Computer Graphics and Game Programming – 508837
2nd Assignment
Advisor: Assoc. Dr. Murat KURT

2nd Assignment Report

04.14.2023

By:
Mehmet ISIKHAN, 91220001906

Table of Contents

Introduction

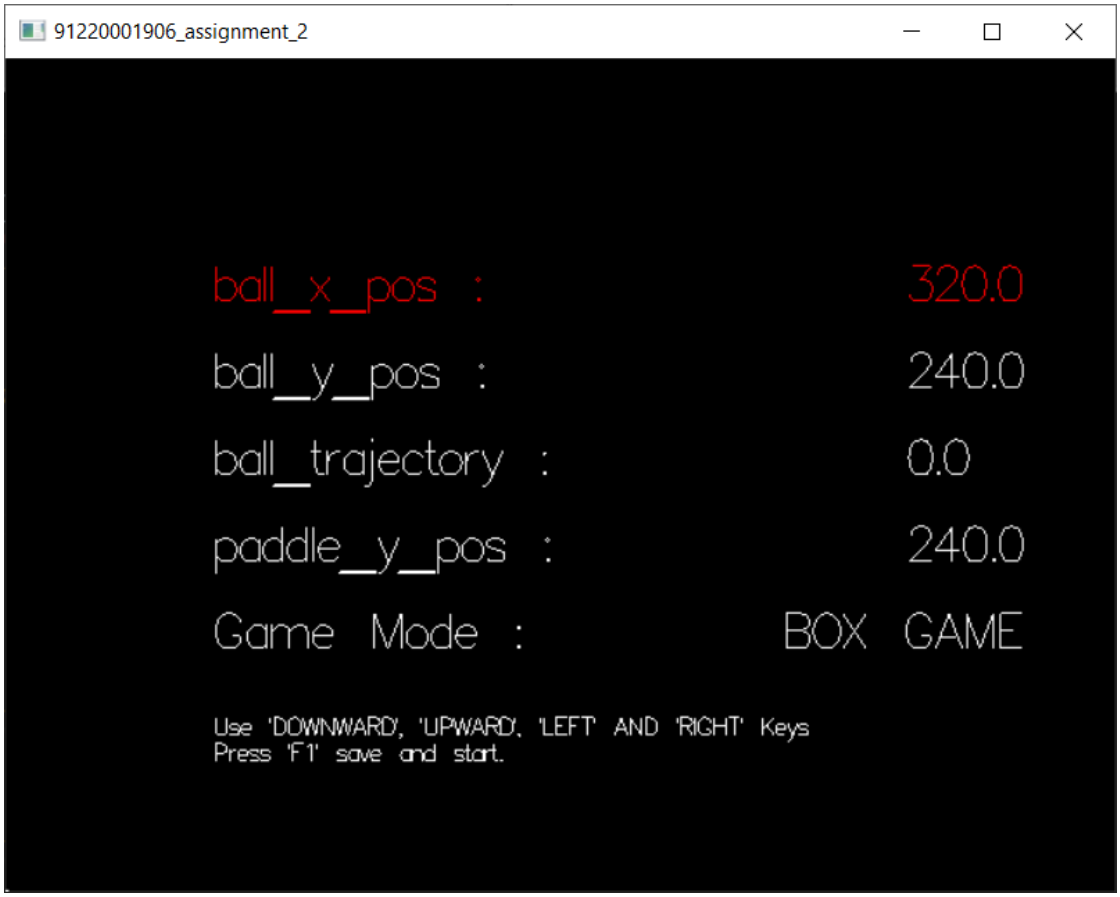
This report includes the classic pong game developing with OpenGL and the different features of the game. Firstly, the pong game was developed in 1958 as a television game by "Ralph Baer" but published in 1972 by ATARI Inc. as the video game [1]. The simple pong game in this article has a ball and a paddle and the player has 15 chances until the game is over. The simple pong game in this article has a ball and a paddle, and the player has 15 chances until the game is over. The mission of the player is preventing the ball from getting behind the paddle, if the ball passes behind of the paddle, the player's chances decrease one. At the end, when the player uses the last chance, the game is over, and it shows the player's game score in second on the screen. But the ball's movement changes according to its slope and touching position on the paddle.

The second game is the "Box Game", and it is again playing with the paddle and ball, so we can say it is a variation of the pong game. Again, the player has 15 chances, but this time the screen is not empty. According to screen size, lots of boxes are located on the screen and the mission of the player is not keeping the ball on the area only at this time because he or she has to destroy the boxes with the ball touching. If the player can destroy the boxes with 15 fail chances, the player will be successful, else the game is over again. In conclusion, I tried to explain the developed by me the pong game and its features developed in this report.

1. The Pong Game

1.1. Main Menu

At the assignment 2 paper, the ball's x and y, paddle's y coordinates and ball's trajectory should be available to take as the console input, but a main menu is developed for taking those variables from the player instead of the console inputs. Also, the ball's x and y, paddle's y coordinates variables change according to screen size and those changes can be seeable on the menu and the player can change those variables on the menu. The player can move with the keyboard's upward and downward arrow keys also he or she can change the variables with the keyboard's left and right keys. In this way, the input process of the variables has become more user friendly. In addition, the game mode selection option has been added to the main menu, players can choose the game mode want to play.



