Національний університет «Києво-Могилянська академія»

Факультет Інформатики

**ЗВІТ**

**З лабораторної роботи №2**

**«Гра Breakout»**

Виконали:

Студенти 1-ого курсу,

Спеціальності «Прикладна математика»

Ярошепта Богдан, Засуха Дар’я

Викладач: Кирієнко О.В.

Київ 2020

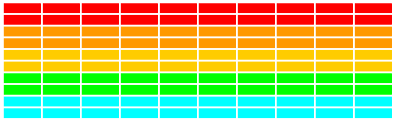
**Постановка задачі**

Написати класичну аркадну гру Break-out, яку розробив Стів Возняк. Гра складається з поля для гри, цеглинок, ракетки та м’яча. Принцип гри полягає в тому, що за допомогою мишки можна рухати ракетку, яка повинна відбивати м’яч. М’яч може відбиватися від ракетки і стінок поля, а якщо він влучає в цеглинку, то цеглинка зникає, тобто м’яч розбиває її. В гравця є три спроби, якщо м’яч впав на підлогу, то вважається, що спроба втрачена. Якщо м’яч розбив всі цеглинки, то вважається, що гру виграною.

**Розподіл задач**

Декомпозиція написання гри полягала в наступному:

1. Установка цеглин.  
   Засуха Дар’я встановлювала цеглинки, враховуючи задані константи їхнього розміру, відступу між ними та координатами в гральному полі та розробила метод їхнього рівномірного розфарбування п’ятьма кольорами (**RED, ORANGE, YELLOW, GREEN, CYAN**).



1. Створення ракетки.  
   Ярошепта Богдан створив ракетку, розробив метод її руху за курсором (курсор визначає рух середини ракетки), обмежив рух відносно границь грального поля.
2. Створення м’яча та його рух.  
   Ярошепта Богдан створив м’яч у вікні без цеглинок та ракетки, примусив його відбиватись від стінок, змінюючи напрямок руху і значення швидкості при зіткненні зі стелею та стінками, і при зіткненні з підлогою, зупинятись.
3. Перевірка колізій.

Для перевірки колізії було створено два методи Ярошептою Богданом. Перший метод перевіряв наявність зіткнення чотирьох точок м’яча з будь-яким елементом поля (текст, цеглинка чи ракетка) і повертав цей елемент у другий метод. Другий метод, перевіряв отриманий елемент, який зіткнувся з м’ячом і, відповідно до того, який цей елемент, робив наступне: якщо цеглинка - прибирав її, якщо ракетка - змушував м’яч відбиватися від ракетки, якщо текст - ігнорувати і пролітати повз нього.

1. Введення в гру кількість життів, набраних балів та виведення цієї інформації на екран – Засуха Дар’я.
2. Виведення на екран результат гри і оформлення тексту –Засуха Дар’я.

При виникненні проблем із задачами, які писались окремо, ми вирішували їх разом дистанційно, пропонуючи інші способи виконання та реалізації. Склейка методів, які писались індивідуально, робилась почергово. Обговорення, перегляд та тестування протягом всього виконання лабораторної робилось спільними зусиллями.

**Структура програми**

Користуючись константами, які контролюють ігрові параметри, такі як розмірність різних об’єктів, створюються всі ці необхідні об’єкти гри, а саме цеглинки, ракетка, м’яч та поле для гри. М’яч з’являється посередині на ракетці та відбивається від нею під довільним кутом і прямує вгору, на шляху відбиваючись від стін, цеглинок та підлоги. М’яч може відбиватись від ракетки, яка рахується за курсором, від цеглинок, знищуючи їх при колізії, від стін та стелі. Відбиваючись від інших елементів, змінює свій напрям руху на протилежний. Рухаючись таким чином, м’яч довільно змінює свою швидкість. Від підлоги м’яч може відбиватись тричі, на третій раз гра завершується програшом, якщо користувач не встиг збити всі цеглинки. По закінченню гри виводиться повідомлення результатом гри. У разі виграшу, коли всі цеглинки знищені, кількість балів при розмірі цеглинок 8x5 ХР=400.

