FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

COMP 410 A

SP24

SOS Game with Parallel Programming

Hafsah Shahbaz

251684784

Introduction

The objective of this project is to implement the **SOS** game using Python and utilize **parallel programming** to enhance performance, especially for larger grids. The game involves two players who alternately place either an 'S' or an 'O' on an 8x8 grid with the goal of forming sequences of 'SOS' horizontally, vertically, or diagonally. The winner is determined by the player who forms the most 'SOS' sequences by the end of the game. Additionally, the project includes a graphical user interface **(GUI)** and a computer opponent to play against.

Design and Implementation

Code Flow

1. **Make Move**:
    - When a player makes a move, the `make_move` method is called, updating the grid and incrementing the move count.
    - The `check_sos_parallel` method is called to check for 'SOS' sequences in parallel.
2. **Check SOS Parallel**:
    - The grid is divided into four chunks, and four processes are created and started.
    - Each process executes the `parallel_check` function on its chunk.
    - Processes join back, and results are checked for any 'SOS' sequences.
3. **Parallel Check Function**:
    - Each process iterates through its assigned chunk and uses `check_direction` to look for 'SOS' sequences.
    - Results are stored in a shared dictionary which is later checked to determine if there is a winner.

Parallel Processing Overview

Parallel processing in this project is utilized to improve the efficiency of checking for 'SOS' sequences within the grid. The winner calculation is parallelized to improve performance. The grid is divided into segments, and each segment is processed in a separate process. This reduces the time complexity and makes the application more efficient for larger grids.

Number of Processes

The number of processes used is four (`num_processes = 4`). This allows the 8x8 grid to be divided into four equal parts, with each part being handled by a separate process.

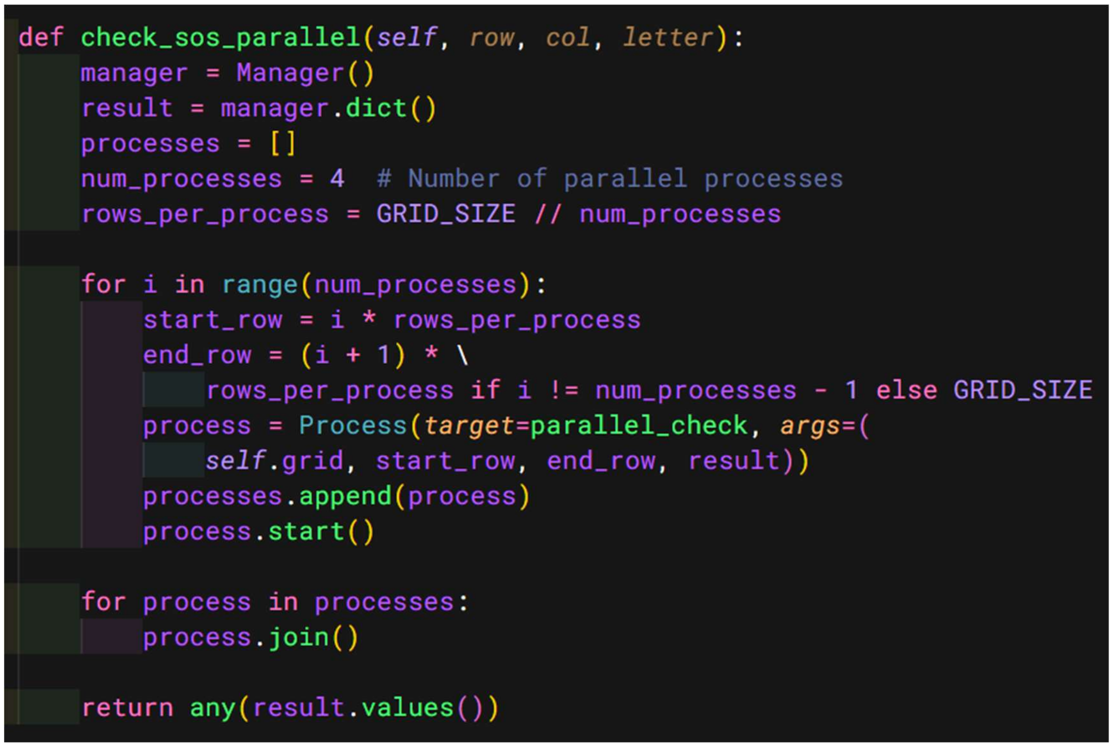
Grid Division

The 8x8 grid is divided into four equal horizontal sections. Each section consists of two rows, given that the grid size (8) divided by the number of processes (4) equals 2 rows per process.

