KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Programavimo kalbų teorija (P175B124)

Laboratorinių darbų ataskaita

Atliko:

IFF-1/8 gr. studentas

Matas Palujanskas

2023 m. kovo 29 d.

Priėmė:

doc. Sajavičius Svajūnas

KAUNAS 2023

TURINYS

1. C++(L1) 3

1.1. Darbo užduotis 3

1.2. Programos tekstas 4

1.3. Pradiniai duomenys ir rezultatai 7

2. Scala(L2) 8

2.1. Darbo užduotis 8

2.2. Programos tekstas 8

2.3. Rezultatai 20

# C++(L1)

## Darbo užduotis

p575

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

## Programos tekstas

#include <iostream>

#include <fstream>

#include <string>

#include <cmath>

#include <chrono>

using namespace std;

using namespace std::chrono;

/// <summary>

/// Skew binary to decimal number class

/// </summary>

struct SkewBinaryToDecimal

{

string line; //binary number

/// <summary>

/// Converting from skew binary to decimal number

/// </summary>

/// <param name="line"> Skew binary number </param>

/// <param name="n"> Amount of numbers </param>

/// <returns> Converted decimal number </returns>

int to\_decimal(string line, int& n)

{

int res = 0;

for (int j = 0; j < n; j++)

{

int l = line.size();

for (int i = 0; i < l; i++) {

res += (((int)line[i] - 48) \* (pow(2, l - i) - 1));

}

return res;

}

}

};

/// <summary>

/// Decimal to skew binary number class

/// </summary>

struct DecimalToSkewBinary {

int decimal; //decimal number

string decimalToSkewBinary(int number) {

if (number == 0) {

return "0";

}

string result = "";

while (number > 0) {

int remainder = number % 3;

result = to\_string(remainder) + result;

number /= 3;

if (remainder == 2 && number > 0) {

number++;

}

}

return result;

}

};

/// <summary>

/// Printing converted numbers to file

/// </summary>

/// <param name="outputFile"> Output file </param>

/// <param name="line"> Binary numbers </param>

/// <param name="convertedNumber"> Decimal numbers </param>

/// <param name="append"></param>

void PrintResult1(string outputFile, string line,

int convertedNumber, bool append)

{

ofstream result;

if (append)

result.open(outputFile, ios\_base::app);

else

result.open(outputFile);

result << "Skew binary: " << line << " " << "decimal: " << convertedNumber << endl;

return;

}

/// <summary>

/// Reading data and performing all tasks

/// </summary>

/// <param name="inputFile"> Data file </param>

/// <param name="outputFile"> Result file </param>

void ReadAndPerformToDecimal(string inputFile, string outputFile)

{

int n;// Amount of numbers

SkewBinaryToDecimal toDecimal[20];

ifstream data(inputFile);

data >> n;

for (int i = 0; i < n; i++)

{

string line;

data.ignore();

data >> toDecimal[i].line;

line = toDecimal[i].line;

int calculated = toDecimal->to\_decimal(line, n);

//string skew\_Binary = toBinary->to\_skew\_binary(calculated, n)

bool append = false;

if (i > 0)

append = true;

PrintResult1(outputFile, line, calculated, append);

}

data.close();

return;

}

// <summary>

/// Printing converted numbers to file

/// </summary>

/// <param name="outputFile"> Output file </param>

/// <param name="line"> Binary numbers </param>

/// <param name="convertedNumber"> Decimal numbers </param>

/// <param name="append"></param>

void PrintResult2(string outputFile, int line,

string convertedNumber, bool append)

{

ofstream result;

if (append)

result.open(outputFile, ios\_base::app);

else

result.open(outputFile);

result << "Decimal: " << line << " " << "skew binary: " << convertedNumber << endl;

return;

}

/// <summary>

/// Reading data and performing all tasks

/// </summary>

/// <param name="inputFile"> Data file </param>

/// <param name="outputFile"> Result file </param>

void ReadAndPerformToBinary(string inputFile, string outputFile)

{

int n;// Amount of numbers

DecimalToSkewBinary toBinary[100];

ifstream data(inputFile);

data >> n;

for (int i = 0; i < n; i++)

{

int line;

data.ignore();

data >> line;

//line = toBinary[i].decimal;

string calculated = toBinary->decimalToSkewBinary(line);

bool append = false;

if (i > 0)

append = true;

