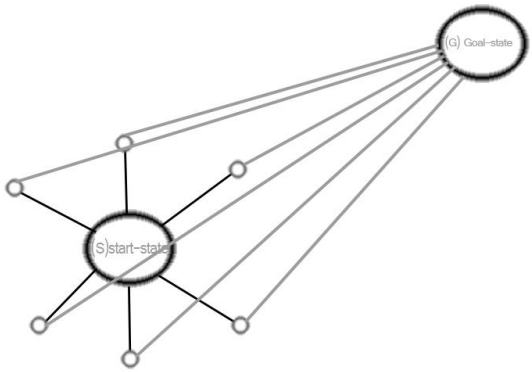
Student: Rayhan Miah
Technical report on: Artificial Intelligence first coursework
Lecturer: Tim Blackwell
Due date: 11/11/2016

Introduction:

The point of interest chosen was on informed search, informed search is where there is a search for the goal this concept looks for a discovery of a goal from the start state to the goal state and the path is irrelevant. It can focus on the following: heuristic search which is a concept in Ai used to solve problems quickly and efficiently and it gives an estimate of the distance function starting from the start state to the goal state and selects the path with shortest distance to the goal. It may also have some information to the goal and path isn't important. The inventive algorithm that will be created will be called Ant hill climbing/gradient descent, this is where it uses a well known algorithm called hill climbing, which computes distance using Euclidean between all successors and goal and chooses the one with smallest distance, which is the closest.

Hill Climbing/Gradient Descent:

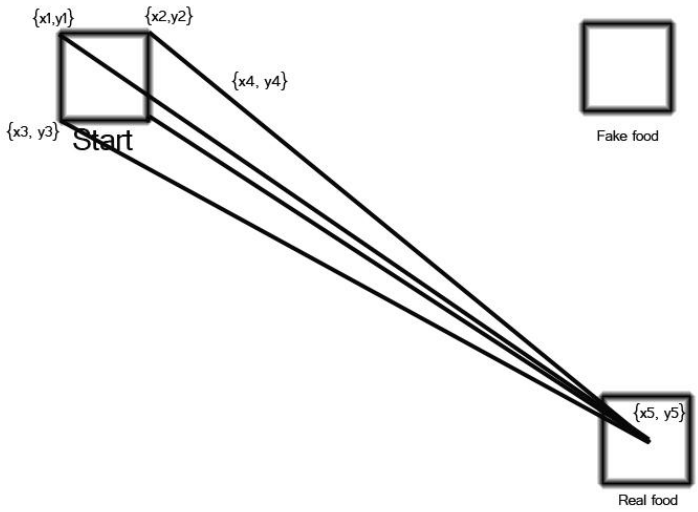


Here is an example of an agent on the (s)start-state trying to get to the (G) Goal-state

Looking at the diagram above suppose the agent has some information and path to the goal isn't important. Suppose the agent understands that in this situation it would have to be a hill climbing approach to the goal, the way in which it would do this is by using hill climbing by computing the distance using Euclidean distance between all the successors (start node) and the goal and choose the one with smallest distance. The start state will have coordinates (x1, y1) and the goal state will have coordinates (x2, y2). Euclidean distance formula is: $\sqrt{[(x2 - x1)^2 + (y2 - y1)^2}$

Sketch plan and pseudocode

Here is a sketch for the project plan of using hill climbing algorithm in the program.



There are 3 states in this: a start state, fake food and real food and a pseudo code will be implemented in showing how you can get from start to real food from 4 positions in the start state and using gradient descent algorithm to find out which is the shortest path.

Pseudo code:

Create width variable set to 640

Create height variable set to width variable * 8

Create playWindowHeight set to width * 65

Create sizeOfCell set to width/20

Create ArrayList of type towers initialise as new ArrayList of towers

Create 3 image variables

Create array of images

Create array of Vector called pathway

Create 2d Array of Cell class

function SETUP()

setUpTheSize(width, height)

set 3 image variables to the images

pass in the pathMask

define array of Images

for(x = 0 to grid.length())