1. Створення всіх необхідних об’єктів гри. Написання методів, де вони викликаються, використовуючи задані константи.
2. Метод рівномірного розфарбування цеглинок п’ятьма кольорами
3. Створення методу опису руху ракетки за курсором, перевіряючи рух ракетки і обмежуючи вихід за межі грального поля.
4. Створення методу опису руху м’яча. Зміна швидкості та напрямку руху після зіткнення з різними елементами.
5. Опис можливих колізій та поведінка м’яча та елементів опісля зіткнення.
6. Метод підрахунку і виводу поточних балів та кількість життів.
7. Метод виводу результату по закінченню гри з набраною кількістю балів.

**Опис методів**

Клас Breakout - це клас, в якому відбувається сама гра.  
В методі run() використано кілька методів:

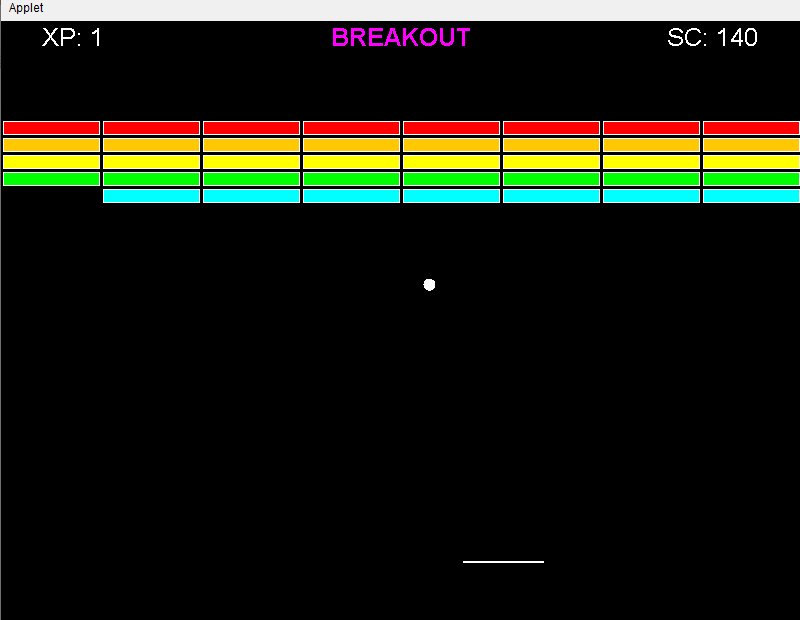
1. setup(), addScoreLabel(), addUpperLabels() - створюють та відображують такі елементи гри: поле для гри, цеглини та ракетка.
2. ballMove()- створює і відображує м’яч, запускає і продовжує рух м`яча, поки є істинним булеан “gameOver” (як тільки він приймає значення хиби(у випадку закінчення життів чи знищенні всіх цеглин), то гра закінчується)
3. checkCollision() – метод, що перевіряє, чи зіткнувся м’яч з якимосб=ь предметом на ігровому полі
4. showEndGameLabel() – метод закінчення гри, що показує написи, які сповіщають гравця, що гра закінчена.

Ці методи складаються з інших методів, які допомагають реалізувати поставлені задачі:

