Προγραμματιστική Άσκηση 1Α

Γραφικά Υπολογιστών Και Συστήματα Αλληλεπίδρασης

Αβανίδη Ιωάννα-Μαρία 4977

Παπαδοπούλου Μαρία 4920

1/11/2024

Περιεχόμενα

Περιγραφή της Εργασίας

Η άσκηση αυτή αφορά τη χρήση της βιβλιοθήκης OpenGL για τη δημιουργία ενός παραθύρου όπου σχεδιάζεται ένας λαβύρινθος και ένας χαρακτήρας που μπορεί να κινείται μέσα στον λαβύρινθο με συγκεκριμένες εντολές πληκτρολογίου.

i. Δημιουργία Παραθύρου

Ξεκινάμε για την δημιουργία του παραθύρου αλλάζοντας τις διαστάσεις που προϋπάρχουν στον κώδικα στις κατάλληλες (750 x 750) και συμπληρώνοντας τον τίτλο που μας ζητείται (Άσκηση 1Α – 2024). Σε αυτό το σημείο υπήρξε μια δυσκολία στην χρήση της ελληνικής γλώσσας. Μετά από έρευνα καταλήξαμε ότι χρειάζεται ο συνδυασμός χαρακτήρων u8 (κωδικοποίηση Unicode).

```
// Open a window and create its OpenGL context
window = glfwCreateWindow(750, 750, u8"Άσκηση 1Α – 2024", NULL, NULL);  // window dimensions and title
```

Στην συνέχεια ορίζουμε το χρώμα του background ως μαύρο (0.0f, 0.0f, 0.0f, 0.0f) διορθώνοντας την δοθείσα κωδικοποίηση.

```
// Black background
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
```

Στο τέλος πηγαίνουμε στο σημείο του κώδικα που αφορά το κλείσιμο του παραθύρου και μέσα στην εντολή while αλλάζουμε την παράμετρο ‘GLFW_KEY_ESCAPE’ σε ‘GLFW_KEY_Q’, ώστε να τερματίζει η εφαρμογή με το πλήκτρο ‘Q’.

```
while (glfwGetKey(window, GLFW_KEY_Q) != GLFW_PRESS &&  glfwWindowShouldClose(window) == 0);    // closing with Q button
```

ii. Σχεδίαση Λαβύρινθου

Ο λαβύρινθος αναπαρίσταται από έναν πίνακα 10x10 με τιμές ‘1’ για τους τοίχους (μπλε τετράγωνα) και ‘0’ για τα μονοπάτια. Ο λαβύρινθος σχεδιάζεται με τρίγωνα που αποτελούν τα τετράγωνα τοίχου. Το κέντρο του λαβύρινθου είναι το σημείο (0,0) του επιπέδου. Κάθε τετράγωνο έχει πλευρά μήκους 1.

Θεωρούμε ότι δουλεύουμε σε 2Δ χώρο και επομένως για όλα τα σημεία η Z συνιστώσα είναι 0.

Περιγραφή Αλγορίθμου:

- Ορισμός του πίνακα του λαβύρινθου.

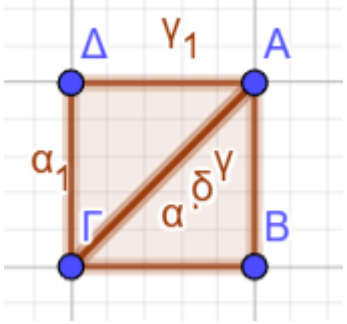
```
//Labyrinth
int labyrinth[10][10] = {
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {0, 0, 1, 1, 1, 1, 0, 1, 0, 1},
    {1, 0, 1, 0, 0, 0, 0, 1, 0, 1},
    {1, 0, 1, 0, 1, 1, 0, 1, 0, 1},
    {1, 0, 0, 0, 0, 1, 0, 0, 0, 1},
    {1, 0, 1, 1, 0, 1, 1, 1, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
    {1, 0, 1, 0, 1, 1, 0, 0, 0, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
};
```

- Ορισμός Συντεταγμένων για το πρώτο τετράγωνο του τοίχου. Οι συντεταγμένες στο ‘shape_0_buffer’ αναπαριστούν δύο τρίγωνα που σχηματίζουν ένα τετράγωνο τοίχου (A-B-C και A-D-C

```
// 1st square
static const GLfloat shape_0_buffer[] = {

    //first triangle
     -4.0f, 5.0f, 0.0f, // A
     -4.0f, 4.0f, 0.0f, // B
     -5.0f, 4.0f, 0.0f, // C

     //second triangle
     -4.0f, 5.0f, 0.0f, // A
     -5.0f, 5.0f, 0.0f, // D
     -5.0f, 4.0f, 0.0f, // C
};
```