for(y = 0 to grid[0].length

Grid[x][y] = new cell(x,y)

```
        end for
end for

function MOUSECHECK()//calculate where the mouse is in relation to the grid
Create variable set x coordinate to mouseX/cellSize
Create variable set y coordinate to mouseY/cellSize
      if X coordinate < grid.length & y coordinate < grid[0].length then
            Grid[x][y].outlineMe()
       end if
end function
function MOUSEPRESSED()//calculate where the mouse is in relation to the grid
      if hovercell ! = null then
            print(hovercell.x +"", hovercell.y)
       end if
end function

function SETUNBUILDABLE(VECTOR V)//calculate where the mouse is in relation to the grid
    grid[v.x][v.y].isPath = true;
end function

function DRAW()
Initialise and set image background
Initalise and set searchPath
Call mouseCheck(function)
      for(i = 0 to allAnts.size())
          AllAnt.get(i).drawMe()
       end for
```

end function

Class cell{

Set Boolean variable called isPath to false

Set X and Y coordinates on grid

Create constructor(x,y)

Set constructor values

function OUTLINEME()

noColor()

if(buildable()) **then**

setOutline color(blue)

else

setOutline(red)

createShape()

end if

end function

function Boolean BUILDABLE()

if Occupant = null **then**

Return true

else

Return false

end if

end function

function BUILDON(Ant a)

```
    if Buildable() then
            Occupant = a
       Add to arrayList of ants
    end if
end function
end class

class ANT{
    create cell x coordinate variable
    create cell y coordinate variable
     create constructor(x,y)
            set constructor values
      function DRAWME()
         //draw images of ants
      end function
end class

class VECTOR{
create x and y variable
create Construtor (x,y)
    set constructor values
end constructor
end class

class ANTSPRITES{
        create PImage array called up = new PImage[2];
        create PImage array called down = new PImage[2];
```

```
        create PImage array called left = new PImage[2];
        create PImage array called right = new PImage[2];
create constructor()
// loadimages for the rest of the directions and each of them has two movements of 2 frame animation
up[0] = loadImage(path +u0.png)
up[1] = loadImage(path + u1.png)
down[0] = loadImage(path +d0.png)
down[1] = loadImage(path + d1.png)
left[0] = loadImage(path +l0.png)
left[1] = loadImage(path + l1.png)
right[0] = loadImage(path +r0.png)
right[1] = loadImage(path + r1.png)
end constructor
end class

class HEURISTICSEARCH{
      create vector array called pathway
       Lane[] allLanes
       Vector spawnPos
Subclass GRADIENTDESCENT{
       Create in variable for endpos
        Direction dir
Create constructor(Direction_dir, Vector cell)
    dir = _dir
if dir == Direction.up || _dir = Direction.down then
```

```
      endPos = round(cellsize * cell.y)
end if
end constructor
end subclass
function GRADIENTDESCENT(VECTOR[] V)
           pathway = v
for(i=0 to v.length -1)
     grid[v[i].x][v[i].y].isPath = true
create variable for difference = v[i].x – v[i+1].x
if  difference ! = 0 then
   for(z = 0 to abs(difference))
if difference < 0 then
     grid[v[i].x + z][v[i].y].isPath = true
else
    grid[v[i].x - z][v[i].y].isPath
end if
end for
else
      difference = v[i].y – v[i+y].y
for(z = 0 to abs(difference))
if difference < 0 then
       grid[v[i].x][v[i].y + z]
else
      grid[v[i].x][v[i].y –z]
end if
end for
end for
```