Picture.1: Main Menu

To enable moving on the menu feature, the integer variable was determined as named “menu_selection” and it was assigned to “1” at the start because the first menu element should be selected when the menu is opening. When the player moves on the menu, this variable moves between 1 and 5. After the downward key pressed, if the “menu_selection” is not equal to 5, it decreases, else it stays to 5. Same process applies to the upward key, when pressing to upward key, if the “menu_selection” variable is not equal to 1, it increases else it stays to 1. Each menu element color changes to red when the “menu_selection” variable is equal to its assigned number. Bottom of the menu, an information text added for the giving information about how the moving and changing menu elements. The “F1” key is to start the game, but the system controls the variables of the ball and peddle when press the “F1” key. If the entered ball and paddle coordinates are outside the available area, an error message is displayed at the bottom of the information text, using the string variable named “errorMessage”. However, if there is no error, the message will not be displayed. At this point, with the “strlen()” function [2], the “errorMessage” string length checking and if the length of the string is not equal to 0, the error message showing, else program assign the entered variables to initial variables then the game start.

```
//MENU//
char errorMessage[50];
int menu_selection = 1;
bool isMenuActive = true;
int gameMode = 1;


/*Main Menu*/
void mainMenu(){
	glColor3d(1,1,1);
	if(menu_selection == 1)
		glColor3d(1,0,0);
	stroke_output((window_width/2)-200, (window_height/2)+100,
2, "ball_x_pos : ");
	stroke_output((window_width/2)+200, (window_height/2)+100,
2, "%.1f",ball_x_pos);
	glColor3d(1,1,1);

	if(menu_selection == 2)
		glColor3d(1,0,0);
	stroke_output((window_width/2)-200, (window_height/2)+50, 2,
"ball_y_pos : ");
	stroke_output((window_width/2)+200, (window_height/2)+50, 2,
"%.1f",ball_y_pos);
	glColor3d(1,1,1);
```

```
	if(menu_selection == 3)
		glColor3d(1,0,0);
	stroke_output((window_width/2)-200, (window_height/2), 2,
"ball_trajectory : ");
	stroke_output((window_width/2)+200, (window_height/2), 2,
"%.1f",ball_trajectory);
	glColor3d(1,1,1);

	if(menu_selection == 4)
		glColor3d(1,0,0);
	stroke_output((window_width/2)-200, (window_height/2)-50, 2,
"paddle_y_pos : ");
	stroke_output((window_width/2)+200, (window_height/2)-50, 2,
"%.1f",paddle_y_pos);
	glColor3d(1,1,1);

	if(menu_selection == 5)
		glColor3d(1,0,0);
	stroke_output((window_width/2)-200, (window_height/2)-100,
2, "Game Mode : ");
	if(gameMode == 0)
		stroke_output((window_width/2)+162, (window_height/2)-
100, 2, "TRAINING");
	if(gameMode == 1)
		stroke_output((window_width/2)+128, (window_height/2)-
100, 2, "BOX GAME");
	glColor3d(1,1,1);

	stroke_output((window_width/2)-200, (window_height/2)-150,
1, "Use 'DOWNWARD', 'UPWARD', 'LEFT' AND 'RIGHT' Keys");
	stroke_output((window_width/2)-200, (window_height/2)-165,
1, "Press 'F1' save and start.");

	if(strlen(errorMessage) != 0){
		glColor3d(1,0,0);
		stroke_output((window_width/2)-200, (window_height/2)-
135, 1, "%s",errorMessage);
		glColor3d(1,1,1);
	}


    glLoadIdentity();
    glutWireTeapot(1);
}
```

Note: I used the “stroke_output” function from the assignment 1 for the creation of the menu's texts [5].

For the main menu keys, the “mainMenuKeys” function allows the moving on the main menu and changing the main menu variables with the arrow keys thanks to glutSpecialFunc from the OpenGL[3]. The “mainMenuKeys” function works like the “key_press” function came from the frame code in assignment 2 paper, but the key char variables are signed like “GLUT_KEY_LEFT” instead of the frame code's unsigned char. The upward key increases the “menu_selection” integer variable if the variable is not equal to 5, and it refreshes the screen with using the “glutPostRedisplay” function, so the menu selection variable is changed, and it is painted with the red color. The downward key decreases the “menu_selection” integer variable if the variable is not equal to 1, and it refreshes and paints the selected menu item to red again. Thus, the program determines which variable to change based on the player's menu selection and with the right and left arrow keys, the player can change the menu elements’ values. When the player wants to start the game, the “F1” key allows the start to the game, but the system checks the entered variables (“ball_x_pos”, “ball_y_pos” and “paddle_y_pos”) before starting the game. If these variables are not on the game area, the system assigns the defined error messages to “errorMessage” char variable using the “strcpy” function [4] and the error messages show on the screen. The playground is defined by the heights, widths, and developer tests of the objects and these controllers are made according to this definition. If all entered variables are in the ground, these variables assign to initial global variables as the start position of the objects and game is starting.

```
//Main Menu Keys//
void mainMenuKeys(int key, int x, int y){
      if(isMenuActive)
            switch(key){
                  case GLUT_KEY_LEFT:
                        switch(menu_selection){
                              case 1:
                                    ball_x_pos -=1;
                                    glutPostRedisplay();
                                    break;
                              case 2:
                                    ball_y_pos -=1;
                                    glutPostRedisplay();
                                    break;
                              case 3:
                                    ball_trajectory -=0.1;
                                    glutPostRedisplay();
                                    break;
```

```
                case 4:
                        paddle_y_pos -=1;
                        glutPostRedisplay();
                        break;
                case 5:
                        if(gameMode == 1){
                                gameMode = 0;
                        }else{
                                gameMode = 1;
                        }
                        glutPostRedisplay();
                        break;
        }
        break;
case GLUT_KEY_RIGHT:
        switch(menu_selection){
                case 1:
                        ball_x_pos +=1;
                        glutPostRedisplay();
                        break;
                case 2:
                        ball_y_pos +=1;
                        glutPostRedisplay();
                        break;
                case 3:
                        ball_trajectory +=0.1;
                        glutPostRedisplay();
                        break;
                case 4:
                        paddle_y_pos +=1;
                        glutPostRedisplay();
                        break;
                case 5:
                        if(gameMode == 1){
                                gameMode = 0;
                        }else{
                                gameMode = 1;
                        }
                        glutPostRedisplay();
                        break;
        }
        break;
case GLUT_KEY_DOWN:
        if(menu_selection==5){

        }else{
                menu_selection += 1;
        }
        glutPostRedisplay();
        break;
```

```
                case GLUT_KEY_UP:
                        if(menu_selection==1){

                        }else{
                                menu_selection -= 1;
                        }
                        glutPostRedisplay();
                        break;
                case GLUT_KEY_F1:
                        if(ball_x_pos < 37 || ball_x_pos >
(window_width-74) ){
                                strcpy(errorMessage,"ball_x_pos is
out of range !");
                                glutPostRedisplay();
                        }else if(ball_y_pos < 37 || ball_y_pos >
(window_height-38) ){
                                strcpy(errorMessage,"ball_y_pos is
out of range !");
                                glutPostRedisplay();
                        }else if(paddle_y_pos < 80 || paddle_y_pos
> (window_height-80) ){
                                strcpy(errorMessage,"paddle_y_pos is
out of range !");
                                glutPostRedisplay();
                        }else{
                                initial_ball_x_pos = ball_x_pos;
                                initial_ball_y_pos = ball_y_pos ;
                                initial_paddle_y_pos = paddle_y_pos;
                                initial_ball_trajectory =
ball_trajectory;
                                isMenuActive = false;
                                glutPostRedisplay();
                        }
                        break;
        }
}
```

The “mainMenuKeys” function calls from the main function as “glutSpecialFunc(mainMenuKeys)” for just the main menu keyboard controller when the ”isMenuActive” variable is true. When a player starts the game, the main menu is deactivated with the “isMenuActive” variable assigned false and the frame code’s “key_press” function use.