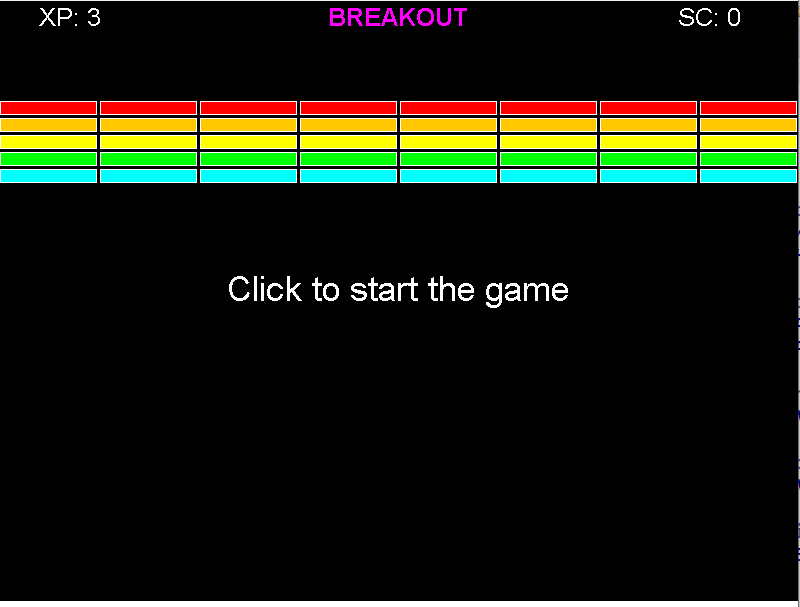
1. loserLabel() – метод, що створює напис, що сповіщає гравця, що він програв. Та розміщує його по центру вікна.
2. winnerLabel() – метод, що створює напис, який сповіщає гравця, що той переміг. Та розміщує його по центру вікна.
3. addPoints() – метод, що додає 10 очків на рахунок гравця(змінна score), якщо він розбив один прямокутник.
4. mouseMoved(MouseEvent e)-описує рух ракетки за курсором і обмежує її рух, не дозволяючи вийти за межі ігрового поля.
5. addBricks() – метод, що генерує та додає на ігрове поле всі цілі з різними кольорами(**RED, ORANGE, YELLOW, GREEN, CYAN**).
6. setNeededColor(int row) – метод, що приймає номер рядка та зафарбовує цеглину в певний колір, відповідно до кольору рядка
7. gameOver() – метод, що повертає булеанове значення в залежності від того, чи програв гравець чи ні.
8. triangle() – метод, що створює та повертає об’єкт типу GPolygon, який збільшує хітбокс ігрової ракетки.
9. Random(double min, max) – допоміжний метод, що повертає випадкове число в діапазоні від min до max.

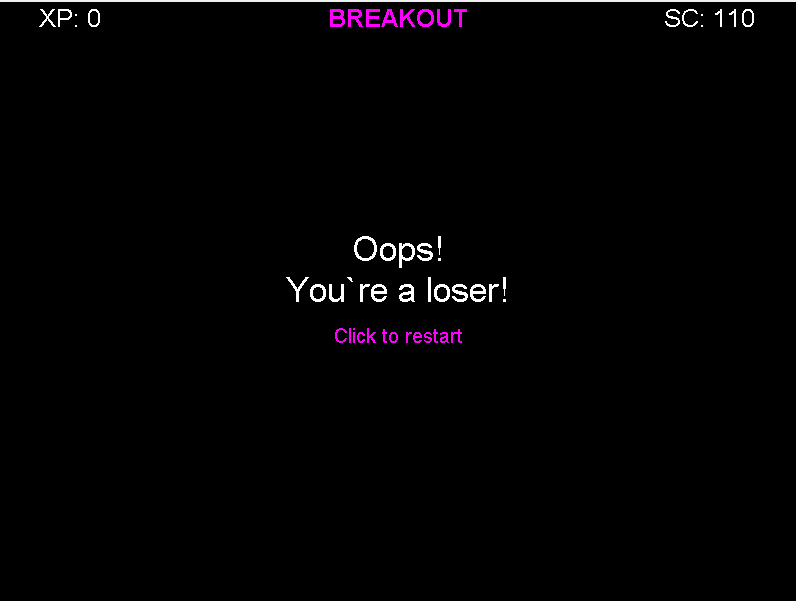
**Інструкція**

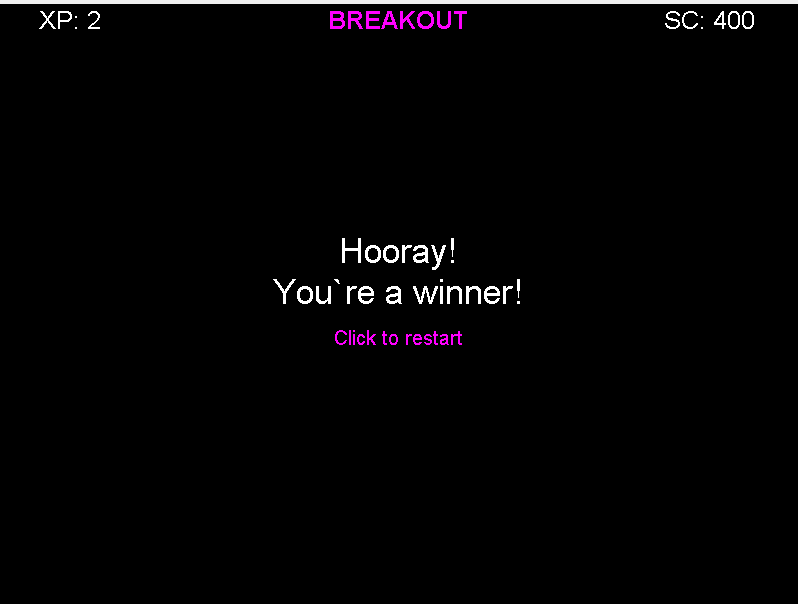
1. Для виграшу необхідно розбити всі цеглинки за три спроби.



