Introduction

In this project, we developed an autonomous agent simulation where multiple agents navigate a maze environment without external input. The primary objective of the project is to implement the movement logic, state tracking, and basic decision-making abilities of agents using fundamental programming constructs without the use of AI libraries or decision trees.

The agents are designed to mimic real-life pathfinding behavior by maintaining their own position, movement history, and response to traps and power-ups using Java. A custom Stack structure is utilized to store past positions, allowing agents to backtrack when needed. The project reinforces essential topics such as data structures (linked list, stack and queue), encapsulation, and object-oriented programming.

Additionally, this simulation provides insight into how simple rules and structures can be used to simulate more complex agent behavior. The randomness of agent movement ensures varied execution paths, making each run unique.

Program Interface

The program runs as a console-based Java application without a graphical user interface (GUI). The interaction is limited to the terminal, where execution can be monitored via printed logs.

The simulation is started by compiling the program with:

javac *. java

java Main

No user input is required during execution. Agents are created with randomized initial positions and begin exploring the maze based on simple movement rules. Their actions, including movements, backtracking, and power-up interactions, are printed to the console for debugging and analysis purposes.

All agent logic and state changes occur internally, and the program structure relies on basic Java classes, custom data structures (Lİnked list, Stack and Queue), and standard control structures. This minimalistic interface design emphasizes algorithm implementation and logic development over interaction or visualization

Program Execution

The program is developed using the Java programming language and is executed through the command line. It does not require any user input; all agents are created automatically when the simulation starts and act independently.

System generates random maze with:

- Walls (W), Traps (T), Power-ups (P), Goal (G)
- 2+ rotating corridors

Compilation Steps:

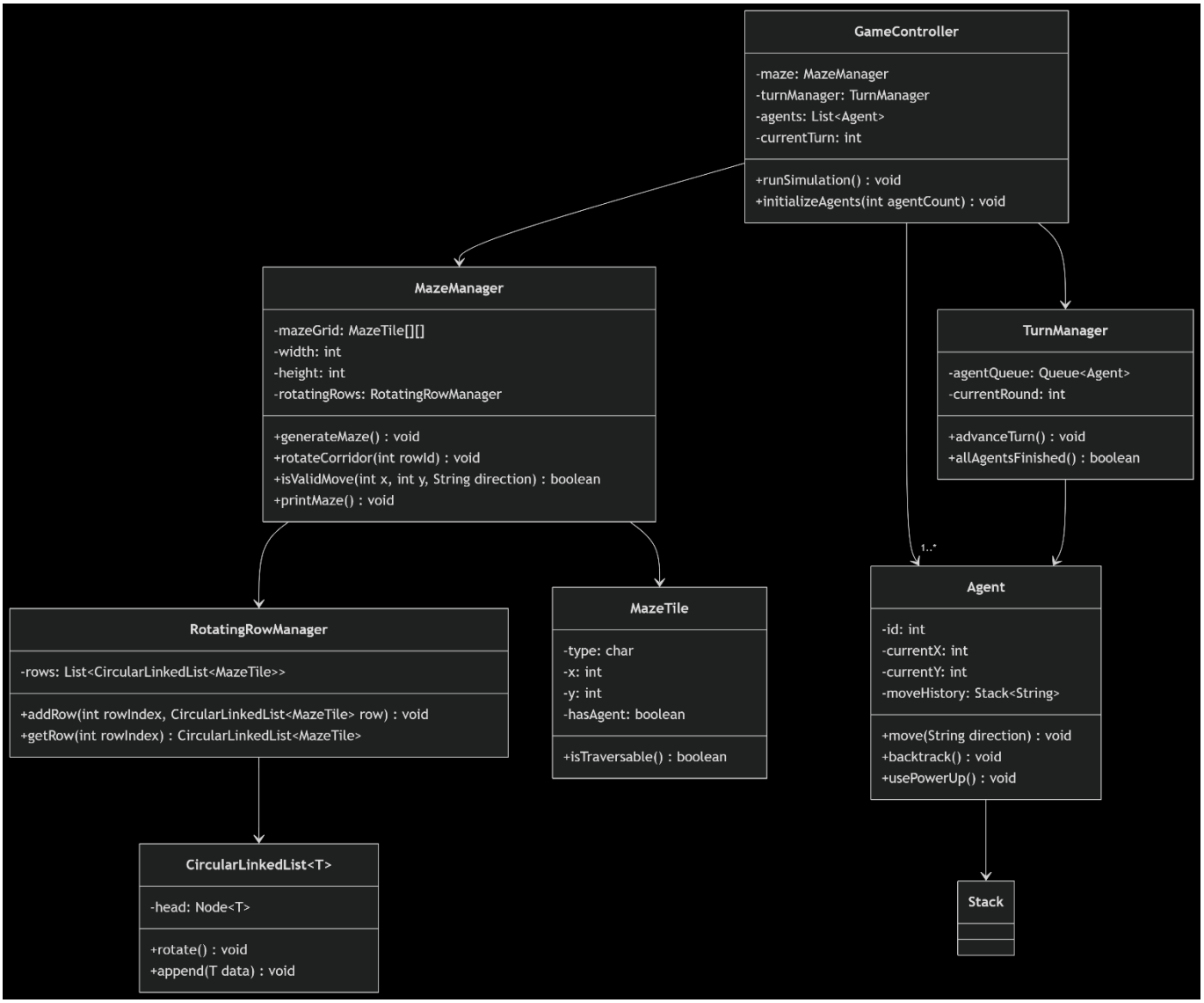
- Open a terminal or command prompt in the directory where the source files are located.
- Compile the program using the following command:
    - javac *. Java
- Run the compiled program with:
    - java Main
- Execution Process:
    - At the beginning of the program, two autonomous agents are created, each assigned a unique id.
    - Agents are placed at random starting positions in the maze.
    - They move randomly in one of the four directions: UP, DOWN, LEFT, or RIGHT.
    - After each move, their position is recorded in a Stack as a string in the format "x,y".
    - Special events like power-ups or traps may be triggered randomly during the simulation.
    - When an agent steps into a trap, it automatically backtracks two steps according to predefined rules.
    - Upon collecting a power-up, the hasPowerUp flag is updated and the applyPowerUp() method may be triggered.
    - During execution, logs are printed to the terminal showing agent position, movement direction, power-up status, and backtrack count

