Arkanoid – Dokumentacja Projektu

Spis treści

Główne założenia aplikacji:

Aplikacja jest wzorowana na popularnej grze Arkanoid z lat 90. Poprzez sterowaniem strzałkami na klawiaturze zmieniamy położenie naszej paletki tak aby odbiła spadającą piłeczkę. Jeśli nam się to uda piłka wędruje w górę i powinna strącić kolejny kafelek. W przypadku jeśli nie odbijemy piłeczki przegrywamy. Gra kończy zakończy się wygraną tylko jeśli strącimy wszystkie kafelki.

Opis klas utworzonych w projekcie

Game.java – klasa tworząca główny interfejs użytkownika

```
private void startBTActionPerformed(java.awt.event.ActionEvent evt) {
    Breakout breakoutFrame = new Breakout();
    breakoutFrame.setVisible(true);
    this.dispose();
}

private void settingsBTActionPerformed(java.awt.event.ActionEvent evt) {
    Settings settingsFrame = new Settings();
    settingsFrame.setVisible(true);
    settingsFrame.pack();
    this.dispose();
}

private void QuitBTActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}
```

Klasa odpowiedzialna za główny interfejs użytkownika. Służy wyłącznie do nawigacji i posiada 3 funkcje kolejno odpowiedzialne za: start gry, przejście do ustawień gry, opuszczenie gry.

Settings.java – klasa odpowiedzialna za ustawienia gry

```
private void saveBTActionPerformed(java.awt.event.ActionEvent evt) {
    FileWriter myWriter;
    try {
        myWriter = new FileWriter("difficulty.txt");
        if(easyRB.isSelected())
        {
        myWriter.write("easy");
        }else if(mediumRB.isSelected()){
        myWriter.write("medium");
        }else if(hardRB.isSelected()){
        myWriter.write("hard");
        }
        myWriter.close();
    } catch (IOException ex) {
        Logger.getLogger(Settings.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        myWriter = new FileWriter("paddlecolor.txt");
        if(blueRB.isSelected())
        {
```

```
        myWriter.write("blue");
        }else if(redRB.isSelected()){
        myWriter.write("red");
        }else if(greenRB.isSelected()){
        myWriter.write("green");
        }
        myWriter.close();
    } catch (IOException ex) {
        Logger.getLogger(Settings.class.getName()).log(Level.SEVERE, null, ex);
    }
    Game gameFrame = new Game();
    gameFrame.setVisible(true);
    this.dispose();
}
```

Posiadamy tutaj dwie główne metody pierwsza z nich odpowiedzialna jest za sczytywanie opcji ustawionych przez użytkownika, a następnie zapisanie ich do pliku.

```
public void set(){
    File myObj;
    try{
    myObj = new File("difficulty.txt");
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
            level = myReader.nextLine();
        }
        myReader.close();
    }catch(IOException ex) {
        Logger.getLogger(Settings.class.getName()).log(Level.SEVERE, null, ex);
    }
    try{
    myObj = new File("paddlecolor.txt");
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
            colorPaddle = myReader.nextLine();
        }
        myReader.close();
    }catch(IOException ex) {
        Logger.getLogger(Settings.class.getName()).log(Level.SEVERE, null, ex);
    }
    switch(level){
        case "easy": easyRB.setSelected(true);break;
        case "medium": mediumRB.setSelected(true);break;
        case "hard": hardRB.setSelected(true);break;
        default: {
        easyRB.setSelected(true);break;
        }
    }
    switch(colorPaddle){
        case "blue": blueRB.setSelected(true);break;
        case "red": redRB.setSelected(true);break;
        case "green": greenRB.setSelected(true);break;
    }
}
```

Druga metoda działa bardzo podobnie lecz w drugą stronę. Z wcześniej już utworzonych plików sczytujemy jakie opcje zapisał poprzednio użytkownik i na ich podstawie ustawiamy odpowiednie pola. Metoda ta jest wywoływana na samym początku klasy tak aby po załadowaniu użytkownik widział już zaznaczone pola zgodne z poprzednim zapisem.