Inside the “draw_scene” function, the program firstly checks the “isMenuActive” boolean variable if it is true in the “mainMenu” function calls, else program runs with the game mode checking and starts the game according to this checking.

```
//Draw Scenes//
void draw_scene(void){

      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      if(isMenuActive){
            mainMenu();
      }else if(gameMode == 1){
.
.
.
```

1.2. Ball Move

The ball moving is the most important feature in the pong game, so the ball moving should change according to game play, but firstly we need to focus on the bounds of the playground and how to create it with the C code. The ball should not pass borders with drawn with “draw_border” function, so the borders should determine before the ball moving features define. According to “MARGIN_SIZE”, “BORDER_SIZE” and developer test, the borders are defined with “window_height” and “window_width” index variables. For keeping the ball on the playground with border's limitations, the ball’s x position (ball_x_pos) and y (ball_y_pos) position increase or decrease much as “BALL_STEP” variable if the ball’s coordinates lower than the defined borders limitations. A boolean variable (ballDirectionX) was defined to track the ball's horizontal movement, and it is initialized to true by default at the beginning of the game. The program checks the value of the “ballDirectionX” variable in the “move_ball” function. If the variable is true, the ball moves horizontally towards the left direction by decreasing the “ball_x_pos” variable by the amount of “BALL_STEP” until it reaches the border variable (37). When the ball reaches the border variable (37), “ballDirectionX” is assigned to false and “ball_x_pos” increase much as “BALL_STEP” until the paddle touching or out of bound. Also, the program keeps the horizontal line of the ball movement as two points (left touch and right-paddle touch) to a float matrix named as “ball_route” for calculation of the slope.

```
void move_ball(int game){
	.
	.
	.

	if(ball_y_pos <= 40){
		ballDirectionY = 1;
	}
	if(ball_y_pos > (window_height-38)){
		ballDirectionY = 2;
	}

	if(ball_x_pos < 37){
		ball_route[0][0] = (float)ball_x_pos;
		ball_route[0][1] = (float)ball_y_pos;
		ballDirectionX = false;
		slope = ((ball_route[0][1]-ball_route[1][1]) /
(ball_route[0][0] - ball_route[1][0]));
	}

	if( (ball_x_pos > (window_width-74)) && (!(ball_x_pos >
(window_width-50))) & ( ball_y_pos < paddle_y_pos+55 && ball_y_pos
> paddle_y_pos-55)){
			.
			.
			.

			ballDirectionX = true;
			ball_route[1][0] = (float)ball_x_pos;
			ball_route[1][1] = (float)ball_y_pos;
			slope = ((ball_route[1][1]-ball_route[0][1]) /
(ball_route[1][0] - ball_route[0][0]));
	}
	.
	.
	.

	if(ballDirectionX){
		ball_x_pos += -BALL_STEP;
	}else{
		ball_x_pos += BALL_STEP;
	}
```

The slope is used to change the ball's movement when it touches the paddle areas. For the ball's vertical movement, an integer variable named “ballDirectionY” is defined, which can take on values of 1, 2, or 0 to represent different vertical directions. If the variable is equal to 1, the “ball_y_pos” variable (which keeps track of the ball's vertical position) increases by “BALL_STEP” multiplied by “applySlope”. Similarly, if the variable is 2, the ball's y position variable decreases by “BALL_STEP” multiplied by

the “applySlope” value. When the “ballDirectionY” variable is equal to 0, the program keeps the “ball_y_pos” variable constant, thus keeping the ball's movement constant on the y-axis and x-axis changes.

```
switch(ballDirectionY){
	case 0:
		break;
	case 1:
		ball_y_pos += BALL_STEP*applySlope;
		break;
	case 2:
		ball_y_pos += -BALL_STEP*applySlope;
		break;

}
```

Picture.2: Ball Movement Line

$$m = \frac{(y_1 - y_2)}{(x_1 - x_2)}$$

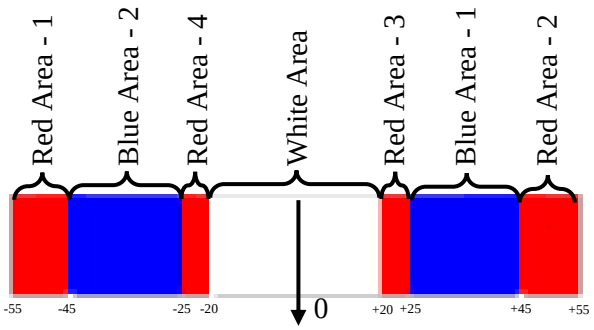
The slope is calculated using the formula for calculating the slope of a line with two known points, and the slope is used to change the ball's movement when it touches the paddle areas and to create special situations. Slope calculation is calculation just for the paddle and left border because up and down borders just change the ball vertical way (positive or negative) they do not intervene the ball slope. The ball moves with the same slope variable until it touches the paddle's areas. Actually, the slope variable changes the ball's movement speed even a little because it multiplied by the "BALL_STEP" variable. So, the ball has the ability of two axis movement on the playground.

1.3. The Paddle

Firstly, for keeping the paddle on the playground, at the "handle_mouse_motion" function, which is used for moving to the paddle, I added limitation for the paddle do not pass to up and bottom border according to paddle length and development tests. In this way, the paddle can not move up more than the border and the same situation is also valid for the bottom border. Also, a check process is added for the understanding if the main menu is active or not. The paddle cannot move with the mouse if the main menu is active, else the paddle can move with the mouse movements thanks to "handle_mouse_motion" and "handle_mouse_click" functions. The bottom border is determined as 80 pixels up from the bottom, and the upper border is 80 pixels down from the window height.

```
void handle_mouse_motion(int x, int y){

	if (left_button_state == GLUT_DOWN && (!isMenuActive)) {
		if(paddle_y_pos < 80 ){
			paddle_y_pos = 80;
		}else if(paddle_y_pos > (window_height-80) ){
			paddle_y_pos = (window_height-80);
		}else{
			paddle_y_pos += left_button_lasty - y;
			left_button_lasty = y;
			glutPostRedisplay();
		}
	}
}
```

Secondly, the paddle must intercept the ball so when the ball reaches the border of the paddle side according to ***1.2. Ball Move*** part if the ball touches the paddle, ball move to the left side simply. But this situation cannot move the ball up and down. Even if moving, it will always move with the same slope variable, so this situation will cause uniform gameplay. For escape of this uniform gameplay, the paddle changes the slope according to paddle touch areas and ball's last slope variable, and this helps to more variation ball movements of the game. These paddle areas are determined by the developer, and we can see the areas in ***picture.3***.



