****

**Національний університет**

**“КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ”**

**Гра “Colorport – in search of happiness”**

Звіт з створення гри “Colorport”

курсу «Навчальна практика».

Виконали студентки 1 курсу

факультету інформатики

спеціальності “Інженерія програмного забезпечення”

Колохіна Катерина Ігорівна,

Шевченко Дар’я Валеріївна та

Бойко Христина Анатоліївна

Викладачка:

Кирієнко Оксана Валентинівна

Зміст

[1. Назва команди “RGB”. 3](#_Toc41913747)

[2. Постановка задачі. 3](#_Toc41913748)

[3. Опис гри. 3](#_Toc41913749)

[4. Розподіл задач. 3](#_Toc41913750)

[5. Структура програми. 4](#_Toc41913751)

[6. Структура даних. 4](#_Toc41913752)

[7. Опис методів та класів. 4](#_Toc41913753)

[8. Робота програми(Інструкція для користувача). 9](#_Toc41913754)

[9. Проблеми та шляхи їх вирішення 15](#_Toc41913755)

[10. Висновки 16](#_Toc41913756)

[11. Список ілюстрацій. 16](#_Toc41913757)

[12. Програмний код. 16](#_Toc41913758)

1. Назва команди “RGB”.
2. Постановка задачі.

Створити гру “Colorport – in search of happiness” з різними ускладненнями та мінімум 5 рівнями складності.

1. Опис гри.

Головний персонаж живе у світі сірості і буденності. Його життя втратило усі барви. Проте одного дня він вирішив змінити своє життя та повернути втрачену радість. Задача гравця полягає в тому, щоб допомогти знайти усі кольори персонажу. Щоправда це не так просто! На шляху до омріяного щастя потрібно пройти безліч перешкод.

Гра складається з 6 рівнів: кольорових кімнат-лабіринтів. Для того, щоб потрапити в нову кімнату потрібно пройти через портал. Шлях до нього охороняють монстри-плями. На початку гравець має 3 житті, які він може втратити(-1 життя за раз) при зустрічі з монстром, але їх можна поновити, знайшовши бонус у вигляді серця в лабіринті. З кожним рівнем гравцю стає складніше проходити кімнату, так як відбувається прискорення руху монстрів, збільшення їх кількості, а також ускладнення лабіринту та змінення місця знаходження порталу.

1. Розподіл задач.

Виконання даного завдання складалося з декількох етапів: розробка графічних об'єктів, навігації для користувача, анімації об'єктів, перевірка колізій, розробка ускладнень, розробка музичного супроводу та тестування.

Задачі кожного студента полягали в:

Христина відповідала за створення графіки лабіринту, розробляла рух та колізії об’єктів.

Катерина відповідала за дизайн графічних об’єктів, розробку анімації та перевірку колізій об’єктів.

Дар'я відповідала за створення навігації для користувача та музичне оформлення програми.

1. Структура програми.

Програма складається з кількох частин (основний фрагмент програми наявний у кінці звіту). До структури належить:

* початкові елементи, тобто призначення констант, змінних;
* створення об’єктів, які в подальшому будуть використовуватись в методах;
* методи, які виконують саме анімацію;
* методи, що перевіряють на зіткнення;
* музичний супровід;
* меню для користувача;
* додаткові файли(зображення та музика).

1. Структура даних.

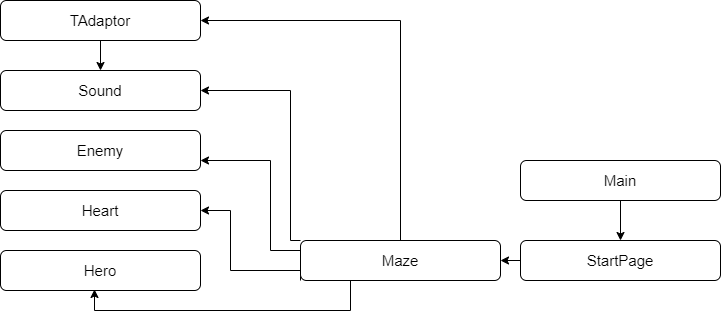


Рис. 1, структура даних

1. Опис методів та класів.

Програма “Colorport” складається з 8 класів: TAdaptor(для відгуку на дії(натискання клавіш) користувача), Sound(для контролю гучності), Enemy(для створення об’єкту монстра), Heart(для створення об’єкту серця), Hero(для створення і регулювання об’єкту головного персонажа), Maze(для створення лабіринту та регулювання дій персонажів у ньому), StartPage(для створення меню, для роботи з користувачем), Main(для запуску проєкту). Детальну інформацію про кожний з методів у класах можна переглянути в Java Doc за покликанням: !!!!потрібно додати покликання на JAVA DOC!!!!!

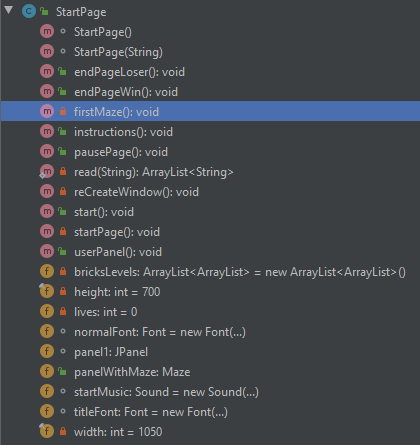


Рис. 2, клас StartPage

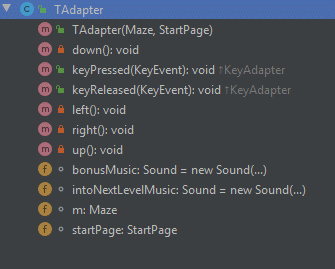


Рис. 3, клас TAdapter

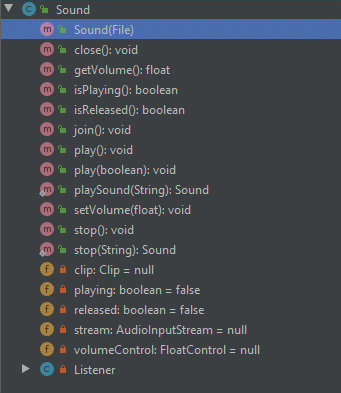


Рис. 4, клас Sound

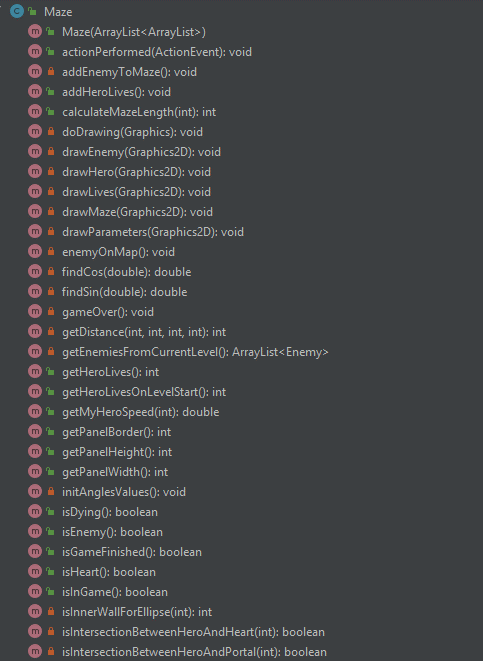


Рис. 5, клас Maze, 1 частина

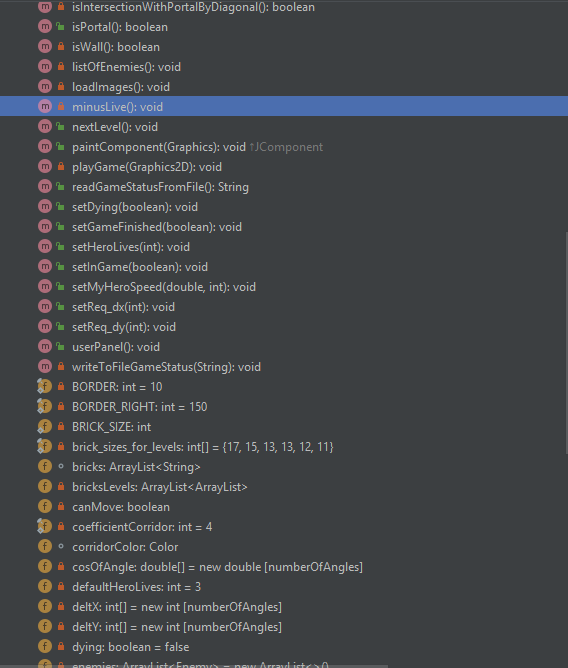


Рис. 6, клас Maze, 2 частина



Рис. 7, клас Main

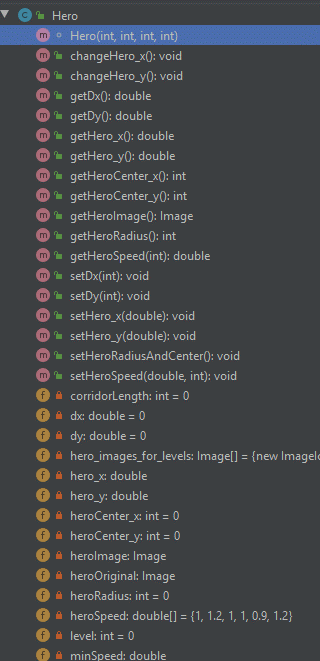


Рис. 8, клас Hero

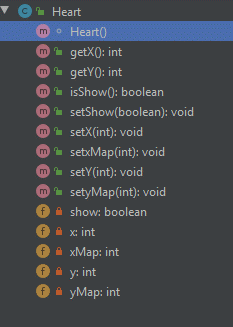


Рис. 9, клас Heart

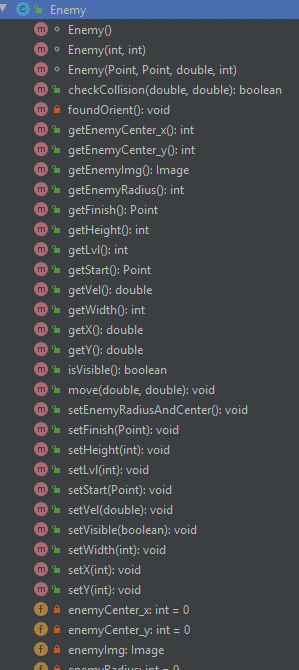


Рис. 10, клас Enemy

1. Робота програми(Інструкція для користувача).

Після запуску програми, користувач бачить стартове вікно гри.