end function

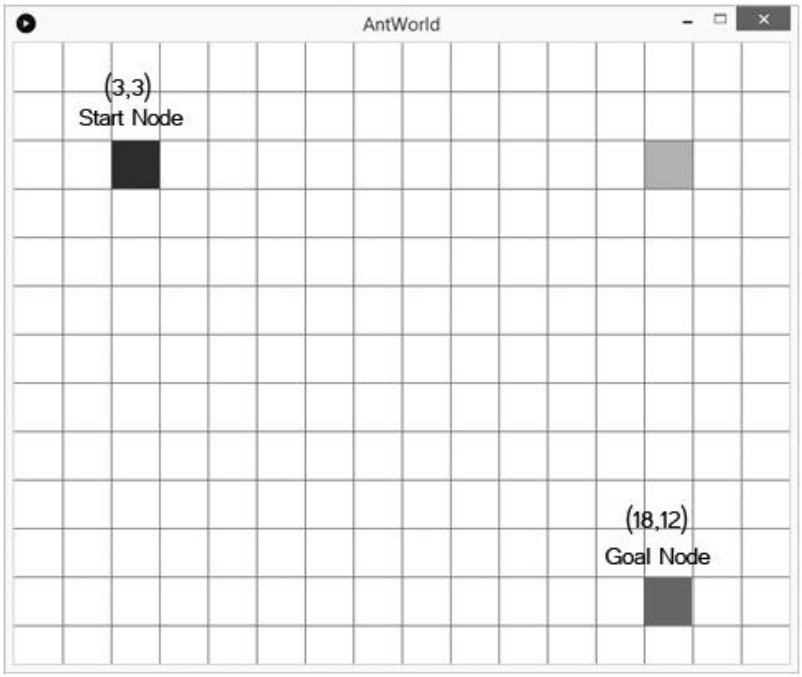
end class

Experiments:

After writing the pseudo code implementation of code was implemented into the original project of ant world with a focus on gradient descent algorithm. 3 experiments were conducted based on sprite animation of ants with them having a 2 frame animation of movement. The experiment was based on the ant having different start nodes and with the same goal node. The ant travels through the start node, which is the starting position to the goal node which is the food by using hill climbing.

3 experiments were conducted using the gradient descent algorithm. The purpose of this experiment was too find the path with the shortest distance from the start node to the end goal node, this was conducted by doing different start and goal nodes taking into account: time, space, quality and completeness.

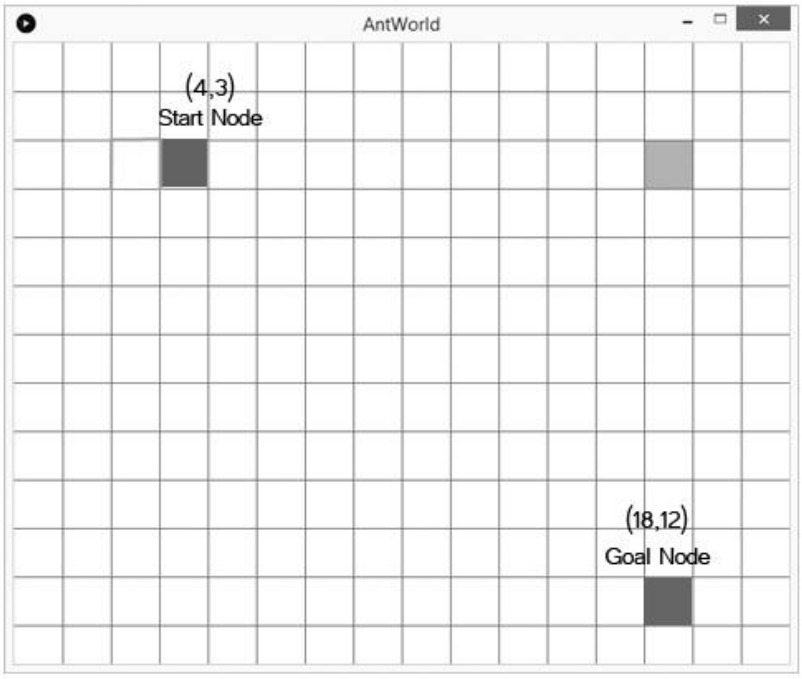
First experiment :



From the first experiment the start node had coordinates (3,3) and goal node had coordinates (18,12) using the formula Euclidean distance: $\sqrt{[(x2 - x1)^2 + (y2 - y1)^2}$ to find the distance between these two points: $\sqrt{[(18 - 3)^2 + (12 - 3)^2=}$

