Breakout

A screenshot of a game

Description automatically generated

Лабораторна робота Федорича Володимира з Основ Програмування.

Задача

Створити гру Breakout використовуючи бібліотеку acm та мову Java.   
A close-up of a screen

Description automatically generated

Гравець керує ракеткою, що переміщається по горизонталі та має відбивати мʼяч, яким у свою чергу можна розбивати цеглинки. Якщо розбити усі цеглинки – гра закінчується перемогою, а якщо м’яч пролітає повз ракетку – гравець втрачає життя.

Структура програми

A screenshot of a computer program

Description automatically generated

Опис методів та класів

Breakout – це головний клас який створює обʼєкти класів Ball, Paddle та Brick й керує ними. Для зручності усі змінні, доступ до яких потребують різні класи було винесено у окремий статичний клас Variables. Також для розвантаження класу Breakout додано допоміжні класи Graphics та Menu, які зберігають у собі статичні функції, якими послуговується головний клас.

В директорії проєкту лежить папка JavaDoc де знаходиться деталий опис кожного метода.

Інструкція користувача

A black screen with white text

Description automatically generated

A screen shot of a phone

Description automatically generatedПісля запуску програми користувач потрапляє в меню вибору рівня, де він може натиснути 1, 2 чи 3 аби запустити рівень відповідної складності.

Користувач має керувати ракеткою за допомогою миші та відбивати мʼяч у цеглинки аби перейти на наступний рівень.

Великі проблеми які виникали

* Проблема – при спробі перевірити чи мʼяч поза екраном, перевірка спрацьовує кожен тік.
  + Вирішення - перестворити мʼячик замість того щоб рухати його.
* Проблема - мʼячик вдаряється об текст
  + Вирішення - ігнорувати цей випадок у перевірці колізій
* Проблема - цеглинки не малюються якщо рядів менше пʼяти
  + Вирішення - їх не було видно через те, що для визначення кольору використовувалася перевірка подільності на 5 без остачі
* Проблеме – рестарт працює тільки один раз
  + Вирішення – змінна життів не оновлювалась після рестарту
* Проблема – мʼячик застрягає
  + Вирішення – повністю вирішити цю проблему неможливо через специфіку реалізації обробку колізій, проте створення 8(а не 4) точок перевірки та зменшення тіку до 1мс зменшило частоту проявлення багу.

Код

Breakout.java

import acm.graphics.GImage;  
import acm.graphics.GLabel;  
import acm.graphics.GObject;  
import acm.program.GraphicsProgram;  
import java.awt.\*;  
import java.awt.event.KeyEvent;  
import java.awt.event.MouseEvent;  
  
public class Breakout extends GraphicsProgram {  
 Paddle paddle;  
 Ball ball;  
 GLabel scoreLabel = new GLabel("Score: " + Variables.*score*, 20, Variables.*appHeight* - 60);  
  
 private boolean gameStarted = false;  
  
 */\*\*  
 \* Handles program initialization and game loop. Runs infinitely operating with params changed during the game.  
 \*/* public void run() {  
 this.setSize(Variables.*appWidth*, Variables.*appHeight*);  
 this.setBackground(Variables.*backgroundColor*);  
  
 while(true){  
 addMouseListeners();  
 addKeyListeners();  
 resetValues();  
  
 //handle game replayability  
 if(Variables.*shouldOpenMenu*){  
 Menu menu = new Menu();  
 add(menu.getStartMenuGObject());  
  
 while(!gameStarted) { pause(5); }  
 initialize();  
  
 remove(menu.getStartMenuGObject());  
 } else{  
 initValues(Variables.*level*);  
 initialize();  
 }  
  
 //game loop  
 while(!Variables.*gameOver*) {  
 ball.move();  
 handleCollisions();  
 paddle.move();  
 handleBallPresence();  
 pause(0.5);  
 }  
  
 //win condition check  
 if(Variables.*won*){  
 win();  
 } else {  
 lose();  
 }  
 }  
 }  
  
 */\*\*  
 \* If ball is not on screen, resets it and decrements lives.  
 \*/* private void handleBallPresence() {  
 if(!ball.isBallOnScreen()){  
 resetBall();  
 decrementHearts();  
 paddle.reset();  
 if(Variables.*lives* <= 0){  
 Variables.*gameOver* = true;  
 }  
 }  
 }  
  