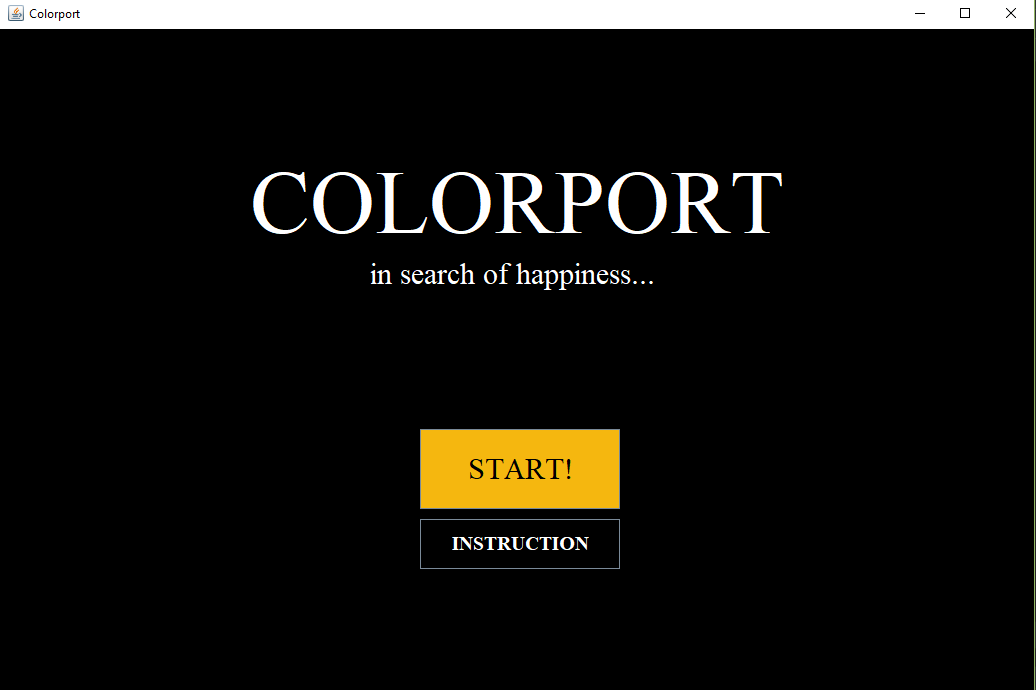
~~~~

Рис. 11, стартове вікно гри

Одразу після відкриття вікна починає звучати музичний супровід. На цьому вікні є дві кнопки: “Start”, що запускає гру, та “Instruction”, що відкриває вікно з інструкцією для користувача.

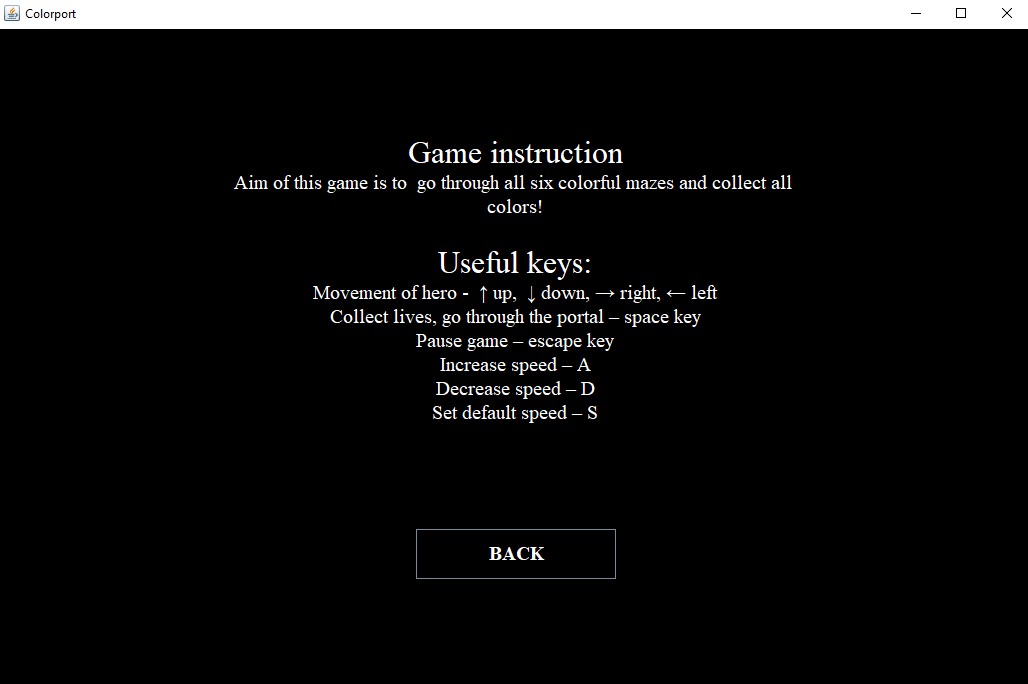


Рис. 12, вікно інструкції

На вікні “Game instruction” можна побачити інструкцію для користувача та кнопку “BACK”, що повертає до початкового меню.

Изображение выглядит как компьютер, монитор, часы, сидит

Автоматически созданное описание

Рис.13, вікно гри

Після натиснення на кнопку “Start” користувач переходить до гри. На цього вікні є сам лабіринт з монстрами, лікувальними сердечками та переходом до наступного рівня, а також панель користувача. На ній є інформація про рівень на якому зараз знаходиться користувач, кількість наявним сердець та кнопка паузи.

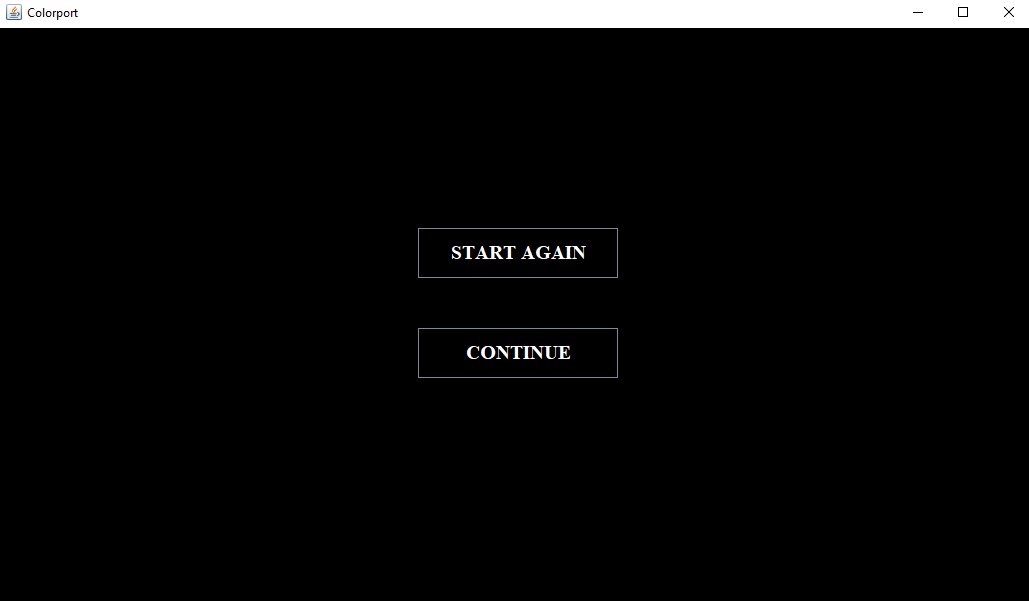
Після натиснення на кнопку паузи виникає вікно з двома кнопками: “START AGAIN”, що розпочинає рівень спочатку, та “CONTINUE” – для продовження гри.

Рис. 14, вікно паузи

**Елементи гри:**

 - головний персонаж, яким керує користувач(його зображення змінюється з кожним рівнем);

 - монстр, при натраплянні на якого головний персонаж втрачає життя.

 - серце, елемент лікування для головного персонажа.

 - портал, для переходу на наступний рівень.

**Кнопки для керування персонажем:**

Рухи героя: ↑ вгору, ↓ вниз, → вправо, ← вліво.

Збирати життя, проходити через портал – space key.

Зупинити гру – escape key.

Збільшити швидкість герою – A.

Зменшити швидкість герою – D.

Встановити швидкість за замовчуванням – S.

Якщо герой натрапляє на перешкоду(монстра), він втрачає життя та повертається на початок лабіринту. Якщо герой втратить усі життя, то гра буде завершена та з’явиться вікно “Loser”, де буде кнопка “TRY AGAIN”, щоб розпочати нову гру.

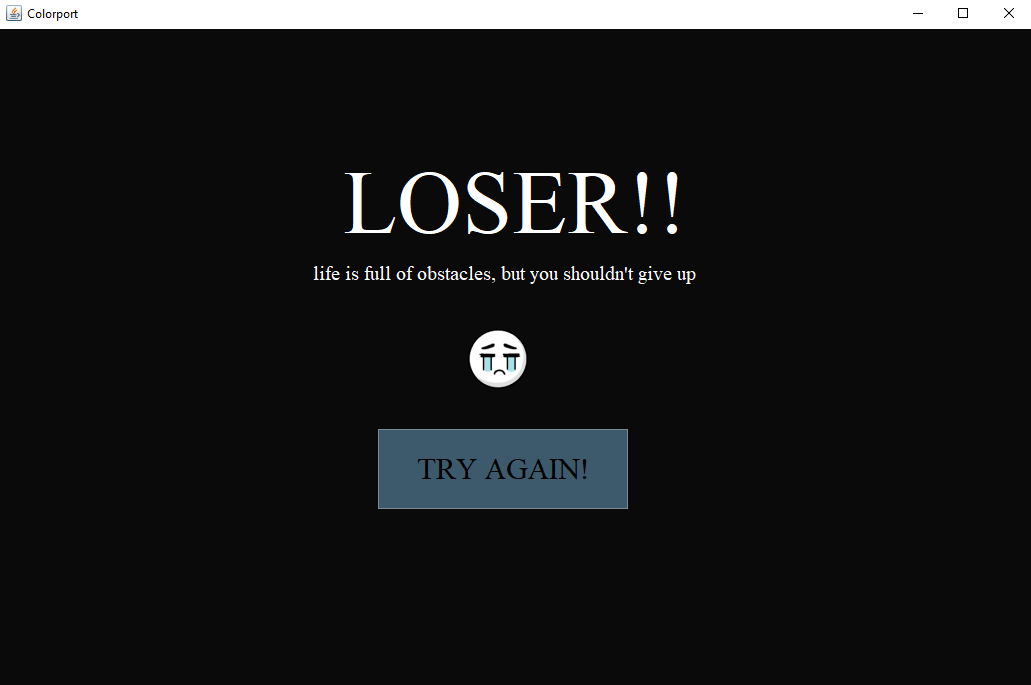


Рис. 15, вікно програшу

Якщо гравець успішно проходить усі 6 лабіринтів, то в нього з’являється вікно виграшу з кнопкою “START AGAIN!”, щоб спробувати пройти всю гру ще раз.

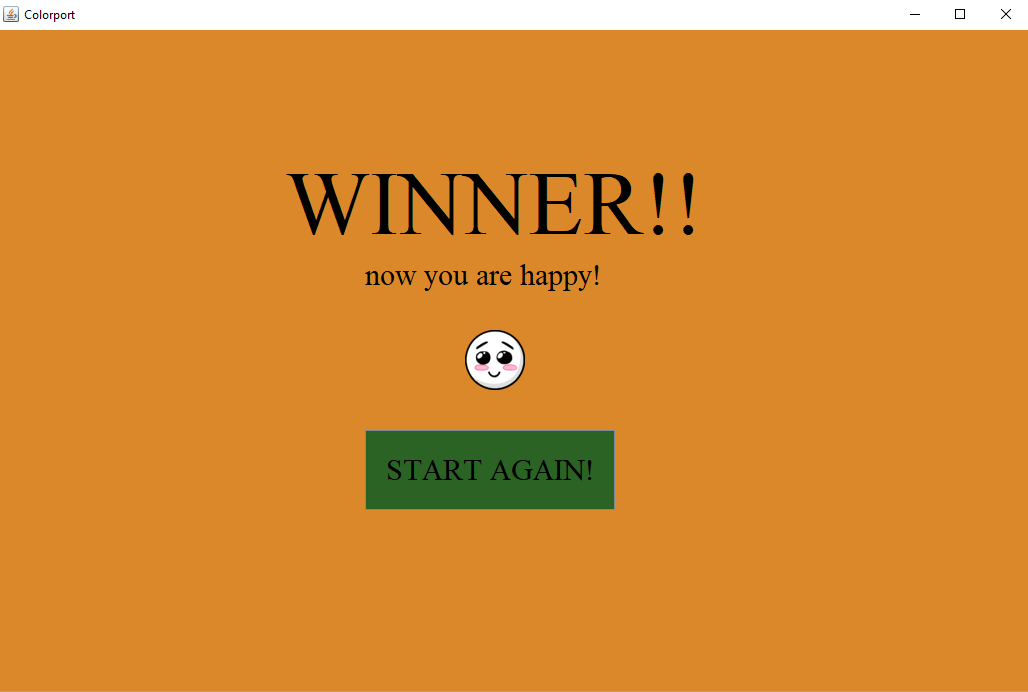


Рис. 16, вікно виграшу

Усього гра має 6 лабіринтів, що відрізняються за рівнем складності та кількістю монстрів:

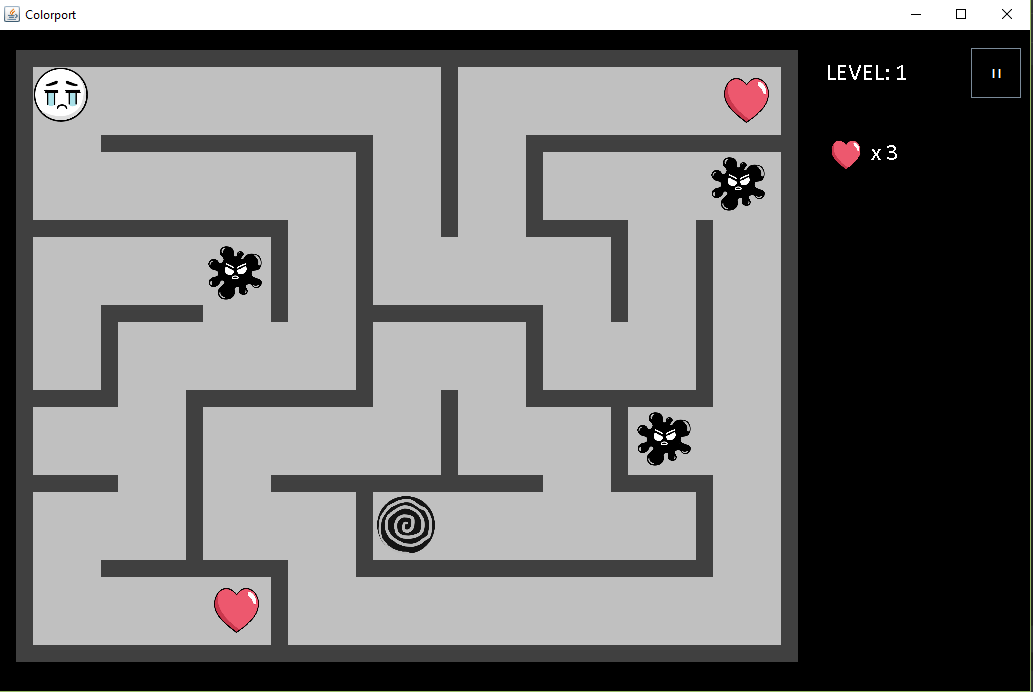


Рис.17, 1 лабіринт

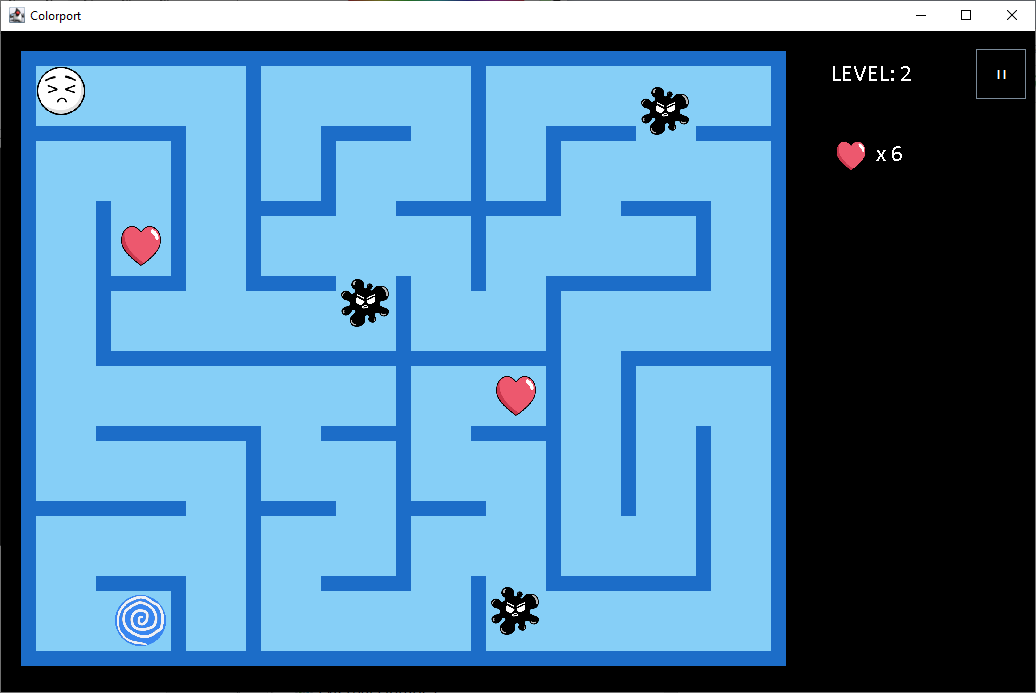


Рис. 18, 2 лабіринт

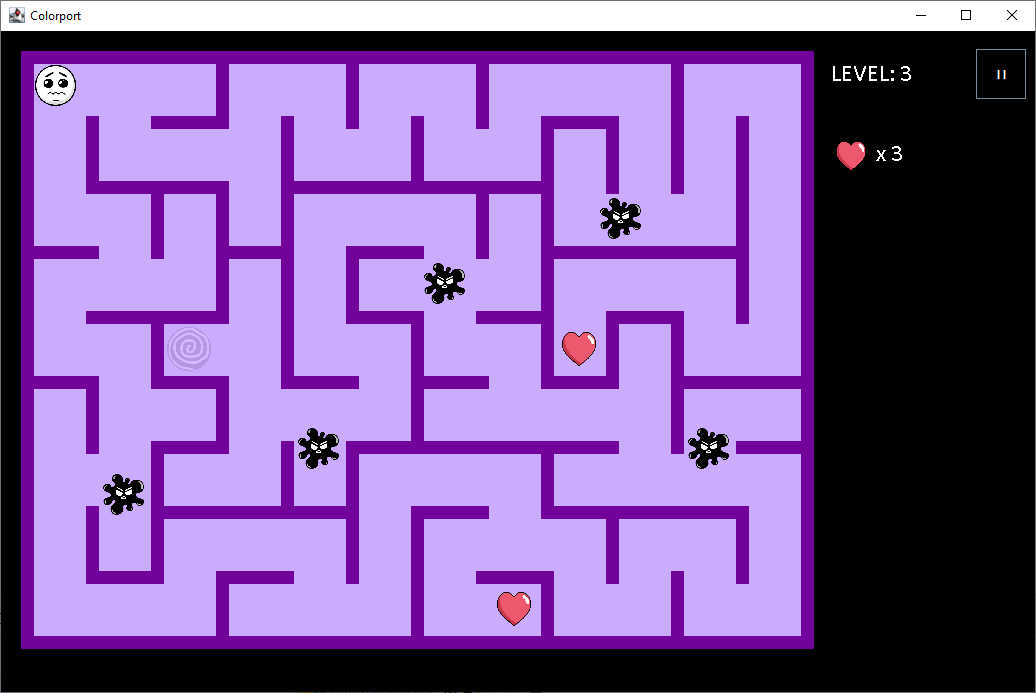


Рис. 19, 3 лабіринт



Рис.20, 4 лабіринт

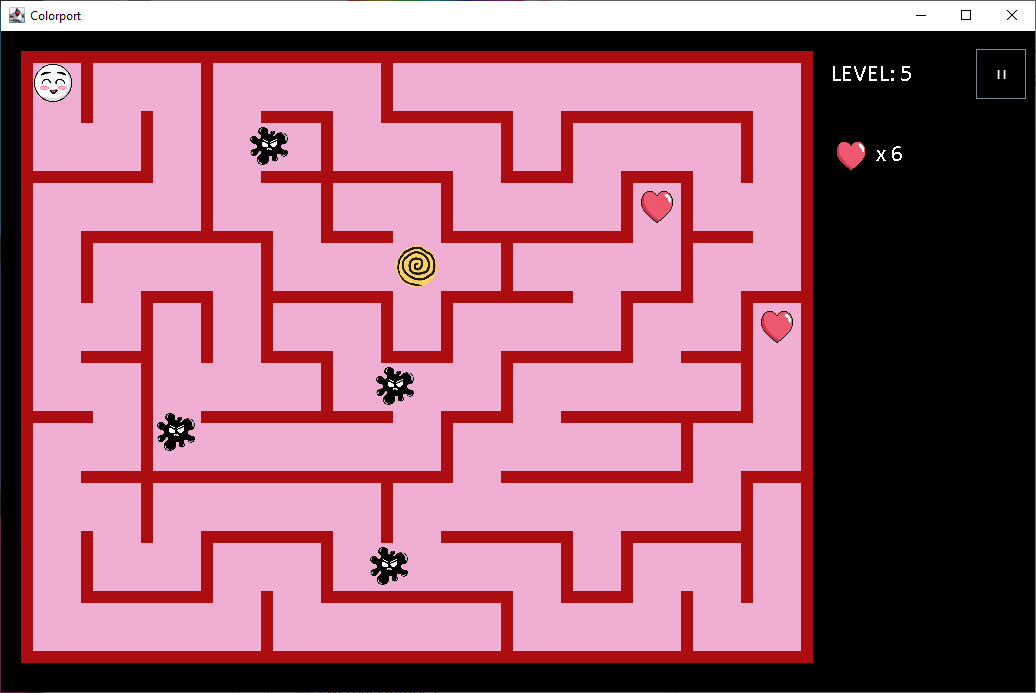


Рис. 21, 5 лабіринт



Рис. 22, 6 лабіринт

1. Проблеми та шляхи їх вирішення

Виникали проблеми з колізією персонажа та монстрів, персонажів та стін та пробудовою лабіринту. Вирішили цю проблему за допомогою корекції координат колізії, адже до того об’єкти накладалися один на одного. Також виникали проблеми зі встановленням звуку. Вирішення полягало в читанні докуменцій, аналізі коду та пошуку прикладів у глобальній мережі. Ще одною з труднощів було застосування платформи Git для контролю версій. Проте після пошуку інформації в глобальній мережі та деякої практики із застовуванням цієї платформи проблем більше не виникало.

1. Висновки

Дане завдання було неймовірно корисним для покращення навичок роботи з об’єктно-орієнтованим програмуванням. Після цієї роботи ми краще опанували графічні бібліотеки java як-от Graphics2D та Swing, навчилися робити більш ефективні інтерфейси для користувача та покращили навички роботи зі звуком у java.

1. Список ілюстрацій.

[Рис. 1, структура даних 4](#_Toc41924059)

[Рис. 2, клас StartPage 5](#_Toc41924060)

[Рис. 3, клас TAdapter 5](#_Toc41924061)

[Рис. 4, клас Sound 5](#_Toc41924062)

[Рис. 5, клас Maze, 1 частина 6](#_Toc41924063)

[Рис. 6, клас Maze, 2 частина 6](#_Toc41924064)

[Рис. 7, клас Main 7](#_Toc41924065)

[Рис. 8, клас Hero 7](#_Toc41924066)

[Рис. 9, клас Heart 8](#_Toc41924067)

[Рис. 10, клас Enemy 8](#_Toc41924068)

[Рис. 11, стартове вікно гри 9](#_Toc41924069)

[Рис. 12, вікно інструкції 9](#_Toc41924070)

[Рис.13, вікно гри 10](#_Toc41924071)

[Рис. 14, вікно паузи 10](#_Toc41924072)

[Рис. 15, вікно програшу 12](#_Toc41924073)

[Рис. 16, вікно виграшу 12](#_Toc41924074)

[Рис.17, 1 лабіринт 13](#_Toc41924075)

[Рис. 18, 2 лабіринт 13](#_Toc41924076)

[Рис. 19, 3 лабіринт 14](#_Toc41924077)

[Рис.20, 4 лабіринт 14](#_Toc41924078)

[Рис. 21, 5 лабіринт 15](#_Toc41924079)

[Рис. 22, 6 лабіринт 15](#_Toc41924080)

1. Програмний код.

Покликання на програмий код на GitHub: <https://github.com/Daria-Shevchenko/Maze/tree/ac4b113420496f25f11e62dbeba0c8c0e172f6d0>

Enemy

package com.rgb;  
  
import java.awt.\*;  
  
  
*/\*\*  
 \* Enemy - element of the game that is responsible for enemy on game screen  
 \** ***@author*** *Kate Kolokhina  
 \*/*public class Enemy {  
  
 private Toolkit t = Toolkit.*getDefaultToolkit*();  
 private Image origin = t.getImage("src/images/other/enemy.png"), enemyImg;  
  
 private int lvl;  
 private int width,height;  
 private double x,y;  
 private double vel;  
  
 private boolean visible;  
  
 private Point start;  
 private Point finish;  
  
 private int enemyRadius = 0;  
 private int enemyCenter\_x = 0;  
 private int enemyCenter\_y = 0;  
  
 */\*\*  
 \* Orientation of moving of enemy  
 \* orient = 0 - if don`t move  
 \* orient = 1 - move on X  
 \* orient = 2 - if move on Y  
 \*/* private int orient;  
  
 */\*\*  
 \* Create a basic enemy, that can`t move  
 \*/* Enemy(){  
 this.visible = false;  
 this.width = 60;  
 this.height = 60;  
 this.start = new Point(0,0);  
 this.finish = new Point(0,0);  
 this.x = start.x;  
 this.y = start.y;  
 this.vel = 0;  
 this.enemyImg = origin.getScaledInstance(width,height,Image.*SCALE\_DEFAULT*);  
 this.orient = 0;  
 foundOrient();  
 // System.out.println("enemy created");  
 }  
  
 */\*\*  
 \* Create a enemy, that can`t move  
 \** ***@param*** *w - width of enemy  
 \** ***@param*** *h - height of enemy  
 \*/* Enemy(int w, int h){  
 this.visible = false;  
 this.width = w;  
 this.height = h;  
 this.start = new Point(0,0);  
 this.finish = new Point(0,0);  
 this.x = start.x;  
 this.y = start.y;  
 this.vel = 0;  
 this.enemyImg = origin.getScaledInstance(width,height,Image.*SCALE\_DEFAULT*);  
 foundOrient();  
 // System.out.println("enemy created");  
 }  
  
 */\*\*  
 \* Create a enemy  
 \** ***@param*** *s - coordinates of the starting point of the enemy trajectory  
 \** ***@param*** *f - coordinates of the end point of the enemy trajectory  
 \** ***@param*** *vel - enemy movement speed  
 \** ***@param*** *lvl - level at which this enemy is  
 \*/* Enemy(Point s, Point f, double vel,int lvl){  
 this.lvl = lvl;  
 this.visible = false;  
 this.width = 60;  
 this.height = 60;  
 this.start = new Point(s);  
 this.finish = new Point(f);  
 this.x = start.x;  
 this.y = start.y;  
 this.vel = vel;  
 this.enemyImg = origin.getScaledInstance(width,height,Image.*SCALE\_DEFAULT*);  
 foundOrient();  
 // System.out.println("enemy created");  
 }  
  
 */\*\*  
 \* Gets enemy width  
 \** ***@return*** *An integer representing the enemy width  
 \*/* public int getWidth() {  
 return width;  
 }  
  
 */\*\*  
 \* Set up enemy width  
 \** ***@param*** *width - enemy width  
 \*/* public void setWidth(int width) {  
 this.width = width;  
 this.enemyImg = origin.getScaledInstance(width,height,Image.*SCALE\_DEFAULT*);  
 }  
  
 */\*\*  
 \* Gets enemy height  
 \** ***@return*** *An integer representing enemy height  
 \*/* public int getHeight() {  
 return height;  
 }  
  
 */\*\*  
 \* Set up enemy height  
 \** ***@param*** *height - enemy height  
 \*/* public void setHeight(int height) {  
 this.height = height;  
 this.enemyImg = origin.getScaledInstance(width,height,Image.*SCALE\_DEFAULT*);  
 }  
  
 */\*\*  
 \* Gets enemy radius  
 \** ***@return*** *An integer representing enemy radius  
 \*/* public int getEnemyRadius() {  
 return enemyRadius;  
 }  
  
 */\*\*  
 \* Calculates enemy radius and coordinates of enemy center  
 \*/* public void setEnemyRadiusAndCenter(){  
 this.enemyRadius = (int)((this.width+this.height)/4);  
 this.enemyCenter\_x = this.width/2;  
 this.enemyCenter\_y = this.height/2;  
 }  
  
 */\*\*  
 \* Gets the x-coordinate of the enemy’s center  
 \** ***@return*** *An integer representing x-coordinate of the enemy’s center  
 \*/* public int getEnemyCenter\_x() {  
 return enemyCenter\_x;  
 }  
  
 */\*\*  
 \* Gets the y-coordinate of the enemy’s center  
 \** ***@return*** *An integer representing y-coordinate of the enemy’s center  
 \*/* public int getEnemyCenter\_y() {  
 return enemyCenter\_y;  
 }  
  
 */\*\*  
 \* Gets enemy image  
 \** ***@return*** *enemy image  
 \*/* public Image getEnemyImg() {  
 return enemyImg;  
 }  
  
 */\*\*  
 \* Gets enemy visibility  
 \** ***@return*** *A boolean representing enemy visibility  
 \*/* public boolean isVisible() {  
 return visible;  
 }  
  
 */\*\*  
 \* Set up enemy visibility  
 \** ***@param*** *visible - enemy visibility  
 \*/* public void setVisible(boolean visible) {  
 this.visible = visible;  
 }  
  
 */\*\*  
 \* Gets coordinates of the starting point of the enemy trajectory  
 \** ***@return*** *A point representing coordinates starting point of the enemy trajectory  
 \*/* public Point getStart() {  
 return start;  
 }  
  
 */\*\*  
 \* Set up coordinates of the starting point of the enemy trajectory  
 \** ***@param*** *start - A point representing coordinates starting point of the enemy trajectory  
 \*/* public void setStart(Point start) {  
  
 this.start = start;  
 this.x = start.x;  
 this.y = start.y;  
 }  
  
 */\*\*  
 \* Gets the x location of enemy on screen  
 \** ***@return*** *An integer representing x location of enemy on screen  
 \*/* public double getX() {  
 return x;  
 }  
  
 */\*\*  
 \* Set up the x location of enemy on screen  
 \** ***@param*** *x - x location of enemy on screen  
 \*/* public void setX(int x) {  
 this.x = x;  
 }  
  
 */\*\*  
 \* Gets the y location of enemy on screen  
 \** ***@return*** *An integer representing y location of enemy on screen  
 \*/* public double getY() {  
 return y;  
 }  
  
 */\*\*  
 \* Set up the y location of enemy on screen  
 \** ***@param*** *y - y location of enemy on screen  
 \*/* public void setY(int y) {  
 this.y = y;  
 }  
  
 */\*\*  
 \* Gets coordinates of the end point of the enemy trajectory  
 \** ***@return*** *A point representing coordinates of the end point of the enemy trajectory  
 \*/* public Point getFinish() {  
 return finish;  
 }  
  
 */\*\*  
 \* Set up coordinates of the end point of the enemy trajectory  
 \** ***@param*** *finish - A point representing coordinates of the end point of the enemy trajectory  
 \*/* public void setFinish(Point finish) {  
 this.finish = finish;  
 }  
  
 */\*\*  
 \* Gets enemy movement speed  
 \** ***@return*** *A double representing enemy movement speed  
 \*/* public double getVel() {  
 return vel;  
 }  
  
 */\*\*  
 \* Set up enemy movement speed  
 \** ***@param*** *vel - enemy movement speed  
 \*/* public void setVel(double vel) {  
 this.vel = vel;  
 }  
  
 */\*\*  
 \* Gets level at which this enemy is  
 \** ***@return*** *An integer representing level at which this enemy is  
 \*/* public int getLvl() {  
 return lvl;  
 }  
  
 */\*\*  
 \* Set up level at which this enemy is  
 \** ***@param*** *lvl - level at which this enemy is  
 \*/* public void setLvl(int lvl) {  
 this.lvl = lvl;  
 }  
  
 */\*\*  
 \* This method check the collision between enemy and hero or game world  
 \** ***@param*** *frameW - width of frame  
 \** ***@param*** *frameH - height of frame  
 \** ***@return*** *\*/* public boolean checkCollision(double frameW,double frameH) {  
 if (orient == 1){  
 if(x < start.x || x +width > finish.x )  
 return true;  