Picture.3: Paddle Areas

The areas of the paddle which are shown in ***picture.3*** developed based by the paddle middle point and left(up) and right(down) lengths. And the main purpose of the areas is giving variation and making more competitive to the game. A developer paddle has been added to the code with giving the special areas' colors in the "draw_paddle" function but these parts added as the command lines if someone wants to play the game with the developer paddle, he or she can make active it and he or she must be deactivated the default white paddle. To handle the paddle's special areas, the "move_ball" function includes logic for detecting when the ball touches the paddle. The different areas of the paddle are also defined in this function. When the ball reaches the border of the paddle side (window_width-74) the function checks the ball and paddle are touching. If they are touching, the special area checks making.

```
void move_ball(int game){
	.
	.
	.
	if(ball_x_pos < 37){
		ball_route[0][0] = (float)ball_x_pos;
		ball_route[0][1] = (float)ball_y_pos;
		ballDirectionX = false;
		slope = ((ball_route[0][1]-ball_route[1][1]) /
(ball_route[0][0] - ball_route[1][0]));
	}

	if( (ball_x_pos > (window_width-74)) && (!(ball_x_pos >
(window_width-50))) & ( ball_y_pos < paddle_y_pos+55 && ball_y_pos
> paddle_y_pos-55)){
```

1.3.1. Red Area - 1

```
if(ball_y_pos > paddle_y_pos+45){ //RED AREA-1//
	ballDirectionY = 1;
	applySlope = 1.1;
}
```

This area covers the paddle positive edge between (paddle's midpoint) + 55 pixels and (paddle's midpoint) + 45 pixels and if the ball touches this area the ball moves up direction (ballDirectionY = 1) with the "1.1 applySlope" variable. The "applySlope" is a float variable for the applying slope to the ball moving according to the calculated "slope" variable. This area is the up edge of the paddle, and the ball always moves up.

1.3.2. Red Area - 2

```
else if(ball_y_pos < paddle_y_pos-45){ //RED AREA-2//
	ballDirectionY = 2;
	applySlope = 1.1;
}
```

The negative area of the paddle is between (paddle's midpoint) - 55 pixels and (paddle's midpoint) - 45 pixels and if the ball touches this area the ball moves down direction (ballDirectionY = 2) with the "1.1 applySlope" variable. This area is the bottom edge of the paddle, and the ball always moves down.

1.3.3. Middle Point

```
else if(ball_y_pos == paddle_y_pos){ //MIDDLE POINT//
      if(slope != 0.0 || slope != -0.0){
            ballDirectionY = 0;
      }else{
            ballDirectionY = rand() % 2 + 1;
      }
}
```

The middle point is the middle of the paddle, and it consists of a single pixel. It is actually inside of the white area but thanks to else if hierarchy the middle point checking is more priority than the white area. This feature is implemented at the start of the game because the ball initially has a slope variable of either "0.0" or "-0.0". To determine the ball's initial direction, a random direction (up-1 or down-2) is selected using the "rand" function [6]. The ball then moves in this direction from the middle of the paddle with the default 1 slope variable. If the player can touch the ball on this point when the play on, the ball's y direction keeps fixed (ballDirectionY = 0) and ball move x-axis as linear.

1.3.4. Blue Area - 1

```
else if(ball_y_pos < paddle_y_pos+45 && ball_y_pos >
paddle_y_pos+25){ //BLUE AREA-1//
      if(slope >= 1){

      }if(slope == 0.0 || slope == -0.0){
            ballDirectionY = 1;
            applySlope =
((float)rand()/(float)(RAND_MAX));
      }else{
            if((rand() % 4) == 1)
                  applySlope = abs(slope)*10;
      }
}
```

When the ball touches the positive side between (paddle's midpoint) + 45 and (paddle's midpoint) + 25, the ball moves up (ballDirectionY = 1) with random slope float variable between 0.0 and 1.0 [7] if the calculated "slope" variable is equal "0.0" or "-0.0". If the slope is equal or greater than the 1, ball's movement keeps at the same. Else, the program generates a number between 0 and 4 [8], if this number equals to 1, the ball's line slope's absolute variable multiply to 10 and assigns to "applySlope" variable. In this way, apply sloping is always changes.

1.3.5. Blue Area - 2

```
else if(ball_y_pos > paddle_y_pos-45 && ball_y_pos < paddle_y_pos-25){ //BLUE AREA-2//

	if(slope >= 1){

	}else if(slope == 0.0 || slope == -0.0){
		ballDirectionY = 2;
		applySlope = ((float)rand()/(float)(RAND_MAX));
	}else{
		if((rand() % 4) == 1)
			applySlope = abs(slope)*10;
	}
}
```

For the negative side of the paddle, between (paddle's midpoint) - 45 and (paddle's midpoint) - 25, the ball moves down (ballDirectionY = 2) with random slope float variable between 0.0 and 1.0 [7] if the calculated "slope" variable is equal "0.0" or "-0.0". If the slope is equal or greater than the 1, the ball's movement stays at the same. Else, the program generates a number between 0 and 4 [8], if this number equals 1, the absolute value of the ball's line slope is multiplied by 10 and assigned to the "applySlope" variable, thus changing the applied slope.

1.3.6. Red Area - 3

```
else if(ball_y_pos < paddle_y_pos+25 && ball_y_pos > paddle_y_pos+20){ //RED AREA-3//

	if(slope == 0.0 || slope == -0.0){
		ballDirectionY = rand() % 2 + 1;
	}
	applySlope = 1.1;
}
```

For the third red area, if the ball's slope is equal to "0.0" or "-0.0", firstly the program chose a way (up or down) as a number 1 or 2 and ball moving there with the 1.1 slope value. If the ball's slope is not equal to zero or minus zero float variable, the ball does not change its vertical movement but "applySlope" becomes 1.1 value. The ball must touch the paddle's positive side between (paddle's midpoint) + 25 and (paddle's midpoint) + 20 for this to happen.