Input and Output

Input sections in the project are set from the main.java file. The height, width and number of agents of the maze can be specified as input. In the

Output section, each move is outputted on the console according to the specified height, width and number of agents, and finally all of these are collected in a file called simulation_log.txt

Program Structure



GameController Class:

The main orchestrator of the simulation that manages the game lifecycle.

Key Features:

- Initializes all game components (maze, agents, turn system)
- Controls the main game loop iteration
- Coordinates between MazeManager and TurnManager
- Handles simulation termination conditions

Critical Methods:

- runSimulation() does not include parameters and executes the game loop until completion.
- initializeAgents() includes int agentCount parameter and creates agents with valid starting positions.

Relationships:

- Aggregates 1 MazeManager
- Aggregates 1 TurnManager
- Contains 1*Agent objects

MazeManager Class:

Manages the maze structure and tile-based operations.

Functionality:

- Generates random maze layouts.
- Handles corridor rotation mechanics.
- Validates movement requests.
- Tracks agent positions.

Data Structures:

- 2D array for maze grid storage
- Circular linked lists for rotating rows

Critical Methods:

- isValidMove () takes int x,y, String dir parameters and checks move legality.
- Rotate Corridor() include int rowId parameter and rotates specified row.

Relationships:,

- Composes width*height MazeTile objects
- Uses 1 RotatingRowManager

TurnManager Class:

Manages turn-based progression and agent sequencing.

Behavior:

- Maintains round-robin turn order
- Processes individual agent turns
- Tracks current round number

Data Structures:

- Queue structure for turn order
- Round counter

Critical Methods:

- advanceTurn() processes next agent's turn
- allAgentsFinished() checks win condition

Relationships:,

- Aggregates 1 Queue<Agent>
- Collaborates with MazeManager

Agent Class:

Represents autonomous actors in the simulation.
Attributes:

- moveHistory type is Stack<String> and it traversal path recording.
- TrapsTriggered type is int and it does statistical tracking.

Behavior:

- Movement in 4 directions
- Backtracking when hitting traps
- Power-up utilization

Critical Methods:

- Move() does changes position.
- Backtrack() does reverts 2 previous moves

Relationships:

- Owns 1 Stack for history
- Associated with MazeTile

MazeTile Class

Fundamental unit of the maze grid.
Tile Types:

- W—Wall—Not Traversable
- P—Power-up--Traversable

Critical Methods:

- isTraversable() checks movement permission
- placeAgent() marks tile occupancy

Relationships:

- Composed by MazeManager

RotatingRowManager Class

Handles circular rotation of maze corridors.
Data Structure: Collection of CircularLinkedList<MazeTile>
Key Functionality:

- Tracks rotatable rows
- Executes rotation operations

Methods:,

- addRow() registers rotatable row
- getRow() retrieves specific row

CircularLinkedLisr Class(Generic)

Implements circular rotation behavior.
Technical Details: Generic implementation(<T>), constant-time rotation
Critical Methods:

- rotate() time complexity is O(1)
- append() time complexity is O(n) used for rotation mechanics

Supporting Structures

Queue:

- Standard FIFO implementation
- Handles turn order management

Stack:

- LIFO structure
- Stores agent movement history

Class Interaction Flow:

1. GameController initializes components
2. TurnManager requests moves from Agent
3. Agent validates moves with MazeManager
4. MazeManager updates tile states
5. Process repeats until termination

Examples

simulation_log.txt

```
=== TURN 1 ===
MAZE STATE:
W W T T T P
T P G P W E
P P T W P A
P E T P T P
W W T W W P
P W T P W P
Agent 0: (5,2) | Moves: 0 | Backtracks: 0 | PowerUp: NO
Last moves: 5,2

=== TURN 2 ===
MAZE STATE:
W W T T T P
T P G P W A
P T W P E P
P E T P T P
W W T W W P
P W T P W P
Agent 0: (5,1) | Moves: 1 | Backtracks: 0 | PowerUp: NO
Last moves: 5,1 5,2
```