 }  
 if (orient == 2){  
 if(y < start.y || y +width > finish.y)  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \* This method move the enemy on screen  
 \** ***@param*** *frameW - width of frame  
 \** ***@param*** *frameH - height of frame  
 \*/* public void move(double frameW,double frameH){  
 //System.out.println(checkCollision(frameW,frameH));  
 if(checkCollision(frameW,frameH) == true){  
 vel = -vel;  
 }  
 // double r = Math.random()-0.5;  
 // int sign = (int) Math.signum(r);  
 if (orient == 1){  
 x+=vel*/\*\*sign\*/*;  
 }  
 if (orient == 2){  
 y+=vel*/\*\*sign\*/*;  
 }  
 }  
  
 */\*\*  
 \* This method is calculated orientation of moving of enemy  
 \*/* private void foundOrient(){  
 if(start.x == finish.x && start.y != finish.y){  
 this.orient = 2;  
 }else if(start.y == finish.y && start.x != finish.x){  
 this.orient = 1;  
 }else {  
 orient = 0;  
 }  
 }  
}

Heart

package com.rgb;  
  
*/\*\*  
 \*  
 \* Heart - element of the game that is responsible for restoring the player’s lives  
 \** ***@author*** *Kate Kolokhina  
 \*/*public class Heart {  
  
 private boolean show;  
 private int x,y;  
 private int xMap,yMap;  
  
 */\*\*  
 \* Create a new heart  
 \*/* Heart(){  
 show = true;  
 }  
  
 */\*\*  
 \* Set up x location of heart in the level map  
 \** ***@param*** *xMap - x location of heart in the level map  
 \*/* public void setxMap(int xMap) {  
 this.xMap = xMap;  
 }  
  
 */\*\*  
 \* Set up y location of heart in the level map  
 \** ***@param*** *yMap - y location of heart in the level map  
 \*/* public void setyMap(int yMap) {  
 this.yMap = yMap;  
 }  
  
 */\*\*  
 \* Set up heart visibility  
 \** ***@param*** *show - heart visibility  
 \*/* public void setShow(boolean show) {  
 this.show = show;  
 }  
  
 */\*\*  
 \* Set up x location of heart on screen  
 \** ***@param*** *x - x location of heart on screen  
 \*/* public void setX(int x) {  
 this.x = x;  
 }  
  
 */\*\*  
 \* Set up y location of heart on screen  
 \** ***@param*** *y - y location of heart on screen  
 \*/* public void setY(int y) {  
 this.y = y;  
 }  
  
 */\*\*  
 \* Gets the heart visibility  
 \** ***@return*** *A boolean representing heart visibility  
 \*/* public boolean isShow() {  
 return show;  
 }  
  
 */\*\*  
 \* Gets the x location of heart on screen  
 \** ***@return*** *A integer representing x location of heart on screen  
 \*/* public int getX() {  
 return x;  
 }  
  
 */\*\*  
 \* Gets the y location of heart on screen  
 \** ***@return*** *A integer representing y location of heart on screen  
 \*/* public int getY() {  
 return y;  
 }  
  
  
}

Hero

package com.rgb;  
*/\*\*  
 \*  
 \* Hero - element of the game that is responsible for hero on game screen  
 \** ***@author*** *Khrystyna Boiko  
 \*/*import javax.swing.\*;  
import java.awt.\*;  
  
public class Hero {  
 */\*\*level is level of the game\*/* private int level = 0;  
 */\*\*heroSpeed is an array of hero speeds on different levels\*/* private double [] heroSpeed = {1, 1.2, 1, 1, 0.9, 1.2};  
 */\*\*minSpeed is minimum speed for the level\*/* private double minSpeed;  
 */\*\*pictureProportionToCell is proportion of picture length to cell length\*/* private double pictureProportionToCell = 0.8;  
 */\*\*hero\_images\_for\_levels is an array of hero images on different levels\*/* private Image [] hero\_images\_for\_levels = {  
 new ImageIcon("src/images/hero/gg1.png").getImage(), new ImageIcon("src/images/hero/gg2.png").getImage(),  
 new ImageIcon("src/images/hero/gg3.png").getImage(), new ImageIcon("src/images/hero/gg4.png").getImage(),  
 new ImageIcon("src/images/hero/gg5.png").getImage(), new ImageIcon("src/images/hero/gg6.png").getImage()};  
 */\*\*heroOriginal is an original image of hero on the level\*/* private Image heroOriginal;  
 */\*\*heroImage is scaled image of hero getting from heroOriginal\*/* private Image heroImage;  
 */\*\*heroRadius is radius of hero image\*/* private int heroRadius = 0;  
 */\*\*heroCenter\_x is half length of the hero image on X in pixels\*/* private int heroCenter\_x= 0;  
 */\*\*heroCenter\_y is half length of the hero image on Y in pixels\*/* private int heroCenter\_y = 0;  
 */\*\*hero\_x is X coordinate of the hero\*/* private double hero\_x;  
 */\*\*hero\_y is Y coordinate of the hero\*/* private double hero\_y;  
 */\*\*dx is displacement of hero on X\*/* private double dx=0;  
 */\*\*dy is displacement of hero on Y\*/* private double dy=0;  
 */\*\*corridorLength is length of the cell (corridor) in maze\*/* private int corridorLength = 0;  
 */\*\*Create a hero  
 \** ***@param*** *new\_level - level of the game  
 \** ***@param*** *new\_corridorLength - length of corridor in a maze  
 \** ***@param*** *hero\_x0 - X coordinate of the hero  
 \** ***@param*** *hero\_y0 - Y coordinate of the hero  
 \* \*/* Hero(int new\_level, int new\_corridorLength, int hero\_x0, int hero\_y0){  
 this.level = new\_level;  
 this.minSpeed =heroSpeed[level-1];  
 this.corridorLength = new\_corridorLength;  
 this.heroOriginal = hero\_images\_for\_levels[this.level-1];  
 double length = corridorLength\*pictureProportionToCell;  
 this.heroImage = heroOriginal.getScaledInstance((int)length, (int)length, Image.*SCALE\_DEFAULT*);  
 this.hero\_x = hero\_x0;  
 this.hero\_y = hero\_y0;  
 }  
 */\*\*Set X coordinate for hero  
 \** ***@param*** *hero\_x - X coordinate of hero\*/* public void setHero\_x(double hero\_x) {  
 this.hero\_x = hero\_x;  
 }  
 */\*\*Set Y coordinate for hero  
 \** ***@param*** *hero\_y - Y coordinate of hero\*/* public void setHero\_y(double hero\_y) {  
 this.hero\_y = hero\_y;  
 }  
 */\*\*Gets X coordinate of the hero  
 \** ***@return*** *An integer representing X coordinate of the hero\*/* public double getHero\_x() {  
 return hero\_x;  
 }  
 */\*\*Gets Y coordinate of the hero  
 \** ***@return*** *An integer representing Y coordinate of the hero\*/* public double getHero\_y() {  
 return hero\_y;  
 }  
 */\*\*Change X coordinate of the hero by displacement of hero on X\*/* public void changeHero\_x() {  
 this.hero\_x = this.hero\_x + this.dx;  
 }  
 */\*\*Change Y coordinate of the hero by displacement of hero on Y\*/* public void changeHero\_y() {  
 this.hero\_y = this.hero\_y + this.dy;  
 }  
 */\*\*Set displacement of hero on X  
 \** ***@param*** *dx - displacement of hero on X\*/* public void setDx(int dx) {  
 this.dx = dx\*this.heroSpeed[this.level-1];  
 }  
 */\*\*Set displacement of hero on Y  
 \** ***@param*** *dy - displacement of hero on Y\*/* public void setDy(int dy) {  
 this.dy = dy\*this.heroSpeed[this.level-1];  
 }  
 */\*\*Gets displacement of hero on X  
 \** ***@return*** *An integer representing a displacement of hero on X\*/* public double getDx() {  
 return dx;  
 }  
 */\*\*Gets displacement of hero on Y  
 \** ***@return*** *An integer representing a displacement of hero on Y\*/* public double getDy() {  
 return dy;  
 }  
 */\*\*Set parameters heroRadius, heroCenter\_x, heroCenter\_y\*/* public void setHeroRadiusAndCenter() {  
 this.heroRadius = (int)(corridorLength\*pictureProportionToCell/2);  
 this.heroCenter\_x = (int)(corridorLength\*pictureProportionToCell/2);  
 this.heroCenter\_y = (int)(corridorLength\*pictureProportionToCell/2);  
 }  
 */\*\*Gets a radius of hero image  
 \** ***@return*** *An integer representing a radius of hero image\*/* public int getHeroRadius() {  
 return heroRadius;  
 }  
 */\*\*Gets center of the hero - half of the hero image on X  
 \** ***@return*** *An integer representing a half length of the hero image on X in pixels\*/* public int getHeroCenter\_x() {  
 return heroCenter\_x;  
 }  
 */\*\*Gets center of the hero - half of the hero image on Y  
 \** ***@return*** *An integer representing a half length of the hero image on Y in pixels\*/* public int getHeroCenter\_y() {  
 return heroCenter\_y;  
 }  
 */\*\*Gets a hero image  
 \** ***@return*** *An image representing a hero image on the level\*/* public Image getHeroImage() {  
 return heroImage;  
 }  
 */\*\*Set hero speed* ***@param*** *new\_heroSpeed A new speed for hero* ***@param*** *level A level of the game \*/* public void setHeroSpeed(double new\_heroSpeed, int level) {  
 if(new\_heroSpeed<=this.minSpeed+0.5 && new\_heroSpeed>=this.minSpeed)  
 this.heroSpeed [level] = new\_heroSpeed;  
 }  
 */\*\*Gets hero speed* ***@param*** *level A level of the game* ***@return*** *An integer representing a hero speed on the level\*/* public double getHeroSpeed(int level){  
 return this.heroSpeed[level];  
 }  
  
}

Main

package com.rgb;  
*/\*\*  
 \*  
 \* Main - public class for launch the game  
 \** ***@author*** *Shevchenko Daria  
 \*/*import java.awt.\*;  
  
public class Main {  
  
 public static void main(String[] args) {  
 EventQueue.*invokeLater*(() -> {  
 /\*ex is a frame with the game "Colorport"\*/  
 StartPage ex = new StartPage("Colorport");  
 ex.setVisible(true);  
 });  
 }  
}

Maze

package com.rgb;  
*/\*\*  
 \*  
 \* Maze - element of the game that is responsible for drawing  
 \* all other elements (maze, hero, enemies, hearts, portal) on game screen  
 \** ***@author*** *Kate Kolokhina and Khrystyna Boiko  
 \*/*import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
import java.awt.image.RescaleOp;  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Scanner;  
  
public class Maze extends JPanel implements ActionListener {  
  
  
  
 */\*\*  
 \* Sound panel  
 \* intoMouster - sound for collidion with monster  
 \*/* Sound intoMonster = new Sound(new File("src/music/intoMonster.wav"));  
 /\*alpha - portal transparency level\*/  
 private int alpha = 255;  
 /\*alphaSign - direction of transparency change\*/  
 private int alphaSign = 1;  
 private int cheat = 0;  
 public void setCheat (int newValue) {  
 this.cheat = newValue;  
 }  
 public final static Sound *lev1* = new Sound(new File("src/music/lev1.wav"));  
 public final static Sound *lev2* = new Sound(new File("src/music/lev2.wav"));  
 public final static Sound *lev3* = new Sound(new File("src/music/lev3.wav"));  
 public final static Sound *lev4* = new Sound(new File("src/music/lev4.wav"));  
 public final static Sound *lev5* = new Sound(new File("src/music/lev5.wav"));  
 public final static Sound *lev6* = new Sound(new File("src/music/lev6.wav"));  
 */\*\*enemySize is size of enemy picture\*/* private int enemySize;  
 */\*\*enemyProportion is proportion of enemy picture to the cell\*/* private double enemyProportion = 0.8;  
 */\*\*enemies is arraylist of all enemies\*/* private ArrayList<Enemy> enemies = new ArrayList<>();  
 */\*\*sf is variable which has start and finish points of enemy movement\*/* private Map<Point, Point> sf;  
 */\*\*bricksLevels is an arraylist of arraylists with maze map strings\*/* private ArrayList<ArrayList> bricksLevels;  
 */\*\*gameLevel is level of the game\*/* public int gameLevel = 0;  
 */\*\*MAX\_gamelevel is maximum level of the game\*/* private int MAX\_gamelevel = 6;  
 */\*\*gameFinished is boolean for finish of the game\*/* private boolean gameFinished = false;  
 */\*\*inGame is boolean for game working\*/* private boolean inGame = false;  
 */\*\*dying is boolean for lose the game\*/* private boolean dying = false;  
 */\*\*canMove is boolean for ability to hero move \*/* private boolean canMove;  
 */\*\*myHero is hero object\*/* private Hero myHero;  
 */\*\*heart1, heart2 are heart objects\*/* private Heart heart1 = new Heart(),  
 heart2 = new Heart();  
 */\*\*defaultHeroLives is number of lives that hero has at the start of the game\*/* private int defaultHeroLives = 3;  
 */\*\*heroLives is current quantity of hero lives\*/* private int heroLives = defaultHeroLives;  