PrintResult2(outputFile, line, calculated, append);

}

data.close();

return;

}

int main()

{

string inputFile1 = "Data1.txt";

string outputFile1 = "Results1.txt";

string inputFile2 = "Data2.txt";

string outputFile2 = "Results2.txt";

// Duration of operations start point

auto start = high\_resolution\_clock::now();

// Main calculations method

ReadAndPerformToDecimal(inputFile1, outputFile1);

ReadAndPerformToBinary(inputFile2, outputFile2);

// Duration of operations end point

auto stop = high\_resolution\_clock::now();

// Duration

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Time taken by function: "

<< duration.count() << " microseconds" << endl;

return 0;

}

## Pradiniai duomenys ir rezultatai

Data1( skew binary numbers)

Data1.txt:

8

10120

200000000000000000000000000000

10

1000000000000000000000000000000

11

100

11111000001110000101101102000

0

Result1.txt:

Skew binary: 10120 decimal: 44

Skew binary: 200000000000000000000000000000 decimal: 2147483646

Skew binary: 10 decimal: 3

Skew binary: 1000000000000000000000000000000 decimal: 2147483647

Skew binary: 11 decimal: 4

Skew binary: 100 decimal: 7

Skew binary: 11111000001110000101101102000 decimal: 1041110737

Skew binary: 0 decimal: 0

Data2 ( binary numbers)

Data2.txt:

8

44

2147483646

3

2147483647

4

7

1041110737

0

Result2.txt:

Decimal: 44 skew binary: 2202

Decimal: 2147483646 skew binary: 20220200022111212100

Decimal: 3 skew binary: 10

Decimal: 2147483647 skew binary: 20220200022111212101

Decimal: 4 skew binary: 11

Decimal: 7 skew binary: 21

Decimal: 1041110737 skew binary: 10201220001020110021

Decimal: 0 skew binary: 0

# Scala(L2)

## Darbo užduotis

Reikalavimai programai/botui:

1. Panaudoti bent kelis master boto išleidžiamus botų padėjėjų tipus

(pvz.: minos, raketos į priešus, "kamikadzės", rinkikai, masalas ir pan.)

2. Panaudoti bet kurį vieną iš kelio radimo algoritmų (DFS, BFS, A\*, Greedy, Dijkstra).

Šiame Scalatron bot‘e buvo panaudotas BFS kelio radimo algoritmas. Taip pat pritaikyti buvo reference boto išleidžiami padėjėjai: raketos. Implementuoti papildomi pagalbiniai botai: minos, bei rinkikai kurie gali keisti savo tipą, o surinkę tam tikrą kiekį taškų grižta pas main botą. Minos nėra dedamos kol pagrindinio boto energija yra didesnė už 50. Raketos nėra šaudomos kol energija didesnė už 10. Gynybinė raketa nėra šaunama kol energija didesnė už 65. Pagalbininkai nėra kviečiami, kol boto energija didesnė už 20.

## Programos tekstas

import scala.collection.mutable.Queue

import scala.collection.mutable.Stack

import scala.util.control.\_

import util.Random

object ControlFunction

{

val random = new Random()

def forMain(bot: Bot) {

val (directionValue, nearestEnemyMain, nearestEnemySlave, \_) = analyzeView(bot.view)

val dontPutMine = bot.inputAsIntOrElse("dontPutMine", -1)

val dontFireRocket = bot.inputAsIntOrElse("dontFireRocket", -1)

val dontFireDefensiveRocket = bot.inputAsIntOrElse("dontFireDefensiveRocket", -1)

val dontSpawnAssistant = bot.inputAsIntOrElse("dontSpawnAssistant", -1)

var direction = BFS\_SearchingAlgorythm(bot,bot.view)

var temp = XY(0,0)

if(temp == direction){

direction = XY(random.nextInt(3)-1, random.nextInt(3)-1)

var cell = bot.view(direction)

while(cell == 'W' || cell == 'b' || cell == 'p'){

direction = XY(random.nextInt(3)-1, random.nextInt(3)-1)

cell = bot.view(direction)

}

bot.say("RANDOM:" + direction.toString())