Breakout.java – klasa wywołująca

```
private void initUI() {

    add(new Board());
    setTitle("Arkanoid");

    setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    this.setLocation(dim.width/2-this.getSize().width/2-155, dim.height/2-
this.getSize().height/2-215);
    setResizable(false);
    pack();
}
```

Jest to klasa odpowiedzialna za utworzenie nowej ramki oraz wywołanie naszej głównej klasy samej gry. Ustawiamy tutaj jeszcze podstawowe opcje takie jak brak możliwości zamknięcia okna przez X jak i brak możliwości zmiany wielkości naszego okienka.

Board.java – Główna klasa odpowiedzialna za inicjalizacje gry

```
private void initBoard() {
    JButton button = new JButton("Go to menu");
    button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            Game gameFrame = new Game();
            gameFrame.setVisible(true);
            gameFrame.pack();
            setVisible(false);

        }
    });
    add(button);
    addKeyListener(new TAdapter());
    setFocusable(true);
    setPreferredSize(new Dimension(Commons.WIDTH, Commons.HEIGHT));
    File myObj;
    try{
        myObj = new File("difficulty.txt");
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
            level = myReader.nextLine();
        }
        myReader.close();
    }catch(IOException ex) {
        Logger.getLogger(Settings.class.getName()).log(Level.SEVERE, null, ex);
    }
```

```
    gameInit();
}
```

Pierwsza metoda wywoławcza naszej klasy. Służy do „obudzenia” najważniejszy komponentów aby cała gra zaczęła prawidłowo działać. To tutaj również sczytujemy poziom trudności ustawiony przez użytkownika.

```
private void gameInit() {
    ball = new Ball();
    paddle = new Paddle();

    int k = 0;
    int sizeI=3;
    size =18;
    if(level.equals("medium")){
        sizeI=10;
        size = 60;
    }
    if(level.equals("hard")){
        sizeI=15;
        size = 90;
    }
    bricks = new Brick[size];
    Random r = new Random();
    int rn = 0;
    boolean bonus = false;
    for (int i = 0; i < sizeI; i++) {
        for (int j = 0; j < 6; j++) {
            rn = r.nextInt() % 10;
            if(rn==1){
                bonus=true;
            }
            bricks[k] = new Brick(j * 40 + 30, i * 10 + 50,bonus);
            bonus=false;
            k++;
        }
    }

    timer = new Timer(Commons.PERIOD, new GameCycle());
    timer.start();
}
```

W tej klasie wywołujemy poszczególne elementy gry takie jak paletka, piłka i kafelki. Na podstawie wcześniej już zapisanego poziomu trudności ustawiamy odpowiednią ilość kafelek do zbicia. Podczas tworzenia się kafelek losowana jest również szansa na uzyskanie bonusowej kafelki. Na samym końcu tworzymy nasz Cykl Życia naszej gry.

```
private class GameCycle implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {

        doGameCycle();
```

```
    }
}

private void doGameCycle() {

    ball.move();
    paddle.move();
    checkCollision();
    repaint();
}
```

Cykl ten jest wywoływany co zadany interwał czasowy i to on steruje naszą grą. Odpowiedzialny jest za wywołanie ruchu przez komponenty gry oraz sprawdzanie kolizji.

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);

    var g2d = (Graphics2D) g;

    if (inGame) {

        drawObjects(g2d);
    } else {

        gameFinished(g2d);
    }

    Toolkit.getDefaultToolkit().sync();
}
```

Na podstawie tego czy gra dalej trwa czy też została już zakończona wywoływana jest odpowiednia metoda za rysowanie danych widoków.

```
private void drawObjects(Graphics2D g2d) {
    var font = new Font("Verdana", Font.BOLD, 10);
    g2d.setColor(Color.BLACK);
    g2d.setFont(font);

    g2d.drawString("Points: "+points,
            5,
            20);

    g2d.drawImage(ball.getImage(), ball.getX(), ball.getY(),
            ball.getImageWidth(), ball.getImageHeight(), this);
    if(bonus==false) {
        g2d.drawImage(paddle.getImage(), paddle.getX(), paddle.getY(),
                paddle.getImageWidth(), paddle.getImageHeight(), this);
    }else if(bonus==true&&time<=600){
        var ii = new ImageIcon("src/resources/bonuspaddle.png");
        g2d.drawImage(ii.getImage(), paddle.getX(), paddle.getY(),
                80, paddle.getImageHeight(), this);
        time++;
    }else if(time>600){
        time =0;
```

```
            bonus=false;
        }

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

            if (!bricks[i].isDestroyed()) {

                g2d.drawImage(bricks[i].getImage(), bricks[i].getX(),
                        bricks[i].getY(), bricks[i].getImageWidth(),
                        bricks[i].getImageHeight(), this);
            }
        }
    }
```

Metoda odpowiedzialna za rysowanie całego widoku takich jak punkty, piłka, paletka oraz kafelki.

