Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**Лабораторна робота №9**

на тему:

«Гра з графічним інтерфейсом»

з курсу:

«Алгоритмізація та програмування»

Виконав:

ст. гр. КН-110

Король Орест

Прийняв:

Кривенчук Ю.П.

Львів – 2017 р.

*Лабораторна робота №9*

***Тема роботи:*** *розробити програму, що реалізовуватиме гру з інтерфейсом на базі гри курсу цс50 ''брейкаут''.*

***Мета роботи:*** навчитись писати програми з зв’язками між функціями, здобути навички у кодуванні окремих об’єктів на мові програмування С.

**Завдання**

Розробити програму, яка викликатиме гру базовану на грі з курсу цс50 ‘’брейкаут’’.

​

**Текст програми**

#define \_XOPEN\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

// Stanford Portable Library

#include "gevents.h"

#include "gobjects.h"

#include "gwindow.h"

// height and width of game's window in pixels

#define HEIGHT 600

#define WIDTH 400

// number of rows of bricks

#define ROWS 5

// number of columns of bricks

#define COLS 10

// height and width of the bricks in pixels

#define BRICK\_HEIGHT 10

#define BRICK\_WIDTH 40

// spaces between the bricks in pixels

#define BRICK\_SPACE (50/11)

// height and width of the paddle in pixels

#define PADDLE\_HEIGHT 5

#define PADDLE\_WIDTH 50

// radius of ball in pixels

#define RADIUS 10

// lives

#define LIVES 3

#define DIV 5

// prototypes

void initBricks(GWindow window);

GOval initBall(GWindow window);

GRect initPaddle(GWindow window);

GLabel initScoreboard(GWindow window);

void updateScoreboard(GWindow window, GLabel label, int points);

GObject detectCollision(GWindow window, GOval ball);

void reverseDirection(double \*velocity);

int main(void)

{

// seed pseudorandom number generator

srand48(time(NULL));

// instantiate window

GWindow window = newGWindow(WIDTH, HEIGHT);

// instantiate bricks

initBricks(window);

// instantiate ball, centered in middle of window

GOval ball = initBall(window);

// instantiate paddle, centered at bottom of window

GRect paddle = initPaddle(window);

// instantiate scoreboard, centered in middle of window, just above ball

GLabel label = initScoreboard(window);

// number of bricks initially

int bricks = COLS \* ROWS;

// number of lives initially

int lives = LIVES;

// number of points initially

int points = 0;

// initial velocity

double magnitude = (drand48()+1)/50;

int direction = (rand()%2) \* 2 - 1;

double x\_velocity = drand48() / DIV;

double y\_velocity = drand48() / DIV;

// keep playing until game over

while (lives > 0 && bricks > 0)

{

// TODO -- entire while loop

// check for mouse event

GEvent event = getNextEvent(MOUSE\_EVENT);

// if event is detected

if (event != NULL)

{

// if event is mouse movement

if (getEventType(event) == MOUSE\_MOVED)

{

// paddle follows mouse along x asis

int x = getX(event) - 30;

setLocation(paddle, x, 570);

// sets special position when mouse is

// off window or near edge

if (x <= 0)

setLocation(paddle, 0, 570);

if (x >= 340)

setLocation(paddle, 339, 570);

}

}

GObject object = detectCollision(window, ball);

move(ball, x\_velocity, y\_velocity);

if (getX(ball) + 20 >= WIDTH)

{

x\_velocity = -drand48() / DIV;

}

if (getX(ball) <= 0)

{

x\_velocity = drand48() / DIV;

}

if (getY(ball) <= 10)

{

y\_velocity = drand48() / DIV;

}

if (object != NULL && object != paddle && object != label)

{

y\_velocity = drand48() / (DIV - 2);

removeGWindow(window, object);

points++;

updateScoreboard(window, label, points);

}

if (object == paddle)

{

y\_velocity = -drand48() / DIV;

}

if (getY(ball) >= 570)

{

lives--;

removeGWindow(window, ball);

ball = initBall(window);

waitForClick();

}

}

// wait for click before exiting

waitForClick();

// game over

closeGWindow(window);

return 0;

}

/\*\*

\* Initializes window with a grid of bricks.