}

bot.move(direction)

if(dontPutMine < bot.time && bot.energy > 50){

nearestEnemyMain match {

case None => // no one nearby

case Some(relPos) => // a main bot is nearby

val unitFlat = relPos.signum

val remainder = relPos - unitFlat // we place slave nearer target, so subtract that from overall flat surface

bot.spawn(unitFlat, "mood" -> "Mine", "target" -> remainder)

bot.set("dontPutMine" -> (bot.time + relPos.stepCount + 1))

}

}

if(dontFireRocket < bot.time && bot.energy > 10) { // fire rocket?

nearestEnemyMain match {

case None => // no one nearby

case Some(relPos) => // a main bot is nearby

val unitFlat = relPos.signum

val remainder = relPos - unitFlat // we place slave nearer target

bot.spawn(unitFlat, "mood" -> "Rocket", "target" -> remainder)

bot.set("dontFireRocket" -> (bot.time + relPos.stepCount + 1))

}

}

else

if(dontFireDefensiveRocket < bot.time && bot.energy > 65) { // fire defensive rocket?

nearestEnemySlave match {

case None => // no one nearby

case Some(relPos) => // an enemy slave is nearby

if(relPos.stepCount < 8) {

val unitFlat = relPos.signum

val remainder = relPos - unitFlat // we place slave nearer target

bot.spawn(unitFlat, "mood" -> "DefensiveRocket", "target" -> remainder)

bot.set("dontFireDefensiveRocket" -> (bot.time +

relPos.stepCount + 1))

}

}

}

if(dontSpawnAssistant < bot.time && bot.energy > 20){

bot.spawn(bot.view.center, "mood" -> "Assistant", "target" -> "", "collector" -> 2000)

bot.set("dontSpawnAssistant" -> (bot.time + 10))

}

}

def slaveBots(bot: MiniBot) {

bot.inputOrElse("mood", "Waiting") match {

case "Mine" => reactAsMine(bot)

case "Rocket" => reactAsRocket(bot)

case "Defensive" => reactAsDefensiveRocket(bot)

case "Assistant" => reactAsAssistant(bot)

case "NonActive" => reactAsNonActive(bot)

case s: String => bot.log("unknown mood: " + s)

}

}

def BFS\_SearchingAlgorythm(bot: Bot, view: View) : XY = {

var queue = Queue[XY]()

var visited = Set[XY]()

var path = Map[XY,XY]()

queue.enqueue(XY(0, 0))

while (!queue.isEmpty) {

val next = queue.dequeue()

if (next.length > 15) {

// if no available path found do not move

return XY(0, 0)

}

for (i <- -1 to 1; j <- -1 to 1) {

val xy = XY(i, j) + next

val cell = view(xy)

// if found food backtrack and return direction

if (cell == 'P' || cell == 'B') {

var currentSource = next

if (xy.length < 1.5) {

return xy

}

while (currentSource.length > 1.5) {

val temp = view(currentSource)

if(temp != 'W')

currentSource = path(currentSource)

}

return currentSource

}

if (cell == '\_' && !visited.contains(xy)) {

queue.enqueue(xy)

visited += xy

path += (xy -> next)

}

}

}

// default return - not move

XY(0, 0)

}

def reactAsMine(bot: MiniBot){

bot.view.offsetToNearest('m') match {

case Some(delta: XY) =>

// another master is visible at the given relative position

if(delta.length <= 2) {

// blowing it up

bot.explode(4)

} else

{

// no

}

case None =>

bot.say("Mine")

}

}

def reactAsRocket(bot: MiniBot) {

bot.view.offsetToNearest('m') match {

case Some(delta: XY) =>

// another master is visible at the given relative position

if(delta.length <= 2) {

bot.explode(4)//exploding

} else

{

// no

bot.move(delta.signum)

bot.set("rx" -> delta.x, "ry" -> delta.y)

}

case None =>

// no target visible

val target = bot.inputAsXYOrElse("target", XY.Zero)

// did we arrive at the target?

if(target.isNonZero) {

// keep going

val unitDelta = target.signum // CellPos(-8,6) => CellPos(-1,1)

bot.move(unitDelta)

// compute the remaining delta and encode it into a new 'target' property

val remainder = target - unitDelta // CellPos(-7,5)

bot.set("target" -> remainder)

}

else{

// yes -- but we did not detonate yet, and are not pursuing anything

bot.set("mood" -> "NonActive", "target" -> "")

bot.say("NonActive")