```
    private void gameFinished(Graphics2D g2d) {

        var font = new Font("Verdana", Font.BOLD, 18);
        FontMetrics fontMetrics = this.getFontMetrics(font);
        g2d.setColor(Color.BLACK);
        g2d.setFont(font);

        g2d.drawString(message,
                (Commons.WIDTH - fontMetrics.stringWidth(message)) / 2,
                Commons.WIDTH / 2);
        g2d.drawString("On difficulty: "+level,
                (Commons.WIDTH - fontMetrics.stringWidth(message)) / 2 -30,
                Commons.WIDTH / 2+50);

    }
```

Drugą z opcji wywołania przez paintComponent jest końcowy widok po zakończeniu rozgrywki.

```
    private void checkCollision() {

        if (ball.getRect().getMaxY() > Commons.BOTTOM_EDGE) {

            stopGame();
        }

        for (int i = 0, j = 0; i < size; i++) {

            if (bricks[i].isDestroyed()) {

                j++;
            }

            if (j == size) {

                message = "Victory";
                stopGame();
            }
```

```
}
int paddleLPosY = (int) paddle.getRect().getMinY();
int ballLPosY = (int) ball.getRect().getMinY();
int paddleLPos = (int) paddle.getRect().getMinX();
int ballLPos = (int) ball.getRect().getMinX();
int first = paddleLPos + 8;
int second = paddleLPos + 16;
int third = paddleLPos + 24;
int fourth = paddleLPos + 32;
int last = paddleLPos +40;
if(bonus==true){
   first = paddleLPos + 16;
   second = paddleLPos + 32;
   third = paddleLPos + 48;
   fourth = paddleLPos + 64;
   last = paddleLPos +80;
}
if(ballLPosY>=paddleLPosY && ballLPos>=paddleLPos && ballLPos<=last){
   if (ballLPos < first) {
      speed++;
      ball.setXDir(-1*speed);
      ball.setYDir(-1*speed);

   }

   if (ballLPos >= first && ballLPos < second) {

      ball.setXDir(-1);
      ball.setYDir(-1 * ball.getYDir());
   }

   if (ballLPos >= second && ballLPos < third) {

      ball.setXDir(0);
      ball.setYDir(-1);
   }

   if (ballLPos >= third && ballLPos < fourth) {

      ball.setXDir(1);
      ball.setYDir(-1 * ball.getYDir());
   }

   if (ballLPos > fourth) {
      speed++;
      ball.setXDir(1*speed);
      ball.setYDir(-1*speed);

   }
}

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

   if ((ball.getRect()).intersects(bricks[i].getRect())) {

      int ballLeft = (int) ball.getRect().getMinX();
      int ballHeight = (int) ball.getRect().getHeight();
      int ballWidth = (int) ball.getRect().getWidth();
      int ballTop = (int) ball.getRect().getMinY();
```

```
            var pointRight = new Point(ballLeft + ballWidth + 1, ballTop);
            var pointLeft = new Point(ballLeft - 1, ballTop);
            var pointTop = new Point(ballLeft, ballTop - 1);
            var pointBottom = new Point(ballLeft, ballTop + ballHeight + 1);

            if (!bricks[i].isDestroyed()) {

                if (bricks[i].getRect().contains(pointRight)) {

                    ball.setXDir(-1*speed);
                } else if (bricks[i].getRect().contains(pointLeft)) {

                    ball.setXDir(1*speed);
                }

                if (bricks[i].getRect().contains(pointTop)) {

                    ball.setYDir(1*speed);
                } else if (bricks[i].getRect().contains(pointBottom)) {

                    ball.setYDir(-1*speed);
                }

                bricks[i].setDestroyed(true);
                points++;
                if(bricks[i].bonusbrick==true){
                    bonus=true;
                }
            }
        }
    }
}
```

Ostatnia i zarazem najdłuższa metoda zawarta w Board.java odpowiada za sprawdzanie kolizji w grze. Tutaj również sprawdzamy czy wszystkie kafelki zostały zbite.

Sprite.java

