Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

**Звіт із лабораторної роботи №2**



**Виконала:** студентка 1 курсу

Тітарчук Вероніка Олександрівна

**Викладач:**

ст. викл. Кирієнко Оксана Валентинівна

Київ – 2018

Зміст

[Постановка задачі 3](#_Toc38300326)

[Структура програми 5](#_Toc38300327)

[Опис методів і класів 6](#_Toc38300328)

[Інструкція 7](#_Toc38300329)

[Проблеми 11](#_Toc38300330)

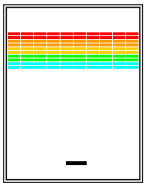
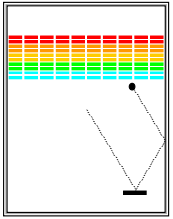
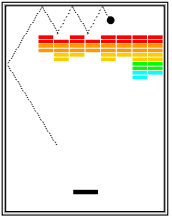
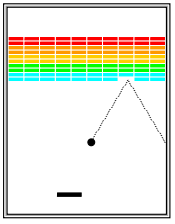
[Програмний код 12](#_Toc38300331)

# Постановка задачі

**Завдання для лабораторної**

Вашим завданням буде написати класичну аркадну гру Breakout, яку розробив Стів Возняк перед заснуванням компанії Apple разом із Стівом Джобсом.

**Гра Breakout**

В Breakout світ виглядає, як показано на малюнках вище. Кольорові чотирикутники вгорі – цеглини, трохи більший, що знизу – ракетка. Ракетка знаходить у фіксованій позиції у вертикальній площині, але може пересуватися в горизонтальній, обмежена стінками.

Гра складається з трьох змін напрямку. Кожного разу м’яч запускається з центру вікна вниз під випадковим кутом. Далі він відбивається від ракетки та стін ігрового світу відповідно до фізичного принципу – «кут падіння дорівнює куту відбивання». Таким чином, після відбивання від ракетки та стін, м’яч має траєкторію, що показана на другому малюнку (лінія, яка показую траєкторію руху м’яча не має відображатися).

Як ви можете бачити, м’яч відбивається від однієї з нижніх цеглин, під час цього – вона зникає. Третій малюнок показує, що відбувається потім.

Гра продовжується до тих пір, поки не виконається одна з наступних умов:

1. Коли м’яч попадає на нижню стінку, що значить, що гравець його не відбив. В цьому випадку дається ще одне «життя» (ще один м’яч), але якщо вони закінчилися, то гра припиняється з програшем.
2. Коли відбита остання цеглина. В цьому випадку гравець виграв.

Після того, як певна кількість колонок розбита, відкривається простір до верхньої стінки. Коли таке трапляється, то м’яч може перелетіти на верхній рівень. На останньому малюнку наведена ілюстрація даної ситуації.

# Структура програми

Були використані наступні бібліотеки:

* java.awt.Color;
* java.awt.event.\*;
* acm.graphics.\*;
* acm.program.GraphicsProgram;
* acm.util.RandomGenerator;

Створено і використано 14 констант, 13 методів. Також у проекті є зображення формату gif.

Назва проекту – LB2Titarchuk. Програма написана в одному класі - Breakout. Метод run 2 власне запускає гру, а в методі init() ініціалізується початков вікно.

Структура програми виглядає наступним чином:

Запускається вікно. Виконується метод init(), що відразу створює об’єкти у вікні:

* background();
* paddle((APPLICATION\_WIDTH - PADDLE\_WIDTH) / 2, APPLICATION\_HEIGHT - PADDLE\_Y\_OFFSET);
* brickets();
* addMouseListeners();
* addKeyListeners().

Потім запускається метод run(), в якому встановлюються розміри вікна та запускається сама гра. Сама гра працює допоки змінна isFinished не стане true. У циклі while(!isFinished) Виконуються наступні методи:

* moveBall();
* detector();
* checkForBallCollision();

Коли isFinished стане true, тоді виводиться на екран результат(виграш/програш) та видаляється м’яч та ракетка.