 */\*\*heroLivesOnLevelStart is quantity of hero lives at the beginning of the level\*/* private int heroLivesOnLevelStart = defaultHeroLives;  
 */\*\*pathToFileWithGameStatus is path to file with game status (level and hero lives)\*/* private String pathToFileWithGameStatus = "src/mazeFiles/levelStatus.txt";  
 */\*\*timer is timer for game\*/* private Timer timer;  
 */\*\*map is array with information about cells in maze  
 0 - corridor  
 1 - wall  
 2 - portal  
 3 - heart  
 7 - corridor with enemy  
 9 - outside wall  
 \*/* public int [][] map;  
 */\*\*bricks is arraylist of strings (map of maze)\*/* ArrayList<String> bricks;  
 */\*\*pictureShift is shift of picture from cell border\*/* private int pictureShift = 6;  
 */\*\*pictureLengthInCell is relative length of picture to cell length\*/* private int pictureLengthInCell = 4;  
 */\*\*heart, portal are images of heart and portal\*/* private Image heart,portal;  
 */\*\*portal\_x - X coordinate of portal  
 \*portal\_y - Y coordinate of portal\*/* private int portal\_x=0, portal\_y=0;  
 */\*\*brick\_sizes\_for\_levels is array of brick sizes for all levels of the game\*/* private static final int [] *brick\_sizes\_for\_levels* ={17, 15, 13, 13, 12, 11};  
 */\*\*BRICK\_SIZE is size of brick on current level\*/* private static int *BRICK\_SIZE*;  
 */\*\*coefficientCorridor is the number that shows how many times the corridor is larger than the inner wall\*/* private static final int *coefficientCorridor* = 4;  
 */\*\*outsideWallCoef is the number that shows how many times the outside wall is larger than the inner wall\*/* private static final int *outsideWallCoef* = 1;  
 */\*\*BORDER is length of border on jpanel\*/* private static final int *BORDER* = 10;  
 */\*\*BORDER\_RIGHT is length of border on jpanel\*/* private static final int *BORDER\_RIGHT* = 150;  
 */\*\*mazeWidth is maze width\*/* private int mazeWidth = 0;  
 */\*\*mazeHeight is maze height\*/* private int mazeHeight = 0;  
 */\*\*enemyW,enemyH are width and height of enemy\*/  
 /\*\*numberOfAngles is all angles on circle\*/* private int numberOfAngles = 360;  
 */\*\*cosOfAngle is array of cosine of all angles on circle\*/* private double [] cosOfAngle = new double [numberOfAngles];  
 */\*\*sinOfAngle is array of sine of all angles on circle\*/* private double [] sinOfAngle = new double [numberOfAngles];  
 */\*\*deltX is distance on X to point on circle border\*/* private int [] deltX = new int [numberOfAngles];  
 */\*\*deltY is distance on Y to point on circle border\*/* private int [] deltY = new int [numberOfAngles];  
 */\*\*enemiesFromCurrentLevel is list of enemies on current level\*/* ArrayList<Enemy> enemiesFromCurrentLevel = new ArrayList<Enemy>();  
 */\*\*portal\_images\_for\_levels is array of portal images for all levels\*/* private Image [] portal\_images\_for\_levels = {  
 new ImageIcon("src/images/portal/portal1.png").getImage(), new ImageIcon("src/images/portal/portal2.png").getImage(),  
 new ImageIcon("src/images/portal/portal4.png").getImage(), new ImageIcon("src/images/portal/portal3.png").getImage(),  
 new ImageIcon("src/images/portal/portal5.png").getImage(), new ImageIcon("src/images/portal/portal6.png").getImage()};  
 */\*\*mazeColorsForWalls is array of colors for walls for all levels\*/* Color [] mazeColorsForWalls = {Color.*DARK\_GRAY*, new Color(28, 109, 200), new Color(113, 2, 155), new Color(20, 150, 14), new Color(172, 13, 17), new Color(193, 96, 6)};  
 */\*\*mazeColorsForCorridors is array of colors for corridors for all levels\*/* Color [] mazeColorsForCorridors = {Color.*LIGHT\_GRAY*, new Color(134, 207, 247), new Color(202, 171, 254), new Color(154, 255, 125), new Color(240, 174, 210), new Color(250, 197, 86)};  
 */\*\*corridorColor is color of corridors on current level\*/* Color corridorColor;  
 */\*\*wallColor is color of walls on current level\*/* Color wallColor;  
 */\*\*Create a maze game  
 \*****@param*** *bricksLevels - an arraylist of arraylists with maze map strings\*/* public Maze(ArrayList<ArrayList> bricksLevels) {  
 // System.out.println("Maze - it is Constructor");  
 this.bricksLevels = bricksLevels;  
 /\*Set delay to timer and start the timer\*/  
 timer = new Timer(5, this);  
 timer.start();  
 /\*Create list of enemies\*/  
 listOfEnemies();  
 /\*Prepare game for next level\*/  
 nextLevel();  
 /\*Calculate and fill sinOfAngle, cosOfAngle, deltX, deltY with values\*/  
 initAnglesValues();  
 }  
 */\*\*Calculate and fill sinOfAngle, cosOfAngle, deltX, deltY with values for each angle on the circle\*/* private void initAnglesValues(){  
 // System.out.println("initAngleValues");  
  
 /\*x\_center is half width of hero image\*/  
 int x\_center = myHero.getHeroImage().getWidth(null) / 2;  
 /\*y\_center is half height of hero image\*/  
 int y\_center = myHero.getHeroImage().getHeight(null) / 2;  
 for(int i=0; i<numberOfAngles; i++) {  
 /\*Fill sinOfAngle, cosOfAngle with cosines and sines\*/  
 sinOfAngle[i] = findSin(i);  
 cosOfAngle[i] = findCos(i);  
 /\*rX, rY are distances from the center line of picture to point on circle border\*/  
 double rX = myHero.getHeroImage().getWidth(null) \* cosOfAngle[i] / 2;  
 double rY = myHero.getHeroImage().getHeight(null) \* sinOfAngle[i] / 2;  
 /\*Fill deltX, deltY with values\*/  
 deltX [i] = x\_center + (int) (rX);  
 deltY [i] = y\_center + (int) (rY);  
 }  
 }  
 */\*\*Calculate and return cosine of angle  
 \** ***@param*** *degree - angle which cosine will be calculated  
 \** ***@return*** *An integer representing cosine of angle\*/* private double findCos(double degree){  
 */\*\*Converting values to radians \*/* double a = Math.*toRadians*(degree);  
 double cos = Math.*cos*(a);  
 */\*\*Consider special angles in calculating\*/* if(degree!=0 && degree % 180 != 0 && degree % 90 == 0){cos = 0;}  
 return cos;  
 }  
 */\*\*Calculate and return sine of angle  
 \** ***@param*** *degree - angle which sine will be calculated  
 \** ***@return*** *An integer representing sine of angle\*/* private double findSin(double degree){  
 */\*\*Converting values to radians \*/* double a = Math.*toRadians*(degree);  
 double sin = Math.*sin*(a);  
 */\*\*Consider special angles in calculating\*/* if(degree == 0 || degree % 180 == 0){sin = 0;}  
 return sin;  
 }  
 */\*\*Gets distance between two points  
 \** ***@param*** *x1 - X coordinate of first point  
 \** ***@param*** *y1 - Y coordinate of first point  
 \** ***@param*** *x2 - X coordinate of second point  
 \** ***@param*** *y2 - Y coordinate of second point  
 \** ***@return*** *An integer representing a distance between two points calculated by their X and Y coordinates\*/* private int getDistance(int x1, int y1, int x2, int y2){  
 return (int)Math.*sqrt*((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));  
 }  
 */\*\*Gets boolean of game finish  
 \** ***@return*** *A boolean representing game finish\*/* public boolean isGameFinished() {  
 return gameFinished;  
 }  
 */\*\*Set gameFinished value  
 \** ***@param*** *gameFinished - value for game finish boolean\*/* public void setGameFinished(boolean gameFinished) {  
 this.gameFinished = gameFinished;  
 }  
 */\*\*Gets boolean of game working  
 \** ***@return*** *A boolean representing game working\*/* public boolean isInGame() {  
 return inGame;  
 }  
 */\*\*Set inGame value  
 \** ***@param*** *inGame - value for game working boolean\*/* public void setInGame(boolean inGame) {  
 this.inGame = inGame;  
 }  
 */\*\*Gets boolean of hero death  
 \** ***@return*** *A boolean representing if hero is alive or dead\*/* public boolean isDying() {  
 return dying;  
 }  
 */\*\*Set dying value  
 \** ***@param*** *dying - value for hero live or death\*/* public void setDying(boolean dying) {  
 this.dying = dying;  
 }  
 */\*\*Add +1 to hero lives and write new information to file\*/* public void addHeroLives(){  
 heroLives++;  
 writeToFileGameStatus((gameLevel-1) + "|" + heroLives);  
 }  
 */\*\*Set heroLives and heroLivesOnLevelStart with new values,  
 \* write new information to file  
 \** ***@param*** *new\_heroLives - quantity of hero lives\*/* public void setHeroLives(int new\_heroLives) {  
 this.heroLives = new\_heroLives;  
 this.heroLivesOnLevelStart = new\_heroLives;  
 writeToFileGameStatus((gameLevel-1) + "|" + heroLives);  
 }  
 */\*\*Gets quantity of hero lives at the beginning of the level  
 \** ***@return*** *An integer representing quantity of hero lives at the beginning of the level\*/* public int getHeroLivesOnLevelStart() {  
 return heroLivesOnLevelStart;  
 }  
 */\*\*Gets quantity of hero lives  
 \** ***@return*** *An integer representing quantity of hero lives\*/* public int getHeroLives(){  
 return heroLives;  
 }  
 */\*\*Set hero speed on current level  
 \** ***@param*** *new\_heroSpeed - new hero speed  
 \** ***@param*** *level - current level\*/* public void setMyHeroSpeed(double new\_heroSpeed, int level) {  
 this.myHero.setHeroSpeed(new\_heroSpeed, level);  
 }  
 */\*\*Gets hero speed* ***@param*** *level - level from which the hero speed is taken* ***@return*** *An integer representing hero speed on the level\*/* public double getMyHeroSpeed(int level){  
 return this.myHero.getHeroSpeed(level);  
 }  
 */\*\*Writes to file game status  
 \** ***@param*** *data - information about game level and number of hero lives\*/* private void writeToFileGameStatus(String data){  
 // System.out.println("writeToFileGameStatus");  
 File file = new File(pathToFileWithGameStatus);  
 FileWriter fr = null;  
 try {  
 fr = new FileWriter(file);  
 fr.write(data);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }finally{  
 //close resources  
 try {  
 fr.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\* Reads from the file data about game level and number of hero lives  
 \** ***@return*** *A string representing the result of reading the file.  
 \* Result is empty when operation is successful and contains error if not.\*/* public String readGameStatusFromFile()  
 {//System.out.println("readGameStatusFromFile");  
 Scanner rd;  
 String result = "";  
 try  
 {  
 rd = new Scanner(new File(pathToFileWithGameStatus));  
 while (rd.hasNextLine())  
 {  
 String str = rd.nextLine();  
 // "|" is a symbol used to separate fields in saved file  
 String[] record = str.split("[|]");  
 gameLevel = Integer.*parseInt*(record[0]);  
 heroLives = Integer.*parseInt*(record[1]);  
  
 }  
 rd.close();  
 }  
 catch (IOException e)  
 {  
 result = "File is not found " + pathToFileWithGameStatus;  
 }  
 return result;  
 }  
  
 */\*\*Loading images for maze\*/* private void loadImages() {  
 // System.out.println("loadImages");  
 Toolkit t=Toolkit.*getDefaultToolkit*();  
 heart = t.getImage("src/images/other/heart\_red\_s.png");  
 portal = portal\_images\_for\_levels[this.gameLevel-1];  
 }  
 */\*\*Prepare for next level. Initialize variables with correct values for that level\*/* public void nextLevel(){  
 // System.out.println("nextLevel");  
 readGameStatusFromFile();  
 if(gameLevel<MAX\_gamelevel) {  
 if(gameLevel == 1) {  
 *lev2*.play();  
 }  
  
 switch(gameLevel) {  
 case 0:  
 if(*lev1*.isPlaying() == false)  
 *lev1*.play();  
 break;  
 case 1:  
 *lev1*.stop();  
 if(*lev2*.isPlaying() == false)  
 *lev2*.play();  
 break;  
 case 2:  
 *lev2*.stop();  
 if(*lev3*.isPlaying() == false)  
 *lev3*.play();  
 break;  
 case 3:  
 *lev3*.stop();  
 if(*lev4*.isPlaying() == false)  
 *lev4*.play();  
 break;  
 case 4:  
 *lev4*.stop();  
 if(*lev5*.isPlaying() == false)  
 *lev5*.play();  
 break;  
 case 5:  
 *lev5*.stop();  
 if(*lev6*.isPlaying() == false)  
 *lev6*.play();  
 break;  
 default:  
 // StartPage.startMusic.play();  
 break;  
 }  
 gameLevel++;  
 *BRICK\_SIZE* = *brick\_sizes\_for\_levels*[gameLevel-1];  
 heart1.setShow(true);  
 heart2.setShow(true);  