Division and Handling of Chunks

1. **Grid Division**:
    - The grid is divided horizontally into four chunks:
        - Process 1: Rows 0-1
        - Process 2: Rows 2-3
        - Process 3: Rows 4-5
        - Process 4: Rows 6-7
2. **Process Creation and Execution**:
    - Four processes are created using Python's `multiprocessing.Process`. Each process is responsible for checking its assigned chunk for 'SOS' sequences.
    - Each process calls the `parallel_check` function, passing in the grid, start row, end row, and a shared dictionary (`result`) to store results.
3. **Parallel Check Function**:
    - The `parallel_check` function iterates through the assigned rows and columns of the grid chunk.
    - For each cell that contains 'S' or 'O', it checks in all eight possible directions (horizontal, vertical, and diagonal) to see if an 'SOS' sequence is formed using the `check_direction` function.
    - If an 'SOS' sequence is found, the result is stored in the shared dictionary (`result`).
4. **Result Aggregation**:
    - After all processes complete their execution, the main process joins them back and aggregates the results from the shared dictionary.
    - The presence of any 'SOS' sequence in the results determines if the current move has resulted in a win.

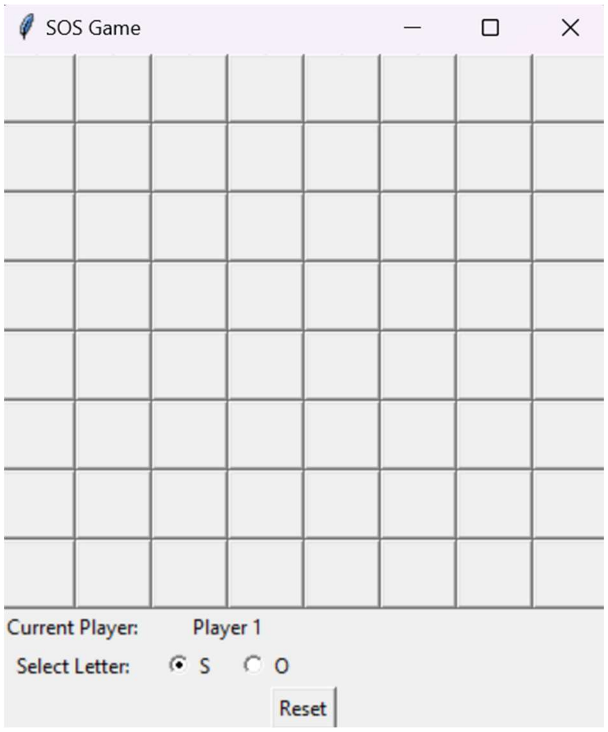
Code snippet:



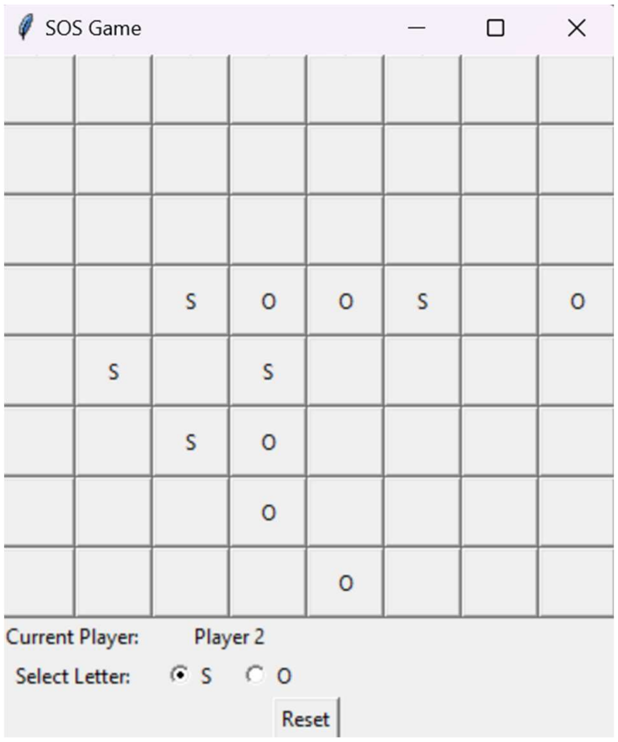
```python
def check_sos_parallel(self, row, col, letter):
    manager = Manager()
    result = manager.dict()
    processes = []
    num_processes = 4  # Number of parallel processes
    rows_per_process = GRID_SIZE // num_processes

    for i in range(num_processes):
        start_row = i * rows_per_process
        end_row = (i + 1) * \
            rows_per_process if i != num_processes - 1 else GRID_SIZE
        process = Process(target=parallel_check, args=(
            self.grid, start_row, end_row, result))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()

    return any(result.values())
```