 */\*\*  
 \* Respawns ball and paddle, resets score, lives and bricks.  
 \*/* private void initialize(){  
 //remove old paddle and ball  
 if(paddle != null){  
 remove(paddle);  
 }  
 if(ball != null){  
 remove(ball);  
 }  
  
 //draw platform  
 paddle = new Paddle((Variables.*appWidth*-15)/2 - Variables.*paddleWidth*/2,  
 (Variables.*appHeight* - (Variables.*paddleHeight* \* 2) - 60) , Variables.*paddleWidth*, Variables.*paddleHeight*);  
 add(paddle);  
  
 //draw ball  
 ball = new Ball(paddle.getX()+  
 paddle.getWidth()/2 - Variables.*radius*/2,  
 paddle.getY() - Variables.*radius* - 15, Variables.*radius*);  
 add(ball);  
  
 Variables.*brickCount* = Variables.*rows* \* Variables.*bricksPerRow*;  
  
 //draw hearts and bricks  
 Graphics.*drawBricks*(getGCanvas());  
 Graphics.*drawHearts*(getGCanvas());  
  
 //draw score  
 add(scoreLabel);  
 scoreLabel.setFont("TimesNewRoman-20");  
 scoreLabel.setColor(Color.*WHITE*);  
 }  
  
 */\*\*  
 \* Handles collisions with paddle, bricks and walls via checking if 8 points of ball are colliding with something.  
 \* These point are located on the edges and corners of ball.  
 \*/* private void handleCollisions() {  
 //define coordinates of 8 points  
 double bx1 = ball.getX() - 1;  
 double by1 = ball.getY() - 1;  
  
 double bx2 = ball.getX() + ball.getWidth() - 1;  
 double by2 = by1;  
  
 double bx3 = ball.getX() + 1;  
 double by3 = ball.getY() + ball.getHeight() + 1;  
  
 double bx4 = ball.getX() + ball.getWidth() + 1;  
 double by4 = by3;  
  
 double bx5 = ball.getX();  
 double by5 = ball.getY();  
  
 double bx6 = ball.getX() + ball.getWidth();  
 double by6 = by5;  
  
 double bx7 = ball.getX();  
 double by7 = ball.getY() + ball.getHeight();  
  
 double bx8 = ball.getX() + ball.getWidth();  
 double by8 = by7;  
  
 GObject obj = null;  
  
 //check if any of 8 points are colliding with something  
 if(getElementAt(bx1, by1) != null) {  
 obj = getElementAt(bx1, by1);  
 }  
 else if(getElementAt(bx2, by2) != null) {  
 obj = getElementAt(bx2, by2);  
 }  
 else if(getElementAt(bx3, by3) != null) {  
 obj = getElementAt(bx3, by3);  
 }  
 else if(getElementAt(bx4, by4) != null) {  
 obj = getElementAt(bx4, by4);  
 }  
 else if(getElementAt(bx5, by5) != null) {  
 obj = getElementAt(bx5, by5);  
 }  
 else if(getElementAt(bx6, by6) != null) {  
 obj = getElementAt(bx6, by6);  
 }  
 else if(getElementAt(bx7, by7) != null) {  
 obj = getElementAt(bx7, by7);  
 }  
 else if(getElementAt(bx8, by8) != null) {  
 obj = getElementAt(bx8, by8);  
 }  
  
 boolean side = Variables.*rg*.nextBoolean(0.5);  
  
 //old code  
 /\*if(obj == paddle) {  
 double direction = side == true ? Math.PI/4 : Math.PI/2;  
 ball.setDirection(direction);  
  