# Опис методів і класів

Назва проекту – LB2Titarchuk;

Програма написана в одному класі - Breakout;

run() - Запускає всю програму.

init() - Викликає методи, які встановлюють всі об’єкти на екран;

background() - додавання GIF зображення на задній план;

brickets() - малює цеглинки;

paddle() - малює ракетку;

ball() - малює м’ячик;

addMouseListeners() - додає можливіть впливати на гру за допомогою миші;

mousePressed() - Дає змогу ракетці рухатися вслід за мишкою лише по вісі X;

mouseDragged() – злідкує за переміщенням курсора;

addKeyListeners(); Додає можливіть впливати на гру за допомогою клавіатури;

game(); Запускає головну частину гри, яка буде зациклена за допомогою delay;

detector(); Перевіряє, з якими об’єктами та якими сторонами описаного навколо кульки прямокутника стикається цей прямокутник, та відповідно змінює координатний рух кульки.

removeElement(GPoint point) - метод, який видаляє об’єкт, отриманий у даній точці за допомогою метода GetElementAt(), якщо це цеглинка. Бере на вхід GPoint.

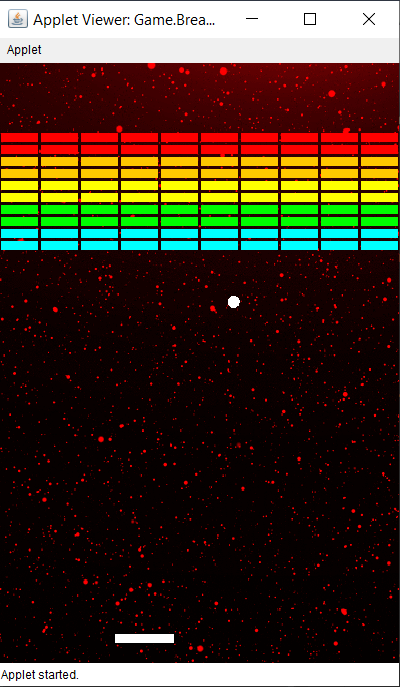
KeyTyped() - дозволяє збільшити швидкість при натисненні на клавішу.

Collide ()- boolean метод перевіряє, чи існує в точці об’єкт відмінний від null. На вхід бере GPoint.

moveBall()- рухає м’яч на vx по осі Ох і на vy по осі Оу.

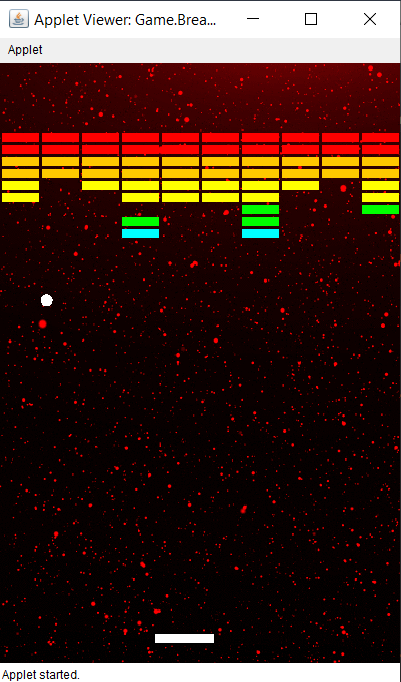
# Інструкція

При запуску з’являється вікно з грою. Гра запускається самостійно. М’яч летить у напрямку ракетки з випадковою швидкістю і напрямком.



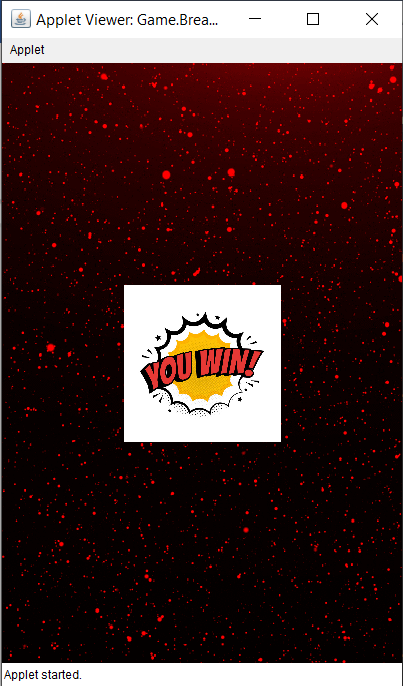
Затиснувши курсор мишки, рухайте вправо та вліво, для переміщення ракетки та намагайтесь відбити м’яча.

