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

Звіт
до лабораторної роботи
з інтеллектуальних систем
на тему «Pacman+»

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

КИЇВ-2020

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.

І друга частина лабораторної:

В лабіринт додати:

- Точки. За кожну точку, що пакман з'їв додавати +1 бал. Якщо пакман з'їв всі точки на карті це означає, що він виграв
- Привидів. Привиди рухають картою і намагаються з'їсти пакмана. Якщо привид з'їв пакмана - програш
- Рівні. З кожним рівнем збільшується складність гри.
- Привиди та пакман мають керуватися мінімаксним алгоритмом або альфа-бета відтинанням.
- В алгоритм привидів, додати рандомний крок.

Виконання

В процессі реалізації було використано два алгоритми для обходу лабіринту – BFS та DFS. За вимогами другої частини, надалі вони були замінені на мінімаксний алгоритм. Їх написанням займався Трєскунов Денис *(src/graphs/*)*.

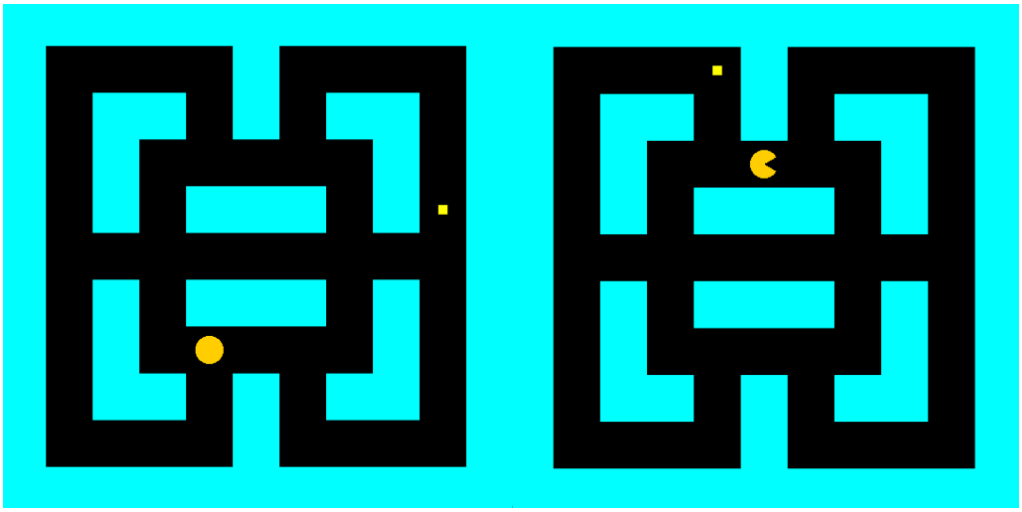
```
// return next vertex to go
public T MinMax(int i, GraphVertex<T> start, GraphData<T> graph, int depth, boolean pacwoman) {
    int bestOption = MMrecursive(start, graph, depth, pacwoman);
    Random random = new Random();
    int prob = random.nextInt() % Finals.PROBABILITY_OF_RANDOM;
    ArrayList<GraphVertex<T>> children = graph.findChildren(start);
    if (prob == 0 && !pacwoman) {
        prob = random.nextInt() % children.size();
        while (prob < 0) prob += children.size();
        return children.get(prob).ghosts.get(i);
    }
    else {
        for (GraphVertex<T> child: children) {
            if (child.value == bestOption) {
                if (pacwoman) {
                    return child.pacwoman;
                }
                else {
                    return child.ghosts.get(i);
                }
            }
        }
        return null;
    }
}
```

```
// 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;
}
```

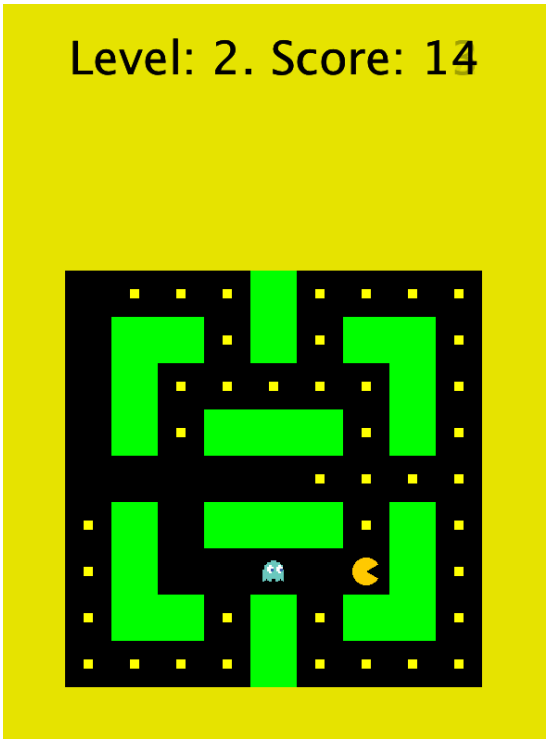
```java
public AlgorithmData<T> DFS(T start, ArrayList<T> targets) {
    if (targets.size() == 0) return null;
    boolean[] visited = new boolean[vertices.size()]; // private field for dfs
    Stack<T> stack = new Stack<>();
    stack.push(start);
    T s = start;
    while (stack.size() != 0) {
        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. Також було реалізовано перехід на наступні рівні, додавання привидів. Виконанням цієї частини займалася Антоненко Вероніка. *(src/maze/*)*

Перша версія



Друга версія – з привидами




```
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, створення лабіринту, виведення даних про роботу програми, написання junit тестів та контроль якості відповідав Франчук Іван. *(src/main/*, src/entities/*).*

Виведення статистики виконання відбувається у файл для обох алгоритмів.

BFSinfo.txt

```
BFS ALGORITHM
Number of steps: 1
Memory consumed: 30138624 bytes
Time used: 1 milliseconds
BFS ALGORITHM
Number of steps: 26
Memory consumed: 30138624 bytes
Time used: 1 milliseconds
BFS ALGORITHM
Number of steps: 54
Memory consumed: 15153504 bytes
Time used: 2 milliseconds
```

DFSinfo.txt

```
DFS ALGORITHM
Number of steps: 1
Memory consumed: 30656536 bytes
Time used: 1 milliseconds
DFS ALGORITHM
Number of steps: 4
Memory consumed: 30656536 bytes
Time used: 1 milliseconds
DFS ALGORITHM
Number of steps: 25
Memory consumed: 31829128 bytes
Time used: 1 milliseconds
```

```java
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);
    }
}
```