 \*/  
  
 //check what object is colliding with ball and change direction accordingly  
 if(obj == paddle) {  
 double direction = ball.getDirection() > Math.*PI* ? Math.*PI*/4 : 3\*Math.*PI*/4;  
 ball.setDirection(direction);  
 } else if(obj instanceof Brick){  
 double direction = side == true ? Math.*PI*/1.1 : Math.*PI*/2.2;  
 ball.setDirection(direction);  
 Brick x = (Brick)obj;  
 x.onDeleteBrick();  
 scoreLabel.setLabel("Score: " + Variables.*score*);  
 remove(obj);  
 } else if(obj != null && !(obj instanceof GLabel) && !(obj instanceof GImage)){  
 double direction = side == true ? Math.*PI*/1.1 : Math.*PI*/2.2;  
 direction += Variables.*rg*.nextBoolean(0.5) ? 0.05 : -0.05; // Add a slight random variation  
 ball.setDirection(direction);  
 }  
 }  
  
 */\*\*  
 \* Handles key presses.  
 \* @param e key event  
 \*/* public void keyPressed(KeyEvent e) {  
 if(!gameStarted && e.getKeyCode() >= 49 && e.getKeyCode() <= 51){  
 openLevel(e.getKeyCode());  
 }  
 if(Variables.*gameOver* && !Variables.*won*){  
 if(e.getKeyCode() == 82){  
 Variables.*shouldRestart* = true;  
 }  
 } else if(Variables.*gameOver* && Variables.*won*){  
 if(e.getKeyCode() == 78){  
 Variables.*shouldOpenLevel* = true;  
 }  
 }  
 if(Variables.*gameOver* && Variables.*won* && Variables.*level* == 3){  
 Variables.*shouldOpenMenu* = true;  
 }  
 }  
  
 */\*\*  
 \* Handles mouse movement.  
 \* @param e mouse event  
 \*/* public void mouseMoved(MouseEvent e) {  
 Variables.*mouseX* = e.getX();  
 Variables.*mouseY* = e.getY();  
 }  
  
 */\*\*  
 \* Handles win condition visualisation and level transition.  
 \* If level is 3, proceeds to final screen.  
 \*/* private void win(){  
 if(Variables.*level* == 3){  
 endGame();  
 } else{  
 Variables.*winSound*.setVolume(1);  
 Variables.*winSound*.play();  
  
 Graphics.*addWinText*(getGCanvas());  
  
 //wait for click to open next level  
 while(!Variables.*shouldOpenLevel*){  
 pause(5);  
 }  
 Variables.*level*++;  
 setDefaultValuesForLevel(Variables.*level*);  
 removeAll();  
 Variables.*shouldOpenMenu* = false;  
 }  
 }  
  
 */\*\*  
 \* Handles endgame screen.  
 \*/* private void endGame(){  
 Variables.*shouldOpenMenu* = false;  
 removeAll();  
 Graphics.*addEndgameUI*(getGCanvas());  
 while(!Variables.*shouldOpenMenu*){  
 pause(5);  
 }  
 removeAll();  
 }  
  
 */\*\*  
 \* Handles lose condition visualisation and restart.  
 \*/* private void lose() {  
 Variables.*loseSound*.setVolume(1);  
 Variables.*loseSound*.play();  
  
 Graphics.*addLoseText*(getGCanvas());  
  
 while(!Variables.*shouldRestart*){  
 pause(5);  
 }  
 removeAll();  
 Variables.*shouldOpenMenu* = false;  
 }  
  
 */\*\*  
 \* Respawns ball.  
 \*/* private void resetBall(){  
 remove(ball);  
 ball = new Ball(paddle.getX()+  
 paddle.getWidth()/2 - Variables.*radius*/2,  
 paddle.getY() - Variables.*radius* - 15, Variables.*radius*);  
 add(ball);  
 }  
  
 */\*\*  
 \* Decrements lives and removes heart from screen.  
 \*/* private void decrementHearts(){  
 --Variables.*lives*;  
 if(Variables.*lives* > 0){  
 Variables.*loseLifeSound*.setVolume(1);  
 Variables.*loseLifeSound*.play();  
 }  
  
 Variables.*hearts*.get(Variables.*lives*).setVisible(false);  
 Variables.*hearts*.remove(Variables.*lives*);  
 }  
  
 */\*\*  
 \* Sets default values for level.  
 \* @param key level number  
 \*/* private void initValues(int key){  
 switch(key){  
 case 49:  
 setDefaultValuesForLevel(1);  
 break;  
 case 50:  
 setDefaultValuesForLevel(2);  
 break;  
 case 51:  
 setDefaultValuesForLevel(3);  
 break;  
 }  
 }  
  