1.3.7. Red Area - 4

```
else if(ball_y_pos > paddle_y_pos-25 && ball_y_pos < 
paddle_y_pos-20){ //RED AREA-4//

      if(slope == 0.0 || slope == -0.0){
            ballDirectionY = rand() % 2 + 1;
      }
      applySlope = 1.1;
}
```

The fourth red area represents the paddle's negative side between (paddle's midpoint) - 25 and (paddle's midpoint) – 20. If the ball's slope is equal to "0.0" or "-0.0", firstly the program chooses a way (up or down) as a number 1 or 2 with the rand function [8] and ball moving there with the 1.1 slope value. If the ball's slope is not equal to zero or minus zero float variable, the ball does not change its vertical movement but "applySlope" becomes 1.1 value.

1.3.8. White Area

```
else{ //WHITE AREA//
      if(slope == 0.0 || slope == -0.0){
            applySlope = 
((float)rand()/(float)(RAND_MAX));
            ballDirectionY = rand() % 2 + 1;
      }else{
            if((rand() % 8) == 1)
                  applySlope = abs(slope)*10;
      }
}
```

The white area is the else statement of the area checking process if else conditions, so it includes the not determined areas between (paddle's midpoint) + 20 and (paddle's midpoint) – 20. If the ball's slope is equal to zero or minus zero float variables, the program chooses a random way for vertical movement of the ball with rand function [8] as a number 1 or 2 and "applySlope" variable again assigns a float variable between 0.0 and 1.0 values as random [7]. On the other hand, if the ball has a slope variable different from zero or minus zero float variable, the program multiplies this slope variable's absolute values by 10 and it is assigned to "applySlope" variable with 1 in 9 chances.

In conclusion, the ball's movement is modified according to these paddle areas, which helps to avoid monotony in the gameplay.

1.4. User Interface

The player has limited chance for playing the game. This limitation keeps in “game_chance” integer global variable and if the ball out of the court from the paddle side it means the player do not touch the ball with the paddle, the defined “game_over_counter” global integer variable increase by one. The system calculates the remains chance of the player with the “game_chance” - “game_over_counter”. The player's remaining chances are displayed as text on the left corner of the screen, using the "stroke_output" function [5]. This text changes every time the player loses a chance. If the player fails, a message is printed in the middle of the screen, indicating the remaining chance and the restart orientation. Until the last chance, this message shows on the screen every failure, when the player failed to last chance, this time a game over message shows in the middle of the screen with the player’s score with the exit the game information.

```
void draw_UI(){
	glColor3d(1,0,0);
	stroke_output((window_width)-200, (window_height)-25, 1,
"Remains chance : %d",game_chance-game_over_counter);
	glColor3d(1,1,1);
	if(game_over_counter == game_chance && (!successTextCheck)){
		if(end == 0){
			end = clock();
			execution_time = (((double)(end -
start))/CLOCKS_PER_SEC) - pause;
		}
		glColor3d(1,0,0);
		stroke_output((window_width/2)-180, (window_height/2),
5, "GAME OVER!");
		stroke_output((window_width/2)-60, (window_height/2)-
50, 1, "Your score %fsn",execution_time);
		glColor3d(1,1,1);
		stroke_output((window_width/2)-60, (window_height/2)-
75, 1, "Press 'ESC' to exit.");
	}
	if(game_over && game_over_counter != 0 && game_over_counter
!= game_chance && (!successTextCheck)){
		glColor3d(1,1,1);
		stroke_output((window_width/2)-100, (window_height/2),
3, "You Failed");
		stroke_output((window_width/2)-80, (window_height/2)-
50, 1, "Your Remain chance %d",game_chance-game_over_counter);
		stroke_output((window_width/2)-80, (window_height/2)-
75, 1, "Press 'R' to restart again.");
		stroke_output((window_width/2)-80, (window_height/2)-
100, 1, "Then press 'S' for the start");
		glColor3d(1,1,1);}}
```

1.4.1. Time Score

For the keeping the players' score, there is a time keeping features in the game. This feature works when the player starts the game, and it keeps the time when the ball is moving until the game is over. At the end, the time score calculated with the end time – (start time + pause times) and it assigns to "execution_time" double variable then it shows on the screen with the game over message. The program uses the computer clock time with the "clock" function [9] for the time score calculations.

The time is kept in five different variables determined with the use of "clock_t" [9]. These variables are "start", "end", "pause_end", "pause_start", and "pause". The "start" variable is assigned when the game starts, and the "end" variable is assigned when the game ends. When the player fails and there is a failed message on the screen, the "pause_start" variable is assigned to the beginning of the failed screen. If the player starts the game again, this time is assigned to "pause_end" variable, and the elapsed time is added to the "pause" variable as the time elapsed during the pause (pause_end - pause_start).

1.4.2. Control Keys

The keyboard press functions are different to each other in the main menu and in the game, so there is a "isMenuActive" check variable in the "key_press" function and this function checks the main menu activity. If the main menu is active, the "key_press" function does not work. For the "s" or "S" key (start key), there is a checking process for the reset button. If the reset button is pressed, the game can start else cannot start, so the game should reset with the reset key before the start. The reset key ("r" or "R") is reset the paddle and ball positions with the initial positions variables also it reset slope, ball's vertical direction (ballDirectionY = 0) and ball's horizontal direction (ballDirectionX = true). But the game will not start when the game is over (when the player sees the gameover screen) because both reset and start buttons check the "game_over_counter" variable. If there is an equality between "game_chance" and "game_over_counter" variables, a new game will not start.

```
//TIME ELEMENTS//
clock_t start, end, pause_end, pause_start, pause;
double execution_time;

void key_press(unsigned char key, int x, int y){
	if(!isMenuActive)
		switch (key) {
			case 's': /* start game */
			case 'S':
				if(start == 0)
					start = clock();
				if(pause_start != 0){
					pause_end = clock();
					pause += ((double)(pause_end -
pause_start))/CLOCKS_PER_SEC;
				}
				if (game_over && (game_chance !=
				game_over_counter) && resetButtonCheck) {
					game_over = 0;
					move_ball(current_game);
				}
				break;
			case 'r': /* reset game */
			case 'R':
				if (game_over && (game_chance !=
				game_over_counter)) {
					resetButtonCheck = true;
					ballDirectionY = 0;
					applySlope = 1;
					ballDirectionX = true;
					current_game++;
					ball_x_pos = initial_ball_x_pos;
					ball_y_pos = initial_ball_y_pos;
					paddle_y_pos = initial_paddle_y_pos;
					ball_trajectory =
					initial_ball_trajectory;
					game_over = 1;
					glutPostRedisplay();
				}
				break;
			case ESCAPE:
				exit(0);
		}
}
```