}

}

}

def reactAsDefensiveRocket(bot: MiniBot) {

bot.view.offsetToNearest('s') match {

case Some(delta: XY) =>

// another slave is visible at the given relative position

// moving closer

bot.move(delta.signum)

bot.set("rx" -> delta.x, "ry" -> delta.y)

case None =>

// no target visible

val target = bot.inputAsXYOrElse("target", XY.Zero)

// did we arrive at the target?

if(target.isNonZero) {

// no

val unitDelta = target.signum // e.g. CellPos(-8,6) => CellPos(-1,1)

bot.move(unitDelta)

// compute the remaining delta and encode it into a new 'target' property

val remainder = target - unitDelta // e.g. = CellPos(-7,5)

bot.set("target" -> remainder)

}

else{

// yes -- but we did not annihilate yet, and are not pursuing anything => switch purpose

bot.set("mood" -> "NonActive", "target" -> "")

bot.say("NonActive")

}

}

}

def reactAsAssistant(bot: MiniBot){

val (directionValue, nearestEnemyMaster, \_, master) = analyzeView(bot.view)

val collector = bot.inputAsIntOrElse("collector", 0)

if(bot.energy > collector){

bot.set("mood" -> "NonActive", "target" -> "")

reactAsNonActive(bot)

}

else if(bot.energy > collector/10 && !master.isEmpty){

bot.set("mood" -> "NonActive", "target" -> "")

reactAsNonActive(bot)

}

else if(!nearestEnemyMaster.isEmpty && bot.energy < collector/10){

bot.set("mood" -> "Rocket", "target" -> "")

reactAsRocket(bot)

}

else{

val lastDirection = bot.inputAsIntOrElse("lastDirection", 0)

directionValue(lastDirection) += 10

val bestDirection45 = directionValue.zipWithIndex.maxBy(\_.\_1).\_2

val direction = XY.fromDirection45(bestDirection45)

bot.move(direction)

bot.set("lastDirection" -> bestDirection45)

}

}

def reactAsNonActive(bot: MiniBot){

val (directionValue, nearestEnemyMaster, master) = analyzeViewAsNonActive(bot, bot.view)

val gather = bot.inputAsIntOrElse("gather", 0)

if(bot.energy < gather && master.isEmpty){

bot.set("mood" -> "Assistant", "target" -> "")

reactAsAssistant(bot)

}

val masterDirectionXY = bot.inputAsXYOrElse("master",XY.Zero)

val masterDirection = masterDirectionXY.toDirection45

val masterDirectionLocal = XY.fromDirection45(masterDirection)

directionValue(masterDirection) += 10

if(bot.view(masterDirectionLocal) == 'W' || bot.view(masterDirectionLocal) == 'p' || bot.view(masterDirectionLocal) == 'b'){

directionValue(masterDirection) -= 100

}

val bestDirection45 = directionValue.zipWithIndex.maxBy(\_.\_1).\_2

val direction = XY.fromDirection45(bestDirection45)

bot.move(direction)

}

def analyzeView(view: View) = {

val directionValue = Array.ofDim[Double](8)

var nearestEnemyMaster: Option[XY] = None

var nearestEnemySlave: Option[XY] = None

var master: Option[XY] = None

view.cells.zipWithIndex foreach {case (c, i) =>

val cellRelPos = view.relPosFromIndex(i)

if (cellRelPos.isNonZero){

val stepDistance = cellRelPos.stepCount

val value: Double = c match{

case 'm' => // another master: not dangerous, but an obstacle

nearestEnemyMaster = Some(cellRelPos)

- 100 / stepDistance

case 's' => // another slave: potentially dangerous?

nearestEnemySlave = Some(cellRelPos)

- 100 / stepDistance

case 'S' => // our own slave

-50 / stepDistance

case 'M' => // our own master

master = Some(cellRelPos)

0.0

case 'B' => // great beast: valuable, but runs away

if (stepDistance == 1) 600

else if (stepDistance == 2) 300

else (150 - stepDistance \* 15).max(10)

case 'P' => // great plant: less valuable, but does not run

if (stepDistance == 1) 500

else if (stepDistance == 2) 300

else (150 - stepDistance \* 10).max(10)

case 'b' => // bad beast: dangerous, but only if very close

if (stepDistance < 4) -400 / stepDistance else -50 / stepDistance

case 'p' => // bad plant: bad, but only if I step on it

if (stepDistance < 2) -1000 else 0

case 'W' => // wall: harmless, just don't walk into it

if (stepDistance < 2) -1000 else 0

case \_ =>

0.0

}

val direction45 = cellRelPos.toDirection45

directionValue(direction45) += value

}

}

(directionValue, nearestEnemyMaster, nearestEnemySlave, master)