simulation_log.txt

```
Agent 1: (7,9) | Moves: 49 | Backtracks: 11 | PowerUp: YES
Last moves: 7,9 7,10 7,9 7,10 7,9

=== FINAL STATISTICS ===
FINAL MAZE:
E E T T T E W P T W T T W T P T
T E P W W P E T T W W W G W T P
P E E P P E E T P E W P W E E W
P E E P E T P P T W A W P W W E
A E T T A T A T A A W T A W W A
P T P T W W E W E W E T P P E E
E E E T E W T W T T A E W P T P
T E W W T E T P A E W W E T W T
E W E T W E T W P W W P P W T T
E E E E T E W E E P E P P E T E
P T W T W W E A T P E E P E W W
W P E E P T E W T T T P P W T W
E P T W W W P T T T W W E P T T
E W W E W W T T T P E E E E W T
P P T T W W W W P E T W T E W E
P E T T T P E W E P E T T E T W

AGENT RESULTS:
Agent 0: FAILED | Total Moves: 37 | Backtracks: 11 | Traps: 14
Agent 1: FAILED | Total Moves: 50 | Backtracks: 11 | Traps: 14

GLOBAL STATS:
Total Turns: 100
First Winner: Agent -1
```

simulation_log.txt

```
E W E T W E T W P W W P P W T T
E E E E T E W A E P E P P E T E
P T W T W W E E T P E E P E W W
W P E E P T E W T T T P P W T W
E P T W W W P T T T W W E P T T
E W W E W W T T T P E E E E W T
P P T T W W W W P E T W T E W E
P E T T T P E W E P E T T E T W
Agent 0: (10,4) | Moves: 7 | Backtracks: 1 | PowerUp: YES
Last moves: 10,4 10,3 10,4 10,3 10,4
Agent 1: (7,9) | Moves: 7 | Backtracks: 1 | PowerUp: YES
Last moves: 7,9 7,10 7,9 7,10 7,9

=== TURN 16 ===
MAZE STATE:
E E T T T E W P T W T T W T P T
W W G W T P T E P W W P E T T W
P E E P P E E T P E W P W E E W
P E E P E T P P T W A W P W W E
E T T A T A T P A W T E W W P E
P T P T W W E W E W E T P P E E
E E E T E W T W T T E E W P T P
T E W W T E T P A E W W E T W T
E W E T W E T W P W W P P W T T
E E E E T E W A E P E P P E T E
P T W T W W E E T P E E P E W W
W P E E P T E W T T T P P W T W
E P T W W W P T T T W W E P T T
E W W E W W T T T P E E E E W T
P P T T W W W W P E T W T E W E
P E T T T P E W E P E T T E T W
Agent 0: (10,3) | Moves: 8 | Backtracks: 1 | PowerUp: YES
Last moves: 10,3 10,4 10,3 10,4 10,3
Agent 1: (7,9) | Moves: 7 | Backtracks: 1 | PowerUp: YES
Last moves: 7,9 7,10 7,9 7,10 7,9

```

Improvements and Extensions

Shortcomings

- Limited agent AI(random movement)
- No GUI(text-only output)

Future Extensions

- Add BFS based pathfinding it's for finding to shortest way out the maze.
- Implement graphical interface

Difficulties

1. Circular Linked List Implementation
   - Initial issues with pointer management during corridor rotation
   - Solved by adding boundary checks
2. Concurrent Agent Tracking
   - Multiple agents on same tile caused bugs
   - Fixed via atomic position updates

Conclusion

This project successfully demonstrates how classical data structures (queues, stacks, circular lists) can model complex game mechanics. The implementation meets all specified requirements while providing a foundation for future enhancements.

References

1. Data Structures and Algorithms in Java – M. Goodrich
2. https://www.geeksforgeeks.org/
3. https://visualgo.net/en
4. https://docs.oracle.com/javase/8/docs/api/
5. Lecture notes
6. Lab Materials

Appendices

Appendix A: includes Main.java-Agent.java-MazeManager.java-MazeTile.java-TurnManager.java

Main.java

```
public class Main {
    public static void main(String[] args) {
        GameController controller = new GameController(16, 16, 2);
        controller.runSimulation();
    }
}
```

Agent.java

```
import java.util.Arrays;

public class Agent {
    private final int id;
    public int currentX;
    public int currentY;
    public boolean hasReachedGoal;
    public int totalMoves;
    public static int backtracks;
    public static int trapsTriggered;
    public boolean hasPowerUp;
    public final Stack moveHistory;

    public Agent(int id, int startX, int startY) {
        this.id = id;
        this.currentX = startX;
        this.currentY = startY;
        this.moveHistory = new Stack();
        this.moveHistory.push(startX + "," + startY); // başlangıçta buraya koyduk
    }
    public String[] convertToStringArray(Object[] array) {
        String[] stringArray = new String[array.length];
        for (int i = 0; i < array.length; i++) {
            stringArray[i] = (String) array[i];
        }
        return stringArray;
    }

    // oto oynatmaya göre ajan yukarı aşağı sağ veya sola doğru hareket ederse hareketi
dğeiştir
    public void move(String direction) {
        direction = direction.toUpperCase();
        if (!Arrays.asList("UP", "DOWN", "LEFT", "RIGHT").contains(direction)) return;

        int newX = currentX;
        int newY = currentY;

        switch (direction) {
            case "UP" -> newY--;
            case "DOWN" -> newY++;
            case "LEFT" -> newX--;
            case "RIGHT" -> newX++;
        }

        currentX = newX;
```

```
    currentY = newY;
    recordMove(newX, newY);
    totalMoves++;
}

// tuzağa yakalanırsa 2 adım geri gitmeli kodu
public void backtrack() {
    if (moveHistory.getSize() < 2) return;

    // stackten 2 pop et
    moveHistory.pop();
    moveHistory.pop();

    if (moveHistory.isEmpty()) return;

    String lastValidMove = moveHistory.peek();
    if (lastValidMove == null || !lastValidMove.contains(",")) return;

    try {
        String[] coords = lastValidMove.split(",");
        currentX = Integer.parseInt(coords[0]);
        currentY = Integer.parseInt(coords[1]);
        backtracks++;
    } catch (NumberFormatException e) {
        System.err.println("Invalid move format in history!");
    }
}

// tuzağı triggerleyıp çalışmasını sağla
public void triggerTrap() {
    trapsTriggered++;
    backtrack(); //bu fonksiyonla da oto 2 adım geri at
}

private void recordMove(int x, int y) {
    moveHistory.push(x + "," + y);
}

// gücü varsa hamle sayısını arttırmadan hareket etmesini sağla
public void usePowerUp() {
    if (hasPowerUp) {

        totalMoves--; // Hamle sayısını artırmadan hareket ettirir
        hasPowerUp = false;
    }
}

// son hareketleri alır gamecontrollerda yazdırmak için gerekli
public String[] getLastNMoves(int n) {
    return moveHistory.getLastNMoves(n);
}

public String getMoveHistoryAsString() {
    StringBuilder sb = new StringBuilder();
    String[] moves = getLastNMoves(moveHistory.getSize());
    for (String move : moves) {
        if (move != null) sb.append(move).append(" -> ");
    }
    return sb.length() > 0 ? sb.substring(0, sb.length() - 4) : "No moves";
}
public void removeAgent(MazeManager mazeManager, int x, int y) {
    mazeManager.updateAgentPosition(this, x, y);
}

public void placeAgent(MazeManager mazeManager, int x, int y) {
    mazeManager.updateAgentPosition(this, x, y);
}
```

```
    public int getId() { return id; }
    public int getCurrentX() { return currentX; }
    public int getCurrentY() { return currentY; }
    public boolean hasReachedGoal() { return hasReachedGoal; }
    public void setReachedGoal(boolean reached) { this.hasReachedGoal = reached; }
    public int getTotalMoves() { return totalMoves; }
    public int getBacktracks() { return backtracks; }
    public int getTrapsTriggered() { return trapsTriggered; }
    public boolean hasPowerUp() { return hasPowerUp; }
    public void setPowerUp(boolean powerUp) { this.hasPowerUp = powerUp; }
    //override yazmayınca sarı uyarı verdi
    @Override
    public String toString() {
        return String.format(
            "Agent %d: (%d,%d) | Moves: %d | Backtracks: %d | Traps: %d | PowerUp: %s |
Goal: %s",
            id, currentX, currentY, totalMoves, backtracks, trapsTriggered,
            hasPowerUp ? "YES" : "NO", hasReachedGoal ? "REACHED" : "PENDING"
        );
    }
}
```

MazeManager.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;
//Halil burdaki kodları dikkatle incele anlamadığını beraber çalışam hoca sorabilir ben
rotatingrows için donsatlist diye bi class oluşturmuşum
//kusura bakma ya adları çok şey yapmamıştım meğer belgede isimler de varmış
düzenlemeye çalışcam
public class MazeManager {
    public  final MazeTile[][] mazeGrid;
    public  final int width;
    public  final int height;
    public  final donsatlist rotatingRows;
    public  int goalX = -1;
    public  int goalY = -1;


    public MazeManager(int width, int height) {
        this.width = width;
        this.height = height;
        this.mazeGrid = new MazeTile[height][width];
        this.rotatingRows = new donsatlist();
        initializeMaze();
    }

    private void initializeMaze() {
        Random rand = new Random();
        char[] tileTypes = {'E', 'W', 'T', 'P'};

        //eastgele labirent oluşturmak için matris dizinin satır ve sütunlarını
oluşturur
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                mazeGrid[y][x] = new
MazeTile(tileTypes[rand.nextInt(tileTypes.length)], x, y);
            }
        }

        // rastgele bir noktaya tek bir hedef koymak için
        placeGoal(rand);

        //dönen zımbırtıyı initialez etmek için bunu kullancaz
        initializeRotatingRows(rand);
    }
```

```
    private void placeGoal(Random rand) {
        goalX = rand.nextInt(width);
        goalY = rand.nextInt(height);
        mazeGrid[goalY][goalX] = new MazeTile('G', goalX, goalY);
    }

    private void initializeRotatingRows(Random rand) {
        while (rotatingRows.getRotatingRowCount() < 2) {
            int row = rand.nextInt(height);
            if (!rotatingRows.contains(row)) {
                CircularLinkedList<MazeTile> rowList = new CircularLinkedList<>();
                for (int x = 0; x < width; x++) {
                    rowList.append(mazeGrid[row][x]);
                }
                rotatingRows.addRow(row, rowList);
            }
        }
    }

    public void rotateRandomCorridor() {
        int[] availableRows = rotatingRows.getAllIndexes();
        if (availableRows.length > 0) {
            int rowToRotate = availableRows[new
Random().nextInt(availableRows.length)];
            rotateCorridor(rowToRotate);
        }
    }

    public void rotateCorridor(int rowId) {
        if (!rotatingRows.contains(rowId)) return;

        CircularLinkedList<MazeTile> row = rotatingRows.getRow(rowId);
        row.rotate();

