**DOKUZ EYLUL UNIVERSITY**

**ENGINEERING FACULTY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CME1252 PROJECT BASED LEARNING – II**

**PROGRESS REPORT**

**PROJECT – II**

**Gravity Game**

**by**

**AHMED PATEL**

**BARIS OLÇAY**

**MUHARREM PEHLEVAN**

**YASAMIN VALISHARIATPANAHI**

**Lecturers**

**Tanzer ONURGİL**

**Dr. Atakan KUBİLAY**

**14.04.2023**

# CHAPTER ONE

PROGRESS DESCRIPTION

# CHAPTER TWO

TASK SUMMARY

## Completed Tasks

YASAMIN VALISHARIATPANAHI:

By controlling the position of Player and Robots, when their positions are the same the game will be over by prompt a message on the console that the game is over.

AHMED PATEL:

Enabled robots to destroy treasures, added a start game screen and testing robots and boulders which has been created by my teammates.

BARIŞ OLÇAY:

I did the rock pushing part and the rock falling part. I helped my friends in some parts where we made mistakes.

MUHARREM PEHLEVAN:

I did the main game array, the input queue, the movements of the newly added robots and the destruction of the robots.

## Incomplete Tasks: Reasons and Explanations

Every tasks has been completed.

## Additional Improvements

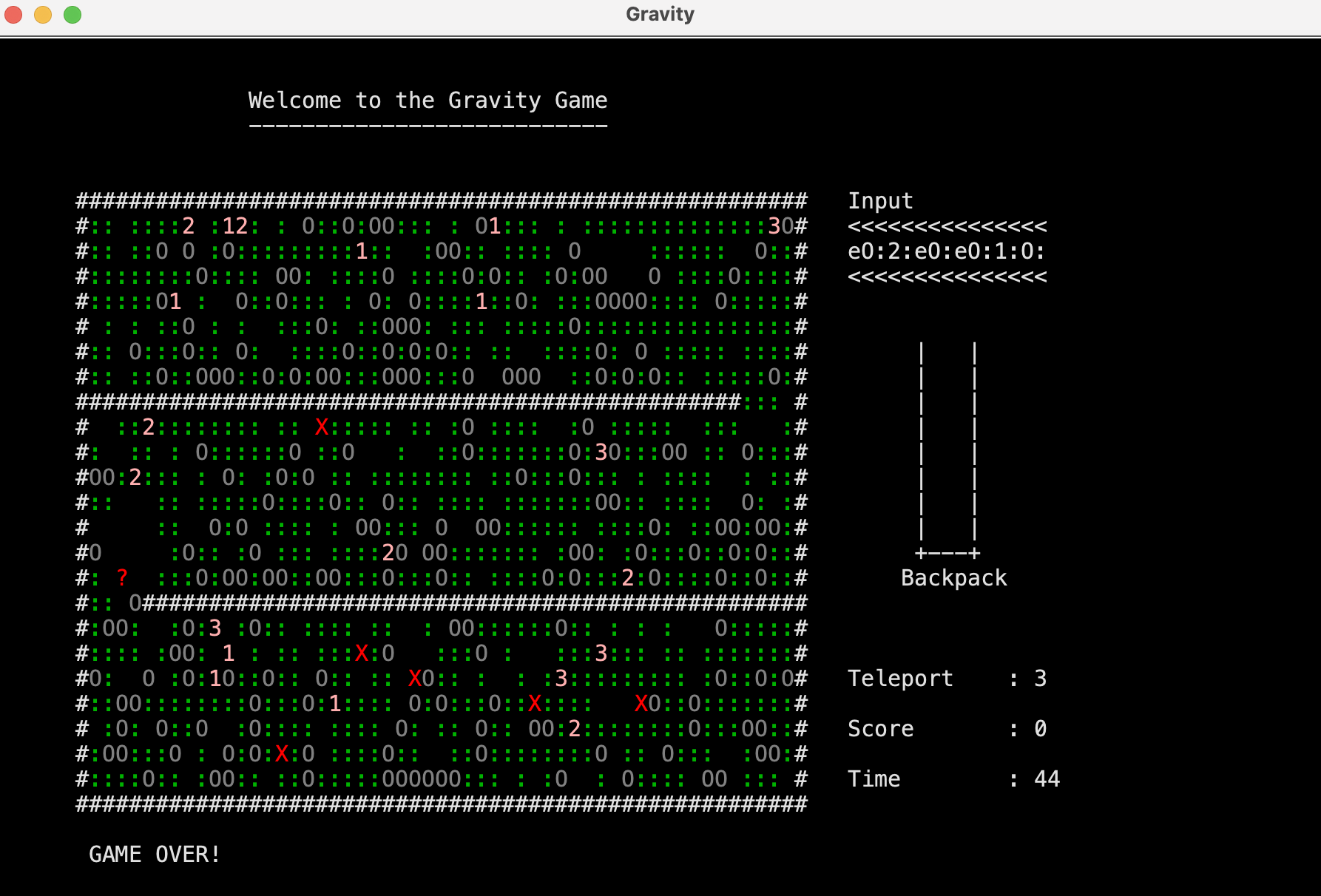
There has been no additional improvement added to this project other than gibing colors to the elements of the game.