```
public class Sprite {

    int x;
    int y;
    int imageWidth;
    int imageHeight;
    Image image;

    protected void setX(int x) {

        this.x = x;
    }

    int getX() {

        return x;
    }
```

```
    protected void setY(int y) {

        this.y = y;
    }

    int getY() {

        return y;
    }

    int getImageWidth() {

        return imageWidth;
    }

    int getImageHeight() {

        return imageHeight;
    }

    Image getImage() {

        return image;
    }

    Rectangle getRect() {

        return new Rectangle(x, y,
                image.getWidth(null), image.getHeight(null));
    }

    void getImageDimensions() {

        imageWidth = image.getWidth(null);
        imageHeight = image.getHeight(null);
    }
}
```

Klasa wspomagająca dla komponentów gry. Każdy z komponentów dziedziczy po tej klasie, a sama klasa jest odpowiedzialna za gettery oraz setery w naszej aplikacji.

Paddle.java – Obiekt naszej paletki

```
1)public Paddle() {
        File myObj;
        try{
          myObj = new File("paddlecolor.txt");
          Scanner myReader = new Scanner(myObj);
          while (myReader.hasNextLine()) {
            color = myReader.nextLine();
          }
          myReader.close();
        }catch(IOException ex) {
          Logger.getLogger(Settings.class.getName()).log(Level.SEVERE, null, ex);
        }
```

```
        initPaddle();
    }

    private void initPaddle() {

        loadImage();
        getImageDimensions();

        resetState();
    }

    private void loadImage() {
        var ii = new ImageIcon("src/resources/paddle.png");
        if(color.equals("red")){
            ii = new ImageIcon("src/resources/redpaddle.png");
        }else if(color.equals("green")){
            ii = new ImageIcon("src/resources/greenpaddle.png");
        }
        image = ii.getImage();
    }
```

Sczytywanie z pliku jaki kolor został ustawiony przez naszego użytkownika, a następnie ustawienie odpowiedniego koloru paletki.

```
void move() {

    x += speedX;

    if (x <= 0) {

        x = 0;
    }

    if (x >= Commons.WIDTH - imageWidth) {

        x = Commons.WIDTH - imageWidth;
    }
}
```

Poruszanie się paletki i blokowanie ruchu przy krańcach ekranu przez sprawdzanie pozycji.

Ball.java – Obiekt piłki

```
void move() {

    x += moveX;
    y += moveY;

    if (x <= 0) {
        if (moveX<=0) moveX=-moveX;

        setXDir(moveX);
    }

    if (x >= Commons.WIDTH - imageWidth) {
        setXDir(-moveX);
    }
```

```
    if (y <= 0) {
        if(moveY<=0) moveY=-moveY;
        setYDir(moveY);
    }
}
```

Sprawdzanie kolizji ze ściankami naszego okienka gry.

Brick.java – Obiekt pojedynczej kafelki

```
private void initBrick(int x, int y,boolean bonus) {
    this.x = x;
    this.y = y;
    bonusbrick=bonus;

    isDestroyed = false;

    loadImage();
    getImageDimensions();
}

private void loadImage() {
    var ii = new ImageIcon("src/resources/brick.png");
    if(bonusbrick==true){
        ii = new ImageIcon("src/resources/brickbonus.png");
    }
    image = ii.getImage();
}

boolean isDestroyed() {
    return isDestroyed;
}

void setDestroyed(boolean val) {
    isDestroyed = val;
}
```

Generowanie kafelki przez ustawienie jej położenia i obrazu. Sprawdzanie i ustawienie zniszczenia kafelki jeśli to potrzebne.

Commons.java – Pomocniczy interface

```
public interface Commons {

    int WIDTH = 300;
    int HEIGHT = 400;
    int BOTTOM_EDGE = 390;
    int INIT_PADDLE_X = 200;
    int INIT_PADDLE_Y = 360;
    int INIT_BALL_X = 230;
    int INIT_BALL_Y = 355;
    int PERIOD = 10;
}
```

Interface posiadający zapisane dane statyczne wykorzystujące głównie przez klasę Board.java. Miedzy innymi znajdziemy tu definicje wielkości okna gry czy też interwał czasowy w naszym cyklu życia.