

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА  
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА КІБЕРНЕТИКИ

**Звіт**  
**до лабораторної роботи №3**  
**з інтелектуальних систем**  
**на тему «Пошук в лабіринті»**

Виконала група студентів 4-го курсу  
Факультету комп'ютерних наук  
та кібернетики  
групи МІ-4  
Антоненко Вероніка,  
Франчук Іван,  
Трескунов Денис.

## **Постановка задачі**

Необхідно створити власний прототип ведення абстрактної гри. Прототип має містити абстрактний клас або інтерфейс, що буде описувати загальну частину (Стан, Початковий стан, Функція визначення нащадка, Перевірка цілі, Вартість шляху).

В якості реалізації необхідно розробити спрощений варіант гри Pac-Man. Задача - в існуючому лабіринті пакмен має знайти шлях до цілі та з'їсти її.

Шлях шукати різними способами використовуючи алгоритми з лекції та виводити статистику:

- Час витрачений на пошук
- Кількість кроків зроблену під час пошуку
- Кількість оперативної пам'яті, що була витрачена на пошук

Необхідно зробити графічний інтерфейс схожий на оригінал.

## Виконання

В процесі реалізації було використано два алгоритми для обходу лабіринту – BFS та DFS. Їх написанням займався Трескунов Денис (клас *Graph.java*).

```
// ALGORITHMS
// return path without start and null if no path was found
public AlgorithmData<T> BFS(T start, ArrayList<T> targets) {
    if (targets.size() == 0) return null;
    boolean visited[] = new boolean[vertices.size()];
    LinkedList<T> queue = new LinkedList<T>();

    // Mark the current node as visited and enqueue it
    visited[vertices.indexOf(start)] = true;
    queue.add(start);

    T s = start;
    ArrayList<T> result = new ArrayList<>();
    HashMap<T, T> parentList = new HashMap<>(); // key - child, value - parent
    while (queue.size() != 0)
    {
        s = queue.poll();

        // Get all adjacent vertices of the dequeued vertex s
        // If a adjacent has not been visited, then mark it
        // visited and enqueue it
        ArrayList<T> neighbours = getNeighbours(s);
        for (T neighbour: neighbours) {
            if (!visited[vertices.indexOf(neighbour)]) {
                parentList.put(neighbour, s); // adding parent
                if (targets.contains(neighbour)) {

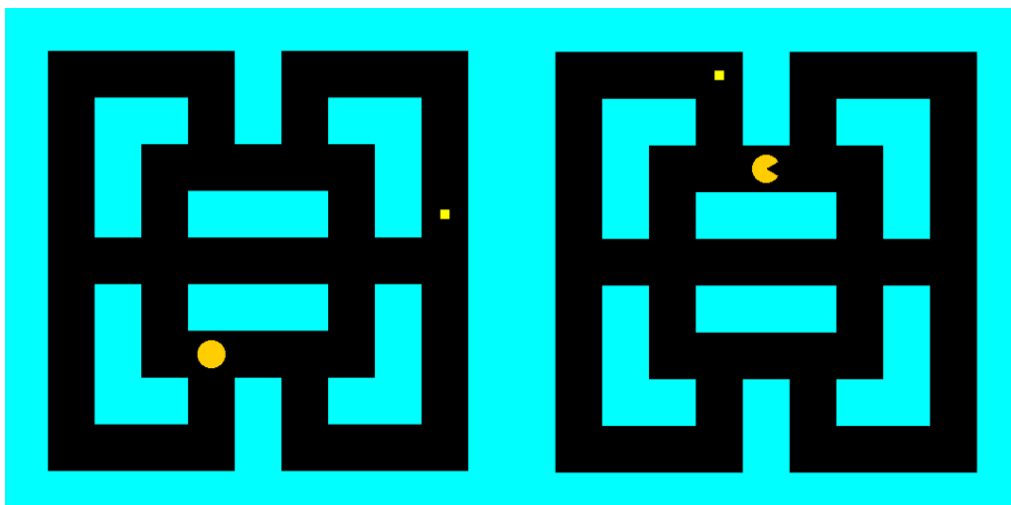
                    result.add(neighbour);
                    T parent = parentList.get(neighbour);
                    while (parent != start) {
                        result.add(parent);
                        parent = parentList.get(parent);
                    }
                    Collections.reverse(result);
                    return new AlgorithmData(parentList.size(), result);
                }
                visited[vertices.indexOf(neighbour)] = true;
                queue.add(neighbour);
            }
        }
    }
    return null;
}
```

```

public AlgorithmData<T> DFS(T start, ArrayList<T> targets) {
    if (targets.size() == 0) return null;
    boolean[] visited = new boolean[vertices.size()]; // private field for dfs
    Stack<T> stack = new Stack<>();
    stack.push(start);
    T s = start;
    while (stack.size() != 0) {
        visited[vertices.indexOf(s)] = true;
        ArrayList<T> neighbours = getNeighbours(s);
        ArrayList<T> unvisitedNeighbours = new ArrayList<>();
        for (T neighbour: neighbours) {
            if (!visited[vertices.indexOf(neighbour)]) unvisitedNeighbours.add(neighbour);
        }
        if (unvisitedNeighbours.size() == 0) {
            stack.pop();
            s = stack.peek();
        }
        for (int i = 0; i < unvisitedNeighbours.size(); i++) {
            T neighbour = unvisitedNeighbours.get(i);
            if (!visited[vertices.indexOf(neighbour)]) {
                stack.push(neighbour);
                s = neighbour;
                if (targets.contains(neighbour)) {
                    ArrayList<T> result = new ArrayList<>(stack);
                    result.remove(0);
                    int count = 0;
                    for (boolean bool: visited) {
                        if (bool) count++;
                    }
                    return new AlgorithmData<T>(count, result);
                }
                break;
            }
        }
    }
    return null;
}