Середині гра виглядає приблизно так:



Якщо гравець не зміг відбити кульку м’ячик з’являється у центрі ігрового поля і починає рухатися з згенерованою швидкістю і напрямком, як і на початку гри.

**Якщо гравець збив усі блоки, виводиться привітання і гра припиняється.**



**Якщо йому це не вдалося, а спроби вичерпані -**

**Виводиться повідомлення про поразку.**

****

# Проблеми

Виникли проблеми із визначенням зіткнень кульки адже GOval в acm задає вписаний в прямокутник овал, Тому було неможливо напряму брати точки на кульці. Було прийнято брати точки на квадраті, адже кулька маленька і розмірами можна було знехтувати. Тому що на око помітити різницю у відбиванні практично не можливо.

Також було врахувано різницю у законах відбивання від стін, ракетки та цеглинок.

При зіткненні із рамкою просто змінюється відповідно швидкість лише y при зіткненні із верхньою стінкою та лише по y.

При зіткненні із лівою із чи правою стінко, лише по х.

Але від нижньою стінки м’ячик не має відбиватися, тому він просто ремувається.

Також окремо прораховане відбиття від ракетки: кут падіння дорівнює куту відбиття, але для цікавості гри і можливості хоча б якось впливати на гру, додано властивість відбивання від краю ракетки: якщо м’ячик падає на край ракетки то змінюється х=-х, y=-y, так що м’ячик починає летіти по тій самій прямій у протилежному напрямку.

Але найтяжчим було реалізувати зіткення із цеглинками. Тому що ми мали рахувати відбиття від різних боків цеглинок та різними сторонами м’яча. А також через те, що м’яч рухався зі статичною швидкістю, іноді всі 4 його краї могли не потряпляти на краї цеглинки, а відразу потрапити в середину цеглинки, де проконтролювати її дії майже не можливо.

Також була проблема з реалізацією руху ракетки, оскільки хотілося, щоб ракетка рухалася саме при затиснутому клівою клавішею мишки. І тому спочатку не вдавалося. Не було зрозуміло як саме використати addMouseListeners(), але розібравшись саме з роботою цього методу – все стало на свої місця.

# Програмний код

package Game;

/\*

\* File: Breakout.java

\* -------------------

\* Name: Veronika Titarchuk

\* Section Leader:

\*

\* This file will eventually implement the game of Breakout.

\*/

import acm.graphics.\*;

import acm.program.\*;

import acm.util.\*;

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

public class Breakout extends GraphicsProgram {

/\*\* Width and height of application window in pixels \*/

public static final int APPLICATION\_WIDTH = 400;

public static final int APPLICATION\_HEIGHT = 600;

/\*\* Dimensions of the paddle \*/

private static final int PADDLE\_WIDTH = 60;

private static final int PADDLE\_HEIGHT = 10;

/\*\* Offset of the paddle up from the bottom \*/

private static final int PADDLE\_Y\_OFFSET = 30;

/\*\* Number of bricks per row \*/

private static final int NBRICKS\_PER\_ROW = 10;

/\*\* Number of rows of bricks \*/

private static final int NBRICK\_ROWS = 10;

/\*\* Separation between bricks \*/

private static final int BRICK\_SEP = 4;

/\*\* Width of a brick \*/

private static final int BRICK\_WIDTH = (APPLICATION\_WIDTH - (NBRICKS\_PER\_ROW - 1) \* BRICK\_SEP) / NBRICKS\_PER\_ROW;

/\*\* Height of a brick \*/

private static final int BRICK\_HEIGHT = 8;

/\*\* Radius of the ball in pixels \*/

private static final int BALL\_RADIUS = 10;

/\*\* Offset of the top brick row from the top \*/

private static final int BRICK\_Y\_OFFSET = 70;

/\*\* Number of turns \*/

private static final int NTURNS = 2;

private double vx, vy;

private RandomGenerator rgen = RandomGenerator.getInstance();

private int NumOfTheTurn;

private boolean isFinished;

private int counterOfBricks;

private static final char c = 32;

/\* Method: run() \*/

/\*\* Runs the Breakout program. \*/

public void run() {

/\* You fill this in, along with any subsidiary methods \*/

this.setSize(APPLICATION\_WIDTH, APPLICATION\_HEIGHT);

game();