# CHAPTER THREE

EXPLANATION of algorıthms

## Screenshots

**

**

## Functions

Main class:

Public static void main(String[] args) 🡪 calling the Game class for running the project.

Game class:

public Game() 🡪 Constructor class, which will do the initialization of walls, earth elements, 200 random boulders, 30 random treasures(1, 2, 3) and 200 random free spaces on the game field. Calling the Player, 7 Robots and Status classes. Input that will hold the elements that are inserted into the maze area from an input queue which is always full of elements and will show the next element that is going to be inserted into the field. Then by showing the map on the console the game is going to be started. When the Robots reach the player that means their positions are the same and the game will be over. When the boulder fall from the gravity if it is on the wall it will stay on it and if it falls on the robot, the robot will be removed and the player will gain 900 points.

public int getTime() 🡪 getter of time when we need to use it in other classes.

public void setTime(int time) 🡪 setter of time when we want to change the value of the time.

Player class:

public Player(Console console, char[][] map) 🡪 standard code for moue and keyboard which in this project the game can’t be played with mouse. At the beginning of the game initializing the backpack, 3 teleport and 0 score. Random Player’s position on the field when the place on the field is empty.

public int getX() 🡪 getter for accessing the value of Player’s X- axis position on the field

public int getY() 🡪 getter for accessing the value of Player’s Y-axis position on the field

public Boolean getAlive() 🡪 getter for player, understanding if he/she is still alive

public Stack getBackpack() 🡪 getter for accessing the value of Player’s backpack

public int getTeleport() 🡪 getter for accessing the value of Player’s teleportation

public int getScore() 🡪 getter for accessing the value of Player’s score

public void setX(int x) 🡪 setting the value of Player’s X-axis position

public void setY(int y) 🡪 setting the value of Player’s Y-axis position

public void setBackpack(Stack backpack) 🡪 setting the value of Player’s backpack

public void setTeleport(int teleport) 🡪 setting the value of Player’s teleportation

public void setScore(int score) 🡪 setting the value of Player’s score

public void playerMovement() 🡪 when keyboard is pressed if will control the movement of the Player up, down, left and right and it make sure that when its wall and boulder the player can’t go through it. In addition to that if there is an empty space next to a boulder from left and right the player can push the boulder. It will also controls that the position is in the wall and not outside of it. When the space on the keyboard is pressed if the player has teleportation it will teleport the Player’s position to a random location that is in the range and not on the walls or on the other elements it can be on an empty spaces in the field.

Public void collect(int y, int x) 🡪 with the help of this method it will give the treasures points and when the player take them it will be added to the backpack. Also when the backpack is full the element on the top will be removed and the new element can be inserted. If the two elements in the backpack are identical numbers then they turn to score and teleport rights.

Robots class:

public Robots(int num, char[][] map) 🡪 we initialize the arguments with the attributes. Random Robots position on the field when the place on the field is empty. Controlling when they get destroyed by the boulder, they will be removed from the field.

public int getX() 🡪 getter for accessing the value of Robots X- axis position on the field

public int getY() 🡪 getter for accessing the value of Robots Y-axis position on the field

public void setX(int x) 🡪 setting the value of Robots X-axis position

public void setY(int y) 🡪 setting the value of Robots Y-axis position

public void robotMovement() 🡪 it checks the 4 different directions (up, down, left, right) because the robots don’t move diagonally. The movements are random when it’s going to move it controls the next position if it’s empty or the player is there it moves other than that it cannot move. Also it controls if we have robots in the field or not if we have they move when they generate after they removed.