 */\*\*  
 \* Opens level.  
 \* @param i level number  
 \*/* private void openLevel(int i){  
 initValues(i);  
 gameStarted = true;  
 }  
  
 */\*\*  
 \* Resets game-handling variables that have been changed during the game to default values.  
 \*/* private void resetValues(){  
 Variables.*won* = false;  
 Variables.*gameOver* = false;  
 Variables.*score* = 0;  
 Variables.*hearts*.clear();  
 scoreLabel.setLabel("Score: ");  
 Variables.*shouldRestart* = false;  
 gameStarted = false;  
 Variables.*shouldOpenLevel* = false;  
 Variables.*lives* = Variables.*getLives*();  
 }  
  
 */\*\*  
 \* Handles starting params for each level.  
 \* @param level level number  
 \*/* private void setDefaultValuesForLevel(int level){  
 switch(level) {  
 case 1:  
 Variables.*rows* = 3;  
 Variables.*bricksPerRow* = 3;  
 Variables.*lives* = 5;  
 Variables.*level* = 1;  
 break;  
 case 2:  
 Variables.*rows* = 5;  
 Variables.*bricksPerRow* = 5;  
 Variables.*lives* = 3;  
 Variables.*level* = 2;  
 break;  
 case 3:  
 Variables.*rows* = 10;  
 Variables.*bricksPerRow* = 10;  
 Variables.*lives* = 1;  
 Variables.*level* = 3;  
 break;  
 }  
 }  
}

Ball.java

import acm.graphics.GObject;  
import acm.graphics.GOval;  
import acm.util.SoundClip;  
  
import java.awt.\*;  
  
public class Ball extends GOval  
{  
 private double posX;  
 private double posY;  
  
 private double size;  
  
 private double direction;  
  
 private double velocity;  
  
 public void setDirection(double direction)  
 {  
 this.direction += direction;  
  
 if(this.direction >= 2 \* Math.*PI*)  
 this.direction -= (2 \* Math.*PI*);  
  
 Variables.*bounceSound*.setVolume(1);  
 Variables.*bounceSound*.play();  
 }  
  
 */\*\*  
 \* Default constructor  
 \* @param posX - x coordinate of the ball  
 \* @param posY - y coordinate of the ball  
 \* @param size - size of the ball  
 \*/* public Ball(double posX, double posY, double size)  
 {  
 super(posX, posY, size, size);  
 this.posX = posX;  
 this.posY = posY;  
 this.size = size;  
 setFilled(true);  
 setColor(Color.*WHITE*);  
 setLocation(posX, posY);  
 direction = Variables.*rg*.nextDouble(0.5, 2.51);  
 velocity = 0.7;  
 }  
  
 */\*\*  
 \* Moves the ball by multiplying the velocity by the sin and cos of the direction  
 \*/* public void move()  
 {  
 checkCollisions();  
 posX += Math.*sin*(direction) \* velocity;  
 posY -= Math.*cos*(direction) \* velocity;  
 setLocation(posX, posY);  
 }  
  
 */\*\*  
 \* Calculates the angle of reflection randomly  
 \*/* public void checkCollisions()  
 {  
 if (posX <= 0 || posX + size >= Variables.*appWidth* - 15 || posY <= 0)  
 {  
 direction += Math.*PI* / 8 + Variables.*rg*.nextDouble(-0.05, 0.05);  
  
 if(direction >= 2 \* Math.*PI*){  
 direction -= (2 \* Math.*PI*);  
 }  
 Variables.*bounceSound*.setVolume(1);  
 Variables.*bounceSound*.play();  
 }  
 }  
  
 */\*\*  
 \* Checks if the ball is on the screen  
 \*/* public boolean isBallOnScreen(){  
 return getY() < Variables.*appHeight*;  
 }  
  
 */\*\*  
 \* Public direction getter  
 \*/* public double getDirection() {  
 return direction;  
 }  
}

Paddle.java

import acm.graphics.GRect;  
  
import java.awt.\*;  
  
public class Paddle extends GRect{  
 private double posX;  
 private double posY;  
 private double width;  
 private double trgX;  
  