 enemySize = (int)(*BRICK\_SIZE*\**coefficientCorridor*\*enemyProportion);  
 myHero = new Hero(gameLevel, *BRICK\_SIZE*\**coefficientCorridor*, *BORDER* + *BRICK\_SIZE* \* *outsideWallCoef* + 1, *BORDER* + *BRICK\_SIZE* \* *outsideWallCoef* + 1);  
 myHero.setHeroRadiusAndCenter();  
 heroLivesOnLevelStart = heroLives;  
 this.bricks = this.bricksLevels.get(gameLevel-1);  
 wallColor = mazeColorsForWalls[gameLevel-1];  
 corridorColor = mazeColorsForCorridors[gameLevel-1];  
 if (bricks.size() != 0) {  
 mazeWidth = calculateMazeLength(this.bricks.get(0).length());  
 mazeHeight = calculateMazeLength(this.bricks.size());  
 }  
 map = new int[this.bricks.size()][this.bricks.get(0).length()];  
  
 enemyOnMap();  
 addEnemyToMaze();  
 enemiesFromCurrentLevel = getEnemiesFromCurrentLevel();  
 // this.setSize(mazeWidth + 2 \* BORDER, mazeHeight + 2 \* BORDER);  
 this.setSize(mazeWidth + 2 \* *BORDER*+*BORDER\_RIGHT*, mazeHeight + 2 \* *BORDER*);  
 loadImages();  
  
 } else {  
 gameOver();  
 }  
 }  
  
  
 */\*\*  
 \* Create a list of enemies for game levels  
 \*/* private void listOfEnemies() {  
 */\*\*  
 \* 1 lvl  
 \*/* enemies.add(new Enemy(new Point(14,4), new Point(18,4),0.6,1));  
 enemies.add(new Enemy(new Point(6,6), new Point(6,8),0.6,1));  
 enemies.add(new Enemy(new Point(16,10), new Point(18,10),0.6,1));  
 */\*\*  
 \* 2 lvl  
 \*/* enemies.add(new Enemy(new Point(18,2), new Point(18,4),0.5,2));  
 enemies.add(new Enemy(new Point(10,4), new Point(10,8),0.5,2));  
 enemies.add(new Enemy(new Point(14,12), new Point(14,16),0.47,2));  
  
 */\*\*  
 \* 3 lvl  
 \*/* enemies.add(new Enemy(new Point(18,6 ), new Point(22,6),0.4,3));  
 enemies.add(new Enemy(new Point(12,8), new Point(16,8),0.6,3));  
 enemies.add(new Enemy(new Point(4,10), new Point(4,16),0.45,3));  
 enemies.add(new Enemy(new Point(10,12), new Point(10,14),0.4,3));  
 enemies.add(new Enemy(new Point(22,12), new Point(22,14),0.4,3));  
 */\*\*  
 \* 4 lvl  
 \*/* enemies.add(new Enemy(new Point(14,6 ), new Point(16,6),0.3,4));  
 //enemies.add(new Enemy(enemyW,enemyH,new Point(14,8), new Point(14,10),0.3,4));  
  
 enemies.add(new Enemy(new Point(22,10), new Point(24,10),0.3,4));  
  
 enemies.add(new Enemy(new Point(22,14), new Point(24,14),0.4,4));  
 enemies.add(new Enemy(new Point(6,16), new Point(6,18),0.5,4));  
  
 */\*\*  
 \* 5 lvl  
 \*/* enemies.add(new Enemy(new Point(8,4 ), new Point(10,4),0.4,5));  
 //enemies.add(new Enemy(enemyW,enemyH,new Point(24,6 ), new Point(26,6),0.4,5));  
 enemies.add(new Enemy(new Point(6,10 ), new Point(6,14),0.4,5));  
 enemies.add(new Enemy(new Point(12,12 ), new Point(16,12),0.5,5));  
 enemies.add(new Enemy(new Point(12,18 ), new Point(18,18),0.5,5));  
  
 */\*\*  
 \* 6 lvl  
 \*/* enemies.add(new Enemy(new Point(14,6 ), new Point(18,6),0.4,6));  
 enemies.add(new Enemy(new Point(14,8 ), new Point(14,10),0.2,6));  
 enemies.add(new Enemy(new Point(24,8), new Point(24,14),0.4,6));  
 enemies.add(new Enemy(new Point(6,12), new Point(6,16),0.4,6));  
 enemies.add(new Enemy(new Point(10,16), new Point(10,20),0.2,6));  
 enemies.add(new Enemy(new Point(14,20), new Point(16,20),0.2,6));  
 enemies.add(new Enemy(new Point(26,18), new Point(26,22),0.5,6));  
  
 }  
  
 */\*\*  
 \* Add enemies to the game map  
 \*/* private void enemyOnMap() {  
 sf = new HashMap<Point, Point>();  
  
  
 for (Enemy enemy : enemies) {  
 enemy.setWidth(enemySize);  
 enemy.setHeight(enemySize);  
 enemy.setEnemyRadiusAndCenter();  
 }  
 for (Enemy enemy : enemies) {  
 if(enemy.getLvl()==gameLevel)  
 sf.put(enemy.getStart(),enemy.getFinish());  
 }  
 System.*out*.println("Level "+gameLevel);  
  
 for(int i=0;i<bricks.size();i++){  
 for(int k=0;k<bricks.get(0).length();k++){  
 if(i==0 || i==bricks.size()-1 || k==0 || k ==bricks.get(0).length()-1)  
 map[i][k]=9;  
 }  
 }  
 }  
  
 */\*\*  
 \* Change the coordinates of enemies trajectories from the map representing to frame representing  
 \*/* private void addEnemyToMaze() {  
  
 int currentBRICK\_SIZE\_Y = *BRICK\_SIZE*;  
 int currentBRICK\_SIZE\_X = *BRICK\_SIZE*;  
 int height = calculateMazeLength(this.bricks.size());  
 int length = calculateMazeLength(this.bricks.get(0).length());  
  
 int y = *BORDER*+*BRICK\_SIZE*\**outsideWallCoef*;  
 int x = 0;  
 boolean evenLine = false;  
 boolean evenColumn = false;  
  
 int j = 0;  
 for (String wall: this.bricks){  
 if (this.bricks.indexOf(wall)>0 && this.bricks.indexOf(wall) < this.bricks.size()-1) {  
  
 char[] symbolsArray = new char[wall.toCharArray().length];  
 symbolsArray = wall.toCharArray();  
  
 if(evenLine) { currentBRICK\_SIZE\_Y = *BRICK\_SIZE*;}  
 else {currentBRICK\_SIZE\_Y = *coefficientCorridor* \* *BRICK\_SIZE*;}  
 evenLine = ! evenLine;  
  
 x = *BORDER* + *BRICK\_SIZE* \* *outsideWallCoef*;  
 evenColumn = false;  
  
 for (int i = 1; i < symbolsArray.length - 1; i++) {  
 if (evenColumn) {currentBRICK\_SIZE\_X = *BRICK\_SIZE*;}  
 else { currentBRICK\_SIZE\_X = *coefficientCorridor* \* *BRICK\_SIZE*;}  
 evenColumn = ! evenColumn;  
 int enemy\_x = x+(currentBRICK\_SIZE\_X/ pictureShift)/2;  
 int enemy\_y = y+(currentBRICK\_SIZE\_Y/ pictureShift)/2;  
 Point test = new Point(i+1,j+1);  
  
 if (sf.containsKey(test)) {  
 //System.out.println(x+" "+y);  
 for (Enemy enemy : enemies) {  
 if(enemy.getStart().equals(test)){  
 enemy.setStart(new Point(enemy\_x, enemy\_y));  
  
 }  
 }  
 }  
  
 if (sf.containsValue(test)) {  
 //System.out.println(test+" "+x+" "+y);  
  
 for (Enemy enemy : enemies) {  
 if(enemy.getFinish().equals(test)){  
 enemy.setFinish(new Point(x+enemySize, y+enemySize));  
 }  
 }  
 }  
 x += currentBRICK\_SIZE\_X;  
 }  
 y += currentBRICK\_SIZE\_Y;  
 }  
 j++;  
 }  
 }  
 */\*\*If player go through all levels and win, this method works.  
 \* Change game status in boolean and write new information to file \*/* private void gameOver(){  
 // System.out.println("gameOver");  
 inGame=false;  
 gameFinished = true;  
 writeToFileGameStatus(0 + "|" + defaultHeroLives);  
 *lev6*.stop();  
 }  
  
 @Override  
 */\*\*Override method that checks when action is performed  
 \** ***@param*** *e -- An event which indicates that some actions were done\*/* public void actionPerformed(ActionEvent e) {  
 canMove = true;  
  
 for (Enemy enemy : enemies) {  
 if(enemy.isVisible() && inGame == true )  
 enemy.move(mazeWidth,mazeHeight);  
 }  
  
 if(isWall()){  
 canMove = false;  
 }  
 if(isEnemy()&& inGame == true){  
 intoMonster.play();  
 minusLive();  
 // StartPage.userPanel();  
 canMove = false;  
 }  
 if(canMove){  
 myHero.changeHero\_x();  
 myHero.changeHero\_y();  
 }  
 repaint();  
 }  
 */\*\*Decrease hero lives by 1 and put hero on another location.  
 \* If there is no lives, then it is hero death.  
 \* Write new information to file\*/* private void minusLive(){  
 // System.out.println("minusLife");  
 if(heroLives>1){  
 heroLives--;  
 writeToFileGameStatus((gameLevel-1) + "|" + heroLives);  
 // System.out.println(heroLives);  
 myHero.setHero\_x(*BORDER* + *BRICK\_SIZE* \* *outsideWallCoef* + 1);  
 myHero.setHero\_y(*BORDER* + *BRICK\_SIZE* \* *outsideWallCoef* + 1);  
 } else if(heroLives == 1){  
 dying = true;  
 writeToFileGameStatus((gameLevel-1) + "|" + defaultHeroLives);  
 }  
 }  
 */\*\*Checks if there is intersection between hero and heart by one point of hero picture  
 \** ***@param*** *angle - angle which indicates the point on the border of hero picture to be checked  
 \** ***@return*** *A boolean representing if there is intersection between hero and heart\*/* private boolean isIntersectionBetweenHeroAndHeart(int angle){  
 // System.out.println("isIntersectionBetweenHeroAndHeart");  
 int x\_center = myHero.getHeroImage().getWidth(null)/2;  
 int y\_center = myHero.getHeroImage().getHeight(null)/2;  
  
 double rX = myHero.getHeroImage().getWidth(null)\*cosOfAngle[angle]/2;  
 double rY = myHero.getHeroImage().getHeight(null)\*sinOfAngle[angle]/2;  
  
 int deltX = x\_center + (int)(rX);  
 int deltY = y\_center + (int)(rY);  
  
 int x0 = deltX+(int)myHero.getHero\_x();  
 int y0 = deltY+(int)myHero.getHero\_y();  
  
 if( heart1.isShow() ==true && x0>=heart1.getX() && x0<=heart1.getX()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && y0>=heart1.getY() && y0<=heart1.getY()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 heart1.setShow(false);  
  
 return true;  
 }  
 if(heart2.isShow() ==true && x0>=heart2.getX() && x0<=heart2.getX()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && y0>=heart2.getY() && y0<=heart2.getY()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 heart2.setShow(false);  
  
 return true;  
 }  
  
 return false;  
 }  
 */\*\*Checks if there is intersection between hero and heart by diagonals  
 \** ***@return*** *A boolean representing if there is intersection between hero and heart by diagonals\*/* private boolean isIntersectionWithHeartByDiagonal(){  
 // System.out.println("isIntersectionWithHeartByDiagonal");  
 /\*horizontal diagonal\*/  
 int xFirst = (int)myHero.getHero\_x();  
 int xLast = (int)myHero.getHero\_x() + myHero.getHeroImage().getWidth(null);  
 int yDiagonal = (int)myHero.getHero\_y()+myHero.getHeroImage().getHeight(null)/2;  
  
 for(int x=xFirst; x <=xLast; x++){  
 if(heart1.isShow() ==true && x>=heart1.getX() && x<=heart1.getX()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && yDiagonal>=heart1.getY() && yDiagonal<=heart1.getY()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 heart1.setShow(false);  
  
 return true;  
  
 }  
 if(heart2.isShow() ==true && x>=heart2.getX() && x<=heart2.getX()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && yDiagonal>=heart2.getY() && yDiagonal<=heart2.getY()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 heart2.setShow(false);  
  
 return true;  
 }  
 }  
  
 /\*vertical diagonal\*/  
 int yFirst = (int)myHero.getHero\_y();  
 int yLast = (int)myHero.getHero\_y() + myHero.getHeroImage().getHeight(null);  
 int xDiagonal = (int)myHero.getHero\_x()+myHero.getHeroImage().getWidth(null)/2;  
  
 for(int y=yFirst; y <=yLast; y++){  
 if(heart1.isShow() ==true && xDiagonal>=heart1.getX() && xDiagonal<=heart1.getX()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && y>=heart1.getY() && y<=heart1.getY()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 heart1.setShow(false);  
 return true;  
 }  
 if(heart2.isShow() ==true && xDiagonal>=heart2.getX() && xDiagonal<=heart2.getX()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && y>=heart2.getY() && y<=heart2.getY()+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 heart2.setShow(false);  
 return true;  
 }  
 }  
  
 return false;  
 }  
 */\*\*Checks if there is intersection between hero and heart  
 \** ***@return*** *A boolean representing if there is intersection between hero and heart\*/* public boolean isHeart(){  
 // System.out.println("isHeart");  
 for (int i = 0; i < numberOfAngles; i++) {  
 if(isIntersectionBetweenHeroAndHeart(i) == true){  
 return true;  
 }  
 }  
  
 if(isIntersectionWithHeartByDiagonal()==true){  
 return true;  
 }  
  
 return false;  
 }  
 */\*\*Checks if there is intersection between hero and portal by one point of hero picture  
 \** ***@param*** *angle - angle which indicates the point on the border of hero picture to be checked  
 \** ***@return*** *A boolean representing if there is intersection between hero and portal\*/* private boolean isIntersectionBetweenHeroAndPortal(int angle){  
 // System.out.println("isIntersectionBetweenHeroAndPortal");  
 int x\_center = myHero.getHeroImage().getWidth(null)/2;  
 int y\_center = myHero.getHeroImage().getHeight(null)/2;  
  
 double rX = myHero.getHeroImage().getWidth(null)\*cosOfAngle[angle]/2;  