}

def analyzeViewAsNonActive(bot: MiniBot, view: View) = {

val directionValue = Array.ofDim[Double](8)

var nearestEnemyMaster: Option[XY] = None

var master: Option[XY] = None

view.cells.zipWithIndex foreach {case (c, i) =>

val cellRelPos = view.relPosFromIndex(i)

if (cellRelPos.isNonZero){

val stepDistance = cellRelPos.stepCount

val value: Double = c match{

case 'm' => // another master: not dangerous, but an obstacle

nearestEnemyMaster = Some(cellRelPos)

-100 / stepDistance

case 's' => // another slave: potentially dangerous?

-100 / stepDistance

case 'S' => // our own slave

-50 / stepDistance

case 'M' => // our own master

master = Some(cellRelPos)

1000

case 'B' => // good beast: valuable, but runs away

if (stepDistance == 1) 600

else if (stepDistance == 2) 300

else (150 - stepDistance \* 15).max(10)

case 'P' => // good plant: less valuable, but does not run

if (stepDistance == 1) 500

else if (stepDistance == 2) 300

else (150 - stepDistance \* 10).max(10)

case 'b' => // bad beast: dangerous, but only if very close

if (stepDistance < 4) -400 / stepDistance else -50 / stepDistance

case 'p' => // bad plant: bad, but only if I step on it

if (stepDistance < 2) -1000 else 0

case 'W' => // wall: harmless, just don't walk into it

if (stepDistance < 2) -1000 else 0

case \_ =>

0.0

}

val direction45 = cellRelPos.toDirection45

directionValue(direction45) += value

}

}

(directionValue, nearestEnemyMaster, master)

}

}

// -------------------------------------------------------------------------------------------------

// Framework

// -------------------------------------------------------------------------------------------------

class ControlFunctionFactory {

def create = (input: String) => {

val (opcode, params) = CommandParser(input)

opcode match {

case "React" =>

val bot = new BotImpl(params)

if( bot.generation == 0 ) {

ControlFunction.forMain(bot)

}

else

{

ControlFunction.slaveBots(bot)

}

bot.toString

case \_ => "" // OK

}

}

}

// -------------------------------------------------------------------------------------------------

trait Bot {

// inputs

def inputOrElse(key: String, fallback: String): String

def inputAsIntOrElse(key: String, fallback: Int): Int

def inputAsXYOrElse(keyPrefix: String, fallback: XY): XY

def view: View

def energy: Int

def time: Int

def generation: Int

// outputs

def move(delta: XY) : Bot

def say(text: String) : Bot

def status(text: String) : Bot

def spawn(offset: XY, params: (String,Any)\*) : Bot

def set(params: (String,Any)\*) : Bot

def log(text: String) : Bot

}

trait MiniBot extends Bot {

// inputs

def offsetToMaster: XY

// outputs

def explode(blastRadius: Int) : Bot

}

case class BotImpl(inputParams: Map[String, String]) extends MiniBot {

// input

def inputOrElse(key: String, fallback: String) = inputParams.getOrElse(key, fallback)

def inputAsIntOrElse(key: String, fallback: Int) = inputParams.get(key).map(\_.toInt).getOrElse(fallback)

def inputAsXYOrElse(key: String, fallback: XY) = inputParams.get(key).map(s => XY(s)).getOrElse(fallback)

val view = View(inputParams("view"))

val energy = inputParams("energy").toInt

val time = inputParams("time").toInt

val generation = inputParams("generation").toInt

def offsetToMaster = inputAsXYOrElse("master", XY.Zero)

// output

private var stateParams = Map.empty[String,Any] // holds "Set()" commands

private var commands = "" // holds all other commands

private var debugOutput = "" // holds all "Log()" output

/\*\* Appends a new command to the command string; returns 'this' for fluent API. \*/

private def append(s: String) : Bot = { commands += (if(commands.isEmpty) s else "|" + s); this }