A = (-4, 5)	
B = (-4, 4)	
Γ = (-5, 4)	
Δ = (-5, 5)	

- Δημιουργούμε διπλό βρόχο με την εντολή for που διατρέχει όλες τις γραμμές και τις στήλες του πίνακα ‘labyrinth’. Για κάθε στοιχείο που είναι ‘1’, δημιουργείται ένα τετράγωνο τοίχου.

```
for (int i = 0; i < 10; i++) { //column
    for (int j = 0; j < 10; j++) { //row
        if (labyrinth[i][j] == 1) { // wall
```

- Ορίζεται ένας προσωρινός πίνακας ‘temp_shape_buffer’ και γίνεται αντιγραφή των τιμών από το ‘shape_0_buffer’, ως αρχικοποίηση των συντεταγμένων για όλα τα τετράγωνα.

```
GLfloat temp_shape_buffer[18]; //temporate array

//copy the vertices
for (int k = 0; k < 18; k++) {
    temp_shape_buffer[k] = shape_0_buffer[k];   // initialization
}
```

- Για κάθε τετράγωνο, μετατοπίζονται οι συντεταγμένες, έτσι ώστε να τοποθετηθεί στη σωστή θέση μέσα στον λαβύρινθο, ανάλογα με την τοποθέτησή του στον πίνακα ‘labyrinth’. Η j προστίθεται στις συντεταγμένες x για τη στήλη, `temp_shape_buffer[k1] = temp_shape_buffer[k1] + j; //x coordinate movement` και η i αφαιρείται από τις y συντεταγμένες για τη γραμμή.

```
temp_shape_buffer[k1+1] = temp_shape_buffer[k1+1] - i; //y coordinate movement
```

Σε αυτό το σημείο χρησιμοποιήσαμε το εργαλείο GeoGebra για την καλύτερη κατανόηση της μετακίνησης των συντεταγμένων του κάθε τετραγώνου στον πίνακα ‘temp_shape_buffer’.

```
//μετακίνηση των συντεταγμενων
for (int k1 = 0; k1 < 18; k1+=3) {
    temp_shape_buffer[k1] = temp_shape_buffer[k1] + j; //x coordinate movement
    temp_shape_buffer[k1+1] = temp_shape_buffer[k1+1] - i; //y coordinate movement
}
```

- Δημιουργία και φόρτωση του Buffer για OpenGL όπως μας έχουν δοθεί από την θεωρία.

```
glGenBuffers(1, &vbo[count]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[count]);
glBufferData(GL_ARRAY_BUFFER, sizeof(temp_shape_buffer), temp_shape_buffer, GL_STATIC_DRAW);
count++;
```

- Συνεχίζουμε παρακάτω στον κώδικα, στην εντολή do-while, και συμπληρώνουμε στον κώδικα μια for εντολή ώστε να σχεδιάζει κάθε τετράγωνο τοίχου (55 τετράγωνα) που έχει αποθηκευτεί στους buffer των VBOs για τον λαβύρινθο.
- Η ‘glDrawArrays(GL_TRIANGLES, 0, 2 * 3)’ σχεδιάζει δύο τρίγωνα (6 κορυφές) για το τετράγωνο τοίχου.

```
for (int i = 0; i < 55; i++){
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[i]);
    glVertexAttribPointer(
        0,                  // attribute 0, must match the layout in the shader.
        3,                  // size
        GL_FLOAT,           // type
        GL_FALSE,           // normalized?
        0,                  // stride
        (void*)0            // array buffer offset
    );

    // Draw the triangle !
    glDrawArrays(GL_TRIANGLES, 0, 2 * 3); // 2 * 3 indices starting at 0 -> 2 triangle

    glDisableVertexAttribArray(0);
}
```

- Χρησιμοποιούμε ένα for loop για τις εντολές που καθαρίζουν την μνήμη που έχει δεσμευτεί για τα VBOs του OpenGL, τα οποία χρησιμοποιήθηκαν για τη σχεδίαση των τοίχων του λαβύρινθου.