 double rY = myHero.getHeroImage().getHeight(null)\*sinOfAngle[angle]/2;  
  
 int deltX = x\_center + (int)(rX);  
 int deltY = y\_center + (int)(rY);  
  
 int x0 = deltX+(int)myHero.getHero\_x();  
 int y0 = deltY+(int)myHero.getHero\_y();  
  
 if(x0>=portal\_x && x0<=portal\_x+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && y0>=portal\_y && y0<=portal\_y+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 return true;  
 }  
  
 return false;  
 }  
 */\*\*Checks if there is intersection between hero and portal by diagonals  
 \** ***@return*** *A boolean representing if there is intersection between hero and portal by diagonals\*/* private boolean isIntersectionWithPortalByDiagonal(){  
 // System.out.println("isIntersectionWithPortalByDiagonal");  
 /\*horizontal diagonal\*/  
 int xFirst = (int)myHero.getHero\_x();  
 int xLast = (int)myHero.getHero\_x() + myHero.getHeroImage().getWidth(null);  
 int yDiagonal = (int)myHero.getHero\_y()+myHero.getHeroImage().getHeight(null)/2;  
  
 for(int x=xFirst; x <=xLast; x++){  
 if(x>=portal\_x && x<=portal\_x+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && yDiagonal>=portal\_y && yDiagonal<=portal\_y+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 return true;  
 }  
 }  
  
 /\*vertical diagonal\*/  
 int yFirst = (int)myHero.getHero\_y();  
 int yLast = (int)myHero.getHero\_y() + myHero.getHeroImage().getHeight(null);  
 int xDiagonal = (int)myHero.getHero\_x()+myHero.getHeroImage().getWidth(null)/2;  
  
 for(int y=yFirst; y <=yLast; y++){  
 if(xDiagonal>=portal\_x && xDiagonal<=portal\_x+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift  
 && y>=portal\_y && y<=portal\_y+*BRICK\_SIZE*\**coefficientCorridor*\* pictureLengthInCell / pictureShift){  
 return true;  
 }  
 }  
  
 return false;  
 }  
 */\*\*Checks if there is intersection between hero and portal  
 \** ***@return*** *A boolean representing if there is intersection between hero and portal\*/* public boolean isPortal(){  
 // System.out.println("isPortal");  
 for (int i = 0; i < numberOfAngles; i++) {  
 if(isIntersectionBetweenHeroAndPortal(i) == true){  
 writeToFileGameStatus(gameLevel + "|" + heroLives);  
 //System.out.println("isPortal true -- next level");  
 return true;  
 }  
 }  
  
 if(isIntersectionWithPortalByDiagonal()==true){  
 return true;  
 }  
  
 return false;  
 }  
 */\*\*Gets arraylist of enemies from current level  
 \** ***@return*** *An arraylist representing all enemies from current level\*/* private ArrayList<Enemy> getEnemiesFromCurrentLevel(){  
 // System.out.println("getEnemiesFromCurrentLevel");  
 ArrayList<Enemy> enemiesFromLevel = new ArrayList<>();  
 for(Enemy oneEnemy: enemies){  
 if(oneEnemy.getLvl() == gameLevel){  
 enemiesFromLevel.add(oneEnemy);  
 }  
 }  
 return enemiesFromLevel;  
 }  
 */\*\*Checks if there is intersection between hero and enemy  
 \** ***@return*** *A boolean representing if there is intersection between hero and enemy\*/* public boolean isEnemy(){  
 for(Enemy theEnemy : enemiesFromCurrentLevel){  
 if(getDistance(myHero.getHeroCenter\_x()+(int)myHero.getHero\_x(), myHero.getHeroCenter\_y()+(int)myHero.getHero\_y(),  
 theEnemy.getEnemyCenter\_x()+(int)theEnemy.getX(), theEnemy.getEnemyCenter\_y()+(int)theEnemy.getY())<=myHero.getHeroRadius()+theEnemy.getEnemyRadius()){  
 return true;  
 }  
 }  
 return false;  
 }  
 */\*\*Checks if there is intersection between hero and wall  
 \** ***@return*** *A boolean representing if there is intersection between hero and wall\*/* private boolean isWall(){  
 /\*Checking with outside walls\*/  
 if(myHero.getHero\_x()+myHero.getDx()<=*BORDER*+*BRICK\_SIZE*\**outsideWallCoef* || myHero.getHero\_y()+myHero.getDy()<=*BORDER*+*BRICK\_SIZE*\**outsideWallCoef*){  
 return true;  
 }  
 int height = calculateMazeLength(this.bricks.size());  
 int length = calculateMazeLength(this.bricks.get(0).length());  
 if(myHero.getHero\_x()+myHero.getHeroImage().getWidth(null)+myHero.getDx() >= *BORDER*+length-*BRICK\_SIZE*\**outsideWallCoef* ||  
 myHero.getHero\_y()+myHero.getHeroImage().getHeight(null)+myHero.getDy() >= *BORDER*+height-*BRICK\_SIZE*\**outsideWallCoef*){  
 return true;  
 }  
 if (cheat == 1) {return false;}  
 /\*Checking with inner walls\*/  
 for (int i = 0; i < numberOfAngles; i++) {  
 if(isInnerWallForEllipse(i) == 1)  
 {  
 return true;  
 }  
 }  
  
 return false;  
 }  
 */\*\*Checks if there is intersection between hero and inner wall  
 \** ***@param*** *angle - angle which indicates the point on the border of hero picture to be checked  
 \** ***@return*** *An integer representing whether the cell, where hero is located, is inner wall or corridor\*/* private int isInnerWallForEllipse(int angle){  
  
 int x\_center = myHero.getHeroImage().getWidth(null)/2;  
 int y\_center = myHero.getHeroImage().getHeight(null)/2;  
  
 double rX = myHero.getHeroImage().getWidth(null)\*cosOfAngle[angle]/2;  
 double rY = myHero.getHeroImage().getHeight(null)\*sinOfAngle[angle]/2;  
  
 int deltX = x\_center + (int)(rX);  
 int deltY = y\_center + (int)(rY);  
  
 //hero\_x - реальная координата персонажа в окне  
 //hero\_x-BORDER-BRICK\_SIZE\*outsideWallCoef - координата персонажа относительно начала лабиринта  
  
 int x0 = (int)(myHero.getDx()+deltX+myHero.getHero\_x()-*BORDER*-*BRICK\_SIZE*\**outsideWallCoef*);  
 int y0 = (int)(myHero.getDy()+deltY+myHero.getHero\_y()-*BORDER*-*BRICK\_SIZE*\**outsideWallCoef*);  
 int sX = (int)(x0/(*BRICK\_SIZE*+*coefficientCorridor* \* *BRICK\_SIZE*));  
 int sY = (int)(y0/(*BRICK\_SIZE*+*coefficientCorridor* \* *BRICK\_SIZE*));  
 int x1 =0;  
 int y1 =0;  
  
 double restX = x0-sX\*(*BRICK\_SIZE*+*coefficientCorridor* \* *BRICK\_SIZE*);  
 if(restX<=*coefficientCorridor* \* *BRICK\_SIZE*)  
 {x1 = sX\*2+1;}else{ x1 = sX\*2+2;}  
  
 double restY = y0-sY\*(*BRICK\_SIZE*+*coefficientCorridor* \* *BRICK\_SIZE*);  
 if(restY<=*coefficientCorridor* \* *BRICK\_SIZE*)  
 {y1 = sY\*2+1;}else{y1 = sY\*2+2;}  
 return map[y1][x1];  
 }  
 */\*\*Draw maze and elements on it using the maze map  
 \** ***@param*** *g2d - A Graphics2D used to drawing maze and other elements on jpanel\*/* private void drawMaze(Graphics2D g2d) {  
 g2d.setPaint(Color.*BLACK*);  
 g2d.fillRect(0, 0, getWidth(), getHeight());  
  
 int currentBRICK\_SIZE\_Y = *BRICK\_SIZE*;  
 int currentBRICK\_SIZE\_X = *BRICK\_SIZE*;  
 int height = calculateMazeLength(this.bricks.size());  
 int length = calculateMazeLength(this.bricks.get(0).length());  
 g2d.setColor(wallColor);  
 g2d.fillRect(*BORDER*, *BORDER*, length, *BRICK\_SIZE*\**outsideWallCoef*);  
 g2d.fillRect(*BORDER*, *BORDER*, *BRICK\_SIZE*\**outsideWallCoef*, height);  
 g2d.fillRect(*BORDER*, *BORDER*+height-*BRICK\_SIZE*\**outsideWallCoef*, length, *BRICK\_SIZE*\**outsideWallCoef*);  
 g2d.fillRect(*BORDER*+length-*BRICK\_SIZE*\**outsideWallCoef*, *BORDER*, *BRICK\_SIZE*\**outsideWallCoef*, height);  
 int y = *BORDER*+*BRICK\_SIZE*\**outsideWallCoef*;  
 int x = 0;  
 boolean evenLine = false;  
 boolean evenColumn = false;  
  
// int enemyCount=0;  
// Point startPoint=new Point (0,0),finishPoint = new Point(0,0);  
  
 int heartCount = 0;  
 int j = 0;  
 for (String wall: this.bricks){  
 if (this.bricks.indexOf(wall)>0 && this.bricks.indexOf(wall) < this.bricks.size()-1) {  
  
 char[] symbolsArray = new char[wall.toCharArray().length];  
 symbolsArray = wall.toCharArray();  
  
 if(evenLine) { currentBRICK\_SIZE\_Y = *BRICK\_SIZE*;}  
 else {currentBRICK\_SIZE\_Y = *coefficientCorridor* \* *BRICK\_SIZE*;}  
 evenLine = ! evenLine;  
  
 x = *BORDER* + *BRICK\_SIZE* \* *outsideWallCoef*;  
 evenColumn = false;  
  
 for (int i = 1; i < symbolsArray.length - 1; i++) {  
 if (evenColumn) {currentBRICK\_SIZE\_X = *BRICK\_SIZE*;}  
 else { currentBRICK\_SIZE\_X = *coefficientCorridor* \* *BRICK\_SIZE*;}  
 evenColumn = ! evenColumn;  
 /\*Draw walls\*/  
 if (symbolsArray[i] == '#') {  
 g2d.setColor(wallColor);  
 g2d.fillRect(x, y, currentBRICK\_SIZE\_X, currentBRICK\_SIZE\_Y);  
 map[j][i] = 1;  
 }  
 /\*Draw corridors\*/  
 if (symbolsArray[i] == '\*') {  
 g2d.setColor(corridorColor);  
 g2d.fillRect(x, y, currentBRICK\_SIZE\_X, currentBRICK\_SIZE\_Y);  
 map[j][i] = 0;  
 }  
 /\*Draw corridor where enemy will be\*/  
 if (symbolsArray[i] == 'E') {  
 g2d.setColor(corridorColor);  
 //g2d.setColor(enemyColor);  
 g2d.fillRect(x, y, currentBRICK\_SIZE\_X, currentBRICK\_SIZE\_Y);  
  
 map[j][i] = 7;  
 }  
 /\*Draw portal\*/  
 if (symbolsArray[i] == 'P') {  
 g2d.setColor(corridorColor);  
 g2d.fillRect(x, y, currentBRICK\_SIZE\_X, currentBRICK\_SIZE\_Y);  
 int picturePortShift = 7;  
 portal\_x = x+(currentBRICK\_SIZE\_X/ picturePortShift)/2;  
 portal\_y = y+(currentBRICK\_SIZE\_Y/ picturePortShift)/2;  
 int pictureLength = picturePortShift-1;  
 g2d.drawImage(portal,portal\_x,portal\_y, pictureLength\*currentBRICK\_SIZE\_X/picturePortShift, pictureLength\*currentBRICK\_SIZE\_Y/picturePortShift,this);  
  
 Color myColour = new Color(corridorColor.getRed(), corridorColor.getGreen(), corridorColor.getBlue(), alpha);  
 g2d.setColor(myColour);  
 g2d.fillRect(x, y, currentBRICK\_SIZE\_X, currentBRICK\_SIZE\_Y);  
 map[j][i] = 2;  
 }  
 /\*Draw hearts\*/  
 if (symbolsArray[i] == 'H') {  
 g2d.setColor(corridorColor);  
 g2d.fillRect(x, y, currentBRICK\_SIZE\_X, currentBRICK\_SIZE\_Y);  
  
 heartCount++;  
  
 if((heart1.isShow()==false && heartCount==1) || (heart2.isShow()==false && heartCount==2)){  
 map[j][i]=0;  
 }  
 if((heart1.isShow()==true && heartCount==1) || (heart2.isShow()==true && heartCount==2)){  
 map[j][i]=3;  
 }  
  
  
 if(heart1.isShow()==true && heartCount==1){  
 heart1.setX(x+currentBRICK\_SIZE\_X/ pictureShift);  
 heart1.setY(y+currentBRICK\_SIZE\_Y/ pictureShift);  
 g2d.drawImage(heart,heart1.getX(),heart1.getY(), pictureLengthInCell \*currentBRICK\_SIZE\_X/ pictureShift, pictureLengthInCell \*currentBRICK\_SIZE\_Y/ pictureShift,this);  
 heart1.setxMap(i);  
 heart1.setyMap(j);  
 }  
  
 if(heart2.isShow()==true && heartCount==2){  
 heart2.setX(x+currentBRICK\_SIZE\_X/ pictureShift);  
 heart2.setY(y+currentBRICK\_SIZE\_Y/ pictureShift);  
 g2d.drawImage(heart,heart2.getX(),heart2.getY(), pictureLengthInCell \*currentBRICK\_SIZE\_X/ pictureShift, pictureLengthInCell \*currentBRICK\_SIZE\_Y/ pictureShift,this);  
 heart2.setxMap(i);  
 heart2.setyMap(j);  
 }  
  
 }  
 x += currentBRICK\_SIZE\_X;  
 }  
 y += currentBRICK\_SIZE\_Y;  
 }  
 j++;  
 }  
 }  
  
 @Override  
 */\*\*Paints all components on jpanel  
 \** ***@param*** *g - A Graphics used to drawing on jpanel\*/* public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 doDrawing(g);  
 }  
 */\*\*Draws all components in different methods  
 \** ***@param*** *g - A Graphics used to drawing on jpanel\*/* private void doDrawing(Graphics g) {  
  
 Graphics2D g2d = (Graphics2D) g;  
 g2d.setColor(Color.*WHITE*);  
 g2d.fillRect(0, 0, getWidth(), getHeight());  
  
 if(alpha >= 254) { alphaSign = -1;}  
 if(alpha <= 1) {alphaSign = 1;}  
 alpha = (int)(alpha + 2\*alphaSign);  
 drawMaze(g2d);  
  
 // drawParameters(g2d);  
 drawLives(g2d);  
  
 if (inGame) {  
 playGame(g2d);  
 } else {  
  
 }  
 if(dying == true){  
 TAdapter.*startPage*.endPageLoser();  
 }  
  
 Toolkit.*getDefaultToolkit*().sync();  
 g2d.dispose();  
 }  
 */\*\*Draws number of hero lives and hero speed  
 \** ***@param*** *g2d - A Graphics2D used to drawing number of hero lives and hero speed on jpanel\*/* private void drawLives(Graphics2D g2d){  
 g2d.setColor(Color.*white*);  
 Font myFont = new Font("Calibri", Font.*PLAIN*, 24);  
 g2d.setFont(myFont);  
 g2d.drawString("LEVEL: " + gameLevel, 820, 40);  
  
 g2d.drawImage(heart,825,100,30,30,Color.*black*,this);  
 g2d.drawString(" x " + heroLives, 860,120);  
  
  
 }  
 public void userPanel(){  
