Εργαστηριακή Εργασία 1

Γενικά Στοιχεία

Μάθημα

Γραφικά Υπολογιστών και Συστήματα Αλληλεπίδρασης

Στοιχεία ομάδας

Α.Μ: 5108
Ονοματεπώνυμο: Κουτσονικολής Νικόλαος

Α.Μ: 4937
Ονοματεπώνυμο: Κωστόπουλος Αλέξανδρος

Ημερομηνία
31 Οκτωβρίου 2024

Περιγραφή Εργασίας

Μέρος 1ο

Το πρώτο μέρος της αναφοράς αφιερώνεται στην αναλυτική και σαφή περιγραφή της υλοποίησης μας τηρώντας τη σειρά των ερωτημάτων της εκφώνησης του project.

Θα παρατεθούν φωτογραφίες και από σημεία του κώδικα για επεξήγηση.

Υποσημείωση: Ο παραδοτέος κώδικας περιέχει επίσης σχόλια με στόχο την καλύτερη ανάγνωση και κατανόηση του

Ερώτημα (α)

(i) (10%) Φτιάξτε ένα πρόγραμμα που θα ανοίγει ένα βασικό παράθυρο 750 x 750. Το background του παραθύρου στην περιοχή εργασίας να είναι μαύρο. Το παράθυρο θα έχει τίτλο «Άσκηση 1Α - 2024» (με ελληνικούς χαρακτήρες – **όχι greeklish**). Με το πλήκτρο **Q** η εφαρμογή τερματίζει.

Καλούμαστε να φτιάξουμε πρόγραμμα στην **OpenGL** το οποίο θα ανοίγει παράθυρο με διαστάσεις **750x750**, θα έχει **μαύρο background**, θα φέρει τίτλο "**Άσκηση 1Α - 2024**" και θα εκτελεί έξοδο με το πάτημα του πλήκτρου **Q**.

Στη δική μας υλοποίηση η έξοδος έχει προγραμματιστεί να γίνεται είτε πατώντας το **X** στο πάνω-δεξιά μέρος του παραθύρου, είτε με το **Q**(όπως ζητείται) είτε πλοηγώντας επιτυχώς τον κινούμενο χαρακτήρα στο τέλος του λαβύρινθου που θα εξηγήσουμε παρακάτω.

Παρατίθεται ο κώδικας:

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

```
// Black background(r=0,g=0,b=0,alpha=0)
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
```

```
while (!finishedFlag && glfwGetKey(window, GLFW_KEY_Q) != GLFW_PRESS && glfwWindowShouldClose(window) == 0);
```

Η έξοδος του προγράμματος τσεκάρεται σε κάθε πέρασμα του main loop και αν έχει γίνει το **finishedFlag = true** ή έχει πατηθεί **Q** τότε κλείνει το παράθυρο και τερματίζει το πρόγραμμα μας.

Ερώτημα (β)

(ii) (30%) Το πρόγραμμα ξεκινάει ζωγραφίζοντας έναν λαβύρινθο (Εικόνα 1) και ένα μικρότερο τετράγωνο (αντιπροσωπεύει έναν χαρακτήρα που κινείται και το αγνοούμε για το ερώτημα αυτό). Ο λαβύρινθος σχηματίζεται ζωγραφίζοντας τετράγωνα μπλε χρώματος, που αντιστοιχούν στα τοιχώματα του λαβύρινθου, οπότε μπορεί να αναπαρασταθεί από έναν διδιάστατο πίνακα (Εικόνα 2) που περιέχει τιμές 0 ή 1. Η τιμή "1" αντιστοιχεί σε τοίχος, ενός η τιμή "0" αντιστοιχεί σε μονοπάτι. Το πρόγραμμά σας ζωγραφίζει μόνο τα μπλε τετράγωνα, χρησιμοποιώντας κατάλληλα τρίγωνα. Το κέντρο του λαβύρινθου είναι το σημείο (0,0) του επιπέδου. Κάθε τετράγωνο έχει πλευρά μήκους 1.

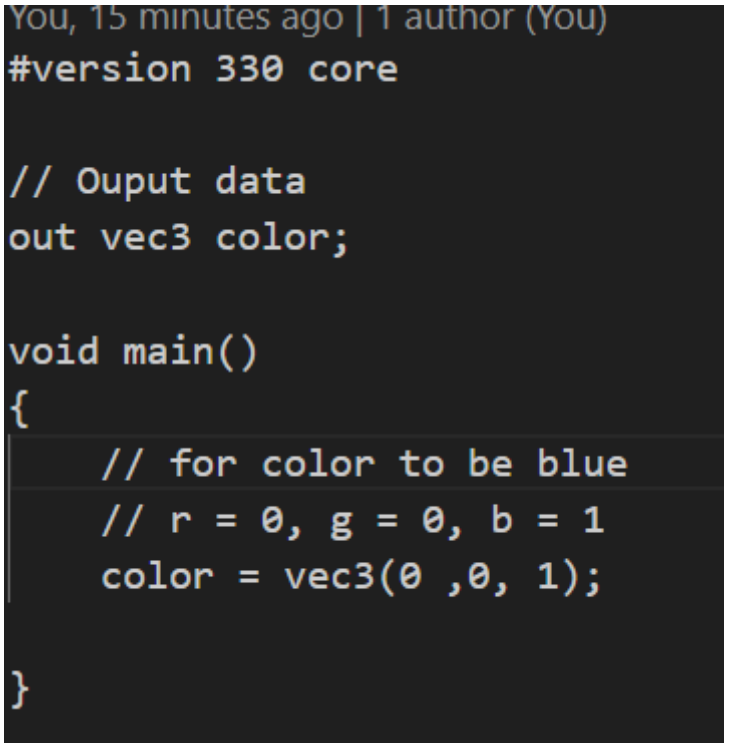
Πρέπει να προσδιορίστε τις συντεταγμένες των σημείων των τριγώνων που σχηματίζουν τα μπλε τετράγωνα (τοίχους) και να τις αποθηκεύσετε σε κατάλληλο πίνακα μέσα στον κώδικά σας. Ο προσδιορισμός των συντεταγμένων να δοθεί αναλυτικά, μαζί με σχέδιο στο readme. Θεωρούμε ότι δουλεύουμε σε 2Δ χώρο και επομένως για όλα τα σημεία η Ζ συνιστώσα είναι 0