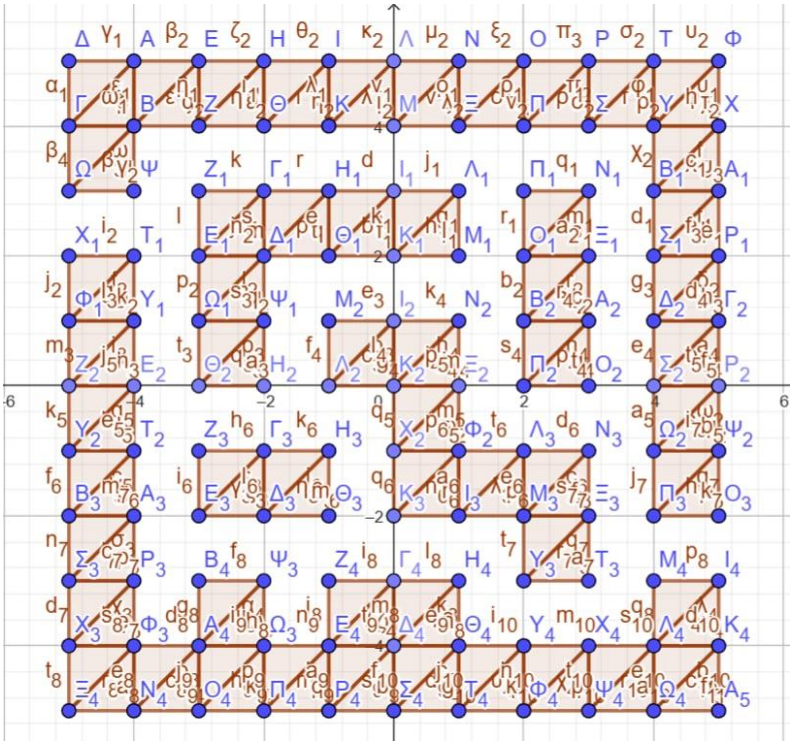
```
for (int i = 0; i < 55; i++) {
    // Cleanup VBO
    glDeleteBuffers(1, &vbo[i]);
}
```

- Αλλάζουμε το χρώμα του λαβύρινθου σε μπλε στον shader ‘ProjectFragmentShader.fragmentshader’.

```
void main()
{

    // Output color = blue
    color = vec3(0,0,1);

}
```



Σημείο

A = (-4, 5)	B_3 = (-5, -2)	Δ_3 = (-2, -2)	Z_3 = (-3, -1)	Θ_3 = (-1, -2)
A_1 = (5, 3)	B_4 = (-3, -3)	Δ_4 = (0, -4)	Z_4 = (-1, -3)	Θ_4 = (1, -4)
A_2 = (3, 1)	Γ = (-5, 4)	E = (-3, 5)	H = (-2, 5)	I = (-1, 5)
A_3 = (-4, -2)	Γ_1 = (-2, 3)	E_1 = (-3, 2)	H_1 = (-1, 3)	I_1 = (0, 3)
A_4 = (-3, -4)	Γ_2 = (5, 1)	E_2 = (-4, 0)	H_2 = (-2, 0)	I_2 = (0, 1)
A_5 = (5, -5)	Γ_3 = (-2, -1)	E_3 = (-3, -2)	H_3 = (-1, -1)	I_3 = (1, -2)
B = (-4, 4)	Γ_4 = (0, -3)	E_4 = (-1, -4)	H_4 = (1, -3)	I_4 = (5, -3)
B_1 = (4, 3)	Δ = (-5, 5)	Z = (-3, 4)	Θ = (-2, 4)	K = (-1, 4)
B_2 = (2, 1)	Δ_1 = (-2, 2)	Z_1 = (-3, 3)	Θ_1 = (-1, 2)	K_1 = (0, 2)
	Δ_2 = (4, 1)	Z_2 = (-5, 0)	Θ_2 = (-3, 0)	K_2 = (0, 0)

K_3 = (0, -2)	M_3 = (2, -2)	Ξ_3 = (3, -2)	Π_3 = (4, -2)	Σ_3 = (-5, -3)
K_4 = (5, -4)	M_4 = (4, -3)	Ξ_4 = (-5, -5)	Π_4 = (-2, -5)	Σ_4 = (0, -5)
Λ = (0, 5)	N = (1, 5)	O = (2, 5)	P = (3, 5)	T = (4, 5)
Λ_1 = (1, 3)	N_1 = (3, 3)	O_1 = (2, 2)	P_1 = (5, 2)	T_1 = (-4, 2)
Λ_2 = (-1, 0)	N_2 = (1, 1)	O_2 = (3, 0)	P_2 = (5, 0)	T_2 = (-4, -1)
Λ_3 = (2, -1)	N_3 = (3, -1)	O_3 = (5, -2)	P_3 = (-4, -3)	T_3 = (3, -3)
Λ_4 = (4, -4)	N_4 = (-4, -5)	O_4 = (-3, -5)	P_4 = (-1, -5)	T_4 = (1, -5)
M = (0, 4)	Ξ = (1, 4)	Π = (2, 4)	Σ = (3, 4)	Y = (4, 4)
M_1 = (1, 2)	Ξ_1 = (3, 2)	Π_1 = (2, 3)	Σ_1 = (4, 2)	Y_1 = (-4, 1)
M_2 = (-1, 1)	Ξ_2 = (1, 0)	Π_2 = (2, 0)	Σ_2 = (4, 0)	Y_2 = (-5, -1)

$Y_3 = (2, -3)$	$X_3 = (-5, -4)$		
$Y_4 = (2, -4)$	$X_4 = (3, -4)$		
$\Phi = (5, 5)$	$\Psi = (-4, 3)$		
$\Phi_1 = (-5, 1)$	$\Psi_1 = (-2, 1)$		
$\Phi_2 = (1, -1)$	$\Psi_2 = (5, -1)$		
$\Phi_3 = (-4, -4)$	$\Psi_3 = (-2, -3)$		
$\Phi_4 = (2, -5)$	$\Psi_4 = (3, -5)$		
$X = (5, 4)$	$\Omega = (-5, 3)$	$\Omega_3 = (-2, -4)$	
$X_1 = (-5, 2)$	$\Omega_1 = (-3, 1)$	$\Omega_4 = (4, -5)$	
$X_2 = (0, -1)$	$\Omega_2 = (4, -1)$		

Αντιμετωπίσαμε προκλήσεις σχετικά με τον υπολογισμό και την οργάνωση των συντεταγμένων των τοίχων, καθώς και τη σωστή τοποθέτησή τους στο παράθυρο. Για αυτόν τον λόγο χρησιμοποιήσαμε το εργαλείο GeoGebra, ώστε να υπάρχει οπτικοποίηση του προβλήματος.

iii. Δημιουργία Χαρακτήρα Α

Ο χαρακτήρας Α αντιπροσωπεύεται ως μπλε τετράγωνο με πλευρά 0.5 και κέντρο του είναι το κέντρο του τετραγώνου στο οποίο βρίσκεται.

Θεωρούμε ότι δουλεύουμε σε 2Δ χώρο και επομένως για όλα τα σημεία η Z συνιστώσα είναι 0.

Περιγραφή Αλγορίθμου:

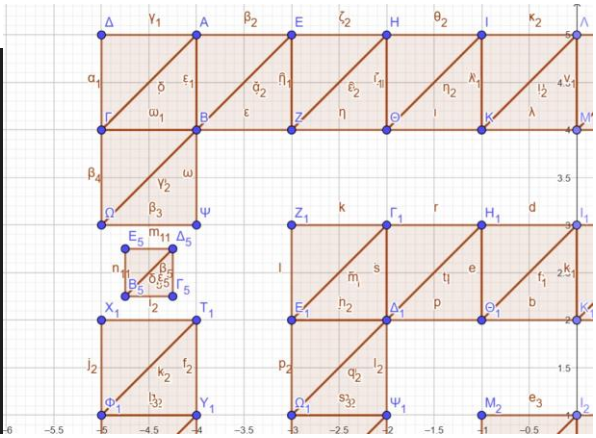
- Το τετράγωνο σχεδιάζεται με δύο τρίγωνα, τα οποία προσδιορίζονται από έξι κορυφές και οι συντεταγμένες τους αποθηκεύονται σε έναν πίνακα buffer, ‘square_buffer’.

```
//Square A
GLfloat square_buffer[] = {

    //first triangle
     -4.75f, 2.25f, 0.0f,
     -4.25f, 2.25f, 0.0f,
     -4.25f, 2.75f, 0.0f,

     //second triangle
     -4.25f, 2.75f, 0.0f,
     -4.75f, 2.75f, 0.0f,
     -4.75f, 2.25f, 0.0f,

};
```