}

public void init() {

NumOfTheTurn = 0;

isFinished = false;

counterOfBricks = NBRICK\_ROWS \* NBRICKS\_PER\_ROW;

background();

paddle((APPLICATION\_WIDTH - PADDLE\_WIDTH) / 2, APPLICATION\_HEIGHT - PADDLE\_Y\_OFFSET);

brickets();

addMouseListeners();

addKeyListeners();

}

/\*\*

\* Body of the game

\*/

public void game() {

while (!isFinished) {

if (ball != null) {

moveBall();

detector();

checkForBallCollision();

} else {

pause(500);

if (vy < 0)

vy = -vy;

ball();

}

}

if (ball != null) {

remove(ball);

}

remove(paddle);

if (counterOfBricks == 0) {

win = new GImage("D:\\Eclipse workspace\\Laboratory2\\src\\Game\\win.jpg");

win.setSize(win.getWidth() / 4, win.getHeight() / 4);

add(win, (APPLICATION\_WIDTH - win.getWidth()) / 2, (APPLICATION\_HEIGHT - win.getHeight()) / 2);

} else {

lose = new GImage("D:\\Eclipse workspace\\Laboratory2\\src\\Game\\lose.jpg");

lose.setSize(lose.getWidth() / 4, lose.getHeight() / 4);

add(lose, (APPLICATION\_WIDTH - lose.getWidth()) / 2, (APPLICATION\_HEIGHT - lose.getHeight()) / 2);

}

}

/\*\*

\* Background of the App

\*/

private void background() {

background = new GImage("D:\\Eclipse workspace\\Laboratory2\\src\\Game\\background1.gif");

background.setSize(background.getWidth(), APPLICATION\_HEIGHT);

add(background, 0, 0);

}

/\*\*

\*

\* @param x - Location on Ox

\* @param y - Location on Oy

\*

\* Shape, size, color of the paddle

\*/

public void paddle(int x, int y) {

paddle = new GRect(x, y, PADDLE\_WIDTH, PADDLE\_HEIGHT);

paddle.setFilled(true);

paddle.setFillColor(Color.white);

add(paddle);

}

/\*\*

\* Creating brickets, their size, shape, color and location

\*/

public void brickets() {

for (int i = 0; i < NBRICKS\_PER\_ROW; i++) {

for (int j = 0; j < NBRICK\_ROWS; j++) {

bricket = new GRect(

(APPLICATION\_WIDTH - NBRICKS\_PER\_ROW \* BRICK\_WIDTH - (NBRICKS\_PER\_ROW - 1) \* BRICK\_SEP) / 2

+ i \* (BRICK\_SEP + BRICK\_WIDTH),

BRICK\_Y\_OFFSET + j \* (BRICK\_HEIGHT + BRICK\_SEP), BRICK\_WIDTH, BRICK\_HEIGHT);

bricket.setFilled(true);

if (j == 0 || j == 1) {

bricket.setFillColor(Color.RED);

bricket.setColor(Color.RED);

}

if (j == 2 || j == 3) {

bricket.setFillColor(Color.ORANGE);

bricket.setColor(Color.ORANGE);

}

if (j == 4 || j == 5) {

bricket.setFillColor(Color.YELLOW);

bricket.setColor(Color.YELLOW);

}

if (j == 6 || j == 7) {

bricket.setFillColor(Color.GREEN);

bricket.setColor(Color.GREEN);

}

if (j == 8 || j == 9) {

bricket.setFillColor(Color.CYAN);

bricket.setColor(Color.CYAN);

}

add(bricket);

}

}

}

/\*\*

\* Creating ball in the App. Size, shape, color and speed

\*/

public void ball() {

if (!(NumOfTheTurn == NTURNS)) {

ball = new GOval((APPLICATION\_WIDTH - BALL\_RADIUS) / 2, (APPLICATION\_HEIGHT - BALL\_RADIUS) / 2, BALL\_RADIUS,

BALL\_RADIUS);

ball.setFilled(true);

ball.setColor(Color.white);

ball.setFillColor(Color.WHITE);

add(ball);

vx = rgen.nextDouble(1.0, 3.0);

if (rgen.nextBoolean(0.5))

vx = -vx;

vy = rgen.nextDouble(1.0, 3.0);