        // döndür satırı labirente geri ekle
        MazeTile[] rotatedRow = new MazeTile[width];
        row.toArray(rotatedRow);
        System.arraycopy(rotatedRow, 0, mazeGrid[rowId], 0, width);
    }

    public boolean isValidMove(Agent agen, int x, int y, String direction) {
        int newX = x, newY = y;

        switch (direction.toUpperCase()) {
            case "UP" -> newY--;
            case "DOWN" -> newY++;
            case "LEFT" -> newX--;
            case "RIGHT" -> newX++;
            default -> { return false; }
        }

        if (newX < 0 || newX >= width || newY < 0 || newY >= height) {
            return false;
        }

        MazeTile targetTile = mazeGrid[newY][newX];
        if (targetTile.getType() == 'T') {
            agen.triggerTrap();
            return false;
        }
        return targetTile.isTraversable() && !targetTile.hasAgent();
    }

    public void updateAgentPosition(Agent agent, int oldX, int oldY) {
        if (isValidPosition(oldX, oldY)) {
            mazeGrid[oldY][oldX].removeAgent();
        }
        if (isValidPosition(agent.getCurrentX(), agent.getCurrentY())) {
```

```
            mazeGrid[agent.getCurrentY()][agent.getCurrentX()].placeAgent();
        }
    }
    //labirentin durumunu yazdırıyoz burda
    public void printMaze() {
        System.out.println("\n=== CURRENT MAZE ===");
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                System.out.print(mazeGrid[y][x] + " ");
            }
            System.out.println();
        }
        System.out.println("Legend: A=Agent, G=Goal, P=Power-up, T=Trap, E=Empty,
W=Wall");
    }
    //halil bu mazesnapshotı senin kodlarında kullanman gerekiyo sanırım ona göre
eklersin
    public void printMazeSnapshot(FileWriter logFile) throws IOException
{           //burda dosyaya yazdırabiliyomuşuz log çıktısı vermek için txtli bi
zımbırtı oluşturcaz
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                MazeTile tile = mazeGrid[y][x];
                char c = tile.hasAgent() ? 'A' : tile.getType();
                logFile.write(c + " ");
            }
            logFile.write("\n");
        }
    }

    public MazeTile getTile(int x, int y) {
        return isValidPosition(x, y) ? mazeGrid[y][x] : null;
    }

    public int[] getRotatingRowIndexes() {
        return rotatingRows.getAllIndexes();
    }

    public int getWidth() { return width; }
    public int getHeight() { return height; }
    public int getGoalX() { return goalX; }
    public int getGoalY() { return goalY; }

    private boolean isValidPosition(int x, int y) {
        return x >= 0 && x < width && y >= 0 && y < height;
    }
}
```

MazeTile.java

```
public class MazeTile {
    public char type; // 'E': Empty, 'W': Wall, 'T': Trap, 'P': Power-up, 'G': Goal
    public int x, y;
    public boolean hasAgent;

    public MazeTile(char type, int x, int y) {
        this.type = type;
        this.x = x;
        this.y = y;
        this.hasAgent = false;
    }
    //Abi buralar seyahat edilebilir noktalar
    public boolean isTraversable() {
        return type == 'E' || type == 'P' || type == 'G';
    }
    //burda ajan dediğimiz şey bulunuyor mu ?
    public boolean hasAgent() {
```

```
        return hasAgent;
    }

    public void setHasAgent(boolean hasAgent) {
        this.hasAgent = hasAgent;
    }

    public char getType() {
        return type;
    }
    @Override//bunu compiler yazmamız gerektiğini söyledi sarı uyarı veriyodu
    public String toString() {
        return hasAgent ? "A" : String.valueOf(type);
    }
    public void removeAgent() {
        this.hasAgent = false;
    }

    public void placeAgent() {
        this.hasAgent = true;
    }
}
```

TurnManager.java

```
import java.util.List;
import java.util.Random;
//burada genel olarak her turnde yönetimi sağlayıp bastırmaya çalıştık
//finallerin ne anlama geldiğini yeni öğrendim onları elleme
public class TurnManager {
    private final Queue<Agent> agentQueue;
    private final MazeManager mazeManager;
    private int currentRound;
    private final Random random;

    public TurnManager(List<Agent> agents, MazeManager mazeManager) {
        this.agentQueue = new Queue<>();
        this.mazeManager = mazeManager;
        this.currentRound = 0;
        this.random = new Random();
        initializeQueue(agents);
    }
    //queue'yı initialize ettik
    private void initializeQueue(List<Agent> agents) {
        for (Agent agent : agents) {
            if (!agent.hasReachedGoal()) {
                agentQueue.enqueue(agent);
            }
        }
    }

    public void advanceTurn() {
        if (agentQueue.isEmpty()) return;

        Agent currentAgent = agentQueue.dequeue();
        if (!currentAgent.hasReachedGoal()) {
            processAgentTurn(currentAgent);
            rotateRandomCorridor();
            if (!currentAgent.hasReachedGoal()) {
                agentQueue.enqueue(currentAgent);
            }
        }
        currentRound++;
        logTurnDetails(currentAgent);
        mazeManager.printMaze(); // her adımda maze'i bastırmak için kullanıyoz
    }

    private void processAgentTurn(Agent agent) {
        String[] directions = {"UP", "DOWN", "LEFT", "RIGHT"};
```

```
        boolean moved = false;

        for (String dir : directions) {
            if (mazeManager.isValidMove(agent,agent.getCurrentX(), agent.getCurrentY(),
dir)) {
                executeAgentMove(agent, dir);
                moved = true;
                break;
            }
        }

        if (!moved) {
            System.out.println("Agent " + agent.getId() + " is stuck!");
        }
    }

    private void executeAgentMove(Agent agent, String direction) {
        int oldX = agent.getCurrentX();
        int oldY = agent.getCurrentY();

        agent.move(direction);
        mazeManager.updateAgentPosition(agent, oldX, oldY);