 JLabel lev = new JLabel("LEVEL: " + gameLevel);  
 lev.setBounds(10,90,100,30);  
 lev.setFont(new Font("Times New Roman", Font.*PLAIN*,20));  
 lev.setForeground(Color.*white*);  
 // panel1.add(lev);  
  
 ImageIcon imageIcon = new ImageIcon("src/images/other/heart\_red\_s.png"); // load the image to a imageIcon  
 Image image = imageIcon.getImage(); // transform it  
 Image newimg = image.getScaledInstance(40, 40, java.awt.Image.*SCALE\_SMOOTH*); // scale it the smooth way  
 imageIcon = new ImageIcon(newimg); // transform it back  
 JLabel img1 = new JLabel(imageIcon);  
 img1.setBounds(10,150,40,40);  
 // panel1.add(img1);  
  
 JLabel heartChecker = new JLabel(" x " + getHeroLives());  
 heartChecker.setBounds(55,150,100,40);  
 heartChecker.setForeground(Color.*white*);  
 heartChecker.setFont(new Font("Times New Roman", Font.*PLAIN*,20));  
 // panel1.add(heartChecker);  
  
 }  
 */\*\*Draws hero location and displacement  
 \** ***@param*** *g2d - A Graphics2D used to drawing hero location and displacement on jpanel\*/* private void drawParameters(Graphics2D g2d){  
 g2d.setColor(Color.*YELLOW*);  
 Font myFont = new Font("Calibri", Font.*BOLD*, 24);  
 g2d.setFont(myFont);  
 g2d.drawString("hero\_x = " + myHero.getHero\_x(), 40, 200);  
 g2d.drawString("hero\_y = " + myHero.getHero\_y(), 40, 220);  
 // g2d.drawString("hero\_DX = " + myHero.getDx(), 40, 240);  
 // g2d.drawString("hero\_DY = " + myHero.getDy(), 40, 260);  
 // g2d.drawString("hero\_Radius = " + myHero.getDy(), 40, 260);  
 // g2d.drawString("hero\_DY = " + myHero.getDy(), 40, 260);  
 int i = 1;  
 int k = 200;  
 for(Enemy theEnemy : enemiesFromCurrentLevel) {  
 g2d.setColor(Color.*WHITE*);  
 g2d.drawString("Enemy radius = "+theEnemy.getEnemyRadius(), 550, 40);  
 g2d.drawString("Enemy center x = "+theEnemy.getEnemyCenter\_x(), 550, 60);  
 g2d.drawString("Enemy center y = "+theEnemy.getEnemyCenter\_y(), 550, 80);  
 g2d.drawString("Hero radius = "+myHero.getHeroRadius(), 550, 100);  
 g2d.drawString("Hero center x = "+myHero.getHeroCenter\_x(), 550, 120);  
 g2d.drawString("Hero center y= "+myHero.getHeroCenter\_y(), 550, 140);  
  
 g2d.drawString("Enemy x = "+(int)theEnemy.getX(), 400, k);  
 g2d.drawString("Enemy y = "+(int)theEnemy.getY(), 400, (k+20));  
  
 g2d.setColor(Color.*YELLOW*);  
 g2d.setFont(myFont);  
 g2d.drawString(i + " radius h + e " + (myHero.getHeroRadius() + theEnemy.getEnemyRadius()), 550, k);  
 g2d.drawString(i + " distance h and e " + (getDistance(myHero.getHeroCenter\_x() + (int) myHero.getHero\_x(), myHero.getHeroCenter\_y() + (int) myHero.getHero\_y(),  
 theEnemy.getEnemyCenter\_x() + (int) theEnemy.getX(), theEnemy.getEnemyCenter\_y() + (int) theEnemy.getY())), 550, (k + 20));  
 i++;  
 k = k + 50;  
 }  
 }  
 */\*\*Draws hero and enemies  
 \** ***@param*** *g2d - A Graphics2D used to drawing hero and enemies on jpanel\*/* private void playGame(Graphics2D g2d) {  
  
 if (dying) {  
  
 } else {  
 drawHero(g2d);  
 drawEnemy(g2d);  
 }  
 }  
 */\*\*Draws hero  
 \** ***@param*** *g2d - A Graphics2D used to drawing hero on jpanel\*/* private void drawHero(Graphics2D g2d) {  
 g2d.drawImage(myHero.getHeroImage(), (int)myHero.getHero\_x(), (int)myHero.getHero\_y(), this);  
 }  
 */\*\*Draws enemies  
 \** ***@param*** *g2d - A Graphics2D used to drawing enemies on jpanel\*/* private void drawEnemy(Graphics2D g2d) {  
 for (Enemy enemy : enemies) {  
 if(enemy.isVisible() && inGame == true && enemy.getLvl() == gameLevel )  
 g2d.drawImage(enemy.getEnemyImg(),(int)enemy.getX(),(int)enemy.getY(),this);  
 enemy.setVisible(true);  
 }  
 }  
 */\*\*Calculate maze length  
 \** ***@param*** *numberOfColons - is number of colons in maze  
 \** ***@return*** *An integer representing maze length\*/* public int calculateMazeLength(int numberOfColons){  
 int mazeLength = 2\**BRICK\_SIZE*\**outsideWallCoef* + (numberOfColons-3)/2\**BRICK\_SIZE*+((numberOfColons-3)/2+1)\**BRICK\_SIZE*\**coefficientCorridor*;  
 return mazeLength;  
 }  
 */\*\*****@return*** *An integer representing panel width\*/* public int getPanelWidth(){  
 return this.mazeWidth+2\**BORDER*;  
 }  
 */\*\*****@return*** *An integer representing panel height\*/* public int getPanelHeight(){  
 return this.mazeHeight+2\**BORDER*;  
 }  
 */\*\*****@return*** *An integer representing panel border\*/* public int getPanelBorder(){  
 return this.*BORDER*;  
 }  
 */\*\*Set Y displacement for hero  
 \** ***@param*** *req\_dy - Y displacement of hero\*/* public void setReq\_dy(int req\_dy) {  
 this.myHero.setDy(req\_dy);  
 }  
 */\*\*Set X displacement for hero  
 \** ***@param*** *req\_dx - X displacement of hero\*/* public void setReq\_dx(int req\_dx) {  
 this.myHero.setDx(req\_dx);  
 }  
}

Sound

package com.rgb;  
  
import javax.sound.sampled.\*;  
import java.io.File;  
import java.io.IOException;  
  
*/\*\* class for creating music  
 \*  
 \*/*public class Sound implements AutoCloseable {  
 private boolean released = false;  
 private AudioInputStream stream = null;  
 private Clip clip = null;  
 private FloatControl volumeControl = null;  
 private boolean playing = false;  
  
 */\*\* constructor  
 \*  
 \*/* public Sound(File f) {  
 try {  
 stream = AudioSystem.*getAudioInputStream*(f);  
 clip = AudioSystem.*getClip*();  
 clip.open(stream);  
 clip.addLineListener(new Listener());  
 volumeControl = (FloatControl) clip.getControl(FloatControl.Type.*MASTER\_GAIN*);  
 released = true;  
 } catch (IOException | UnsupportedAudioFileException | LineUnavailableException exc) {  
 exc.printStackTrace();  
 released = false;  
  
 close();  
 }  
 }  
  
 */\*\* true - if track was uploaded  
 \* false - if it was't  
 \*/* public boolean isReleased() {  
 return released;  
 }  
  
 */\*\* check for playing  
 \*  
 \*/* public boolean isPlaying() {  
 return playing;  
 }  
  
  
 */\*\* breakOld check if treck is playing, if not - start play  
 \*  
 \*/* public void play(boolean breakOld) {  
 if (released) {  
 if (breakOld) {  
 clip.stop();  
 clip.setFramePosition(0);  
 clip.start();  
 playing = true;  
 } else if (!isPlaying()) {  
 clip.setFramePosition(0);  
 clip.start();  
 playing = true;  
 }  
 }  
 }  
  
 */\*\* play sound  
 \*  
 \*/* public void play() {  
 play(true);  
 }  
  
 */\*\* stop clip  
 \*  
 \*/* public void stop() {  
 if (playing) {  
 clip.stop();  
 }  
 }  
  
 */\*\* close clip  
 \*  
 \*/* public void close() {  
 if (clip != null)  
 clip.close();  
  
 if (stream != null)  
 try {  
 stream.close();  
 } catch (IOException exc) {  
 exc.printStackTrace();  
 }  
 }  
  
 */\*\* set volume(from 1 to 0)  
 \*  
 \*/* public void setVolume(float x) {  
 if (x<0) x = 0;  
 if (x>1) x = 1;  
 float min = volumeControl.getMinimum();  
 float max = volumeControl.getMaximum();  
 volumeControl.setValue((max-min)\*x+min);  
 }  
  
 */\*\* getter for volume (from 0 to 1)  
 \*  
 \*/* public float getVolume() {  
 float v = volumeControl.getValue();  
 float min = volumeControl.getMinimum();  
 float max = volumeControl.getMaximum();  
 return (v-min)/(max-min);  
 }  
  
 */\*\* wait for ending of the clip  
 \*  
 \*/* public void join() {  
 if (!released) return;  
 synchronized(clip) {  
 try {  
 while (playing)  
 clip.wait();  
 } catch (InterruptedException exc) {}  
 }  
 }  
  
 */\*\* static method for play  
 \*  
 \*/* public static Sound playSound(String path) {  
 File f = new File(path);  
 Sound snd = new Sound(f);  
 snd.play();  
 return snd;  
 }  
 */\*\* static method that stop clip  
 \*  
 \*/* public static Sound stop(String path) {  
 File f = new File(path);  
 Sound snd = new Sound(f);  
 snd.stop();  
 return snd;  
 }  
 */\*\* class listener  
 \*  
 \*/* private class Listener implements LineListener {  
 public void update(LineEvent ev) {  
 if (ev.getType() == LineEvent.Type.*STOP*) {  
 playing = false;  
 synchronized(clip) {  
 clip.notify();  
 }  
 }  
 }  
 }  
}

StartPage

package com.rgb;  
  
import javax.swing.\*;  
import javax.swing.text.SimpleAttributeSet;  
import javax.swing.text.StyleConstants;  
import javax.swing.text.StyledDocument;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import java.io.\*;  
import java.util.ArrayList;  
  
  
*/\*\*  
 \* Created by Shevchenko Daria on 22.05.2020.  
 \*/*public class StartPage extends JFrame{  
  
 //Sound panel;  
  
 static Sound *startMusic* = new Sound(new File("src/music/start.wav"));  
  
 JPanel panel1;  
 Font titleFont = new Font("Times New Roman", Font.*PLAIN*,90);  
 Font normalFont = new Font("Times New Roman", Font.*PLAIN*,30);  
  
 private final int width = 1050;  
 private final int height = 700;  
  
 private ArrayList<ArrayList> bricksLevels = new ArrayList<ArrayList>();  
  
 public Maze panelWithMaze;  
  
 private int lives = 0;  
  
 StartPage(){}  
  
 StartPage(String title){  
 super(title);  
 // System.out.println("StartPage - it is Constructor");  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 this.addWindowListener(new WindowAdapter() {  
 @Override  
 public void windowClosed(WindowEvent e) {  
 panelWithMaze.setInGame(false);  
 panelWithMaze.setHeroLives(panelWithMaze.getHeroLivesOnLevelStart());  
 }  
  
 @Override  
 public void windowClosing(WindowEvent e) {  
 panelWithMaze.setInGame(false);  
 panelWithMaze.setHeroLives(panelWithMaze.getHeroLivesOnLevelStart());  
 }  
 });  
 this.setSize(width,height);  
  
  
 bricksLevels.add(*read*("src/mazeFiles/maze\_level\_1.txt"));  
 bricksLevels.add(*read*("src/mazeFiles/maze\_level\_2.txt"));  
 bricksLevels.add(*read*("src/mazeFiles/maze\_level\_3.txt"));  
 bricksLevels.add(*read*("src/mazeFiles/maze\_level\_4.txt"));  
 bricksLevels.add(*read*("src/mazeFiles/maze\_level\_5.txt"));  
 bricksLevels.add(*read*("src/mazeFiles/maze\_level\_6.txt"));  
 start();  
  
  
 }  
  
  
 private static ArrayList<String> read(String filename){  
 ArrayList<String> lines = new ArrayList<String>();  
 try{  
 FileInputStream fstream = new FileInputStream(filename);  
 DataInputStream in = new DataInputStream(fstream);  
 BufferedReader br = new BufferedReader(new InputStreamReader(in));  
 String line;  
 while((line = br.readLine()) != null){  
 lines.add(line);  
 }  
 in.close();  
 } catch (Exception e){  
 System.*err*.println("Error: "+ e.getMessage());  
 }  
  
 return lines;  
 }  
  
 */\*\*  
 \* start program  
 \*/* public void start(){  
 // System.out.println("start");  
 // remove(panelWithMaze);  
 panelWithMaze = null;  
 if (panel1!=null) {  
 remove(panel1);  
 panel1=null;  
 }  
 setPreferredSize(new Dimension(width,height));  
 reCreateWindow();  
 startPage();  
 add(panel1);  
  
 this.pack();  
 panel1.setVisible(true);  
 this.setVisible(true);  
 }  
  
 */\*\*  
 \* Start mainPage  
 \*/* private void startPage(){  
 // System.out.println("startPage");  
 panel1 = new JPanel();  
 panel1.setBackground(Color.*black*);  
  
 if(*startMusic*.isPlaying() == false){  
 // startMusic.play();  
 }  
  
  
 JLabel title = new JLabel("COLORPORT");  
 title.setBounds(250,100,600,150);  
 title.setForeground(Color.*WHITE*);  
 title.setFont(titleFont);  
  
 JLabel titleSmall = new JLabel("in search of happiness...");  
 titleSmall.setBounds(370,170,600,150);  
 titleSmall.setForeground(Color.*WHITE*);  
 titleSmall.setFont(new Font("Times New Roman", Font.*PLAIN*, 30));  
  
 JButton startButton =new JButton("START!");  
 startButton.setBackground(new Color(245, 183, 15));  
 startButton.setForeground(Color.*BLACK*);  
 startButton.setFont(normalFont);  
 startButton.setBounds(420,400,200,80);  
 startButton.addActionListener(new ActionListener(){  
 public void actionPerformed(ActionEvent e){  
 *startMusic*.stop();  
 firstMaze();  
 }  
 });  
  
  
 JButton instruction =new JButton("INSTRUCTION");  