1. Вигляд гри на початку



1. Кінець гри, коли витрачаються всі три спроби (м’ячик тричі підряд потрапляє на підлогу і не збиває всі цеглинки)
2. Кінець гри, коли гравець знищив всі цеглинки за три або менше спроб. При цьому гравець набирає 400 очок.



**Проблеми, які виникали, та шляхи їх вирішення**

1. Гра не закінчувалася перемогою, тобто boolean GameOver() не працював коректно.

Для вирішення проблеми потрібно було зменшувати змінну numberOfBricks на один після кожного зіткнення з цеглинкою.

Наприклад:

**if** (l1 != player & l1 != player1 & l1 != **null** & l1 != line) {

remove(getElementAt(l));

numberOfBricks--;

1. Після закінчення 1ого раунду гра не продовжувалася.

Для вирішення потрібно були зменшувати змінну XP на один після кожного закінчення гри.

1. М’яч збивав верхні написи з кількістю спроб, які залишилися, з поточною кількістю очок та з назвою гри.

Для вирішення проблему потрібно було провести лінію під цими написами та трохи змінити умову, аби програма ігнорувала цю лінію. Наприклад:

**if** (l1 != player & l1 != player1 & l1 != **null** & l1 != line) {

remove(getElementAt(l));

**Програмний код**

/\*

\* File: Breakout.java

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

\* Developers: Yaroshepta Bohdan, Zasukha Dasha

\*

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

\*/

**import** acm.graphics.\*;

**import** acm.program.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** Breakout **extends** GraphicsProgram {

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

**private** **static** **final** **int** ***WIDTH*** = 800;

**private** **static** **final** **int** ***HEIGHT*** = 600;

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

**private** **static** **final** **int** ***PLAYER\_WIDTH*** = 80;

**private** **static** **final** **int** ***PLAYER\_HEIGHT*** = 1;

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

**private** **static** **final** **int** ***NumberOfBricksPerRow*** = 8;

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

**private** **static** **final** **int** ***NumberRowOfBricks*** = 5;

/\*\* Separation between bricks \*/

**private** **static** **final** **int** ***BRICK\_SEPARATION*** = 4;

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

**private** **static** **final** **int** ***BRICK\_WIDTH*** = (***WIDTH*** - (***NumberOfBricksPerRow*** - 1) \* ***BRICK\_SEPARATION***)

/ ***NumberOfBricksPerRow***;

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

**private** **static** **final** **int** ***BRICK\_HEIGHT*** = 13;

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

**private** **static** **final** **int** ***BALL\_DIAM*** = 10;

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

**private** **static** **final** **int** ***BRICK\_Y\_OFFSET*** = 100;

/\*\* Number of turns (lives) \*/

**private** **static** **int** *XP* = 3;

/\*\* whole amount of bricks \*/

**private** **int** numberOfBricks = ***NumberOfBricksPerRow*** \* ***NumberRowOfBricks***;

/\*\* value of balls` speed \*/

**private** **double** ballSpeed = 1.5;

/\*\* ball speed by X \*/

**private** **double** ballSpeedX = 1;

/\*\* ball speed by Y \*/

**private** **double** ballSpeedY = 1;

/\*\* value of speed increase of the ball \*/

**private** **double** speedIncrease = 1.0;

/\*\* value of player`s score \*/

**private** **int** score = 0;

/\*\* line to separate upper labels and game field \*/

**private** GRect line;

/\*\* label to show score \*/

**private** GLabel SCOREshow;

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

**public** **void** run() {

**this**.setSize(***WIDTH***, ***HEIGHT***);

**this**.setBackground(Color.***black***);

/\* cycle of a game \*/

**while** (**true**) {

addScoreLabel();

addUpperLabels();

setup();

**while** (!gameOver()) {

ballMove();

checkCollision();

// player.setLocation(ball.getX() + BALL\_DIAM - PLAYER\_WIDTH/2, player.getY());

// player1.setLocation(ball.getX() + BALL\_DIAM - PLAYER\_WIDTH/2,

// player1.getY());

pause(5);

}

showEndGameLabel();

**if** (*XP* == 0) {

*XP* = 3;

score = 0;

}

}

}

/\*\* adds a label with score after every collision \*/

**private** **void** addScoreLabel() {

SCOREshow = **new** GLabel("SC: " + score);

SCOREshow.setFont("Arial-25");

SCOREshow.setColor(Color.***WHITE***);

**double** xSC = SCOREshow.getBounds().getWidth();

SCOREshow.getBounds().getHeight();

add(SCOREshow, 9.3\****WIDTH*** / 11 - xSC/2 + 35, ***BRICK\_Y\_OFFSET*** / 4);

}

/\*\* adds a label with info for player after every turn

\* according to win of lose of the player\*/

**private** **void** showEndGameLabel() {

**if** (numberOfBricks == 0) {

removeAll();

*XP*--;

addScoreLabel();

addUpperLabels();

winnerLabel();

waitForClick();

removeAll();

numberOfBricks = ***NumberOfBricksPerRow*** \* ***NumberRowOfBricks***;

}

**if** (ball.getY() > getHeight() - ***BALL\_DIAM***) {

**this**.removeAll();

*XP*--;

addScoreLabel();

addUpperLabels();

loserLabel();

waitForClick();

removeAll();

numberOfBricks = ***NumberOfBricksPerRow*** \* ***NumberRowOfBricks***;

}

}

/\*\* adds loser label \*/

**private** **void** loserLabel() {

GLabel gameOver2 = **new** GLabel("Oops!");

gameOver2.setFont("Arial-35");

gameOver2.setColor(Color.***WHITE***);

**double** x1 = gameOver2.getBounds().getWidth();

**double** y1 = gameOver2.getBounds().getHeight();

gameOver2.setLocation(***WIDTH*** / 2 - x1 / 2, ***HEIGHT*** / 2 - y1);

add(gameOver2);

GLabel gameOver3 = **new** GLabel("You`re a loser!");

gameOver3.setFont("Arial-35");

gameOver3.setColor(Color.***WHITE***);

**double** x = gameOver3.getBounds().getWidth();

**double** y = gameOver3.getBounds().getHeight();

gameOver3.setLocation(***WIDTH*** / 2 - x / 2, ***HEIGHT*** / 2);

add(gameOver3);

/\* after all turns adds restart label, in other case - continue label \*/

**if** (*XP* == 0) {

GLabel restart = **new** GLabel("Click to restart");

restart.setFont("Arial-20");

restart.setColor(Color.***MAGENTA***);

**double** xR = restart.getBounds().getWidth();

restart.setLocation(***WIDTH*** / 2 - xR / 2, ***HEIGHT*** / 2 + y);

add(restart);

} **else** {

GLabel continueL = **new** GLabel("Click to continue the game");

continueL.setFont("Arial-20");

continueL.setColor(Color.***MAGENTA***);

**double** xC = continueL.getBounds().getWidth();

continueL.setLocation(***WIDTH*** / 2 - xC / 2, ***HEIGHT*** / 2 + y);

add(continueL);

}

}

/\*\* adds a simple info bar on the top of the screen,

\* shows score, XPs left and name of the game \*/

**private** **void** addUpperLabels() {

GLabel XPshow = **new** GLabel("XP: " + *XP*);

XPshow.setFont("Arial-25");

XPshow.setColor(Color.***WHITE***);

**double** xXP = XPshow.getBounds().getWidth();

add(XPshow, ***WIDTH*** / 11 - xXP / 2, ***BRICK\_Y\_OFFSET*** / 4);

GLabel gameName = **new** GLabel("BREAKOUT");

gameName.setFont("Arial-Bold-25");

gameName.setColor(Color.***MAGENTA***);

**double** xGN = gameName.getBounds().getWidth();

add(gameName, ***WIDTH***/2 - xGN/2, ***BRICK\_Y\_OFFSET***/4);

line = **new** GRect(0, ***BRICK\_Y\_OFFSET***/3, ***WIDTH***, 3);

line.setColor(Color.***BLACK***);

add(line);

}

/\*\* adds winner label \*/

**private** **void** winnerLabel() {

GLabel gameOver1 = **new** GLabel("Hooray!");

gameOver1.setFont("Arial-35");

gameOver1.setColor(Color.***WHITE***);

**double** x1 = gameOver1.getBounds().getWidth();

**double** y1 = gameOver1.getBounds().getHeight();

gameOver1.setLocation(***WIDTH*** / 2 - x1 / 2, ***HEIGHT*** / 2 - y1);

add(gameOver1);

GLabel gameOver = **new** GLabel("You`re a winner!");

gameOver.setFont("Arial-35");

gameOver.setColor(Color.***WHITE***);

**double** x = gameOver.getBounds().getWidth();

**double** y = gameOver.getBounds().getHeight();

gameOver.setLocation(***WIDTH*** / 2 - x / 2, ***HEIGHT*** / 2);

add(gameOver);

GLabel restart = **new** GLabel("Click to restart");

restart.setFont("Arial-20");

restart.setColor(Color.***MAGENTA***);

**double** xR = restart.getBounds().getWidth();

restart.setLocation(***WIDTH*** / 2 - xR / 2, ***HEIGHT*** / 2 + y);

add(restart);

}

/\*\* checks for collision and removes an object if it isn`t a player and upper info bar

\* also adds points to score after brick is removed \*/

**private** **void** checkCollision() {

GPoint l, r, u, d;

GObject l1, r1, u1, d1;

l = **new** GPoint(ball.getX() - 0.5, ball.getY() + ***BALL\_DIAM*** / 2);

r = **new** GPoint(ball.getX() + ***BALL\_DIAM*** + 0.5, ball.getY() + ***BALL\_DIAM*** / 2);

u = **new** GPoint(ball.getX() + ***BALL\_DIAM*** / 2, ball.getY() - 0.5);

d = **new** GPoint(ball.getX() + ***BALL\_DIAM*** / 2, ball.getY() + ***BALL\_DIAM*** + 0.5);

**if** ((getElementAt(l) != **null**) || (**int**) l.getX() <= 0) {

l1 = getElementAt(l);

**if** (l1 != player & l1 != player1 & l1 != **null** & l1 != line) {

remove(getElementAt(l));

numberOfBricks--;

addPoints();

remove(SCOREshow);

addScoreLabel();

}

ballSpeedX = -ballSpeedX \* speedIncrease;

} **else** **if** ((getElementAt(r) != **null**) || (**int**) r.getX() >= ***WIDTH***) {

r1 = getElementAt(r);

**if** (r1 != player & r1 != player1 & r1 != **null** & r1 != line) {

remove(r1);

numberOfBricks--;

addPoints();

remove(SCOREshow);

addScoreLabel();

}

ballSpeedX = -ballSpeedX \* speedIncrease;

} **else** **if** ((getElementAt(u) != **null**) || (**int**) u.getY() <= 0) {

u1 = getElementAt(u);

**if** (u1 != player & u1 != player1 & u1 != **null** & u1 != line) {

remove(u1);

numberOfBricks--;

addPoints();

remove(SCOREshow);

addScoreLabel();

}

**if** (u1 != player)

ballSpeedY = -ballSpeedY \* speedIncrease;

} **else** **if** (getElementAt(d) != **null**) {

d1 = getElementAt(d);

**if** (d1 != player & d1 != player1 & d1 != **null** & d1 != line) {

remove(d1);

numberOfBricks--;

addPoints();

remove(SCOREshow);

addScoreLabel();

}

ballSpeedY = -ballSpeedY \* speedIncrease;

}

**if** (ballSpeedY >= 6 || ballSpeedX >= 6) {

speedIncrease = 1;

}

}

/\*\* add points after brick is removed \*/

**private** **void** addPoints() {

score += 10;

}

/\*\* moving of the player \*/

**public** **void** mouseMoved(MouseEvent a) {

**int** x = a.getX();

player.setLocation(x - ***PLAYER\_WIDTH*** / 2, player.getY());

player1.setLocation(player.getX(), player1.getY());

}

/\*\* setting up and preparing for the game

\* adding welcome label, player, ball, bricks and mouse listeners \*/

**private** **void** setup() {

GLabel welcome = **new** GLabel("Click to start the game");

welcome.setFont("Arial-35");

welcome.setColor(Color.***WHITE***);

**double** x = welcome.getBounds().getWidth();

welcome.setLocation(***WIDTH*** / 2 - x / 2, ***HEIGHT*** / 2);

addBricks();

**if** (*XP* == 3) {

add(welcome);

waitForClick();

remove(welcome);

}

player = **new** GRect(***WIDTH*** / 2 - ***PLAYER\_WIDTH*** / 2, ***HEIGHT*** - ***HEIGHT*** / 10, ***PLAYER\_WIDTH***, ***PLAYER\_HEIGHT***);

player1 = triangle();

player1.setVisible(**false**);

ball = **new** GOval(***WIDTH*** / 2 - ***BALL\_DIAM*** / 2, (***HEIGHT*** \* 8) / 10 - ***BALL\_DIAM*** + 2, ***BALL\_DIAM***, ***BALL\_DIAM***);

ball.setFilled(**true**);

ball.setColor(Color.***WHITE***);

ball.setFillColor(Color.***white***);

player.setFilled(**true**);

player.setColor(Color.***WHITE***);

player.setFillColor(Color.***white***);

add(ball);

add(player);

add(player1, ***WIDTH*** / 2 - ***PLAYER\_WIDTH*** / 2, ***HEIGHT*** - ***HEIGHT*** / 10);

ballSpeedX = random(0, 1.5);

ballSpeedY = -Math.*abs*(Math.*sqrt*(Math.*pow*(ballSpeed, 2) - Math.*pow*(ballSpeedX, 2)));

**if** (random(0, 10) % 2 == 0)

ballSpeedX = -ballSpeedX;

addMouseListeners();

}

/\*\* adds bricks in the cycle \*/

**private** **void** addBricks() {

**for** (**int** column = 0; column < ***NumberRowOfBricks***; column++) {

**for** (**int** row = 0; row < ***NumberOfBricksPerRow***; row++) {

GRect brick = **new** GRect(row \* (***BRICK\_WIDTH*** + ***BRICK\_SEPARATION***) + ***BRICK\_SEPARATION*** / 2,

column \* (***BRICK\_HEIGHT*** + ***BRICK\_SEPARATION***) + ***BRICK\_Y\_OFFSET***, ***BRICK\_WIDTH***, ***BRICK\_HEIGHT***);

brick.setColor(Color.***WHITE***);

brick.setFilled(**true**);

brick.setFillColor(setNeededColor(column));

add(brick);

}

}

}

/\*\* changes the color of the bricks according to the number of the row \*/

**private** Color setNeededColor(**int** row) {

**int** numberRow = row \* 5 / ***NumberRowOfBricks***;

**if** (numberRow <= 0) {

**return** Color.***RED***;

} **else** **if** (numberRow == 1) {

**return** Color.***ORANGE***;

} **else** **if** (numberRow == 2) {

**return** Color.***YELLOW***;

} **else** **if** (numberRow == 3) {

**return** Color.***GREEN***;

} **else** {

**return** Color.***CYAN***;

}

}

/\*\* boolean to check if the game is over \*/

**private** **boolean** gameOver() {

**if** (numberOfBricks == 0){

**return** **true**;

}

**else** **if** (ball.getY() > getHeight() - ***BALL\_DIAM***) {

**return** **true**;

}

**return** **false**;

}

/\*\* moving of the ball \*/

**private** **void** ballMove() {

ball.move(ballSpeedX, ballSpeedY);

}

/\*\* adding a triangle to the player to check for collision \*/

**private** GPolygon triangle() {

GPolygon player2 = **new** GPolygon();

player2.addVertex(0, 0);

player2.addEdge(***PLAYER\_WIDTH***, 0);

player2.addEdge(-***PLAYER\_WIDTH*** / 2, 50);

player2.addEdge(-***PLAYER\_WIDTH*** / 2, -50);

// player2.setFilled(true);

// player2.setColor(Color.WHITE);

**return** player2;

}

/\*\* returning random number to start the game with the random speed of the ball \*/

**private** **double** random(**double** min, **double** max) {

**double** n = 0;

n = ((Math.*random*() \* (max - min)) + min); // the random from 1 to 100

print(n);

**return** (n);

}

**private** GPolygon player1;

**private** GOval ball;

**private** GRect player;

}