- Δημιουργία και φόρτωση του buffer για OpenGL όπως μας έχουν δοθεί από την θεωρία.

```
GLuint sq;
glGenBuffers(1, &sq);
glBindBuffer(GL_ARRAY_BUFFER, sq);
glBufferData(GL_ARRAY_BUFFER, sizeof(square_buffer), square_buffer, GL_STATIC_DRAW);
```

- Συνεχίζουμε παρακάτω στον κώδικα, στην εντολή do-while, και προσαρμόζουμε τις εντολές κατάλληλα ώστε να γίνεται απεικόνιση του χαρακτήρα Α.
- Η ‘glDrawArrays(GL_TRIANGLES, 0, 2 * 3)’ σχεδιάζει δύο τρίγωνα (6 κορυφές) για το τετράγωνο του χαρακτήρα Α.

```
//Τετραγωνάκι
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, sq);
glVertexAttribPointer(
    0,                  // attribute 0, must match the layout in the shader.
    3,                  // size
    GL_FLOAT,           // type
    GL_FALSE,           // normalized?
    0,                  // stride
    (void*)0            // array buffer offset
);

// Draw the triangle !
glDrawArrays(GL_TRIANGLES, 0, 2 * 3); // 2 * 3 indices starting at 0 -> 2 triangle

glDisableVertexAttribArray(0);
```

- Χρησιμοποιούμε την εντολή που καθαρίζει την μνήμη που έχει δεσμευτεί για το VBO του OpenGL, το οποίο χρησιμοποιήθηκε για τη σχεδίαση του τετραγώνου του χαρακτήρα A .

```
glDeleteBuffers(1, &sq);
```

iv. Κίνηση Χαρακτήρα

Τα πλήκτρα 'L', 'J', 'K' και 'I' ελέγχουν την κίνηση του χαρακτήρα δεξιά, αριστερά, κάτω και πάνω, αντίστοιχα. Η κίνηση επιτρέπεται μόνο αν το κελί προορισμού δεν είναι τοίχος.

Θεωρούμε ότι δουλεύουμε σε 2Δ χώρο και επομένως για όλα τα σημεία η Z συνιστώσα είναι 0.

Περιγραφή Αλγορίθμου:

- Ορίζουμε την τρέχουσα θέση του τετραγώνου στη γραμμή και την στήλη του λαβυρίνθου και χρησιμοποιούμε ένα flag που δείχνει αν το πλήκτρο έχει ήδη πατηθεί, για να αποφευχθεί η επανάληψη κίνησης.

```
int row = 2;
int column = 0;

int key_pressed = 0; //flag
```

- Ο κώδικας περιέχει έναν βρόχο do-while που ανανεώνει την οθόνη και ελέγχει τις εισόδους του χρήστη. Μέσα σε αυτόν τον βρόχο γίνεται καθαρισμός οθόνης, χρήση shader και διαχείρηση των κλειδιών κίνησης (έλεγχος flag).

```
do {

    // Clear the screen
    glClear(GL_COLOR_BUFFER_BIT);

    // Use our shader
    glUseProgram(programID);

    glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);  /// Αυτό αφορά την κάμερα  - το αγνοείτε

    //When all keys released, flag = 0.
    if ((glfwGetKey(window, GLFW_KEY_L) == GLFW_RELEASE) && (glfwGetKey(window, GLFW_KEY_J) == GLFW_RELEASE) &&
        (glfwGetKey(window, GLFW_KEY_K) == GLFW_RELEASE) && (glfwGetKey(window, GLFW_KEY_I) == GLFW_RELEASE)) {
        key_pressed = 0;
    }
```

- Κίνηση του τετραγώνου:
  Πλήκτρο L (δεξιά κίνηση). Όταν το πλήκτρο L πατηθεί ελέγχεται αν μπορεί να μετακινηθεί το τετράγωνο προς τα δεξιά (ελέγχει αν η επόμενη στήλη είναι κενή). Αν ναι, η στήλη αυξάνεται (column++). Στη συνέχεια, οι συντεταγμένες του τετραγώνου ενημερώνονται και το buffer ανανεώνεται.

```
if (glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS && !key_pressed) {
    if ((labyrinth[row][column + 1] == 0) && (column + 1 < 10)) {
        column++;
        for (int k1 = 0; k1 < 18; k1 += 3) {
            square_buffer[k1] = square_buffer[k1] + 1; //move 1 square right
        }

        glGenBuffers(1, &sq);
        glBindBuffer(GL_ARRAY_BUFFER, sq);
        glBufferData(GL_ARRAY_BUFFER, sizeof(square_buffer), square_buffer, GL_STATIC_DRAW);
        key_pressed = 1;//flag true
    }
}
```