Το συγκεκριμένο ερώτημα μας ζητάει να κατασκευάσουμε σε πρώτη φάση τον **λαβύρινθο**, μέσα στον οποίο, ο χαρακτήρας που θα φτιάξουμε σε επόμενο βήμα, να μπορεί να κινηθεί μέσα με στόχο από την **Α.Θ**(αρχική θέση) να φτάσει στην **Τ.Θ**(τελική θέση).

Ο λαβύρινθος θέλουμε να έχει κέντρο το (0,0) και να εκτείνεται στον άξονα x-άξονα από (-5,5) και στον y-άξονα από (-5, 5). Τα τοιχώματα του λαβυρίνθου σχηματίζονται από παρατεταγμένα μπλε τετράγωνα με πλευρές μήκους 1, το καθένα από τα οποία κατασκευάζεται από 2 τρίγωνα των οποίων τα σημεία καλούμαστε να βρούμε.

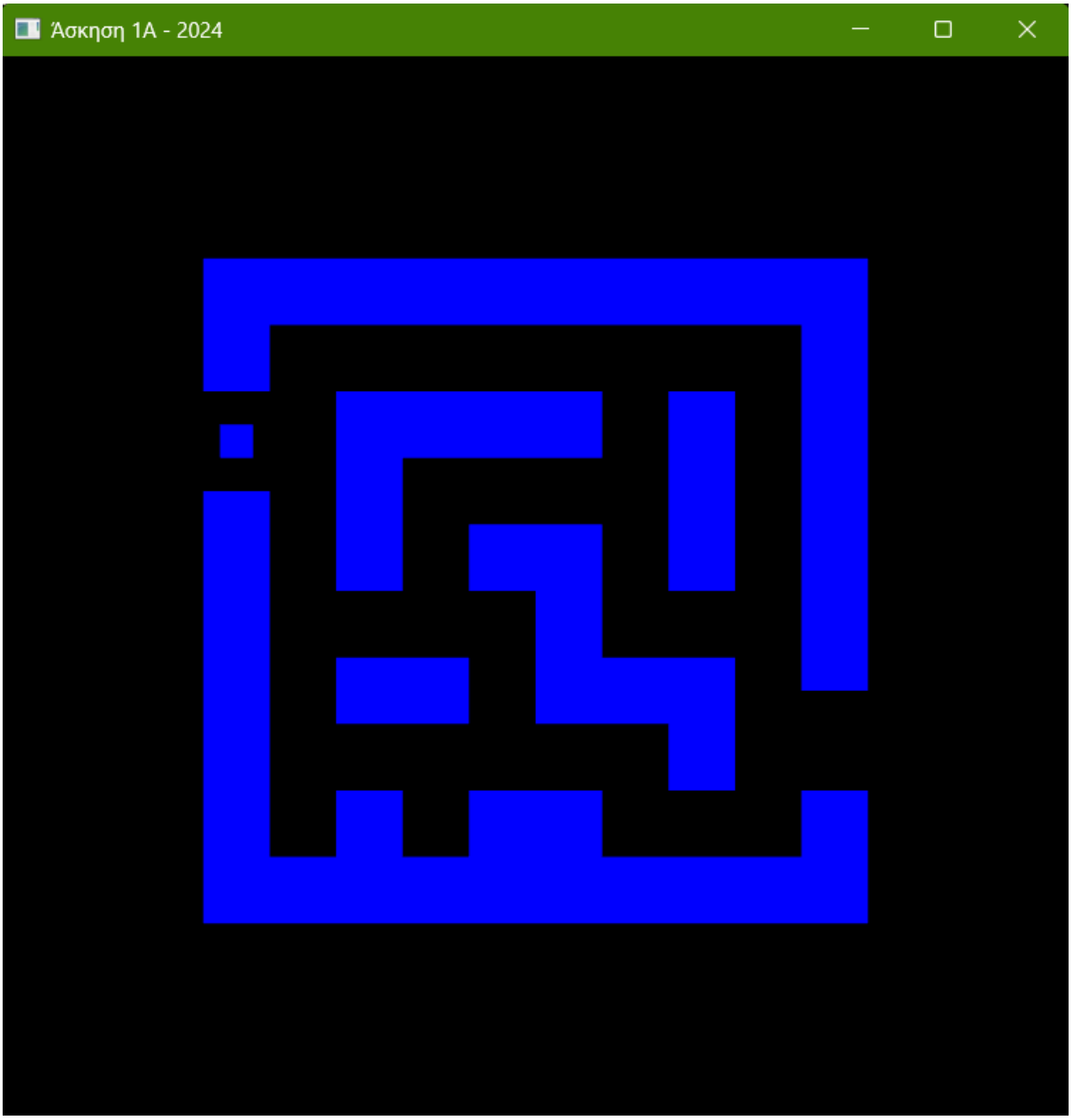
Υποσημείωση: Εργαζόμαστε στον 2D χώρο και επομένως η συνιστώσα στον z-αξονα είναι 0, ωστόσο στα vertex data arrays του λαβυρίνθου και του χαρακτήρα(αργότερα) συμπεριλάβαμε και τη μηδενική αυτή συνιστώσα για κάθε vertex.

Αρχικά αλλάζουμε κατάλληλα τον κώδικα στον fragmentShader, ώστε να σετάρουμε το μπλε χρώμα

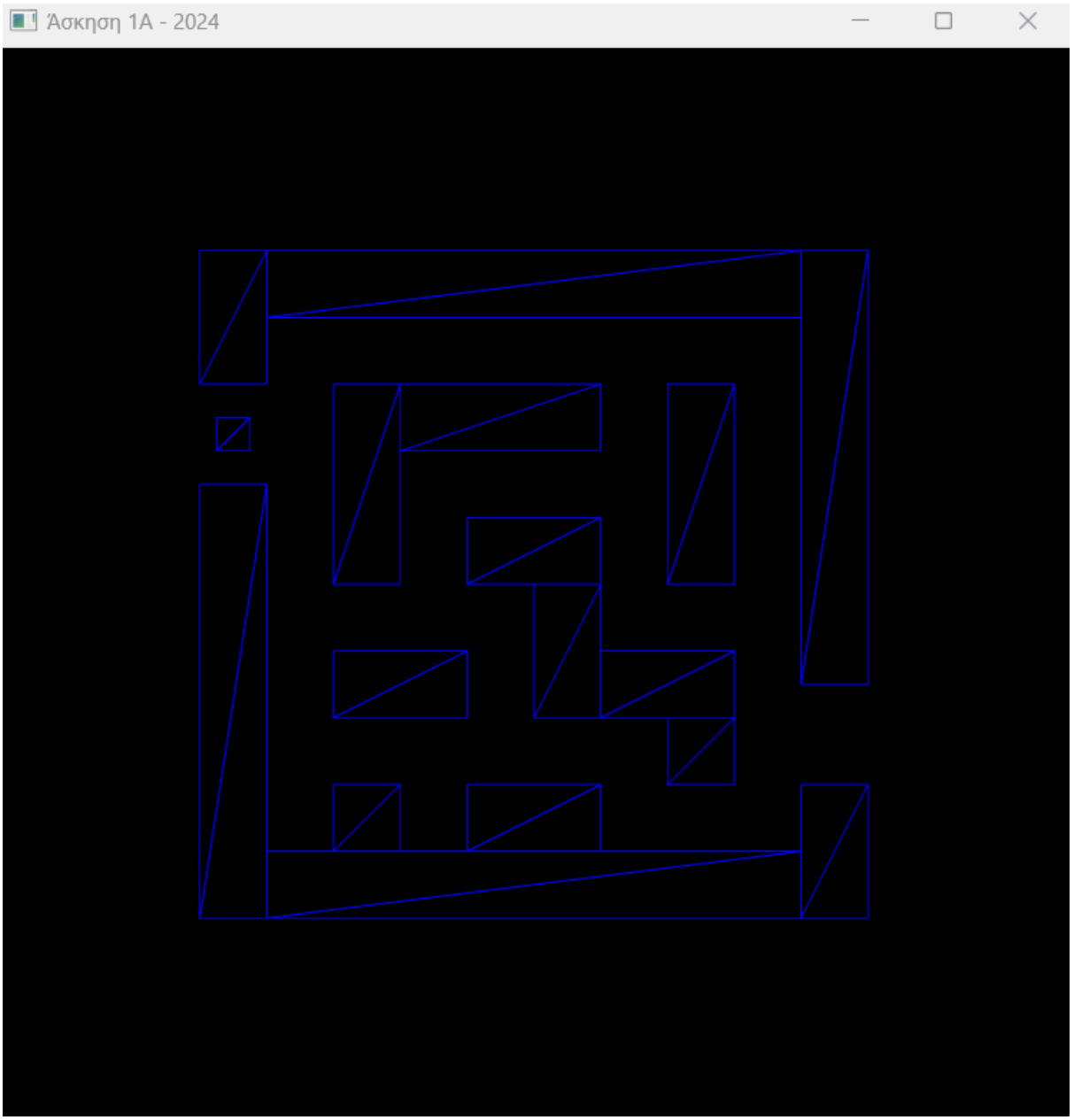


Παρακάτω θα δείτε την αναπαράσταση που θέλουμε να πετύχουμε(μην δώσετε για τώρα σημασία στο μικρό τετράγωνο - χαρακτήρα).

Τελική αναπαράσταση με χρώμα



Αναπαράσταση εσωτερικών τριγώνων



(Σύνολο 32 τριγωνα του λαβυρίνθου)

Πίνακας συντεταγμένων του λαβυρίνθου

```
// amount of vetrices(every triangle has 3)
int number_of_maze_vertices = 96; // 32 triangles
* 3 vertices

GLfloat maze_vertex_buffer_data[] = {
    /*1st rectangle - top-left border*/
    -5.0f, 5.0f, 0.0f,
    -5.0f, 3.0f, 0.0f,
    -4.0f, 5.0f, 0.0f,
    -4.0f, 3.0f, 0.0f,

    /*2nd rectangle - bottom-left border*/
    -5.0f, 1.5f, 0.0f,
    -5.0f, -5.0f, 0.0f,
    -4.0f, 1.5f, 0.0f,
    -4.0f, -5.0f, 0.0f,

    /*3nd rectangle - top border*/
    -4.0f, 5.0f, 0.0f,
    -4.0f, 4.0f, 0.0f,
    4.0f, 5.0f, 0.0f,
    4.0f, 4.0f, 0.0f,

    /*4th rectangle - top-right border*/
    4.0f, 5.0f, 0.0f,
    4.0f, -1.5f, 0.0f,
```

```
    5.0f, 5.0f, 0.0f,
    5.0f, -1.5f, 0.0f,

    /*5th rectangle - bottom-right border*/
    4.0f, -3.0f, 0.0f,
    4.0f, -5.0f, 0.0f,
    5.0f, -3.0f, 0.0f,
    5.0f, -5.0f, 0.0f,

    /*6th rectangle - bottom border*/
    -4.0f, -4.0f, 0.0f,
    -4.0f, -5.0f, 0.0f,
    4.0f, -4.0f, 0.0f,
    4.0f, -5.0f, 0.0f,

    /*7th rectangle - bottom bump #1*/
    -3.0f, -3.0f, 0.0f,
    -3.0f, -4.0f, 0.0f,
    -2.0f, -3.0f, 0.0f,
    -2.0f, -4.0f, 0.0f,

    /*8th rectangle - bottom bump #2*/
    -1.0f, -3.0f, 0.0f,
    -1.0f, -4.0f, 0.0f,
    1.0f, -3.0f, 0.0f,
    1.0f, -4.0f, 0.0f,

    /*9th rectangle - horizontal mid-left wall*/
    -3.0f, -1.0f, 0.0f,
    -3.0f, -2.0f, 0.0f,
```

```
    -1.0f, -1.0f, 0.0f,
    -1.0f, -2.0f, 0.0f,

    /*10th & 11th rectangles - 90deg mid-top
walls*/
    -3.0f, 3.0f, 0.0f,
    -3.0f, 0.0f, 0.0f,
    -2.0f, 3.0f, 0.0f,
    -2.0f, 0.0f, 0.0f,
     //////
    -2.0f, 3.0f, 0.0f,
    -2.0f, 2.0f, 0.0f,
    1.0f, 3.0f, 0.0f,
    1.0f, 2.0f, 0.0f,

    /*12th to 15th rectangles - zig zag walls*/
    -1.0f, 1.0f, 0.0f,
    -1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    /////
    0.0f, 0.0f, 0.0f,
    0.0f, -2.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    1.0f, -2.0f, 0.0f,
    /////
    1.0f, -1.0f, 0.0f,
    1.0f, -2.0f, 0.0f,
    3.0f, -1.0f, 0.0f,
    3.0f, -2.0f, 0.0f,
```

```
    /////
    2.0f, -2.0f, 0.0f,
    2.0f, -3.0f, 0.0f,
    3.0f, -2.0f, 0.0f,
    3.0f, -3.0f, 0.0f,

    /*16th rectangle - vertical top-right wall*/
    2.0f, 3.0f, 0.0f,
    2.0f, 0.0f, 0.0f,
    3.0f, 3.0f, 0.0f,
    3.0f, 0.0f, 0.0f,
};

unsigned int maze_indices[] = {
    0, 1, 2,
    1, 2, 3,

    4, 5, 6,
    5, 6, 7,

    8, 9, 10,
    9, 10, 11,

    12, 13, 14,
    13, 14, 15,

    16, 17, 18,
    17, 18, 19,

    20, 21, 22,
```

```
    21, 22, 23,

    24, 25, 26,
    25, 26, 27,

    28, 29, 30,
    29, 30, 31,

    32, 33, 34,
    33, 34, 35,
    36, 37, 38,
    37, 38, 39,

    40, 41, 42,
    41, 42, 43,

    44, 45, 46,
    45, 46, 47,

    48, 49, 50,
    49, 50, 51,

    52, 53, 54,
    53, 54, 55,

    56, 57, 58,
    57, 58, 59,

    60, 61, 62,
    61, 62, 63,};
```

Αρχικοποίηση του buffer, VAO, EBO και σχεδιασμός

```
// setup maze
    glBindVertexArray(VAOs[0]);
    glBindBuffer(GL_ARRAY_BUFFER,
mazevertexbuffer);
    glBufferData(GL_ARRAY_BUFFER,
sizeof(maze_vertex_buffer_data),
maze_vertex_buffer_data, GL_STATIC_DRAW); //
GL_STATIC_DRAW makes buffer immutable
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,
EBOs[0]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
sizeof(maze_indices), maze_indices,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT,
GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
```

Ο συγκεκριμένος κώδικας χρησιμοποιείται για τη ρύθμιση της απεικόνισης του λαβύρινθου.

Ακολουθούν τα βήματα που εκτελούνται:

1. **Bind Vertex Array Object (VAO)**:
    - `glBindVertexArray(VAOs[0]);`: Ορίζει ότι θα χρησιμοποιηθεί το πρώτο VAO, το οποίο περιέχει τα απαραίτητα δεδομένα για την απόδοση του λαβύρινθου.

2. **Bind Vertex Buffer Object (VBO)**:
    - `glBindBuffer(GL_ARRAY_BUFFER, mazevertexbuffer);`: Συνδέει το VBO που θα περιέχει τα δεδομένα των κορυφών του λαβύρινθου

3. **Γέμισμα VBO με Δεδομένα**:
    - `glBufferData(GL_ARRAY_BUFFER, sizeof(maze_vertex_buffer_data), maze_vertex_buffer_data, GL_STATIC_DRAW);`: Γεμίζει το VBO με τα δεδομένα των κορυφών του λαβύρινθου(βλέπε **σελ. 7-10**) και καθορίζει ότι αυτά τα δεδομένα δεν θα αλλάξουν με το GL_STATIC_DRAW attribute

4. **Bind Element Buffer Object (EBO)**:
    - `glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBOs[0]);`: Συνδέει το EBO που θα περιέχει τους δείκτες των στοιχείων του λαβύρινθου

5. **Γέμισμα EBO με τα Indices**:
    - `glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(maze_indices), maze_indices, GL_STATIC_DRAW);`: Γεμίζει το EBO με τους δείκτες των στοιχείων του λαβύρινθου

6. **Προσδιορισμός Vertex Attribute Pointer**:
    - `glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);`: Καθορίζει κάποια χαρακτηριστικά σχετικά με τον τρόπο με τον οποίο τα δεδομένα κορυφών είναι αποθηκευμένα στο VBO. Εδώ, κάθε κορυφή έχει 3 συντεταγμένες (x, y, z) και είναι αποθηκευμένη ως float.

7. **Ενεργοποίηση Vertex Attribute Array**:
    - `glEnableVertexAttribArray(0);`: Ενεργοποιεί τα χαρακτηριστικά που καθορίστηκαν στο προηγούμενο βήμα.

Έτσι, ορίζουμε και φορτώνουμε τα δεδομένα του λαβύρινθου στους αντίστοιχους buffers, έτοιμα για απεικόνιση.

```
// draw maze
glBindVertexArray(VAOs[0]);
glDrawElements(GL_TRIANGLES, 96, GL_UNSIGNED_INT, 0);
```

Τελικά, μέσα στο render loop γίνεται η σχεδίαση/απεικόνιση του λαβυρίνθου.
Μέσω της **glDrawElements** η οποία είναι πιο ελαφριά από την **glDrawArrays** στην περίπτωση μας, καθώς έχουμε πολλά τετράγωνα και με την χρήση της glDrawArrays θα χρειαζόμασταν 6 vertices(2 τρίγωνα * 3 vertices ανά τρίγωνο) εκ των οποίων κάποια είναι διπλότυπα σε κάθε τετράγωνο.
Αντιθέτως δεν χρειάζονται παρά μόνο 4 vertices για να σχεδιάσουμε το κάθε τετράγωνο, συνδυάζοντας τα κατάλληλα τρίγωνα. Εκεί μας είναι χρήσιμος ο πίνακας των indices, ο οποίος κάνει αυτή την αντιστοίχιση.

Ερώτημα (γ)

(iii) (20%) Το μικρότερο μπλε τετράγωνο, χαρακτήρας **A** (Εικόνα 1), αντιστοιχεί σε έναν κινούμενο αντικείμενο που θα διασχίζει τον λαβύρινθο. Το Α είναι ένα τετράγωνο με πλευρά μήκους 0.5 και το κέντρο του είναι το κέντρο του τετραγώνου στο οποίο βρίσκεται.

Πρέπει να προσδιορίσετε τις συντεταγμένες του τετραγώνου Α (εκφράζοντας το ως 2 τρίγωνα) τις οποίες θα αποθηκεύσετε κατάλληλα. Στο readme να προστεθεί απεικόνιση/σχέδιο με τον προσδιορισμό των συντεταγμένων των σημείων των τριγώνων του Α.

Σε αυτό το ερώτημα, καλούμαστε να κατασκευάσουμε τον κινούμενο χαρακτήρα, τον οποίο θα αναπαριστά ένα μπλε τετράγωνο με πλευρά μήκους 0.5 και κέντρο το κέντρο του τετραγώνου που θα βρίσκεται.

Πίνακας συντεταγμένων χαρακτήρα

```
GLfloat char_vertex_buffer_data[] = {
    -4.75f, 2.5f, 0.0f, // top-left
    -4.75f, 2.0f, 0.0f, // bottom-left
    -4.25f, 2.5f, 0.0f, // top-right
    -4.25f, 2.0f, 0.0f, // bottom-right
};
```

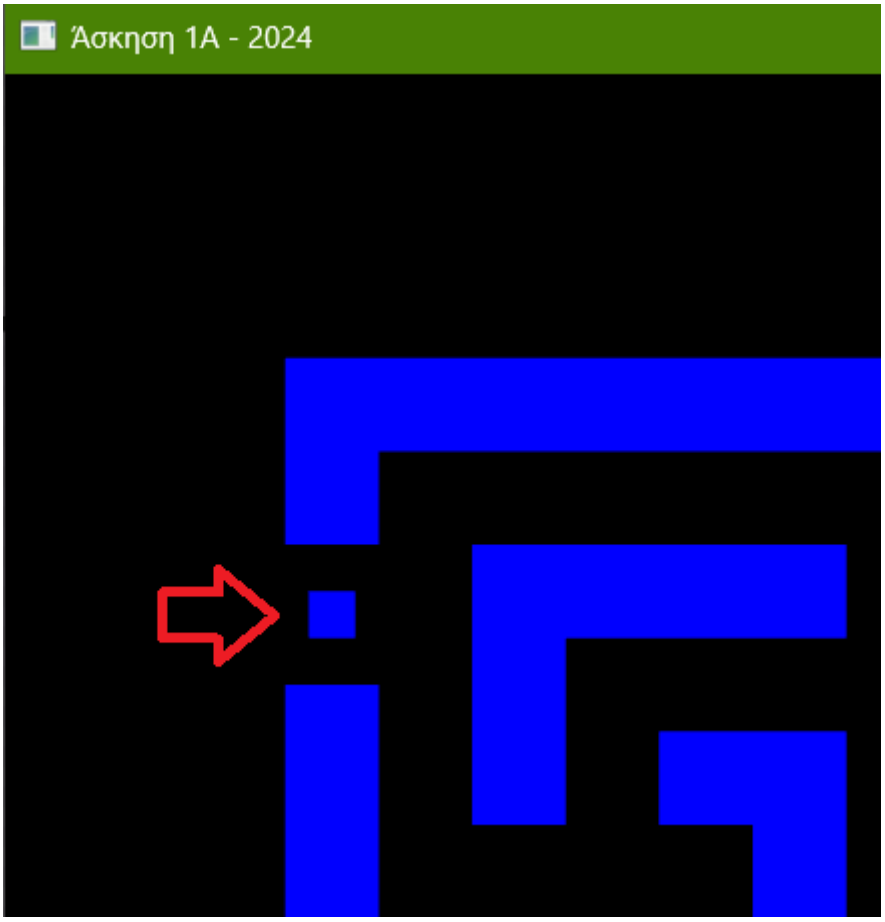
```
unsigned int char_indices[] = {
    0, 1, 2,
    1, 2, 3,
};
```

(τα 2 τρίγωνα που το αποτελούν)

Με παρόμοιο τρόπο όπως στο προηγούμενο ερώτημα, κατασκευάζουμε τον buffer, τη VAO, το EBO και ρυθμίζουμε όλες τις παραμέτρους για να μπορέσουμε να σχεδιάσουμε τον χαρακτήρα στην οθόνη. Η διαδικασία μέχρι εδώ είναι ίδια με τον λαβύρινθο.

```
// setup moveable character
    glBindVertexArray(VAOs[1]);
    glBindBuffer(GL_ARRAY_BUFFER,
charvertexbuffer);
    // GL_DYNAMIC_DRAW makes the buffer mutable,
    // and since we move our character that means
we need to make it dynamic
    glBufferData(GL_ARRAY_BUFFER,
sizeof(char_vertex_buffer_data),
char_vertex_buffer_data, GL_DYNAMIC_DRAW);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,
EBOs[1]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
sizeof(char_indices), char_indices,
GL_DYNAMIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT,
GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);
```

Η διαφορά είναι πως αυτή τη φορά η VBO(**char_vertex_buffer_data**) είναι δυναμική, που σημαίνει ότι μπορούμε να αλλάζουμε τα δεδομένα της και επομένως τις συντεταγμένες του τετραγώνου. Έτσι επιτυγχάνουμε κίνηση στους δύο άξονες(x,y).



(Αρχική θέση του χαρακτήρα)

Έπειτα καλούμαστε να υλοποιήσουμε αλγορίθμους για την κίνηση του χαρακτήρα και την ανίχνευση συγκρούσεων με τα τοιχώματα του λαβύρινθου, με σκοπό στην κίνηση του μόνο μέσα σε αυτόν και όχι μέσα από τα τοιχώματα.