NumOfTheTurn++;

}

}

public void mousePressed(MouseEvent e) {

last = new GPoint(e.getPoint());

}

public void mouseDragged(MouseEvent e) {

if (paddle.getX() <= APPLICATION\_WIDTH - PADDLE\_WIDTH && paddle.getX() >= 0) {

paddle.move(e.getX() - last.getX(), 0);

last = new GPoint(e.getPoint());

} else {

if (paddle.getX() > APPLICATION\_WIDTH - PADDLE\_WIDTH)

paddle.move(APPLICATION\_WIDTH - PADDLE\_WIDTH - paddle.getX(), 0);

if (paddle.getX() < 0)

paddle.move(-paddle.getX(), 0);

}

}

public void keyTyped(KeyEvent e) {

if (e.getKeyChar() == c) {

vx \*= 1.1;

vy \*= 1.1;

}

}

/\*\*

\* Ball moving and finding dots of the ball

\*/

private void moveBall() {

if (ball != null) {

ball.move(vx, vy);

pause(10);

ballUpperLeft = new GPoint(ball.getX(), ball.getY());

ballUpperRight = new GPoint(ball.getX() + BALL\_RADIUS, ball.getY());

ballLowerLeft = new GPoint(ball.getX(), ball.getY() + BALL\_RADIUS);

ballLowerRight = new GPoint(ball.getX() + BALL\_RADIUS, ball.getY() + BALL\_RADIUS);

}

}

private boolean collide(GPoint point) {

return (getElementAt(point) != null

&& getElementAt(point) != background);

}

/\*\*

\* Changing direction depending on an angle of collision.

\*/

private void detector() {

boolean collisionTop = collide(ballUpperLeft) && collide(ballUpperRight);

boolean collisionBottom = collide(ballLowerLeft) && collide(ballLowerRight);

boolean collisionLeft = collide(ballUpperLeft) && collide(ballLowerLeft);

boolean collisionRight = collide(ballUpperRight) && collide(ballLowerRight);

boolean collisionAngle = collide(ballUpperRight) || collide(ballUpperLeft)

|| collide(ballLowerLeft) || collide(ballLowerRight);

// remove bricks

removeElement(ballUpperLeft);

removeElement(ballUpperRight);

removeElement(ballLowerLeft);

removeElement(ballLowerRight);

// frame

if (ball.getX() <= 0

|| ball.getX() + ball.getWidth() >= APPLICATION\_WIDTH){

vx = -vx;

}

else if (ball.getY() <= 0) {

vy = -vy;

}

//

else if (collisionTop) {

vy = -vy;

removeElement(ballUpperLeft);

} else if (collisionBottom) {

vy = -vy;

removeElement(ballLowerLeft);

} else if (collisionLeft) {

vx = -vx;

removeElement(ballLowerLeft);

} else if (collisionRight) {

vx = -vx;

removeElement(ballLowerRight);

} else if (collisionAngle) {

vx = -vx;

vy = -vy;

}

/\*

\* paddle fix

\*/

if (ball.getY() + ball.getHeight() >= paddle.getY()

&& ball.getY() <= paddle.getY() + paddle.getHeight()

&& ball.getX() + ball.getWidth() >= paddle.getX()

&& ball.getX() <= paddle.getX() + paddle.getWidth())

ball.setLocation(ball.getX(), paddle.getY() - ball.getHeight());

}

/\*\*

\* Method for deleting brick after collision.

\*

\* @param point

\* of collision

\*/

private void removeElement(GPoint point) {

// defining colliding object.

GObject collide = getElementAt(point);

// collisions with bricks

if (collide != null && collide != paddle && collide!=background) {

remove(collide);

counterOfBricks--;

if(counterOfBricks==0)

isFinished = true;

}

}

private void checkForBallCollision() {

if (ball.getY() >= APPLICATION\_HEIGHT && ball != null) {

remove(ball);

ball = null;

if (NumOfTheTurn == NTURNS) {

isFinished = true;

}

}

}

GObject collide;

GImage lose;

GImage win;

GImage background;

GPoint ballUpperLeft;

GPoint ballUpperRight;

GPoint ballLowerLeft;

GPoint ballLowerRight;

GPoint last;

GOval ball;

GRect bricket;

GRect paddle;

}