1.4.3. Game Over

As mentioned before, when the ball goes out of bounds from the paddle border and the player has no chances left, the game is over. If the ball's x position becomes greater than the window's width (window_width), the "game_over_counter" variable is increased by 1, and the global "game_over" variable is set to 1 by the program. In the "move_ball" function, the "game_over_counter" is increased by 1, and the "resetButtonCheck" variable is set to false due to the resetting process of the paddle and ball positions.

```
void move_ball(int game){
      .
      .
      .
      else if(ball_x_pos > (window_width)){
            pause_start = clock();
            game_over_counter++;
            game_over = 1;
            resetButtonCheck = false;
      }
```

If the player has a chance, the game starts again with the start key else there is no, the game over message shows in the middle of the screen with the player's score and game does not start.

1.5. Box Game

The last part of the game is about the game's extra scenario. For this scenario, A box game added to the pong game and players can choose the game mode on the main menu. The purpose of the box game is that the player tries to destroy all boxes on the screen with the ball touches. According to box sides, the ball moving direction changes. If the player destroys all boxes with a limited chance of failing, a success message shows to the player with her or his score.

When box game chooses, the "boxRowCounter" and "boxColumnCounter" are determined for the creation of the boxes according to window's height and weight values. The boxes have 60 pixels width and 100 heights, and they have spaces 20 pixels each other. The variable "boxRowCounter" is calculated by dividing the window height by 150, while the "boxColumnCounter" is calculated by subtracting 1 from the result of

dividing the window width by 150. For the calculation of the total boxes area to locate the boxes, the vertical border spaces are found with the subtraction of the boxes' heights and their spaces each of them from the window's height and this variable divide to 2 for finding the one side border space. The horizontal space variable is static as 70 and the boxes are located according to those spaces. In this way, the number of the boxes are located according to window's size. A box has four points and in a loop the point variables increase according to the last drew box's width, height and determined space variables. For the finding number of the boxes, the "boxRowCounter" and "boxColumnCounter" multiply and the result assigns to "boxCounter" integer variable.

Then, four float matrices are defined using the "boxCounter" variable. These matrices are determined for keeping the boxes' sides coordinates. The matrix which is defined for keeping the boxes' up and down coordinates is 3-dimensional. For a box's up coordinates, the upper two corner coordinates are kept on the x-axis and the upper y coordinate is kept of the box. If the ball is between these two corner x-coordinates and 5 units above or below the y-coordinate, it means the ball has touched the top of the box and the ball will move upwards with the slope value reset for a change when it touches the paddle. The same matrix is defined for the keeping bottom edge of the box and this matrix keeps the bottom two corner coordinates are kept on the x-axis and the upper y coordinate is kept. When the ball is positioned between the two corner x-coordinates and its y-coordinate is 5 units above or below, it indicates that the ball has hit the top of the box. In this case, the ball will move downwards with the slope value being reset for a change when it comes into contact with the paddle. The left side and right side matrices are the 4-dimensional for keeping all coordinates of the box. When the ball touches the left or right side of the box, the ball's y direction remains the same, but the "ballDirectionX" (the ball's x direction) is reversed, and the "slope" variable is assigned to "0.0" again for new random slope variables when the paddle is touched. Every box touching, the "removedBox" variable increases 1 and the defined "boxCheck" boolean array assigns to false on the relating box index. The "boxCheck" boolean array keeps the boxes' availability information, and it's all elements are assigned as true at the beginning of the box game. The program checks the related index of boxes with the "boxCheck" boolean array. If the "boxCheck" is true, it means the box is available and the program draws it. Otherwise, if it is false, the box is not drawn (which means the ball has touched the box).