\*/

void initBricks(GWindow window)

{

// TODO -- entire function

// defines colors to be used in bricks

char\* colors[5];

colors[0] = "YELLOW";

colors[1] = "BLUE";

colors[2] = "GREEN";

colors[3] = "RED";

colors[4] = "VIOLET";

// defines variables to set brick location and size

int s = 10;

int w = 29;

int h = 5;

// creates bricks

for (int i = 0; i < ROWS; i++)

{

for (int j = 0; j < COLS; j++)

{

int x = s + (w \* j) + (s \* j);

int y = s + (h \* i) + (s \* i);

GRect brick = newGRect(x, y, w, h);

setColor(brick, colors[i]);

setFilled(brick, true);

add(window, brick);

}

}

}

/\*\*

\* Instantiates ball in center of window. Returns ball.

\*/

GOval initBall(GWindow window)

{

// TODO -- entire function

GOval ball = newGOval(190, 290, 2 \* RADIUS, 2 \* RADIUS);

setColor(ball, "PINK");

setFilled(ball, true);

add(window, ball);

return ball;

}

/\*\*

\* Instantiates paddle in bottom-middle of window.

\*/

GRect initPaddle(GWindow window)

{

// TODO -- entire function

GRect paddle = newGRect(170, 570, 60, 4);

setColor(paddle, "BLACK");

setFilled(paddle, true);

add(window, paddle);

return paddle;

}

/\*\*

\* Instantiates, configures, and returns label for scoreboard.

\*/

GLabel initScoreboard(GWindow window)

{

GLabel label = newGLabel("0");

setFont(label, "Calibri-40");

double x = (getWidth(window) - getWidth(label)) / 2;

double y = (getHeight(window) - getHeight(label)) / 2;

setLocation(label, x, y);

setColor(label, "ORANGE");

add(window, label);

return label;

}

/\*\*

\* Updates scoreboard's label, keeping it centered in window.

\*/

void updateScoreboard(GWindow window, GLabel label, int points)

{

// update label

char s[12];

sprintf(s, "%i", points);

setLabel(label, s);

// center label in window

double x = (getWidth(window) - getWidth(label)) / 2;

double y = (getHeight(window) - getHeight(label)) / 2;

setLocation(label, x, y);

}

/\*\*

\* Detects whether ball has collided with some object in window

\* by checking the four corners of its bounding box (which are

\* outside the ball's GOval, and so the ball can't collide with

\* itself). Returns object if so, else NULL.

\*/

GObject detectCollision(GWindow window, GOval ball)

{

// ball's location

double x = getX(ball);

double y = getY(ball);

// for checking for collisions

GObject object;

// check for collision at ball's top-left corner

object = getGObjectAt(window, x, y);

if (object != NULL)

{

return object;

}

// check for collision at ball's top-right corner

object = getGObjectAt(window, x + 2 \* RADIUS, y);

if (object != NULL)

{

return object;

}

// check for collision at ball's bottom-left corner

object = getGObjectAt(window, x, y + 2 \* RADIUS);

if (object != NULL)

{

return object;

}

// check for collision at ball's bottom-right corner

object = getGObjectAt(window, x + 2 \* RADIUS, y + 2 \* RADIUS);

if (object != NULL)

{

return object;

}

// no collision

return NULL;

}

/\*\*

\* Switches the ball's velocity in the opposite direction

\*/

void reverseDirection(double \*velocity)

{

// dereferencing in order to get the values at the arguments' addresses and store them

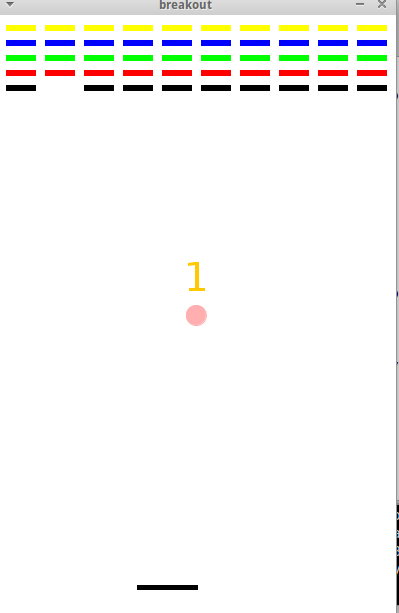
double velocity\_copy = \*velocity;

\*velocity \*= -1;

return;

}

**Приклад виконання програми**

****

***Висновок:*** завдяки набутим знанням з програмування, ми навчились реалізовувати складні програми.