Screenshots

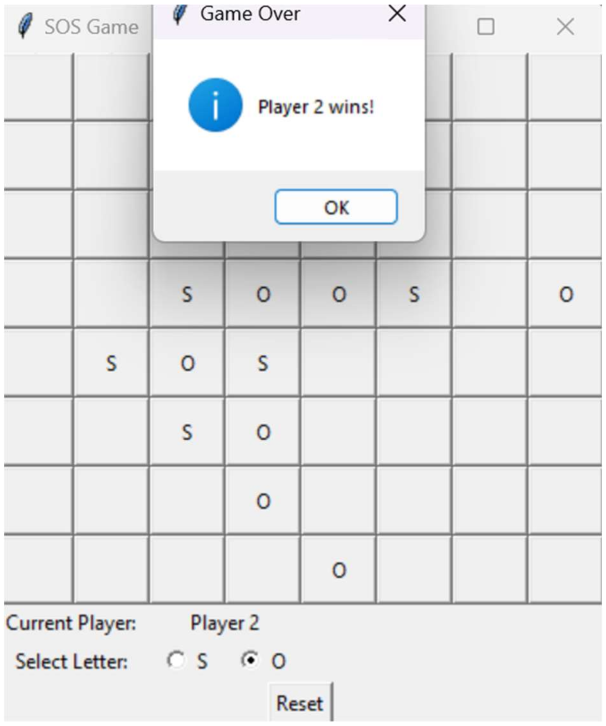
1. **Initial Game Screen**:



2. **Game in Progress**:



3. **Winning Screen**:

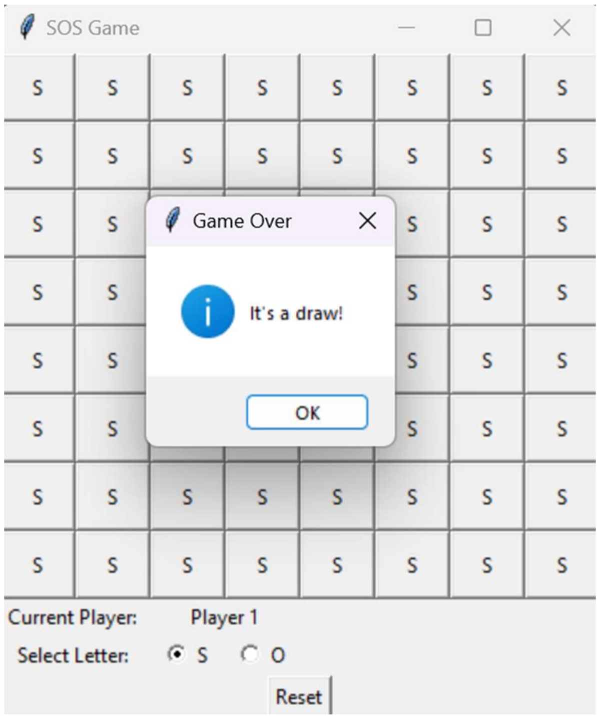


Player 2 wins!

Current Player: Player 2

Select Letter: S [✓ O]

4. **Draw Screen**:



It's a draw!

Current Player: Player 1

Select Letter: [✓ S] O