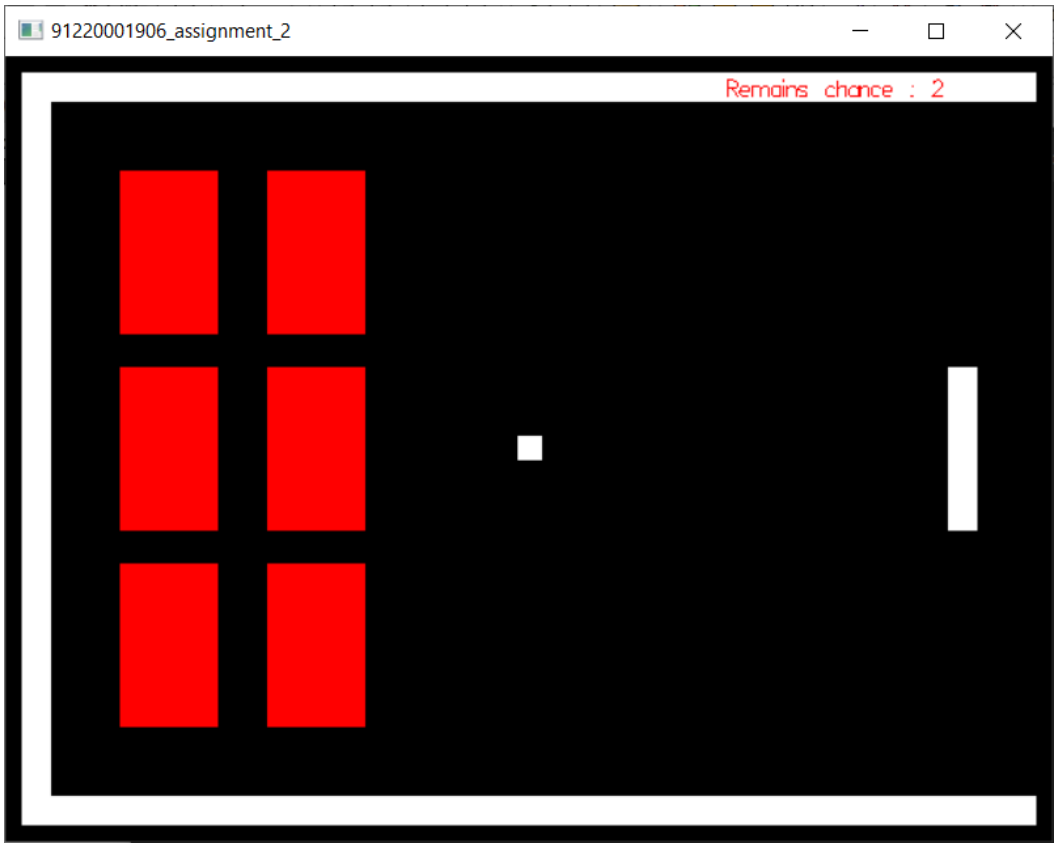
At the end of the box game, if the player destroys all the boxes (meaning that the "removedBox" variable equals the "boxCounter" variable), the program sets the "successTextCheck" variable to true to prevent the game over message from being displayed. It then calculates the player's score and prints a success message using the "stroke_output" function [5].

```
//BOX GAME//
bool boxCheck[100];
int removedBox = 0;
bool successTextCheck = false;

int main(int argc, char **argv){
	.
	.
	.
	for(int i = 0 ; i < 100 ; i ++){
		boxCheck[i] = true;
	}

/*Box Game*/
void boxGame(){
	int boxRowCounter = (window_height / 150);
	int spaceBtwBoxes = (boxRowCounter-1)*20;
	int borderSpace = (window_height - (spaceBtwBoxes +
	(boxRowCounter * 100)))/2;
	int btwSpace = 20;
	int boxColumnCounter = (window_width / 200) - 1;
	int upborderSpace = borderSpace+100;
	int boxCounter = boxRowCounter * boxColumnCounter;
	float boxRightSides[boxCounter][4];
	float boxLeftSides[boxCounter][4];
	float boxUp[boxCounter][3];
	float boxDown[boxCounter][3];
	boxCounter = 0;
	for(int a = 0 ; a < boxRowCounter ; a++){
		btwSpace = 20;
		for(int i = 0 ; i < boxColumnCounter ; i++){
			if(boxCheck[boxCounter]){
				glColor3d(1,0,0);
				glBegin(GL_QUADS);
				glVertex3f(50+btwSpace,borderSpace,0);
				glVertex3f(110+btwSpace,borderSpace,0);
				glVertex3f(110+btwSpace,upborderSpace,0);
				glVertex3f(50+btwSpace,upborderSpace,0);
				glEnd();
				//for right sides//
				boxRightSides[boxCounter][0]=110+btwSpace;
				boxRightSides[boxCounter][1]=borderSpace;
				boxRightSides[boxCounter][2]=upborderSpace;
				boxRightSides[boxCounter][3]=50+btwSpace;
```

```
                        //for left sides//
                        boxLeftSides[boxCounter][0]=50+btwSpace;
                        boxLeftSides[boxCounter][1]=borderSpace;
                        boxLeftSides[boxCounter][2]=upborderSpace;
                        boxLeftSides[boxCounter][3]=110+btwSpace;
                        //for up//
                        boxUp[boxCounter][0]=upborderSpace;
                        boxUp[boxCounter][1]=55+btwSpace;
                        boxUp[boxCounter][2]=115+btwSpace;
                        //for down//
                        boxDown[boxCounter][0]=borderSpace;
                        boxDown[boxCounter][1]=55+btwSpace;
                        boxDown[boxCounter][2]=115+btwSpace;
                        glColor3d(1,1,1);
                  }
                  boxCounter++;
                  btwSpace +=90;

            }
            borderSpace = upborderSpace + 20;
            upborderSpace = borderSpace+100;
      }

      for(int i = 0 ; i < boxCounter ; i++){
            if( (boxUp[i][0]+5 > ball_y_pos) && (boxUp[i][0]-5 <
            ball_y_pos) && (boxUp[i][1] < ball_x_pos) &&
            (boxUp[i][2] > ball_x_pos)){
                  printf("touch to (up) %d \n",i);
                  slope = 1.1;
                  ballDirectionY = 1;
                  boxCheck[i] = false;
                  removedBox++;
            }
            else if( (boxDown[i][0]+5 > ball_y_pos) &&
            (boxDown[i][0]-5 < ball_y_pos) && (boxDown[i][1] <
            ball_x_pos) && (boxDown[i][2] > ball_x_pos)){
                  printf("touch to (down) %d \n",i);
                  slope = 1.1;
                  ballDirectionY = 2;
                  boxCheck[i] = false;
                  removedBox++;
            }
            else if( (boxLeftSides[i][0] < ball_x_pos) &&
            (boxLeftSides[i][3] > ball_x_pos) &&
            (boxLeftSides[i][1] < ball_y_pos) &&
            (boxLeftSides[i][2] > ball_y_pos)){
                  printf("touch to (left) %d \n",i);
                  slope = 0.0;
                  ballDirectionX = !ballDirectionX;
                  boxCheck[i] = false;
                  removedBox++;
            }
```

```
            else if( (boxRightSides[i][0] > ball_x_pos) &&
            (boxRightSides[i][3] < ball_x_pos) &&
            (boxRightSides[i][1] < ball_y_pos) &&
            (boxRightSides[i][2] > ball_y_pos)){
                  if(fourthBoxBug || i== 4) //index 4. box bug fix
                        continue;
                  slope = 0.0;
                  ballDirectionX = !ballDirectionX;
                  boxCheck[i] = false;
                  removedBox++;
                  if(i == 4) //index 4. box bug fix
                        fourthBoxBug = true;
            }else{

            }
      }
            if(removedBox == boxCounter){
                  successTextCheck = true;
                  if(end == 0){
                        end = clock();
                        execution_time = (((double)(end -
start))/CLOCKS_PER_SEC) - pause;
                  }
                  glColor3d(0,1,0);
                  stroke_output((window_width/2)-150,
(window_height/2), 5, "SUCCESS!!");
                  stroke_output((window_width/2)-60,
(window_height/2)-50, 1, "Your score %fsn",execution_time);
                  glColor3d(1,1,1);
                  stroke_output((window_width/2)-60,
(window_height/2)-75, 1, "Press 'ESC' to exit.");
            }
}
```