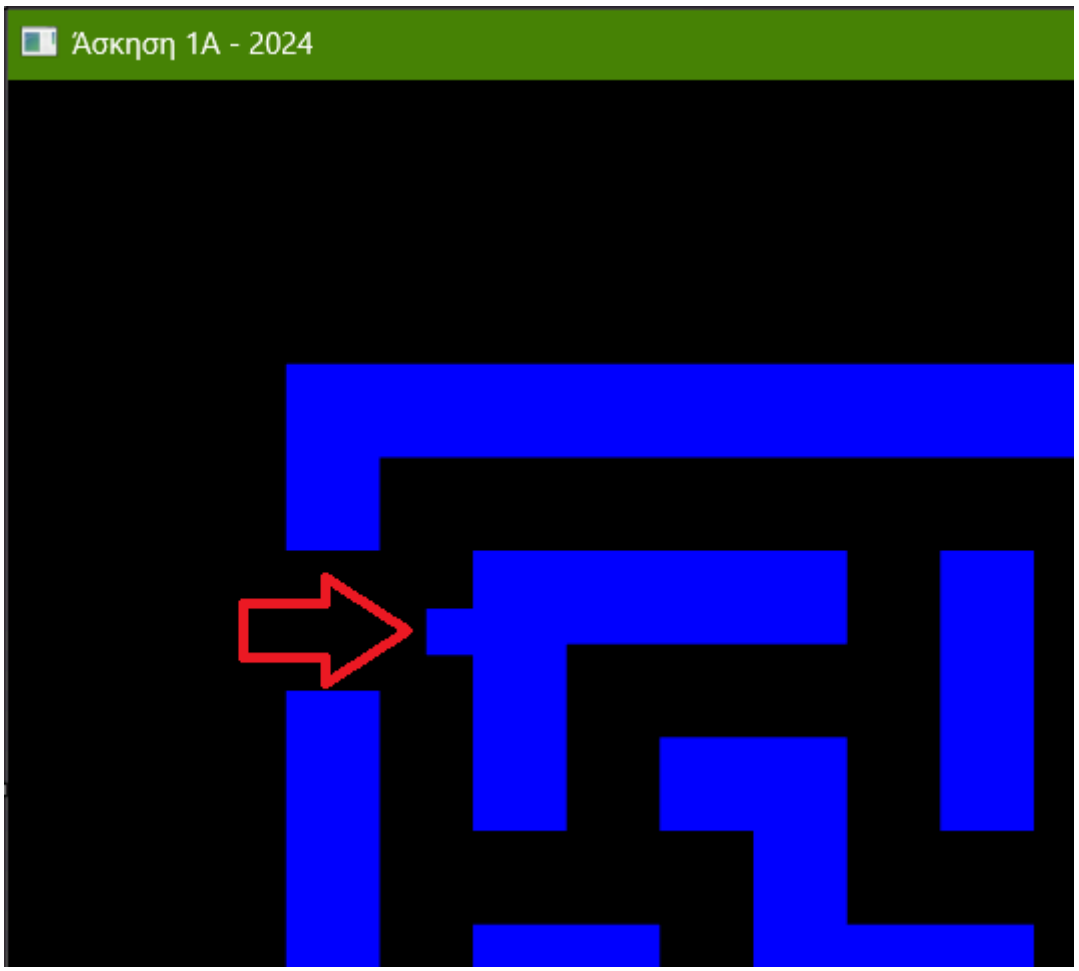
Ερώτημα (δ)

(iv) (30%) Ο χαρακτήρας Α κινείται μέσα στον λαβύρινθο. Η κίνησή του ελέγχεται από το πληκτρολόγιο, και συγκεκριμένα:

- Αν πατηθεί το πλήκτρο L, κινείται μία θέση δεξιά.
- Αν πατηθεί το πλήκτρο J, κινείται μία θέση αριστερά.
- Αν πατηθεί το πλήκτρο K, κινείται μία θέση προς τα κάτω.
- Αν πατηθεί το πλήκτρο I, κινείται μία θέση προς τα πάνω.

Το τετράγωνο Α δεν μπορεί να κινηθεί έξω από τον λαβύρινθο και δεν μπορεί να περάσει μέσα από τοίχο.
(Σημείωση: Προτείνεται το ερώτημα να υλοποιηθεί χωρίς την χρήση μετασχηματισμών).

(Ο χαρακτήρας κινείται)



(Ο χαρακτήρας δεν περνάει από τον τοίχο)

Επεξήγηση κώδικα της **processInput**

```
// this function does all the movement
// checking for key pressing and setting the next
coordinates of the moveable character
void processInput(GLFWwindow *window, GLfloat
*char_vertex_buffer_data, GLfloat
*maze_vertex_buffer_data) {
    float moveX, moveY = 0.0f;
    GLfloat new_vertex_buffer_data[12];
```

```
    if (glfwGetKey(window, GLFW_KEY_I) ==
GLFW_PRESS)
        moveY = 0.001f;
    if (glfwGetKey(window, GLFW_KEY_K) ==
GLFW_PRESS)
        moveY = -0.001f;
    if (glfwGetKey(window, GLFW_KEY_J) ==
GLFW_PRESS)
        moveX = -0.001f;
    if (glfwGetKey(window, GLFW_KEY_L) ==
GLFW_PRESS)
        moveX = 0.001f;

    // calculate new char pos and store them
seperately
    for (int i = 0; i < 12; i += 3) {
        new_vertex_buffer_data[i] =
char_vertex_buffer_data[i] + moveX;
        new_vertex_buffer_data[i + 1] =
char_vertex_buffer_data[i + 1] + moveY;
    }

    // Create bounding box for the character
    Rectangle charRect =
createRectangle(new_vertex_buffer_data, 0);

    // Create bounding boxes for the maze walls
    std::vector<Rectangle> mazeWalls = {
        createRectangle(maze_vertex_buffer_data,
0),
```

```
        createRectangle(maze_vertex_buffer_data,
12),
        createRectangle(maze_vertex_buffer_data,
24),
        createRectangle(maze_vertex_buffer_data,
36),
        createRectangle(maze_vertex_buffer_data,
48),
        createRectangle(maze_vertex_buffer_data,
60),
        createRectangle(maze_vertex_buffer_data,
72),
        createRectangle(maze_vertex_buffer_data,
84),
        createRectangle(maze_vertex_buffer_data,
96),
        createRectangle(maze_vertex_buffer_data,
108),
        createRectangle(maze_vertex_buffer_data,
120),
        createRectangle(maze_vertex_buffer_data,
132),
        createRectangle(maze_vertex_buffer_data,
144),
        createRectangle(maze_vertex_buffer_data,
156),
        createRectangle(maze_vertex_buffer_data,
168),
        createRectangle(maze_vertex_buffer_data,
180),
```

```
        createRectangle(maze_vertex_buffer_data,
192)
    };

    // Check collision
    bool collision = false;
    for (const auto& wall : mazeWalls) {
        if (checkRectCollision(wall, charRect)) {
            collision = true;
            break;
        }
    }

    // update pos if no collision
    if (!collision) {
        for (int i = 0; i < 12; i++) {
            char_vertex_buffer_data[i] =
new_vertex_buffer_data[i];
        }
    }
}
```