 */\*\*  
 \* Default constructor  
 \* @param posX - x coordinate of the paddle  
 \* @param posY - y coordinate of the paddle  
 \* @param width - width of the paddle  
 \* @param height - height of the paddle  
 \*/* public Paddle(double posX, double posY, double width, double height)  
 {  
 super(posX, posY, width, height);  
 this.posX = posX;  
 this.posY = posY;  
 this.width = width;  
 this.trgX = posX;  
  
 setFilled(true);  
 setColor(Color.*WHITE*);  
 }  
  
 */\*\*  
 \* Moves the paddle smoothly towards the mouse  
 \*/* public void move()  
 {  
 trgX = Variables.*mouseX* - width/2;  
  
 double angle = Math.*atan*((70) / (trgX - this.posX));  
  
 if(trgX >= posX) {  
 if(posX + width < Variables.*appWidth*-15)  
 posX += Math.*cos*(angle) \* 3;  
 }  
 else if(posX > 0)  
 posX -= Math.*cos*(angle) \* 3;  
  
 setLocation(posX, posY);  
 }  
  
 */\*\*  
 \* Resets the paddle to the center of the screen  
 \*/* public void reset() {  
 setLocation((Variables.*appHeight*-15)/2 - Variables.*paddleWidth*/2,  
 (Variables.*appHeight* - (Variables.*paddleHeight* \* 2) - 60));  
 }  
}

Brick.java

import acm.graphics.GRect;  
import java.awt.\*;  
  
public class Brick extends GRect {  
 private int weight = 0;  
  
 */\*\*  
 \* Constructor for the Brick class. Sets the color and weight of the brick based on row.  
 \* @param x  
 \* @param y  
 \* @param nrow  
 \*/* Brick(double x, double y, int nrow) {  
 super(x, y, Variables.*brickWidth*, Variables.*brickHeight*);  
 setFilled(true);  
  
 int colorRows = Math.*max*(Variables.*rows* / 5, 1);  
 nrow /= colorRows;  
  
 if (nrow > 4) nrow -= 5;  
  
 switch (nrow) {  
 case 0 -> {  
 setColor(Color.*RED*);  
 weight = 5;  
 }  
 case 1 -> {  
 setColor(Color.*ORANGE*);  
 weight = 4;  
 }  
 case 2 -> {  
 setColor(Color.*YELLOW*);  
 weight = 3;  
 }  
 case 3 -> {  
 setColor(Color.*GREEN*);  
 weight = 2;  
 }  
 case 4 -> {  
 setColor(Color.*CYAN*);  
 weight = 1;  
 }  
 }  
 }  
  
 */\*\*  
 \* Called when a brick is hit. Decrements the brick count, plays a sound, and adds to the score.  
 \*/* public void onDeleteBrick(){  
 Variables.*brickCount*--;  
 Variables.*brickSound*.setVolume(0.5);  
 Variables.*brickSound*.play();  
 Variables.*score* += weight;  
 if(Variables.*brickCount* <= 0) {  
 Variables.*won* = true;  
 Variables.*gameOver* = true;  
 }  
 }  
}

Menu.java

import acm.graphics.GCompound;  
import acm.graphics.GLabel;  
import acm.graphics.GObject;  
import acm.graphics.GRect;  
import acm.program.GraphicsProgram;  
  
import java.awt.\*;  
  
  
public class Menu extends GraphicsProgram {  
 double menuHeight = Variables.*appHeight* / 2;  
 GLabel startLabel = new GLabel("Breakout");  
 GLabel startDescription = new GLabel("Use mouse to control the paddle. Press 1, 2 or 3 to select level");  
 GLabel authorLabel = new GLabel("Made by: Fedorych Volodymyr");  
 GCompound startMenu = new GCompound();  
  