Status class:

public Status(Console console, Player player, Player backpack, Player teleport, Player score, Game time) 🡪 we initialize the variables with our attributes

public showStatus() 🡪 we show the input, backpack, teleport, score and time values on the console.

public Player getBackpack() 🡪 it gets the backpack value of the Player

public Player getTeleport() 🡪 it gets the teleport value of the Player

public Player getScore() 🡪 it gets the score value of the Player

public Game getTime() 🡪 it gets the time value of the Game

Stack class:

public Stack(int capacity) 🡪 initialize the capacity of the stack at the beginning

public Boolean isEmpty() 🡪 checks if the stack is full or not

public Boolean isFull() 🡪 checks if the stack is empty or not

public int size() 🡪 set the size of the stack

public void push(Object data) 🡪 push, adds object into the stack

public Object pop() 🡪 pop, removes object out of the stack

public Object peek() 🡪 give the top data of the stack

CircularQueue class:

public CircularQueue(int capacity) 🡪 initialize the capacity of the queue at the beginning

public Boolean isEmpty() 🡪 checks if the queue is full or not

public Boolean isFull() 🡪 checks if the queue is empty or not

public int size() 🡪 set the size of the queue

public void enqueue(Object data) 🡪 enqueue, adds object into the queue

public Object dequeue() 🡪 dequeue, removes object out of the queue

public Object peek() 🡪 give the top data of the queue

## Algorithms and Solution Strategies

YASAMIN VALISHARIATPANAHI:

For the Robot Class the robotMovement() method, because the robots cant move diagonally and their movements are random so it randomly chooses a direction (up, down, left, right) then moves if that location is empty. Also by controlling which sides of the robot is free for moving it resulted the program to not crash.

AHMED PATEL:

For the Robot movement condition I’ve tested Yasmin's robotMovement() and enabled the robot to destroy treasures by allowing robots to move through the treasures destroying the collect ability and removing from the map

BARIŞ OLÇAY:

I used similar algorithms for rock pushing and rockfall parts. I checked where the rock was going and made it move accordingly. I used the if structure and created a situation to act or not according to the appropriate conditions.

MUHARREM PEHLEVAN:

I combined all game functions in a single array for stable printing and synchronization of the game, I used the queue class and random method for the input queue, and I recorded the ids of the old and newly created robots to perform the movements and destruction of the new robots.

# CHAPTER FOUR

PROBLEMS ENCOUNTERED

YASAMIN VALISHARIATPANAHI:

At first the robots don’t go over the Player with by controlling the movements of the robots, the problem solved. Other than that, everything went as planed and there has been no other problems that I have encountered.

AHMED PATEL:

During testing I’ve encountered the boulders just passing through the robots rather than killing them, Later this problem has been solved by Muharrem.

BARIŞ OLÇAY:

As the rocks fell, not a single unit was going directly all the way at once. I added break inside the loop and made it go one by one. I solved the problem.

MUHARREM PEHLEVAN:

While the game screen was rendering, the screen was not working properly because there were multiple prints to the screen, I resolved it by combining all the prints in one place. Input queue was inverting every game second, I solved it.

# CHAPTER FIVE

conclusıon

In the group projects we keep learning how to work with each other to achieve our goal, which is coding of the project in this case the Gravity game. In addition to that we manage to understand object oriented programming much better in using our knowledge by doing projects and gain experiments. Getting to know Enigma library and how to use it.

REFERENCES

* [https://stackoverflow.com](https://stackoverflow.com/)
* <https://www.java.com/tr/>
* [https://dev.java](https://dev.java/)
* [https://www.eclipse.org](https://www.eclipse.org/)
* [https://pitch.com](https://app.pitch.com/)​
* <https://learn.microsoft.com/en-us/azure/developer/java/learning-resources/fundamentals>​

**AppendIx A**

Poster/Web page of the Project

In this project we didn’t make a poster, we made a video for our project.

**AppendIx B**

Code of the Project

public class Main {

public static void main(String[] args) throws Exception {

Game myGame = new Game();

}

}

public class Stack {

// Attributes

private int top;

private Object[] elements;

// Constructor

public Stack(int capacity) {

elements = new Object[capacity];

top = -1;

}

// Checking if our stack is Empty or not

public boolean isEmpty() {

return (top == -1);

}

// Checking if our stack is Full or not

public boolean isFull() {

return (top + 1 == elements.length);

}

// Set the Size of the stack

public int size() {

return top + 1;

}

// Push in to the stack

public void push(Object data) {

if(isFull())

System.***out***.println("Stack Overflow!");

else {

top++;

elements[top] = data;

}

}

// Pop object out of the stack

public Object pop() {

if(isEmpty()) {

System.***out***.println("Stack is Empty!");

return null;

}

else {

Object retData = elements[top];

top--;

return retData;

}

}

// Peek the top data in the stack

public Object peek() {

if(isEmpty()) {

System.***out***.println("Stack is Empty!");

return null;

}

else

return elements[top];

}

}

public class CircularQueue {

// Attributes

private int rear, front;

private Object[] elements;

// Constructor

public CircularQueue(int capacity) {

elements = new Object[capacity];

rear = -1;

front = 0;

}

// Checking if our queue is Empty or not

public boolean isEmpty() {

return elements[front] == null;

}

// Checking if our queue is Full or not

public boolean isFull() {

return (front == (rear + 1) % elements.length && elements[front] != null && elements[rear] != null);

}

// Set the Size of the queue

int size() {

if(elements[front] == null) {

return 0;

}

else {

if(rear >= front)

return rear - front + 1;

else

return elements.length - (front - rear) + 1;

}

}

// Push in to the queue

public void enqueue(Object data) {

if(isFull())

System.***out***.println("Queue Overflow!");

else {

rear = (rear + 1) % elements.length;

elements[rear] = data;

}

}

// dequeue object out of the stack

public Object dequeue() {

if(isEmpty()) {

System.***out***.println("Queue is Empty!");

return null;

}

else {

Object retData = elements[front];

elements[front] = null;

front = (front + 1) % elements.length;

return retData;

}

}

// Peek the top data in the stack

public Object peek() {

if(isEmpty()) {

System.***out***.println("Queue is Empty!");

return null;

}

else

return elements[front];

}

}

import enigma.console.Console;

import enigma.event.TextMouseEvent;

import enigma.event.TextMouseListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.util.Random;

public class Player {

// attributes

private Console console;

public TextMouseListener tmlis;

public KeyListener klis;

// ------ Standard variables for mouse and keyboard ------

public int mousepr; // mouse pressed?

public int mousex, mousey; // mouse text coordination

public int keypr; // key pressed?

public int rkey; // key (for press/release)

// coordination

private int x;

private int y;

private boolean Alive = true;

public static Stack *backpack*;

private int teleport;

private static int *score*;

private char[][] map;

private Random rnd = new Random();

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

// constructor

public Player(Console console, char[][] map) {

// ------ Standard code for mouse and keyboard ------ Do not change

tmlis=new TextMouseListener() {

public void mouseClicked(TextMouseEvent arg0) {}

public void mousePressed(TextMouseEvent arg0) {

if(mousepr==0) {

mousepr=1;

mousex=arg0.getX();

mousey=arg0.getY();

}

}

public void mouseReleased(TextMouseEvent arg0) {}

};

console.getTextWindow().addTextMouseListener(tmlis);

klis=new KeyListener() {

public void keyTyped(KeyEvent e) {}

public void keyPressed(KeyEvent e) {

if(keypr==0) {

keypr=1;

rkey=e.getKeyCode();

}

}

public void keyReleased(KeyEvent e) {}

};

console.getTextWindow().addKeyListener(klis);

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

this.console = console;

this.map = map;

*backpack* = new Stack(9);

teleport = 3;

*score* = 0;

// Player's position

while(true) {

x = rnd.nextInt(54) + 1;

y = rnd.nextInt(24) + 1;

if(map[y][x] == ' ') {

map[y][x] = 'P';

break;

}

}

}

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

// getters

public int getX() {

return x;

}

public int getY() {

return y;

}

public boolean getAlive() {

return Alive;

}

public Stack getBackpack() {

return *backpack*;

}

public int getTeleport() {

return teleport;

}

public int getScore() {

return *score*;

}

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

// setters

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

public void setBackpack(Stack backpack) {

Player.*backpack* = backpack;

}

public void setTeleport(int teleport) {

this.teleport = teleport;

}

public void setScore(int score) {

Player.*score* = score;

}

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

public void playerMovement() throws InterruptedException {

if (keypr == 1) { // if keyboard button pressed

int newx = x, newy = y;

if (rkey == KeyEvent.***VK\_LEFT*** && x > 1 && map[y][x - 1] != '#' && map[y][x - 1] != 'O') {

newx--;

collect(y, x - 1);

map[y][x - 1] = 'P';

map[y][x] = ' ';

}

if (rkey == KeyEvent.***VK\_RIGHT*** && x < 58 && map[y][x + 1] != '#' && map[y][x + 1] != 'O') {

newx++;

collect(y, x + 1);

map[y][x + 1] = 'P';

map[y][x] = ' ';

}

if (rkey == KeyEvent.***VK\_UP*** && y > 1 && map[y - 1][x] != '#' && map[y - 1][x] != 'O') {

newy--;

collect(y - 1, x);

map[y - 1][x] = 'P';

map[y][x] = ' ';

}

if (rkey == KeyEvent.***VK\_DOWN*** && y < 28 && map[y + 1][x] != '#' && map[y + 1][x] != 'O') {

if (map[y - 1][x] == 'O')

{

Alive = false;

}

newy++;

collect(y + 1, x);

map[y + 1][x] = 'P';

map[y][x] = ' ';

}

// push boulder

if (rkey == KeyEvent.***VK\_LEFT*** && x > 1 && map[y][x - 1] == 'O' && map[y][x - 2] == ' ') {

map[y][x - 2] = 'O';

newx--;

map[y][x - 1] = 'P';

map[y][x] = ' ';

}

if (rkey == KeyEvent.***VK\_RIGHT*** && x < 58 && map[y][x + 1] == 'O' && map[y][x + 2] == ' ') {

map[y][x + 2] = 'O';

newx++;

map[y][x + 1] = 'P';

map[y][x] = ' ';

}

// check if the new position is within bounds and not a wall

if (newx >= 1 && newx <= 54 && newy >= 1 && newy <= 23 && map[newy][newx] != '#') {

// Move player to new position

x = newx;

y = newy;

}

if (rkey == KeyEvent.***VK\_SPACE***) {

if (teleport > 0) { // check if player has any teleport left

teleport--;

int prevX = x;

int prevY = y;

while (true) {

x = rnd.nextInt(54) + 1;

y = rnd.nextInt(24) + 1;

if (map[y][x] == ' ' || map[y][x] == ':') {

map[y][x] = 'P';

break;

}

}

map[prevY][prevX] = ' ';

}

}

keypr = 0; // last action

}

}

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

public void collect(int y, int x) {

if (map[y][x] == '1') {

if (*backpack*.isFull())

*backpack*.pop();

if (!*backpack*.isEmpty()) {

if ((int) *backpack*.peek() == 1) {

*backpack*.pop();

//teleport++;

*score* += 10;

} else

*backpack*.push(1);

} else

*backpack*.push(1);

map[y][x] = ' ';

} else if (map[y][x] == '2') {

if (*backpack*.isFull())

*backpack*.pop();

if (!*backpack*.isEmpty()) {

if ((int) *backpack*.peek() == 2) {

*backpack*.pop();

*score* += 40;

//teleport++;

} else

*backpack*.push(2);

} else

*backpack*.push(2);

map[y][x] = ' ';

} else if (map[y][x] == '3') {

if (*backpack*.isFull())

*backpack*.pop();

if (!*backpack*.isEmpty()) {

if ((int) *backpack*.peek() == 3) {

*backpack*.pop();

teleport++;

*score* += 90;

} else

*backpack*.push(3);

} else

*backpack*.push(3);

//teleport++;

map[y][x] = ' ';

}

}

}

import enigma.console.Console;

import java.util.Random;

public class Robots {

// attributes

private Console console;

// coordination

private int x;

private int y;

private char[][] map;

private Random rnd = new Random();

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

public Robots(int num, char[][] map) {

this.map = map;

// Robot's position

while(num == 1) {

x = rnd.nextInt(54) + 1;

y = rnd.nextInt(24) + 1;

if(map[y][x] == ' ') {

map[y][x] = 'X';

break;

}

}

if (num == 2){

map[getY()][getX()]= ' ';

}

}

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

// getters

public int getX() {

return x;

}

public int getY() {

return y;

}

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

// setters

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

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

public void robotMovement() throws InterruptedException {

if(map[y][x] == 'X'){

int counter = 0; // count the number of invalid moves

while(counter < 4) { // try 4 times to move in different directions

// Generate a random direction

int direction = rnd.nextInt(4);

// Calculate the new position based on the chosen direction

int newX = x;

int newY = y;

if(direction == 0) { // Move up

newY = y - 1;

} else if(direction == 1) { // Move right

newX = x + 1;

} else if(direction == 2) { // Move down

newY = y + 1;

} else { // Move left

newX = x - 1;

}

// Check if the new position is valid and move the robot if it is

if(newX >= 0 && newX < map[0].length && newY >= 0 && newY < map.length && map[newY][newX] == ' ' || map[newY][newX] == 'P' || map[newY][newX] == '1'||map[newY][newX] == '2'||map[newY][newX] == '3' ) {

map[y][x] = ' ';

x = newX;

y = newY;

map[y][x] = 'X';

return; // exit the robotMovement() method

} else {

counter++;

}

}

// If all moves are invalid, do nothing so the program will continued and wont stop working

}

}

}

import enigma.console.Console;

public class Status {

// attributes

private Console console;

public Player backpack = this.getBackpack();

private Player teleport;

private Player score;

private Game time;

private static Stack *temp*;

public Status(Console console, Player backpack, Player teleport, Player score, Game time) {

this.console = console;

this.backpack = backpack;

this.teleport = teleport;

this.score = score;

this.time = time;

}

public void showStatus() {

*temp* = new Stack(8);

console.getTextWindow().setCursorPosition(64, 6);

console.getTextWindow().output("Input");

console.getTextWindow().setCursorPosition(64, 7);

console.getTextWindow().output("<<<<<<<<<<<<<<<");

console.getTextWindow().setCursorPosition(64, 8);

// input in game class

console.getTextWindow().setCursorPosition(64, 9);

console.getTextWindow().output("<<<<<<<<<<<<<<<");

// backpack print part

int y = 12;

for(int i = 0; i < 8; i++) {

console.getTextWindow().setCursorPosition(64, y);

if(!Player.*backpack*.isEmpty() && y+Player.*backpack*.size()>=20) {

console.getTextWindow().output(" | " + Player.*backpack*.peek()+ " |");

*temp*.push(Player.*backpack*.pop());

} else

console.getTextWindow().output(" | " + "" + " |");

y++;

}

while(!*temp*.isEmpty())

Player.*backpack*.push(*temp*.pop());

console.getTextWindow().setCursorPosition(64, 20);

console.getTextWindow().output(" +---+");

console.getTextWindow().setCursorPosition(64, 21);

console.getTextWindow().output(" Backpack");

console.getTextWindow().setCursorPosition(64, 25);

console.getTextWindow().output("Teleport : " + teleport.getTeleport());

console.getTextWindow().setCursorPosition(64, 27);

console.getTextWindow().output("Score : " + score.getScore());

console.getTextWindow().setCursorPosition(64, 29);

console.getTextWindow().output("Time : " + (int)time.getTime());

}

// getters

public Player getBackpack() {

return backpack;

}

public Player getTeleport() {

return teleport;

}

public Player getScore() {

return score;

}

public Game getTime() {

return time;

}

}

import enigma.console.TextAttributes;

import enigma.core.Enigma;

import java.awt.Color;

import java.util.Random;

public class Game {

// Attributes

public static char[][] *map* = new char[25][55];

private Player player;

private int time;

private Status status;

private CircularQueue input = new CircularQueue(150);

private Random rnd = new Random();

public enigma.console.Console cn = Enigma.*getConsole*("Gravity", 100, 40, 15);

public enigma.console.TextWindow cnt = cn.getTextWindow();

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

public Game() throws Exception { // --- Constructor

//Game

cn.getTextWindow().setCursorPosition(7, 2);

cn.getTextWindow().output(" Welcome to the Gravity Game");

cn.getTextWindow().setCursorPosition(7, 3);

cn.getTextWindow().output(" ---------------------------");

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

cn.getTextWindow().setCursorPosition(6, 6);

// initialization

for(int y = 0; y < 25; y++) { // walls

for(int x = 0; x < 55; x++) {

*map*[y][x]=' ';

if(x == 0 || x == 54 || y == 0 || y == 24) {

*map*[y][x] = '#';

}

if(y == 8 && x < 50) {

*map*[y][x] ='#';

}

if(y == 16 && x > 4) {

*map*[y][x] ='#';

}

}

}

for(int y = 1; y < 24; y++) { // earth

for(int x = 1; x < 54; x++) {

if(*map*[y][x] == ' ') {

*map*[y][x] = ':';

}

}

}

for (int i = 0; i < 200; i++) { // boulder

while (true) {

int x = rnd.nextInt(54) + 1;

int y = rnd.nextInt(24) + 1;

if (*map*[y][x] == ':') {

*map*[y][x] = 'O';

break;

}

}

}

for(int i = 0; i < 30; i++) { // 1,2,3

while(true) {

int x = rnd.nextInt(54) + 1;

int y = rnd.nextInt(24) + 1;

int e = rnd.nextInt(3) + 1;

if(*map*[y][x] == ':') {

*map*[y][x] = (char)(e + 48);

break;

}

}

}

for (int i = 0; i < 200; i++) { // empty

while (true) {

int x = rnd.nextInt(54) + 1;

int y = rnd.nextInt(24) + 1;

if (*map*[y][x] == ':') {

cnt.output(x + 6, y + 6, ' ');

*map*[y][x] = ' ';

break;

}

}

}

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

player = new Player(cn, *map*); // player

int numberofrobots =7;

Robots[] robots = new Robots[100];

int[][] robots1 = new int[100][100];

for(int i = 0; i < numberofrobots; i++) { // robots

robots[i] = new Robots(1, *map*);

robots1[i][0] = i;

robots1[i][1]= robots[i].getX();

robots1[i][2]= robots[i].getY();

}

status = new Status(cn, player, player, player, this); // status

// input

for (int i = 0; i < 15; i++) {

int inp = rnd.nextInt(40) + 1;

if (inp <= 6) {

input.enqueue('1');

}

else if (inp > 6 && inp <= 11) {

input.enqueue('2');

}

else if (inp > 11 && inp <= 15) {

input.enqueue('3');

}

else if (inp > 15 && inp <= 16) {

input.enqueue('X');

}

else if (inp > 16 && inp <= 26) {

input.enqueue('O');

}

else if (inp > 26 && inp <= 35) {

input.enqueue(':');

}

else if (inp > 35 && inp <= 40) {

input.enqueue('e');

}

}

Boolean gameBegin = true;

Boolean isAlive = true;

time = 0;

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

while(isAlive) {

//---INPUT---

if(time % 12 == 0)

{

boolean isdone = false;

boolean generate = false;

boolean delete = false;

while(isdone == false)

{

int x = rnd.nextInt(54) + 1;

int y = rnd.nextInt(24) + 1;

if((char)input.peek() != ':' || (char)input.peek() != ' ' && (char)input.peek() != 'O')

{

while(*map*[y][x] == ' ' || *map*[y][x] == ':')

{

if((char)input.peek()=='e'){

input.dequeue();

*map*[y][x] = ' ';

generate = true;

break;

}

else if((char)input.peek()!='X'){

*map*[y][x] = (char)input.dequeue();

generate = true;

break;

} else if ((char)input.peek()=='X') {

numberofrobots++;

robots[numberofrobots] = new Robots(1, *map*);

generate = true;

break;

}

}

}

else if((char)input.peek() == 'O')

{

while(*map*[y][x] == ' ' || *map*[y][x] == ':')

{

*map*[y][x] = (char)input.dequeue();

generate = true;

delete = true;

break;

}

if(delete == true)

{

x = rnd.nextInt(54) + 1;

y = rnd.nextInt(24) + 1;

while(*map*[y][x] == 'O')

{

*map*[y][x] = ':';

break;

}

delete = false;

}

}

else if((char)input.peek() == ':')

{

while(*map*[y][x] == ' ')

{

*map*[y][x] = (char)input.dequeue();

generate = true;

break;

}

x = rnd.nextInt(54) + 1;

y = rnd.nextInt(24) + 1;

}

else if((char)input.peek() == ' ')

{

while(*map*[y][x] == ':')

{

*map*[y][x] = (char)input.dequeue();

generate = true;

break;

}

}

if(generate == true)

{

int inp = rnd.nextInt(40) + 1;

if (inp <= 6) {

input.enqueue('1');

}

else if (inp > 6 && inp <= 11) {

input.enqueue('2');

}

else if (inp > 11 && inp <= 15) {

input.enqueue('3');

}

else if (inp > 15 && inp <= 16) {

input.enqueue('X');

}

else if (inp > 16 && inp <= 26) {

input.enqueue('O');

}

else if (inp > 26 && inp <= 35) {

input.enqueue(':');

}

else if (inp > 35 && inp <= 40) {

input.enqueue('e');

}

}

break;

}

cnt.setCursorPosition(64, 8);

for(int i = 0; i<15;i++)

{

System.***out***.print(input.peek());

input.enqueue(input.dequeue());

}

}

//---FALL---

for (int i = 0; i < 55; i++)

{

for (int j = 0; j < 25; j++)

{

if(*map*[j][i] == 'O' && *map*[j+1][i] != '#' && *map*[j+1][i] != ':')

{

if(*map*[j+1][i] == ' ' || *map*[j+1][i] == '1' || *map*[j+1][i] == '2' || *map*[j+1][i] == '3')

{

*map*[j+1][i] = 'O';

*map*[j][i] = ' ';

break;

}

else if(*map*[j+1][i] == 'O')

{

int a = rnd.nextInt(2);

if(a == 1)

{

if(*map*[j+1][i+1] == ' ' || *map*[j+1][i+1] == '1' || *map*[j+1][i+1] == '2' || *map*[j+1][i+1] == '3')

{

*map*[j][i] = ' ';

*map*[j+1][i+1] = 'O';

break;

}

}

if(a == 0)

{

if(*map*[j+1][i-1] == ' ' || *map*[j+1][i-1] == '1' || *map*[j+1][i-1] == '2' || *map*[j+1][i-1] == '3')

{

*map*[j][i] = ' ';

*map*[j+1][i-1] = 'O';

break;

}

}

}

else if(*map*[j+1][i] == 'X')

{

*map*[j+1][i] = 'O';

*map*[j][i] = ' ';

player.setScore(player.getScore()+ 900);

for(int g =0; g< numberofrobots;g++){

if(robots1[g][1] == i&&robots1[g][2]==j+1){

robots[g]= new Robots(2,*map*);

cnt.setCursorPosition(60, 60);

}

}

break;

}

}

}

}

if(player.getAlive()== false)

{

isAlive = false;

cnt.setCursorPosition(7, 32);

System.***out***.print("GAME OVER!");

}

status.showStatus();

player.playerMovement();

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

if(robots[i]!=null){

robots[i].robotMovement();

}

}

// map

for(int i = 0; i < 25; i++){

for(int j = 0; j < 55; j++){

cnt.setCursorPosition(j+6, i+6);

char c = *map*[i][j];

if(c == ':') { // earth

cnt.output(c, new TextAttributes(Color.***GREEN***.darker()));

} else if(c == 'O') { // boulder

cnt.output(c, new TextAttributes(Color.***GRAY***));

} else if(c >= '1' && c <= '3') { // 1, 2, 3

cnt.output(c, new TextAttributes(Color.***PINK***));

}else if(c == 'P') { // player

cnt.output(c, new TextAttributes(Color.***YELLOW***));

}else if(c == 'X') { // robot

cnt.output(c, new TextAttributes(Color.***RED***));

} else {

System.***out***.print(c);

}

}

}

//Press Enter to start the game

if(gameBegin) {

cn.getTextWindow().setCursorPosition(23, 4);

cnt.output("PRESS ENTER TO START");

System.***in***.read();

cn.getTextWindow().setCursorPosition(23, 4);

cnt.output(" ");

gameBegin = false;

}

// Game Over...

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

if(robots[i] != null && *map*[robots[i].getY()][robots[i].getX()] == 'X' && *map*[player.getY()][player.getX()] == *map*[robots[i].getY()][robots[i].getX()]) {

isAlive = false;

cnt.setCursorPosition(player.getX() + 6, player.getY() + 6);

cnt.output('?', new TextAttributes(Color.***RED***));

cnt.setCursorPosition(7, 32);

System.***out***.print("GAME OVER!");

}

}

Thread.*sleep*(250);

time += 1;

}

}

// getters

public int getTime() {

return time;

}

// setters

public void setTime(int time) {

this.time = time;

}

}