/\*\* Renders commands and stateParams into a control function return string. \*/

override def toString = {

var result = commands

if(!stateParams.isEmpty) {

if(!result.isEmpty) result += "|"

result += stateParams.map(e => e.\_1 + "=" + e.\_2).mkString("Set(",",",")")

}

if(!debugOutput.isEmpty) {

if(!result.isEmpty) result += "|"

result += "Log(text=" + debugOutput + ")"

}

result

}

def log(text: String) = { debugOutput += text + "\n"; this }

def move(direction: XY) = append("Move(direction=" + direction + ")")

def say(text: String) = append("Say(text=" + text + ")")

def status(text: String) = append("Status(text=" + text + ")")

def explode(blastRadius: Int) = append("Explode(size=" + blastRadius + ")")

def spawn(offset: XY, params: (String,Any)\*) = append("Spawn(direction=" + offset +

(if(params.isEmpty) "" else "," + params.map(e => e.\_1 + "=" + e.\_2).mkString(",")) +")")

def set(params: (String,Any)\*) = { stateParams ++= params; this }

def set(keyPrefix: String, xy: XY) = { stateParams ++= List(keyPrefix+"x" -> xy.x, keyPrefix+"y" -> xy.y); this }

}

// -------------------------------------------------------------------------------------------------

/\*\* Utility methods for parsing strings containing a single command of the format

\* "Command(key=value,key=value,...)"

\*/

object CommandParser {

/\*\* "Command(..)" => ("Command", Map( ("key" -> "value"), ("key" -> "value"), ..}) \*/

def apply(command: String): (String, Map[String, String]) = {

/\*\* "key=value" => ("key","value") \*/

def splitParameterIntoKeyValue(param: String): (String, String) = {

val segments = param.split('=')

(segments(0), if(segments.length>=2) segments(1) else "")

}

val segments = command.split('(')

if( segments.length != 2 )

throw new IllegalStateException("invalid command: " + command)

val opcode = segments(0)

val params = segments(1).dropRight(1).split(',')

val keyValuePairs = params.map(splitParameterIntoKeyValue).toMap

(opcode, keyValuePairs)

}

}

// -------------------------------------------------------------------------------------------------

/\*\* Utility class for managing 2D cell coordinates.

\* The coordinate (0,0) corresponds to the top-left corner of the arena on screen.

\* The direction (1,-1) points right and up.

\*/

case class XY(x: Int, y: Int) {

override def toString = x + ":" + y

def isNonZero = x != 0 || y != 0

def isZero = x == 0 && y == 0

def isNonNegative = x >= 0 && y >= 0

def updateX(newX: Int) = XY(newX, y)

def updateY(newY: Int) = XY(x, newY)

def addToX(dx: Int) = XY(x + dx, y)

def addToY(dy: Int) = XY(x, y + dy)

def +(pos: XY) = XY(x + pos.x, y + pos.y)

def -(pos: XY) = XY(x - pos.x, y - pos.y)

def \*(factor: Double) = XY((x \* factor).intValue, (y \* factor).intValue)

def distanceTo(pos: XY): Double = (this - pos).length // Phythagorean

def length: Double = math.sqrt(x \* x + y \* y) // Phythagorean

def stepsTo(pos: XY): Int = (this - pos).stepCount // steps to reach pos: max delta X or Y

def stepCount: Int = x.abs.max(y.abs) // steps from (0,0) to get here: max X or Y

def signum = XY(x.signum, y.signum)

def negate = XY(-x, -y)

def negateX = XY(-x, y)

def negateY = XY(x, -y)

/\*\* Returns the direction index with 'Right' being index 0, then clockwise in 45 degree steps. \*/

def toDirection45: Int = {

val unit = signum

unit.x match {

case -1 =>

unit.y match {

case -1 =>

if(x < y \* 3) Direction45.Left

else if(y < x \* 3) Direction45.Up

else Direction45.UpLeft

case 0 =>

Direction45.Left

case 1 =>

if(-x > y \* 3) Direction45.Left

else if(y > -x \* 3) Direction45.Down

else Direction45.LeftDown

}

case 0 =>

unit.y match {

case 1 => Direction45.Down

case 0 => throw new IllegalArgumentException("cannot compute direction index for (0,0)")