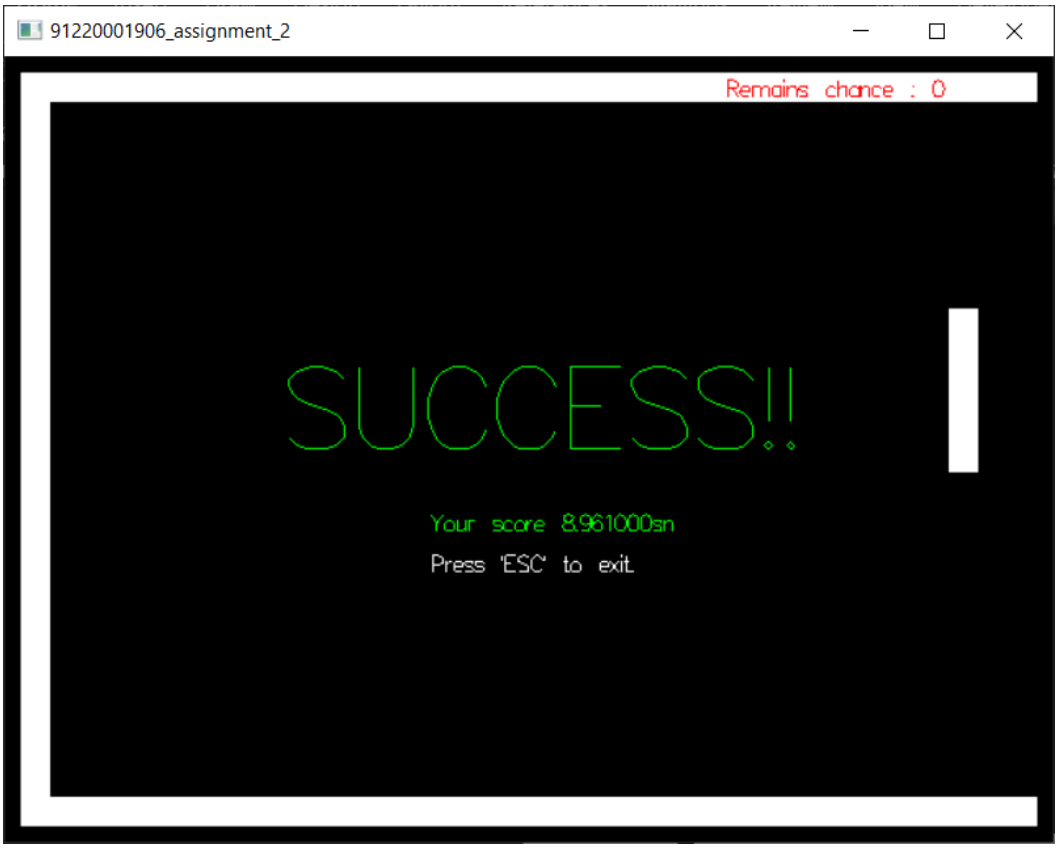
Note: A bug is experienced about the fourth index of box, and the code is included bug fix code piece.



Picture.4: The Box Game

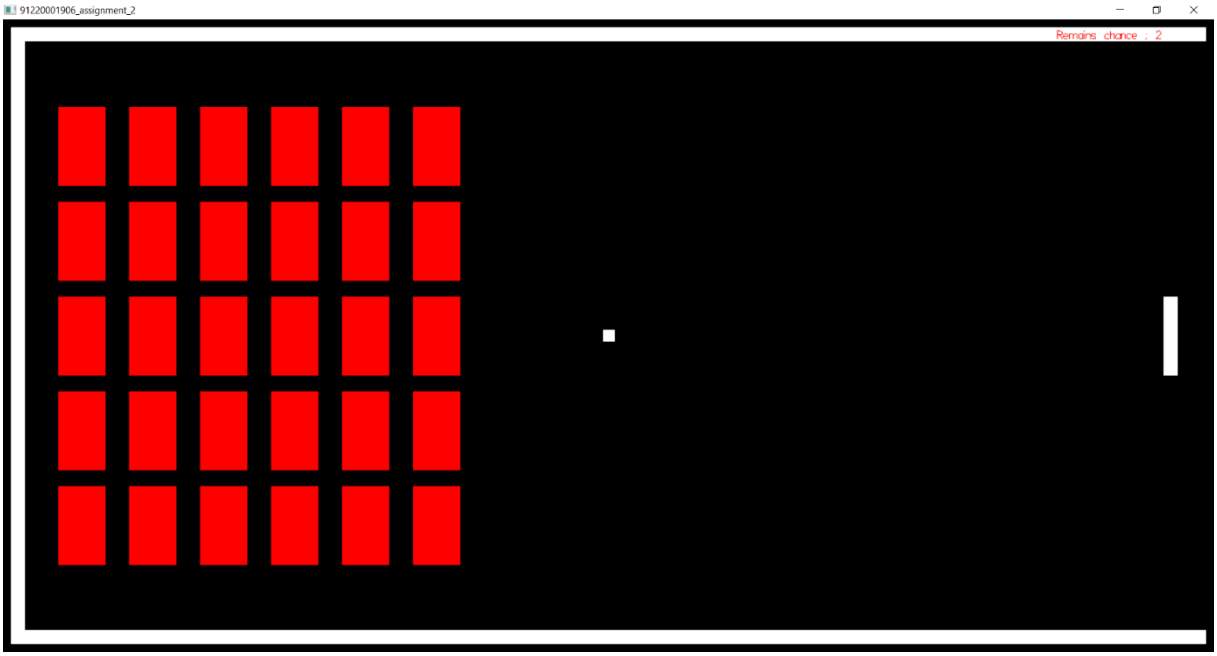
Picture.5: Failed Message

Picture.6: Training Mode



Picture.7: Success Message

Picture.8: Game Over Message



Picture.9: Full Windowed Box Game

References

1. https://www.britannica.com/topic/Pong
2. https://stackoverflow.com/questions/1793867/best-way-to-check-if-a-character-array-is-empty
3. https://www.opengl.org/resources/libraries/glut/spec3/node54.html
4. https://www.scaler.com/topics/c/c-string-declaration/
5. 2023.gfx.hw1.pdf
6. https://stackoverflow.com/questions/822323/how-to-generate-a-random-int-in-c
7. https://stackoverflow.com/questions/13408990/how-to-generate-random-float-number-in-c
8. https://www.javatpoint.com/random-function-in-c
9. https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm