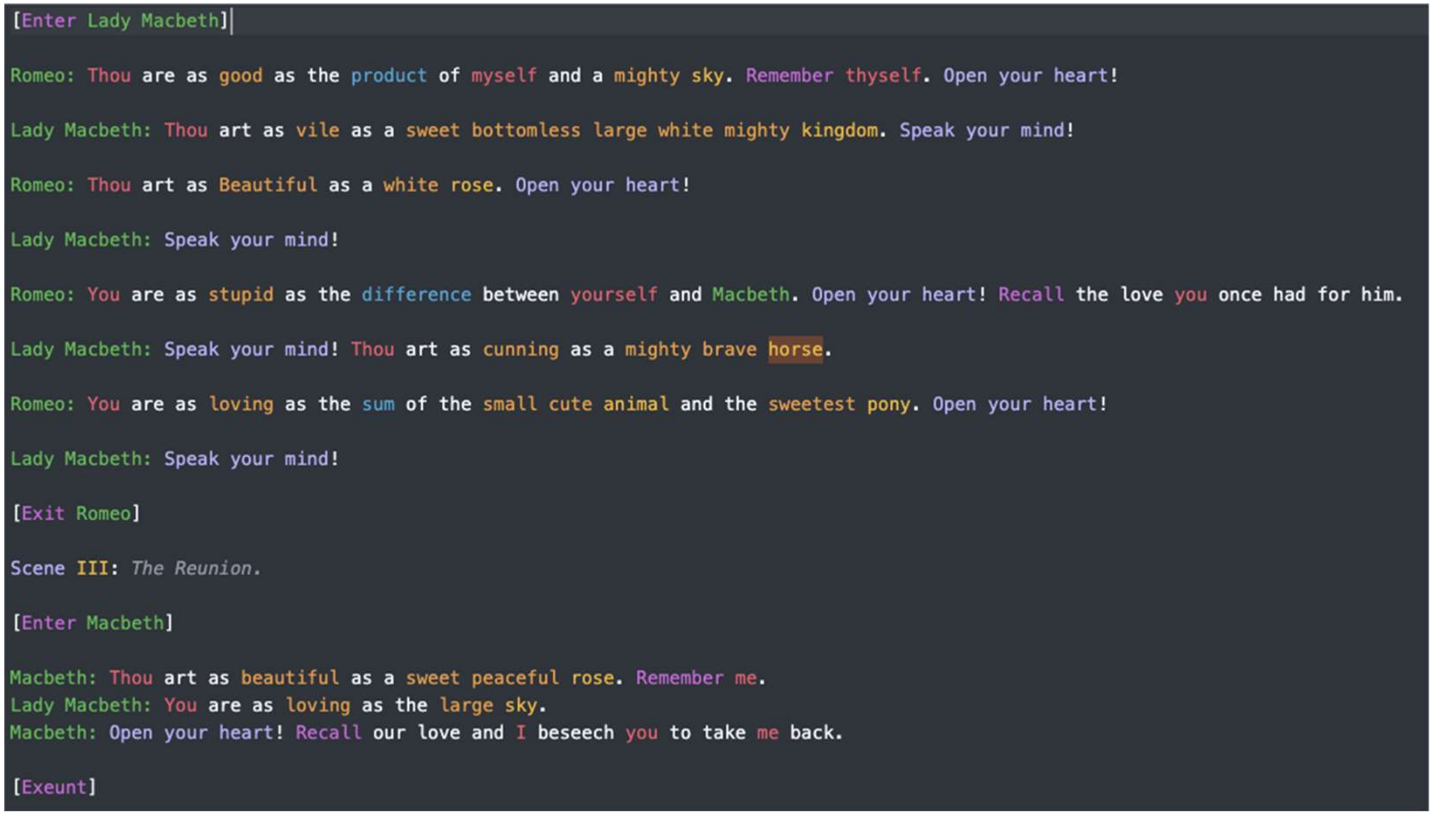
Title: Starts with a title name

Introductions: introduce characters → restricted to ones from Shakespeare plays,

Acts: Denoted as *Act RomanNumeral: description.*

Scenes: Scenes are used as goto labels.

Some of our plot



```
[Enter Lady Macbeth]

Romeo: Thou are as good as the product of myself and a mighty sky. Remember thyself. Open your heart!

Lady Macbeth: Thou art as vile as a sweet bottomless large white mighty kingdom. Speak your mind!

Romeo: Thou art as Beautiful as a white rose. Open your heart!

Lady Macbeth: Speak your mind!

Romeo: You are as stupid as the difference between yourself and Macbeth. Open your heart! Recall the love you once had for him.

Lady Macbeth: Speak your mind! Thou art as cunning as a mighty brave horse.

Romeo: You are as loving as the sum of the small cute animal and the sweetest pony. Open your heart!

Lady Macbeth: Speak your mind!

[Exit Romeo]

Scene III: The Reunion.

[Enter Macbeth]

Macbeth: Thou art as beautiful as a sweet peaceful rose. Remember me.
Lady Macbeth: You are as loving as the large sky.
Macbeth: Open your heart! Recall our love and I beseech you to take me back.

[Exeunt]
```

Shakespeare - Problems

- Only three team members, not enough time for this one.
- Instructions very unclear
- No answer when messaging MEC McGill on instagram

Any questions ?

Note: we had enough snacks to last a week :)