$\sqrt{(15)^2 + (9)^2} = \sqrt{(306)} = 17.49286$

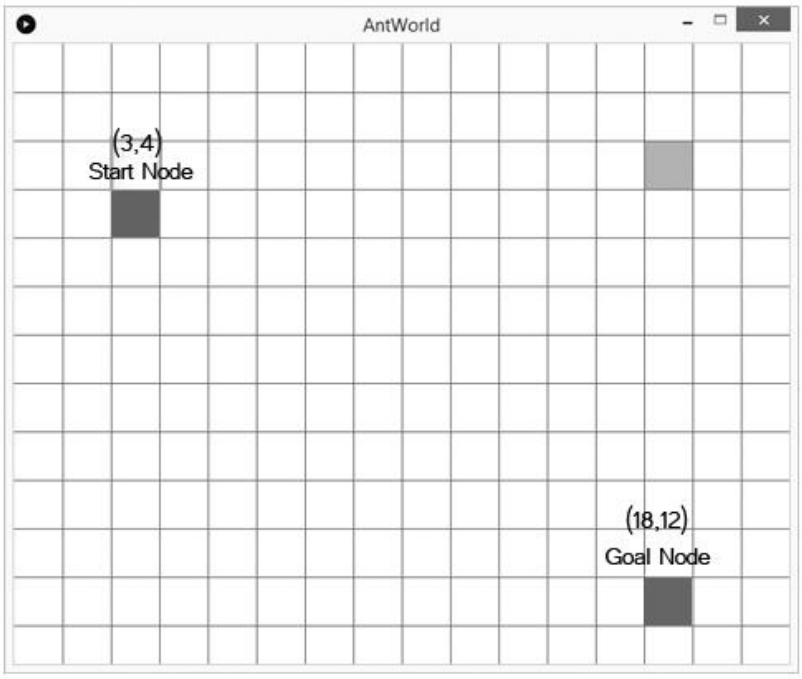
Second experiment :



From the second experiment the start node had coordinates (4,3) and goal node had coordinates (18,12) using the formula Euclidean distance: $\sqrt{[(x2 - x1)^2 + (y2 - y1)^2}$ to find the distance between these two points: $\sqrt{[(18 - 4)^2 + (12 - 3)^2=}$

$\sqrt{(14)^2 + (9)^2 = \sqrt{(277)} = 16.64332}$

Third experiment :



From the second experiment the start node had coordinates (3,4) and goal node had coordinates (18,12) using the formula Euclidean distance: $\sqrt{[(x2 - x1)^2 + (y2 - y1)^2}$ to find the distance between these two points: $\sqrt{[(18 - 3)^2 + (12 - 4)^2=}$

$\sqrt{(15)^2 + (8)^2 = \sqrt{(289)} = 17}$

Results :

After obtaining these experiments you can see that the first start position to the goal had a distance of 17.49286. The second start position to the goal had a distance of 16.64332 and the third start position to the goal had a distance of 17, it is best to go with the second start position as it has the smallest distance to the goal node out of the thee.

Evaluation:

The project was focused on Informed search and the purpose was to use gradient descent algorithm and sprite animation. starting at different nodes and working out the node with the minimal distance to the goal node, evaluating the algorithm it is best to go with start node 2 as it has a minimal distance of 16.64332 this is efficient in terms of space. In terms of time this depends on the magnitude of time in particular taking into account of the HeuristicSearch algorithm this depends on how long the ant sprite animation takes to travel to the goal node. In terms completeness the algorithm addresses all possible inputs and it doesn't miss any for example the algorithm knows which way to turn in terms of the ant animation. The algorithm has a start node and it compares it to the next movement and determines whether it needs to go up, down, left or right and how many, the algorithm does this by figuring out if the difference is in the x-axis or y-axis, therefore knowing which way to turn given some source of knowledge. Therefore, it is complete. Furthermore, the

algorithm is sound as it yields results, which are true by always moving in the right direction based on some knowledge to the goal. However, if there was more time, time would need to be taken into account to measure how long the ant takes to reach the food as different start nodes to the goals nodes may vary. Significantly, to improve this algorithm another algorithm would be invented to measure how long a particular ant takes to find the goal.