case -1 => Direction45.Up

}

case 1 =>

unit.y match {

case -1 =>

if(x > -y \* 3) Direction45.Right

else if(-y > x \* 3) Direction45.Up

else Direction45.RightUp

case 0 =>

Direction45.Right

case 1 =>

if(x > y \* 3) Direction45.Right

else if(y > x \* 3) Direction45.Down

else Direction45.DownRight

}

}

}

def rotateCounterClockwise45 = XY.fromDirection45((signum.toDirection45 + 1) % 8)

def rotateCounterClockwise90 = XY.fromDirection45((signum.toDirection45 + 2) % 8)

def rotateClockwise45 = XY.fromDirection45((signum.toDirection45 + 7) % 8)

def rotateClockwise90 = XY.fromDirection45((signum.toDirection45 + 6) % 8)

def wrap(boardSize: XY) = {

val fixedX = if(x < 0) boardSize.x + x else if(x >= boardSize.x) x - boardSize.x else

x

val fixedY = if(y < 0) boardSize.y + y else if(y >= boardSize.y) y - boardSize.y else

y

if(fixedX != x || fixedY != y) XY(fixedX, fixedY) else this

}

}

object XY {

/\*\* Parse an XY value from XY.toString format, e.g. "2:3". \*/

def apply(s: String) : XY = { val a = s.split(':'); XY(a(0).toInt,a(1).toInt) }

val Zero = XY(0, 0)

val One = XY(1, 1)

val Right = XY( 1, 0)

val RightUp = XY( 1, -1)

val Up = XY( 0, -1)

val UpLeft = XY(-1, -1)

val Left = XY(-1, 0)

val LeftDown = XY(-1, 1)

val Down = XY( 0, 1)

val DownRight = XY( 1, 1)

def fromDirection45(index: Int): XY = index match {

case Direction45.Right => Right

case Direction45.RightUp => RightUp

case Direction45.Up => Up

case Direction45.UpLeft => UpLeft

case Direction45.Left => Left

case Direction45.LeftDown => LeftDown

case Direction45.Down => Down

case Direction45.DownRight => DownRight

}

def fromDirection90(index: Int): XY = index match {

case Direction90.Right => Right

case Direction90.Up => Up

case Direction90.Left => Left

case Direction90.Down => Down

}

def apply(array: Array[Int]): XY = XY(array(0), array(1))

}

object Direction45 {

val Right = 0

val RightUp = 1

val Up = 2

val UpLeft = 3

val Left = 4

val LeftDown = 5

val Down = 6

val DownRight = 7

}

object Direction90 {

val Right = 0

val Up = 1

val Left = 2

val Down = 3

}

// -------------------------------------------------------------------------------------------------

case class View(cells: String) {

val size = math.sqrt(cells.length).toInt

val center = XY(size / 2, size / 2)

def apply(relPos: XY) = cellAtRelPos(relPos)

def indexFromAbsPos(absPos: XY) = absPos.x + absPos.y \* size

def absPosFromIndex(index: Int) = XY(index % size, index / size)

def absPosFromRelPos(relPos: XY) = relPos + center

def cellAtAbsPos(absPos: XY) = cells.charAt(indexFromAbsPos(absPos))

def indexFromRelPos(relPos: XY) = indexFromAbsPos(absPosFromRelPos(relPos))

def relPosFromAbsPos(absPos: XY) = absPos - center

def relPosFromIndex(index: Int) = relPosFromAbsPos(absPosFromIndex(index))

def cellAtRelPos(relPos: XY) = cells.charAt(indexFromRelPos(relPos))

def offsetToNearest(c: Char) = {

val matchingXY = cells.view.zipWithIndex.filter(\_.\_1 == c)

if( matchingXY.isEmpty )

None

else {

val nearest = matchingXY.map(p => relPosFromIndex(p.\_2)).minBy(\_.length)

Some(nearest)

}

}

}

## Rezultatai

Suprojektuotas botas (Bot1) buvo paliktas kovoti su reference botu. Naujasis botas surenka daugiau taškų. Taip yra todėl, kad naujasis botas turi pagalbininkus renkančius maistą, taip taškai pradeda kilti eksponentiškai – tai ir yra didžiausias pranašumas prieš reference botą, kadangi jis tik vienas ir vieninteliai pagalbininkai kuriuos jis naudoja yra tik raketos, kurių naujasis botas nebijo, turi savas tiek atakuojančias tiek gynybines.

Paveikslėlis, kuriame yra diagrama

Automatiškai sugeneruotas aprašymas