```

Інтерфейс програми зроблений за зразком гри Пакман. Зчитування лабіринта відбувається з файлу, а її генерацією займається окремий клас.. Виконанням цієї частини займалася Антоненко Вероніка. (класи *DrawManager*, entity *PacMan*, *Direction*)



```

public static void drawMaze(Maze maze, Graphics g) {

    // filling background with green
    g.setColor(Color.CYAN);
    g.fillRect(0, 0, Finals.WINDOW_WIDTH, Finals.WINDOW_HEIGHT);

    // filling every cell
    for(Cell cell: maze.getCells()) {
        switch (cell.getType()) {
            case Wall:
                g.setColor(Color.CYAN);
                g.fillRect(Finals.SIDE_GAP + cell.getX()*Finals.CELL_SIZE, Finals.TOP_GAP + cell.getY()*Finals.CELL_SIZE,
                    Finals.CELL_SIZE, Finals.CELL_SIZE);
                break;
            case Empty:
                g.setColor(Color.BLACK);
                g.fillRect(Finals.SIDE_GAP + cell.getX()*Finals.CELL_SIZE, Finals.TOP_GAP + cell.getY()*Finals.CELL_SIZE,
                    Finals.CELL_SIZE, Finals.CELL_SIZE);
                break;
            case Tablet:
                g.setColor(Color.BLACK);
                g.fillRect(Finals.SIDE_GAP + cell.getX()*Finals.CELL_SIZE, Finals.TOP_GAP + cell.getY()*Finals.CELL_SIZE,
                    Finals.CELL_SIZE, Finals.CELL_SIZE);
                g.setColor(Color.YELLOW);
                g.fillRect(Finals.SIDE_GAP + cell.getX()*Finals.CELL_SIZE + Finals.TABLET_GAP,
                    Finals.TOP_GAP + cell.getY()*Finals.CELL_SIZE + Finals.TABLET_GAP,
                    Finals.TABLET_SIZE, Finals.TABLET_SIZE);
                break;
        }
    }
}

```

```

public static void drawEntity(Entity entity, Graphics g) {

    // reading correct image
    BufferedImage PW_image = null;
    try {
        PW_image = ImageIO.read(new File(entity.getPicturePath()));
    }
    catch (Exception e) {
        System.err.println(e.getMessage());
    }

    g.drawImage(PW_image, entity.getCurrentPosition().getX()*Finals.CELL_SIZE + Finals.SIDE_GAP + Finals.PACMAN_GAP,
        entity.getCurrentPosition().getY()*Finals.CELL_SIZE + Finals.TOP_GAP + Finals.PACMAN_GAP,
        Finals.PACMAN_SIZE, Finals.PACMAN_SIZE, null);
}

```

За забезпечення коректного зв'язку усіх елементів між собою, виведення даних про роботу програми та написання junit тестів відповідав Франчук Іван. (класи *GameMain*, *Finals*, *AlgorithmData*, *test1-5*).

Виведення статистики виконання відбувається у файл для обох алгоритмів.

BFSInfo.txt	DFSInfo.txt
BFS ALGORITHM	DFS ALGORITHM
Number of steps: 1	Number of steps: 1
Memory consumed: 30138624 bytes	Memory consumed: 30656536 bytes
Time used: 1 milliseconds	Time used: 1 milliseconds
BFS ALGORITHM	DFS ALGORITHM
Number of steps: 26	Number of steps: 4
Memory consumed: 30138624 bytes	Memory consumed: 30656536 bytes
Time used: 1 milliseconds	Time used: 1 milliseconds
BFS ALGORITHM	DFS ALGORITHM
Number of steps: 54	Number of steps: 25
Memory consumed: 15153504 bytes	Memory consumed: 31829128 bytes
Time used: 2 milliseconds	Time used: 1 milliseconds

```

public void paint(Graphics g) {
    try {
        DrawManager.drawMaze(maze, g);
        DrawManager.drawEntity(pacMan, g);

        while (maze.areThereTablets()) {

            // writing info and counting time and memory
            File file = new File(Finals.PROJECT_PATH.concat("results/" + Finals.ALGORITHM_USED + "info.txt"));
            FileWriter fr = new FileWriter(file, true);
            fr.write(Finals.ALGORITHM_USED + " ALGORITHM\n");
            fr.close();

            // time
            long timeBefore = System.currentTimeMillis();
            // memory
            Runtime rt = Runtime.getRuntime();
            long memoryBefore = (rt.totalMemory() - rt.freeMemory());

            pacMan.pathToTarget(Graph.class.getMethod(Finals.ALGORITHM_USED, Object.class, ArrayList.class), maze);

            long memoryAfter = (rt.totalMemory() - rt.freeMemory());
            long memory = memoryAfter - memoryBefore;

            long timeAfter = System.currentTimeMillis();
            long time = timeAfter - timeBefore;
            fr = new FileWriter(file, true);
            fr.write("Memory consumed: " + memoryAfter + " bytes \n");
            fr.write("Time used: " + time + " milliseconds \n");
            fr.close();

            while (pacMan.getPath().size() != 0) {
                TimeUnit.MILLISECONDS.sleep(100);

                pacMan.nextPosition(Graph.class.getMethod(Finals.ALGORITHM_USED, Object.class, ArrayList.class), maze);
                DrawManager.drawMaze(maze, g);
                DrawManager.drawEntity(pacMan, g);
            }
            if (!maze.areThereTablets()) maze.newRandomTablet(pacMan.getCurrentPosition());
        }
    } catch (Exception e) {
        System.err.println(e);
    }
}

```