Πλήκτρο J (αριστερή κίνηση). Παρόμοια λογική με το πλήκτρο L, αλλά για την αριστερή κίνηση.

```
if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS && !key_pressed ) {
    if ((labyrinth[row][column - 1] == 0) && (column - 1 >= 0)) {
        column--;
        for (int k2 = 0; k2 < 18; k2 += 3) {
            square_buffer[k2] = square_buffer[k2] - 1; //move one square left
        }

        glGenBuffers(1, &sq);
        glBindBuffer(GL_ARRAY_BUFFER, sq);
        glBufferData(GL_ARRAY_BUFFER, sizeof(square_buffer), square_buffer, GL_STATIC_DRAW);
        key_pressed = 1;//flag true
    }
}
```

Πλήκτρο K (κάτω κίνηση). Ελέγχει την κίνηση προς τα κάτω, αντίστοιχα.

```
if (glfwGetKey(window, GLFW_KEY_K) == GLFW_PRESS && !key_pressed) {
    if ((labyrinth[row + 1][column] == 0) && (row + 1 < 10)) {
        row++;
        for (int k3 = 0; k3 < 18; k3 += 3) {
            square_buffer[k3 + 1] = square_buffer[k3 + 1] - 1; //move one square down
        }

        glGenBuffers(1, &sq);
        glBindBuffer(GL_ARRAY_BUFFER, sq);
        glBufferData(GL_ARRAY_BUFFER, sizeof(square_buffer), square_buffer, GL_STATIC_DRAW);
        key_pressed = 1;//flag true
    }
}
```

Πλήκτρο I (άνω κίνηση). Παρόμοια λογική για την κίνηση προς τα πάνω.

```
if (glfwGetKey(window, GLFW_KEY_I) == GLFW_PRESS && !key_pressed) {
    if ((labyrinth[row - 1][column] == 0) && (row - 1 >= 0)) {
        row--;
        for (int k4 = 0; k4 < 18; k4 += 3) {
            square_buffer[k4 + 1] = square_buffer[k4 + 1] + 1; //move one square up
        }

        glGenBuffers(1, &sq);
        glBindBuffer(GL_ARRAY_BUFFER, sq);
        glBufferData(GL_ARRAY_BUFFER, sizeof(square_buffer), square_buffer, GL_STATIC_DRAW);
        key_pressed = 1;//flag true
    }
}
```

Κατά την υλοποίηση, αντιμετωπίστηκαν προκλήσεις σχετικά με τον χειρισμό της κίνησης, ώστε να γίνεται ομαλά και να εμποδίζεται η διέλευση του χαρακτήρα από τους τοίχους.

Πληροφορίες σχετικά με την υλοποίηση

Λειτουργικό Σύστημα: Windows 10/11

Περιβάλλον/Editor: Visual Studio x86

Ιδιαιτερότητες στον τρόπο ανάλυσης/χρήσης: --

Άλλες ιδιαιτερότητες: --

Σύντομη αξιολόγηση της λειτουργίας της ομάδας

Κατά την υλοποίηση της άσκησης, η ομάδα μας εργάστηκε σε κλίμα συνεργασίας. Η κύρια δυσκολία που αντιμετωπίσαμε ήταν ο περιορισμένος χρόνος, καθώς λειτουργήσαμε ταυτόχρονα και δεν καταμερίσαμε το φόρτο εργασίας. Επίσης, υπήρξαν διαφωνίες ως προς τη μεθοδολογία ανάλυσης, τις οποίες επιλύσαμε μέσω συζήτησης και συμβιβασμών. Σε γενικές γραμμές, η ομάδα δούλεψε με ενότητα και υπευθυνότητα. Ωστόσο, για μελλοντικές εργασίες θα ήταν χρήσιμο να οργανώσουμε τον καταμερισμό των εργασιών για εξοικονόμηση χρόνου.

Αναφορές-Πηγές

- ✓ https://www.opengl-tutorial.org/beginners-tutorials/tutorial-2-the-first-triangle/
- ✓ https://www.geogebra.org/classic?lang=el
- ✓ https://stackoverflow.com/questions/42538578/unicode-character-greek-capital-letter-delta-is-rendered-as-white-vertical-re