        MazeTile currentTile = mazeManager.getTile(agent.getCurrentX(),
agent.getCurrentY());
        handleTileEffects(agent, currentTile);
    }
    //burda pozisyon durumlarını update etmeye çalışıyoz
    private void handleTileEffects(Agent agent, MazeTile tile) {
        switch (tile.getType()) {
            case 'T' -> {
                System.out.println("Agent " + agent.getId() + " triggered a trap!");
                agent.triggerTrap();
                // Update position after backtrack
                mazeManager.updateAgentPosition(agent, agent.getCurrentX(),
agent.getCurrentY());
            }
            case 'P' -> {
                System.out.println("Agent " + agent.getId() + " collected a power-
up!");
                agent.setPowerUp(true);
            }
            case 'G' -> {
                System.out.println("Agent " + agent.getId() + " reached the goal!");
                agent.setReachedGoal(true);
            }
        }
    }

    private void rotateRandomCorridor() {
        int[] rotatingRows = mazeManager.getRotatingRowIndexes();
        if (rotatingRows.length > 0) {
            int rowToRotate = rotatingRows[random.nextInt(rotatingRows.length)];
            System.out.println("Rotating row: " + rowToRotate);
            mazeManager.rotateCorridor(rowToRotate);
        }
    }

    public boolean allAgentsFinished() {
        for (Agent agent : agentQueue.toList()) {
            if (!agent.hasReachedGoal()) {
                return false;
            }
        }
        return true;
    }

    private void logTurnDetails(Agent agent) {
```

```
        System.out.println("\n=== Turn " + currentRound + " ===");
        System.out.println("Current Agent: " + agent.getId());
        System.out.printf("Position: (%d, %d)%n", agent.getCurrentX(),
agent.getCurrentY());
        System.out.println("Total Moves: " + agent.getTotalMoves());
        System.out.println("Backtracks: " + agent.getBacktracks());
        System.out.println("Traps Triggered: " + agent.getTrapsTriggered());
        System.out.println("Power-Up: " + (agent.hasPowerUp() ? "Active" : "None"));
        System.out.println("Goal Status: " + (agent.hasReachedGoal() ? "REACHED" : "Not
reached"));

        System.out.print("Recent Path: ");
        String[] lastMoves = agent.getLastNMoves(5);
        for (String move : lastMoves) {
            if (move != null) System.out.print(move + " → ");
        }
        System.out.println("Current");
    }

    public int getCurrentRound() {
        return currentRound;
    }

    public Agent getCurrentAgent() {
        return agentQueue.peek();
    }
}
```

Appendix B: includes Stack, Queue and CircularLinkedList structures

Stack Struct

```
//stack tanımını hocanın slaytlarından yaptım direkt
public class Stack {

    public class Node {
        String data;
        Node next;

        Node(String data) {
            this.data = data;
            this.next = null;
        }
    }

    public Node top;
    public int size = 0;

    public Stack() {
        this.top = null;
    }

    public boolean isEmpty() {
        return top == null;
    }

    public void push(String data) {
        Node newNode = new Node(data);
        newNode.next = top;
        top = newNode;
        size++;
    }

    public String pop() {
        if (isEmpty()) {
            return null;
        }
        String data = top.data;
        top = top.next;
        size--;
```

```
        return data;
    }

    public String peek() {
        if (isEmpty()) {
            return null;
        }
        return top.data;
    }

    public String[] getLastNMoves(int n) {
        String[] moves = new String[n];
        Node current = top;
        int i = 0;
        while (current != null && i < n) {
            moves[i++] = current.data;
            current = current.next;
        }
        return moves;
    }
    public int getSize() {
        return size; // Stack boyutunu döndürür
    }
    //override yazmayınca sarı uyarı verdi
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        Node current = top;
        while (current != null) {
            sb.append(current.data).append(" -> ");
            current = current.next;
        }
        sb.append("END");
        return sb.toString();
    }
}
```

CircularLinkedList Struct

```
//abi buranın ismini değiştirmedim ya diğer her yeri buna göre yazmışız o yüzden böyle
kaldı raporda dikkat ederiz
public class donsatlist{
    private class node{
        int rowIndex;
        CircularLinkedList<MazeTile> rowList;
        node next;

        node(int rowIndex, CircularLinkedList<MazeTile> rowList){
            this.rowIndex=rowIndex;
            this.rowList=rowList;
        }
    }

    private node head;
    public void addRow(int rowIndex,CircularLinkedList<MazeTile> list){  //linkedliste
satır ekleme işlemi burda yapılıyo
        node newNode = new node(rowIndex, list);
        newNode.next = head;
        head = newNode;
    }
    public CircularLinkedList<MazeTile> getRow(int rowIndex){
        node current = head;
        while(current != null){
            if(current.rowIndex==rowIndex){
                return current.rowList;
            }
            current = current.next;
        }
```