Snippets of original codes:

```
//generating variables
static int _width = 640;//width variable is auto-generated
static int _height = (int)(_width * .8);//height variable
static float cellSize = _width/20;
CellOne[][] Grid = new CellOne[20][13];
PImage[] AntSprite; //image variable to store ant picture
AntSpriteOne[] AntSpriteOne;//array object to store sprite animation of ants
CellOne hoverCell = null;//not part of the cell
HeuristicSearch Level;//
ArrayList<AntOne> AllAnts = new ArrayList<AntOne>();
ArrayList<Creep> AllCreeps = new ArrayList<Creep>();

//-------code implementation of the grid using double for loop under setup-------//
  for (int x = 0; x < Grid.length; x++) {//using double for loop to loop through all the cells
    for (int y = 0; y < Grid[0].length; y++) {
      Grid[x][y] = new CellOne(x, y);
    }
  }



  AntSprite = new PImage[] {
    loadImage("Ants/ants.png"),//setting the ant image for debugging

  };

  AntSpriteOne = new AntSpriteOne[]{ //creating  object for sprite animation
    new AntSpriteOne("ant"),

  };
     //Creating a vector object inside of this code it is where the ant starts for example at (3,3) and
reaches point by travelling through these two points (18,12)
     //use euclidean distance between each of these points and choose the one with lowest distance
  Vector[] _path = new Vector[] {
    //point one
    new Vector(3, 3), //start point
    new Vector(18, 12), //end point
    //point two
    new Vector(4, 3),
    new Vector(18, 12),
    //point three
    new Vector(3, 4),
    new Vector(18, 12),
```

```
  };

  Level = new HeuristicSearch(_path);
    //these are the paths for it not to go on
  setUnbuildable(new Vector(0, 0));
  setUnbuildable(new Vector(1, 0));
  setUnbuildable(new Vector(1, 2));
  setUnbuildable(new Vector(0, 2));
  setUnbuildable(new Vector(0, 3));


  //under draw
for (int i = 0; i < AllAnts.size(); i++) {
    AllAnts.get(i).drawMe();
  }

  for (int i = 0; i < AllCreeps.size(); i++) AllCreeps.get(i).move();

  mouseCheck();

//checking where the mouse is
void mouseCheck() {
  int x = (int)(mouseX / cellSize);
  int y = (int)(mouseY / cellSize);

  if (x < Grid.length && y < Grid[0].length) {
    hoverCell = Grid[x][y];
    hoverCell.outlineMe();
  }
}

void mousePressed() {//used for debugging checking coordinates
  if (hoverCell != null) {
    println("new Vector(" + hoverCell.x + ", " + hoverCell.y + "),");
    if (hoverCell.buildable()) hoverCell.buildOn(new AntOne(hoverCell.x, hoverCell.y, 0));
  }
}

void setUnbuildable(Vector v) {
  Grid[v.x][v.y].isPath = true;
}

public void keyPressed() {

    if(key == 'a') AllCreeps.add(new Creep(0));
  else if (key == 's') AllCreeps.add(new Creep(1));
}

class AntOne {
  //where the ant is
```

```
  int cellX;
  int cellY;

  int spriteIndex;
  int tint;

  void drawMe() {
    image(AntSprite[spriteIndex], cellX * cellSize, cellY * cellSize, cellSize, cellSize);//this draws image
of ant
  }

  AntOne(int x, int y, int sprite) {
    cellX = x;
    cellY = y;
    spriteIndex = sprite;
  }
}

class Vector {//vector for x and y coordinates
  int x;
  int y;

  Vector(int _x, int _y) {
    x = _x;
    y = _y;
  }
}

//class used to create the ant sprite animation
class AntSpriteOne {
  //creating PImage variable for storing ants
  PImage[] up = new PImage[2];
  PImage[] down = new PImage[2];
  PImage[] left = new PImage[2];
  PImage[] right = new PImage[2];

  PImage getImage(Direction dir, int anim){
    if(dir == Direction.up) return up[anim];
    else if(dir == Direction.down) return down[anim];
    else if(dir == Direction.left) return left[anim];
    else return right[anim];
  }

  AntSpriteOne(String name) {//constructor and passing in two images each for up, down, left or
right depending on which way the ant is going
    String path = "creeps/" + name + "/" + name + "_";
    up[0] = loadImage(path + "u0.png");
    up[1] = loadImage(path + "u1.png");
    down[0] = loadImage(path + "d0.png");
    down[1] = loadImage(path + "d1.png");
    left[0] = loadImage(path + "l0.png");
```

```
    left[1] = loadImage(path + "l1.png");
    right[0] = loadImage(path + "r0.png");
    right[1] = loadImage(path + "r1.png");
  }
}

//creating a class to determine where the cell is
class CellOne {
  int x;
  int y;

  boolean isPath = false;//check if it is part of the path

  AntOne occupant = null;

  void buildOn(AntOne t) {
    if (buildable()) {
      occupant = t;
      AllAnts.add(occupant);
    }
  }

  boolean buildable() {
    if (occupant == null && !isPath)return true;
    else return false;
  }

  void outlineMe() {//called when hover over a cell
    noFill();
    if (buildable())stroke(#00FF00);
    else stroke(#FF0000);
    rect(x * cellSize, y * cellSize, cellSize, cellSize);
  }

  CellOne(int _x, int _y) {
    x = _x;
    y = _y;
  }
}

class Creep{
  int posX;
  int posY;
  int onLane = 0;
  Direction dir;

  int speed = 1;

  int spriteIndex;
  int anim = 0;
```

```
  int animTimer = 0;
  int animDelay = 5;

  void move(){
    PathStatus status = Level.checkPos(new Vector(posX, posY), onLane);
    if(status == PathStatus.finished){

      return;
    } else if (status ==  PathStatus.next){
      onLane ++;
      dir = Level.getDir(onLane);
    }

    if(dir == Direction.up) posY -= speed;
    else if(dir == Direction.down) posY += speed;
    else if(dir  == Direction.left) posX -= speed;
    else posX += speed;

    image(AntSpriteOne[spriteIndex].getImage(dir, anim), posX, posY, cellSize, cellSize);

    animTimer++;
    if(animTimer > animDelay){
      animTimer = 0;
      if(anim == 0) anim = 1;
      else anim = 0;
    }

  }

  Creep(int sprite){
    spriteIndex = sprite;
    Vector p = Level.getSpawn();
    posX = p.x;
    posY = p.y;
    dir = Level.getDir(0);
  }
}

class HeuristicSearch {
  Vector[] Pathway;
  gradientDescentAlgo[] allLanes;
  Vector spawnPos;

  Vector getSpawn(){
    return spawnPos;
  }

  Direction getDir(int i){
    if(i < allLanes.length) return allLanes[i].dir;
    else return allLanes[allLanes.length - 1].dir;
  }
```

```
PathStatus checkPos(Vector v, int lane){
  if(lane > allLanes.length) return PathStatus.finished; //done with path

  boolean check = allLanes[lane].checkPos(v);
  if(check) return PathStatus.next; // turn
  else return PathStatus.stay; // keep going
}

class gradientDescentAlgo{//subclass of heuristicSearch
  int endPos;
  Direction dir;

  boolean checkPos(Vector v){
    if(dir == Direction.up){
      if(endPos > v.y) return true;
      else return false;
    } else if(dir == Direction.down){
      if(endPos < v.y) return true;
      else return false;
    } else if(dir == Direction.left){
      if(endPos > v.x) return true;
      else return false;
    } else {
      if(endPos < v.x) return true;
      else return false;
    }
  }

  gradientDescentAlgo(Direction _dir, Vector cell){
    dir = _dir;
    if(_dir ==  Direction.up || _dir == Direction.down) endPos = round(cellSize * cell.y);
    else endPos = round(cellSize * cell.x);
  }
}

HeuristicSearch(Vector[] v) {
  Pathway = v;
  Direction[] directions = new Direction[v.length - 1];
  spawnPos = new Vector(round(v[0].x * cellSize), round(v[0].y * cellSize));

  for (int i = 0; i < v.length - 1; i++) {
    Grid[v[i].x][v[i].y].isPath = true;

    int difference = v[i].x - v[i + 1].x;
    if (difference  != 0) {//using x
      for (int z = 0; z < abs(difference); z++) {
        if (difference < 0){
          Grid[v[i].x + z][v[i].y].isPath = true;//path goes right
          directions[i] = Direction.right;
        }
```

```
        else{
          Grid[v[i].x - z][v[i].y].isPath = true;//path goes left
          directions[i] = Direction.left;
        }
       }
      }
      else { //using y
       difference = v[i].y - v[i + 1].y;

       for (int z = 0; z < abs(difference); z++) {
        if (difference < 0){
          Grid[v[i].x][v[i].y + z].isPath = true; //path goes down
          directions[i] = Direction.down;
        }
        else {
          Grid[v[i].x][v[i].y - z].isPath = true;// path goes up
          directions[i] = Direction.up;
        }
       }
      }
     }

    if (v.length > 0)Grid[v[v.length - 1].x][v[v.length - 1].y].isPath = true;

    allLanes = new gradientDescentAlgo[directions.length];
    for(int i = 0; i < allLanes.length; i++) allLanes[i] = new gradientDescentAlgo(directions[i], v[i + 1]);
   }
}

//enums class determine direction of movement and what to do in path
enum Direction {up, down, left, right};
enum PathStatus {stay, next, finished};
```