 */\*\*  
 \* Constructor for the Menu class  
 \*/* public Menu(){  
 int kegel = (int)((Variables.*appWidth*/2.7) \* (menuHeight/2.5) / 1000);  
  
 menuTextSetup(kegel);  
 }  
  
 */\*\*  
 \* Method that sets up the menu text  
 \* @param kegel - font size  
 \*/* private void menuTextSetup(int kegel) {  
 startLabel.setFont("TimesNewRoman-" + kegel);  
 double startLabel\_x = 20;  
 double startLabel\_y = (menuHeight / 2) + Variables.*brickYOffset* + (startLabel.getAscent() / 2);  
 startLabel.setLocation(startLabel\_x, startLabel\_y);  
 startLabel.setColor(Color.*WHITE*);  
  
 startDescription.setFont("TimesNewRoman-" + kegel/2);  
 double startDescription\_y = startLabel\_y + 25;  
 startDescription.setLocation(startLabel\_x, startDescription\_y);  
 startDescription.setColor(Color.*decode*("#fdfefe"));  
  
 authorLabel.setFont("TimesNewRoman-" + kegel/2);  
 double authorLabel\_y = startLabel\_y + 50;  
 authorLabel.setLocation(startLabel\_x, authorLabel\_y);  
 authorLabel.setColor(Color.*decode*("#fdfefe"));  
  
 startMenu.add(startLabel);  
 startMenu.add(startDescription);  
 startMenu.add(authorLabel);  
 }  
  
 */\*\*  
 \* Menu getter  
 \*/* public GObject getStartMenuGObject(){  
 return startMenu;  
 }  
}

Graphics.java

import acm.graphics.GCanvas;  
import acm.graphics.GImage;  
import acm.graphics.GLabel;  
  
import java.awt.\*;  
  
public class Graphics {  
 */\*\*  
 \* Draws the visual representation of lives  
 \* @param canvas the canvas to draw on  
 \*/* public static void drawHearts(GCanvas canvas){  
 for(int i = 0; i < Variables.*lives*; ++i){  
 GImage heart = new GImage("../images/heart.png");  
 heart.scale(0.1, 0.1);  
 canvas.add(heart, i \* 25, 0);  
 Variables.*hearts*.add(heart);  
 }  
 }  
  
 */\*\*  
 \* Draws bricks  
 \* @param canvas the canvas to draw on  
 \*/* public static void drawBricks(GCanvas canvas){  
 double totalBricksWidth = Variables.*bricksPerRow* \* (Variables.*brickWidth* + Variables.*brickDelta*) - Variables.*brickDelta*;  
 if (totalBricksWidth > Variables.*appWidth*) {  
 Variables.*bricksPerRow* = ((Variables.*appWidth* + Variables.*brickDelta*) / (Variables.*brickWidth* + Variables.*brickDelta*));  
 totalBricksWidth = Variables.*bricksPerRow* \* (Variables.*brickWidth* + Variables.*brickDelta*) - Variables.*brickDelta*;  
 }  
 double startX = (Variables.*appWidth* - totalBricksWidth) / 2;  
 for (int x = 0; x < Variables.*bricksPerRow*; x++)  
 for (int y = 0; y < Variables.*rows*; y++) {  
 double bx = startX + x \* (Variables.*brickWidth* + Variables.*brickDelta*);  
 double by = y \* (Variables.*brickHeight* + Variables.*brickDelta*) + Variables.*brickYOffset*;  
 canvas.add(new Brick(bx, by, y));  
 }  
 }  
  
 */\*\*  
 \* Draws win-level text  
 \* @param canvas the canvas to draw on  
 \*/* public static void addWinText(GCanvas canvas){  
 GLabel winLabel = new GLabel("You won!");  
 winLabel.setLocation((Variables.*appWidth* - winLabel.getWidth())/2, Variables.*appHeight*/2);  
 winLabel.setFont("TimesNewRoman-20");  
 winLabel.setColor(Color.*WHITE*);  
 canvas.add(winLabel);  
  
 GLabel nextLevelLabel = new GLabel("Press N to go to next level");  
 nextLevelLabel.setLocation((Variables.*appWidth* - nextLevelLabel.getWidth())/2, Variables.*appHeight*/2 + 25);  
 nextLevelLabel.setFont("TimesNewRoman-20");  
 nextLevelLabel.setColor(Color.*WHITE*);  
 canvas.add(nextLevelLabel);  
 }  
  
 */\*\*  
 \* Draws lose-level text  
 \* @param canvas the canvas to draw on  
 \*/* public static void addLoseText(GCanvas canvas) {  
 GLabel loseLabel = new GLabel("You lost!");  
 loseLabel.setLocation((Variables.*appWidth* - loseLabel.getWidth())/2, Variables.*appHeight*/2);  
 loseLabel.setFont("TimesNewRoman-20");  
 loseLabel.setColor(Color.*WHITE*);  
 canvas.add(loseLabel);  
  
 GLabel restartLabel = new GLabel("Press R to restart");  
 restartLabel.setLocation((Variables.*appWidth* - restartLabel.getWidth())/2, Variables.*appHeight*/2 + 25);  
 restartLabel.setFont("TimesNewRoman-20");  
 restartLabel.setColor(Color.*WHITE*);  
 canvas.add(restartLabel);  
 }  
  
 */\*\*  
 \* Draws endgame text  
 \* @param canvas the canvas to draw on  
 \*/* public static void addEndgameUI(GCanvas canvas){  
 GLabel endgameLabel = new GLabel("You beat the game! Love you <3");  
 endgameLabel.setFont("TimesNewRoman-24");  
 endgameLabel.setLocation((Variables.*appWidth* - endgameLabel.getWidth())/2, Variables.*appHeight*/2 - 200);  
 endgameLabel.setColor(Color.*PINK*);  
 canvas.add(endgameLabel);  
  
 GImage cat = new GImage("../images/cat.gif");  
 cat.scale(0.5, 0.5);  
 cat.setLocation((Variables.*appWidth* - cat.getWidth())/2, Variables.*appHeight*/2 - 175);  
 canvas.add(cat);  
  
 GLabel restartLabel = new GLabel("Press any key to open menu ^^");  
 restartLabel.setFont("TimesNewRoman-20");  
 restartLabel.setLocation((Variables.*appWidth* - restartLabel.getWidth())/2, Variables.*appHeight*/2 + 300);  
 restartLabel.setColor(Color.*WHITE*);  
 canvas.add(restartLabel);  
 }  
}

Variables.java

import acm.graphics.GImage;  
import acm.util.RandomGenerator;  
import acm.util.SoundClip;  
import java.awt.\*;  
import java.util.ArrayList;  
  
public class Variables {  
  
 //--------------------GAME--------------------  
 public static RandomGenerator *rg* = new RandomGenerator();  
  
 public static final int *appWidth* = 600;  
  
 public static final int *appHeight* = 800;  
  
 public static final int *paddleWidth* = 100;  
  
 public static final int *paddleHeight* = 10;  
  
 public static double *mouseX*;  
  
 public static double *mouseY*;  
  
 public static Color *backgroundColor* = Color.*BLACK*;  
  
 public static int *level* = 1;  
  
 public static int *score*;  
  
 public static int *lives* = 9;  
  
 public static int getLives() {  
 switch (*level*) {  
 case 1:  
 return 5;  
 case 2:  
 return 3;  
 case 3:  
 return 2;  
 }  
 return 0;  
 }  
  
 public static ArrayList<GImage> *hearts* = new ArrayList<>();  
 //--------------------BRICKS--------------------  
 public static int *bricksPerRow* = 5;  
  
 public static int *rows* = 10;  
  
 public static int *brickDelta* = 10;  
  
 public static int *brickWidth* = 75;  
  
 public static int *brickHeight* = 25;  
  
 public static int *brickYOffset* = 70;  
  
 public static int *brickCount*;  
 //--------------------BALL--------------------  
 public static final int *radius* = 15;  
  
 //--------------------FLAGS--------------------  
 public static boolean *shouldOpenMenu* = true;  
  
 public static boolean *shouldRestart* = false;  
  
 public static boolean *shouldOpenLevel* = false;  
  
 public static boolean *gameOver* = false;  
  
 public static boolean *won* = false;  
  
 //--------------------SOUNDS--------------------  
 public static SoundClip *bounceSound* = new SoundClip("../sounds/click.wav");  
  
 public static SoundClip *brickSound* = new SoundClip("../sounds/break.wav");  
  
 public static SoundClip *loseLifeSound* = new SoundClip("../sounds/loseLife.wav");  
  
 public static SoundClip *winSound* = new SoundClip("../sounds/win.wav");  
  
 public static SoundClip *loseSound* = new SoundClip("../sounds/lose.wav");  
}