```
        return null;
    }
    public boolean contains(int rowIndex){
        node current = head;
        while(current != null){
            if(current.rowIndex==rowIndex) return true;
            current = current.next;
        }
        return false;
    }
    public int[] getAllIndexes(){
        int count = 0;
        node current = head;
        while(current!=null){
            count++;
            current = current.next;
        }
        int[] indexes = new int[count];
        current = head;
        int i = 0;
        while(current!=null){
            indexes[i++]=current.rowIndex;
            current=current.next;
        }
        return indexes;
    }
    public int getRotatingRowCount() {  //bunu senin kodlarının birinde kullanılırken
gördüm sanırım o yüzden ekledim
        int count = 0;
        node current = head;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }
}
```

Queue Struct

```
import java.util.ArrayList;
import java.util.List;

public class Queue<T> { //<T> olunca generic oluyomuş diğer veri tiplerini de döndürmek
için kullanılabiliyomuş


    private class Node {
        T data;
        Node next;

        Node(T data) {
            this.data = data;
            this.next = null;
        }
    }

    private Node front;
    private Node rear;
    private int size;

    public Queue() {
        front = null;
        rear = null;
        size = 0;
    }

    //kuyruğa eleman ekliyoruz
```

```
    public void enqueue(T data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            front =  newNode;
            rear = newNode;
        } else {
            rear.next = newNode;
            rear = newNode;
        }
        size++;
    }

    // kuyruktan eleman çıkar
    public T dequeue() {
        if (isEmpty()) {
            return null;
        }
        T data = front.data;
        front = front.next;
        if (front == null) { //boş mu bakıp sıfırladık
            rear = null;
        }
        size--;
        return data;
    }

    // en öndeki elemanı yayınlatma
    public T peek() {
        if (isEmpty()) return null;
        return front.data;
    }

    //queeue boş mu
    public boolean isEmpty() {
        return front == null;
    }

    // kaç eleman var
    public int getSize() {
        return size;
    }

    // kuyruğu yazdırma
    public void printQueue() {
        System.out.print("Queue [front -> rear]: ");
        Node current = front;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public List<T> toList() {
    List<T> list = new ArrayList<>();
    Node current = front;
    while (current != null) {
        list.add(current.data);
        current = current.next;
    }
    return list;
}

}
```

Appendix C: Some Outputs

First Output:

```
=== TURN 1 ===
MAZE STATE:
E T E E E T E E
E T W G P P P W
W T W A T E E P
E T P E T P P W
E T P W E W P T
W W E P E W P W
W E W P T E W W
P W E W E P E E
Agent 0: (3,2) | Moves: 0 | Backtracks: 0 | PowerUp: NO
Last moves: 3,2

=== FINAL STATISTICS ===
FINAL MAZE:
T E E E T E E E
E T W A P P P W
W T W P T E E P
E T P E T P P W
E T P W E W P T
W W E P E W P W
W E W P T E W W
P W E W E P E E

AGENT RESULTS:
Agent 0: GOAL | Total Moves: 1 | Backtracks: 0 | Traps: 0

GLOBAL STATS:
Total Turns: 1
First Winner: Agent 0
```

Second Output:

```
=== TURN 1 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
E T E P P W E W E T T T W G W E
T E W W E W P T P P W W E T W E
A P E W T E E P E E P P T W E P
E P W P W W T E P P P T P E W T
W E P E W T T W W E E E P E P W
Agent 0: (0,5) | Moves: 0 | Backtracks: 0 | PowerUp: NO
Last moves: 0,5

=== TURN 2 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T E P P W E W E T T T W G W E E
T E W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
A P W P W W T E P P P T P E W T
W E P E W T T W W E E E P E P W
Agent 0: (0,6) | Moves: 1 | Backtracks: 0 | PowerUp: NO
Last moves: 0,6 0,5

=== TURN 3 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
```

```
T E P P W E W E T T T W G W E E
T E W W E W P T P P W W E T W E
A P E W T E E P E E P P T W E P
P W P W W T E P P P T P E W T E
W E P E W T T W W E E E P E P W
Agent 0: (0,5) | Moves: 2 | Backtracks: 0 | PowerUp: NO
Last moves: 0,5 0,6 0,5

=== TURN 4 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
E P P W E W E T T T W G W E E T
T E W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
A W P W W T E P P P T P E W T E
W E P E W T T W W E E E P E P W
Agent 0: (0,6) | Moves: 3 | Backtracks: 1 | PowerUp: YES
Last moves: 0,6 0,5

=== TURN 5 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
E P P W E W E T T T W G W E E T
T E W W E W P T P P W W E T W E
A P E W T E E P E E P P T W E P
W P W W T E P P P T P E W T E P
W E P E W T T W W E E E P E P W
Agent 0: (0,5) | Moves: 4 | Backtracks: 1 | PowerUp: YES
Last moves: 0,5 0,6 0,5

=== TURN 6 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
P P W E W E T T T W G W E E T E
T E W W E W P T P P W W E T W E
E A E W T E E P E E P P T W E P
W P W W T E P P P T P E W T E P
W E P E W T T W W E E E P E P W
Agent 0: (1,5) | Moves: 5 | Backtracks: 2 | PowerUp: YES
Last moves: 1,5 0,5

=== TURN 7 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
P W E W E T T T W G W E E T E P
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
W P W W T E P P P T P E W T E P
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 6 | Backtracks: 2 | PowerUp: YES
Last moves: 1,4 1,5 0,5

=== TURN 8 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
P W E W E T T T W G W E E T E P
T E W W E W P T P P W W E T W E
E A E W T E E P E E P P T W E P
```

```
P W W T E P P P T P E W T E P W
W E P E W T T W W E E E P E P W
Agent 0: (1,5) | Moves: 7 | Backtracks: 2 | PowerUp: YES
Last moves: 1,5 1,4 1,5 0,5

=== TURN 9 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
W E W E T T T W G W E E T E P P
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
P W W T E P P P T P E W T E P W
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 8 | Backtracks: 2 | PowerUp: YES
Last moves: 1,4 1,5 1,4 1,5 0,5

=== TURN 10 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
W A W E T T T W G W E E T E P P
T E W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
W W T E P P P T P E W T E P W P
W E P E W T T W W E E E P E P W
Agent 0: (1,3) | Moves: 9 | Backtracks: 2 | PowerUp: YES
Last moves: 1,3 1,4 1,5 1,4 1,5

=== TURN 11 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
W E W E T T T W G W E E T E P P
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
W T E P P P T P E W T E P W P W
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 10 | Backtracks: 2 | PowerUp: YES
Last moves: 1,4 1,3 1,4 1,5 1,4

=== TURN 12 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
W A W E T T T W G W E E T E P P
T E W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
T E P P P T P E W T E P W P W W
W E P E W T T W W E E E P E P W
Agent 0: (1,3) | Moves: 11 | Backtracks: 2 | PowerUp: YES
Last moves: 1,3 1,4 1,3 1,4 1,5

=== TURN 13 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
E W E T T T W G W E E T E P P W
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
T E P P P T P E W T E P W P W W
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 12 | Backtracks: 2 | PowerUp: YES
```

```
Last moves: 1,4 1,3 1,4 1,3 1,4

=== TURN 14 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
W E T T T W G W E E T E P P W E
T E W W E W P T P P W W E T W E
E A E W T E E P E E P P T W E P
T E P P P T P E W T E P W P W W
W E P E W T T W W E E E P E P W
Agent 0: (1,5) | Moves: 13 | Backtracks: 2 | PowerUp: YES
Last moves: 1,5 1,4 1,3 1,4 1,3

=== TURN 15 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
E T T T W G W E E T E P P W E W
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
T E P P P T P E W T E P W P W W
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 14 | Backtracks: 2 | PowerUp: YES
Last moves: 1,4 1,5 1,4 1,3 1,4

=== TURN 16 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T T T W G W E E T E P P W E W E
T E W W E W P T P P W W E T W E
E A E W T E E P E E P P T W E P
T E P P P T P E W T E P W P W W
W E P E W T T W W E E E P E P W
Agent 0: (1,5) | Moves: 15 | Backtracks: 3 | PowerUp: YES
Last moves: 1,5 1,4 1,3 1,4 1,3

=== TURN 17 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T T T W G W E E T E P P W E W E
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
E P P P T P E W T E P W P W W T
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 16 | Backtracks: 3 | PowerUp: YES
Last moves: 1,4 1,5 1,4 1,3 1,4

=== TURN 18 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T T T W G W E E T E P P W E W E
T E W W E W P T P P W W E T W E
E A E W T E E P E E P P T W E P
P P P T P E W T E P W P W W T E
W E P E W T T W W E E E P E P W
Agent 0: (1,5) | Moves: 17 | Backtracks: 4 | PowerUp: YES
Last moves: 1,5 1,4 1,3 1,4 1,3

=== TURN 19 ===
```

```
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T T W G W E E T E P P W E W E T
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
P P P T P E W T E P W P W W T E
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 18 | Backtracks: 4 | PowerUp: YES
Last moves: 1,4 1,5 1,4 1,3 1,4

=== TURN 20 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T T W G W E E T E P P W E W E T
T E W W E W P T P P W W E T W E
E A E W T E E P E E P P T W E P
P P T P E W T E P W P W W T E P
W E P E W T T W W E E E P E P W
Agent 0: (1,5) | Moves: 19 | Backtracks: 5 | PowerUp: YES
Last moves: 1,5 1,4 1,3 1,4 1,3

=== TURN 21 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T W G W E E T E P P W E W E T T
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
P P T P E W T E P W P W W T E P
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 20 | Backtracks: 5 | PowerUp: YES
Last moves: 1,4 1,5 1,4 1,3 1,4

=== TURN 22 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
T W G W E E T E P P W E W E T T
T E W W E W P T P P W W E T W E
E A E W T E E P E E P P T W E P
P T P E W T E P W P W W T E P P
W E P E W T T W W E E E P E P W
Agent 0: (1,5) | Moves: 21 | Backtracks: 5 | PowerUp: YES
Last moves: 1,5 1,4 1,5 1,4 1,3

=== TURN 23 ===
MAZE STATE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
W W E T E E T P P P E W W W E E
W G W E E T E P P W E W E T T T
T A W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
P T P E W T E P W P W W T E P P
W E P E W T T W W E E E P E P W
Agent 0: (1,4) | Moves: 22 | Backtracks: 5 | PowerUp: YES
Last moves: 1,4 1,5 1,4 1,5 1,4

=== FINAL STATISTICS ===
FINAL MAZE:
P P E T E P T W P T E T P P P E
T P E T T E E W E T W T E P E E
```

```
W W E T E E T P P P E W W W E E
A W E E T E P P W E W E T T T W
T E W W E W P T P P W W E T W E
E P E W T E E P E E P P T W E P
P T P E W T E P W P W W T E P P
W E P E W T T W W E E E P E P W

AGENT RESULTS:
Agent 0: GOAL | Total Moves: 23 | Backtracks: 5 | Traps: 6

GLOBAL STATS:
Total Turns: 23
```
First Winner: Agent 0