 instruction.setBounds(420,490,200,50);  
 instruction.setBackground(Color.*black*);  
 instruction.setForeground(Color.*WHITE*);  
 instruction.setFont(new Font("Times New Roman", Font.*BOLD*,20));  
 instruction.addActionListener(new ActionListener(){  
 public void actionPerformed(ActionEvent e){  
 instructions();  
  
 }  
 });  
  
 panel1.add(instruction);  
 panel1.add(titleSmall);  
 panel1.add(title);  
 panel1.add(startButton);  
 panel1.setLayout(null);  
  
 }  
  
 */\*\*  
 \* upload page with maze  
 \* \*/* private void firstMaze(){  
 // System.out.println("firstMaze");  
 panel1.removeAll();  
 revalidate();  
 repaint();  
 panelWithMaze.setInGame(true);  
 System.*out*.println("in game - " + panelWithMaze.isInGame());  
 panelWithMaze.setBounds(10,10, 950,700);  
  
  
  
 JButton back =new JButton("I I");  
 // back.setIcon(new ImageIcon("pause"));  
 back.setBounds(975,18,50,50);  
 back.setBackground(Color.*black*);  
 back.setForeground(Color.*WHITE*);  
  
 back.addActionListener(new ActionListener(){  
 public void actionPerformed(ActionEvent e){  
 pausePage();  
 }  
 });  
 panel1.add(back);  
 panel1.add(panelWithMaze);  
 panel1.setLayout(null);  
 revalidate();  
 repaint();  
 }  
  
 */\*\*  
 \* open winning page  
 \*/* public void endPageWin(){  
 // System.out.println("endPageWin");  
 panel1.removeAll();  
 panelWithMaze.removeAll();  
 panelWithMaze.revalidate();  
 panelWithMaze.repaint();  
 revalidate();  
 repaint();  
  
 panel1.setBackground(new Color(219, 136, 42));  
  
 JLabel title = new JLabel("WINNER!!");  
 title.setBounds(290,100,600,150);  
 title.setForeground(Color.*black*);  
 title.setFont(titleFont);  
  
 ImageIcon imageIcon = new ImageIcon("src/images/hero/gg4.png"); // load the image to a imageIcon  
 Image image = imageIcon.getImage(); // transform it  
 Image newimg = image.getScaledInstance(60,60, java.awt.Image.*SCALE\_SMOOTH*); // scale it the smooth way  
 imageIcon = new ImageIcon(newimg); // transform it back  
 JLabel img1 = new JLabel(imageIcon);  
 img1.setBounds(470,300,60,60);  
 panel1.add(img1);  
  
  
 JLabel titleSmall = new JLabel("now you are happy!");  
 titleSmall.setBounds(370,170,600,150);  
 titleSmall.setForeground(Color.*black*);  
 titleSmall.setFont(new Font("Times New Roman", Font.*PLAIN*, 30));  
  
 JButton startButton =new JButton("START AGAIN!");  
 startButton.setBackground(new Color(43, 99, 37));  
 startButton.setForeground(Color.*BLACK*);  
 startButton.setFont(normalFont);  
  
 startButton.setBounds(370,400,250,80);  
 startButton.addActionListener(e -> start());  
  
 panel1.add(titleSmall);  
 panel1.add(title);  
 panel1.add(startButton);  
  
  
 panel1.setLayout(null);  
 revalidate();  
 repaint();  
 }  
  
 */\*\*  
 \* open loser page  
 \*/* public void endPageLoser(){  
 // System.out.println("endPageLoser");  
 panelWithMaze.setInGame(false);  
 panel1.removeAll();  
 revalidate();  
 repaint();  
  
 panel1.setBackground(new Color(10, 10, 10));  
  
 JLabel title = new JLabel("LOSER!!");  
 title.setBounds(345,100,600,150);  
 title.setForeground(Color.*white*);  
 title.setFont(titleFont);  
  
 ImageIcon imageIcon = new ImageIcon("src/images/hero/gg1.png"); // load the image to a imageIcon  
 Image image = imageIcon.getImage(); // transform it  
 Image newimg = image.getScaledInstance(60,60, java.awt.Image.*SCALE\_SMOOTH*); // scale it the smooth way  
 imageIcon = new ImageIcon(newimg); // transform it back  
 JLabel img1 = new JLabel(imageIcon);  
 img1.setBounds(470,300,60,60);  
 panel1.add(img1);  
  
  
  
 JLabel titleSmall = new JLabel("life is full of obstacles, but you shouldn't give up");  
 titleSmall.setBounds(315,170,600,150);  
 titleSmall.setForeground(Color.*WHITE*);  
 titleSmall.setFont(new Font("Times New Roman", Font.*PLAIN*, 20));  
  
  
  
 JButton startButton =new JButton("TRY AGAIN!");  
 startButton.setBackground(new Color(60, 90, 107));  
 startButton.setForeground(Color.*BLACK*);  
 startButton.setFont(normalFont);  
  
 startButton.setBounds(380,400,250,80);  
 startButton.addActionListener(e -> start());  
  
  
 panel1.add(titleSmall);  
 panel1.add(title);  
 panel1.add(startButton);  
  
 panel1.setLayout(null);  
 revalidate();  
 repaint();  
 }  
  
 */\*\*method that recreate window  
 \*/* private void reCreateWindow(){  
 // System.out.println("reCreationWindow");  
 panelWithMaze = new Maze(bricksLevels);  
 this.addKeyListener(new TAdapter(panelWithMaze,this));  
 setFocusable(true);  
 setLocationRelativeTo(null);  
 }  
  
 */\*\*  
 \* open pause page  
 \*/* public void pausePage(){  
 // System.out.println("pausePage");  
 panelWithMaze.setInGame(false);  
 lives = panelWithMaze.getHeroLivesOnLevelStart();  
 panel1.removeAll();  
 revalidate();  
 repaint();  
  
 JButton startAgain =new JButton("START AGAIN");  
 startAgain.setBounds(420,200,200,50);  
 startAgain.setBackground(Color.*black*);  
 startAgain.setForeground(Color.*WHITE*);  
 startAgain.setFont(new Font("Times New Roman", Font.*BOLD*,20));  
  
 startAgain.addActionListener(new ActionListener(){  
 public void actionPerformed(ActionEvent e){  
 reCreateWindow();  
 firstMaze();  
 panelWithMaze.setHeroLives(lives);  
 }  
 });  
  
 JButton continueGame =new JButton("CONTINUE");  
 continueGame.setBounds(420,300,200,50);  
 continueGame.setBackground(Color.*black*);  
 continueGame.setForeground(Color.*WHITE*);  
 continueGame.setFont(new Font("Times New Roman", Font.*BOLD*,20));  
 continueGame.addActionListener(new ActionListener(){  
 public void actionPerformed(ActionEvent e){  
 firstMaze();  
 }  
 });  
  
  
 panel1.add(continueGame);  
 panel1.add(startAgain);  
  
 panel1.setLayout(null);  
 revalidate();  
 repaint();  
 }  
  
 */\*\*  
 \* open page with instructions  
 \*/* public void instructions(){  
 // System.out.println("pausePage");  
 panel1.removeAll();  
 revalidate();  
 repaint();  
  
  
 JTextPane textArea = new JTextPane();  
 textArea.setText("Game instruction" + "\n" +  
 "Aim of this game is to go through all six colorful mazes and collect all colors!" + "\n" +  
 "\n" +"Useful keys:" + "\n" +  
 "Movement of hero - ↑ up, ↓ down, → right, ← left" + "\n" +  
 "Collect lives, go through the portal – space key" + "\n" +  
 "Pause game – escape key" + "\n" +  
 "Increase speed – A" + "\n" +  
 "Decrease speed – D" + "\n" +  
 "Set default speed – S");  
 textArea.setFont(new Font("Times New Roman", Font.*PLAIN*,20));  
 textArea.setBounds(220,100,600,300);  
  
 textArea.setBackground(Color.*black*);  
 textArea.setForeground(Color.*WHITE*);  
  
 SimpleAttributeSet aSet = new SimpleAttributeSet();  
 StyleConstants.*setFontFamily*(aSet, "Times New Roman");  
 StyleConstants.*setAlignment*(aSet, StyleConstants.*ALIGN\_CENTER*);  
 StyleConstants.*setFontSize*(aSet, 32);  
  
 SimpleAttributeSet bSet = new SimpleAttributeSet();  
 StyleConstants.*setAlignment*(bSet, StyleConstants.*ALIGN\_CENTER*);  
 StyledDocument doc = textArea.getStyledDocument();  
 doc.setParagraphAttributes(0, doc.getLength(), bSet, false);  
 doc.setParagraphAttributes(0, 12, aSet, false);  
 doc.setParagraphAttributes(100, 12, aSet, false);  
  
 textArea.setOpaque(false);  
 textArea.setEditable(false);  
  
 panel1.add(textArea);  
  
  
  
 JButton continueGame =new JButton("BACK");  
 continueGame.setBounds(420,500,200,50);  
 continueGame.setBackground(Color.*black*);  
 continueGame.setForeground(Color.*WHITE*);  
 continueGame.setFont(new Font("Times New Roman", Font.*BOLD*,20));  
 continueGame.addActionListener(new ActionListener(){  
 public void actionPerformed(ActionEvent e){  
  
 start();  
 }  
 });  
  
  
 panel1.add(continueGame);  
  
  
 panel1.setLayout(null);  
 revalidate();  
 repaint();  
 }  
  
}

TAdaptor

package com.rgb;  
  
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
import java.io.File;  
  
*/\*\*  
 \* This class of game control through the keyboard  
 \*/*public class TAdapter extends KeyAdapter {  
 Maze m;  
 static StartPage *startPage*;  
  
 */\*\* Sound panel  
 \*  
 \*/* Sound bonusMusic = new Sound(new File("src/music/bonus.wav"));  
 Sound intoNextLevelMusic = new Sound(new File("src/music/intoNextLevel.wav"));  
  
 */\*\*  
 \* Constructor  
 \** ***@param*** *maze - reference to the maze class  
 \** ***@param*** *start - reference to the game frame class  
 \*/* public TAdapter(Maze maze, StartPage start){  
 m = maze;  
 *startPage* = start;  
 }  
  
 */\*\*  
 \* Change the hero’s displacement by y if the hero moves up  
 \*/* private void up(){  
 m.setReq\_dy(-1);  
 }  
  
 */\*\*  
 \* Change the hero’s displacement by y if the hero moves down  
 \*/* private void down(){  
 m.setReq\_dy(1);  
 }  
  
 */\*\*  
 \* Change the hero’s displacement by x if the hero moves left  
 \*/* private void left(){  
 m.setReq\_dx(-1);  
 }  
  
 */\*\*  
 \* Change the hero’s displacement by x if the hero moves right  
 \*/* private void right(){  
 m.setReq\_dx(1);  
 }  
  
 */\*\*  
 \* Override method that check when the key goes down  
 \** ***@param*** *e -- An event which indicates that a keystroke occurred in a component  
 \*/* @Override  
 public void keyPressed(KeyEvent e) {  
 m.setReq\_dx(0);  
 m.setReq\_dy(0);  
 int code = e.getKeyCode();  
 //up  
 if(code == KeyEvent.*VK\_UP* && m.isInGame()){  
 up();  
 }  
 //down  
 if(code == KeyEvent.*VK\_DOWN* && m.isInGame()){  
 down();  
 }  
 //left  
 if(code == KeyEvent.*VK\_LEFT* && m.isInGame()){  
 left();  
 }  
 //right  
 if(code == KeyEvent.*VK\_RIGHT* && m.isInGame()){  
 right();  
 }  
  
 if(code == KeyEvent.*VK\_SPACE* && m.isInGame()){  
 if(m.isInGame()==true && m.isDying() == false){  
  
 if(m.isHeart()==true) {  
 bonusMusic.play();  
 m.addHeroLives();  
 }  
 if(m.isPortal()){  
 intoNextLevelMusic.play();  
 m.nextLevel();  
 }  
  
 }  
  
 if(m.isGameFinished() == true && m.isDying() == false) {  
 *startPage*.endPageWin();  
 }  
 }  
 if(code == KeyEvent.*VK\_ENTER* && m.isInGame()){  
 // startPage.endPageLoser();  
 }  
  
 if(code == KeyEvent.*VK\_ESCAPE* && m.isInGame())  
 {  
 *startPage*.pausePage();  
 }  
 if(code == KeyEvent.*VK\_1* && m.isInGame())  
 {  
 m.setCheat(1);  
 }  
  
 if(code == KeyEvent.*VK\_0* && m.isInGame())  
 {  
 m.setCheat(0);  
 }  
 if(code == KeyEvent.*VK\_A* && m.isInGame())  
 {  
 double new\_speed = m.getMyHeroSpeed(m.gameLevel-1);  
 m.setMyHeroSpeed(new\_speed\*1.05, m.gameLevel-1);  
 }  
 if(code == KeyEvent.*VK\_D* && m.isInGame())  
 {  
 double new\_speed = m.getMyHeroSpeed(m.gameLevel-1);  
 m.setMyHeroSpeed(new\_speed/1.05, m.gameLevel-1);  
 }  
 if(code == KeyEvent.*VK\_S* && m.isInGame())  
 {  
 m.setMyHeroSpeed(1, m.gameLevel-1);  
 }  
 /\* if(code == KeyEvent.VK\_PAGE\_UP)  
 {  
 }  
  
 if(code == KeyEvent.VK\_PAGE\_DOWN)  
 {  
 }  
 \*/  
 if(code == KeyEvent.*VK\_BACK\_SPACE* && m.isInGame()){  
 System.*out*.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-");  
 for(int i=0;i<m.bricks.size();i++){  
 for(int k=0;k<m.bricks.get(0).length();k++){  
 System.*out*.print(m.map[i][k]+"");  
 }  
 System.*out*.println("");  
 }  
 System.*out*.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  
 }  
 }  
  
 */\*\*  
 \* Stop hero if key is released  
 \** ***@param*** *e - An event which indicates that a keystroke occurred in a component  
 \*/* @Override  
 public void keyReleased(KeyEvent e)  
 {  
 int code = e.getKeyCode();  
  
 if(code == KeyEvent.*VK\_UP* && m.isInGame()){  
 m.setReq\_dy(0);  
 }  
 //down  
 if(code == KeyEvent.*VK\_DOWN* && m.isInGame()){  
 m.setReq\_dy(0);  
 }  
 //left  
 if(code == KeyEvent.*VK\_LEFT* && m.isInGame()){  
 m.setReq\_dx(0);  
 }  
 //right  
 if(code == KeyEvent.*VK\_RIGHT* && m.isInGame()){  
 m.setReq\_dx(0);  
 }  
  
 }  
  
}