Η συνάρτηση **processInput** διαχειρίζεται την κίνηση του χαρακτήρα, εκτελώντας τα εξής βήματα:

1. **Ανίχνευση πατήματος πλήκτρου**: Ελέγχει αν πατηθούν συγκεκριμένα πλήκτρα(**I**, **J**, **K**, **L**) με χρήση της **glfwGetKey** για να προσδιορίσει την κατεύθυνση της κίνησης (**πάνω**, **κάτω**, **αριστερά**, **δεξιά**). Οι

αυξήσεις της κίνησης επιλέχθηκαν αυθαίρετα να είναι 0.001 μονάδες με σκοπό την ομαλή κίνηση.

2. **Υπολογισμός νέας θέσης**: Υπολογίζει τις νέες συντεταγμένες για τον χαρακτήρα με βάση τις εισαγωγές από τα πλήκτρα και τις αποθηκεύει σε έναν προσωρινό πίνακα **new_char_vertex_buffer_data**. Αργότερα και εφόσον βεβαιωθούμε ότι δεν υπάρχει σύγκρουση, θα ενημερώσει τον πίνακα που περιέχει τις ενεργές συντεταγμένες.

3. **Δημιουργία ορθογώνιου για τον χαρακτήρα**: Δημιουργεί ένα περιβάλλον ορθογώνιο(**bounding box**) για τη νέα θέση του χαρακτήρα χρησιμοποιώντας τη συνάρτηση **createRectangle**.

4. **Δημιουργία ορθογώνιων για τους τοίχους του λαβύρινθου**: Δημιουργεί περιβάλλοντα ορθογώνια για κάθε τμήμα των τοίχων του λαβυρίνθου χρησιμοποιώντας τα δεδομένα των κορυφών τους και την **createRectangle**.

```
// Function to create a rectangle from vertex data
Rectangle createRectangle(GLfloat* vertices, int startIdx) {
    return {
        vertices[startIdx], vertices[startIdx + 6], vertices[startIdx + 4], vertices[startIdx + 1]
    };
};
```

5. **Ανίχνευση σύγκρουσης**: Ελέγχει αν το ορθογώνιο του χαρακτήρα τέμνεται με κάποιο από τα ορθογώνια των τοίχων του λαβύρινθου καλώντας τη συνάρτηση **checkRectCollision**. Αν εντοπιστεί σύγκρουση, η θέση του χαρακτήρα δεν ενημερώνεται. Επίσης αυτή διαδοχικά καλεί πριν επιστρέψει την **checkIfReachedEnd** η οποία τσεκάρει αν έχει φτάσει ο χαρακτήρας στο τέλος του λαβυρίνθου. Αν ναι, αλλάζει το **finishedFlag** σε **true**, ώστε να βγούμε από το loop.

```
// Function to check if two rectangles overlap/collide
bool checkRectCollision(Rectangle border, Rectangle character) {
    checkIfReachedEnd(character);
    return ((
        (character.minX < -5.0f || character.maxX > 5.0f) ||
        (character.minX < border.maxX &&
        character.maxX > border.minX &&
        character.minY < border.maxY &&
        character.maxY > border.minY)
    ));
}
```

```
// check if we have reached the end of the maze
// if yes set the flag to true so we can terminate the window
bool checkIfReachedEnd(Rectangle character) {
    if(character.maxX >= 5.0f) { // we have reached the end
        finishedFlag = true;
        return true;
    }
    return false;
}
```

6. **Ενημέρωση θέσης**: Αν δεν εντοπιστεί σύγκρουση, ενημερώνει τη θέση του χαρακτήρα στον αρχικό πίνακα δεδομένων των κορυφών **char_vertex_buffer_data** με τις νέες συντεταγμένες.

Αυτή η συνάρτηση εξασφαλίζει ότι ο χαρακτήρας κινείται μέσα στο λαβύρινθο χωρίς να συγκρούεται με τους τοίχους.

```
// struct to describe collision rectangles
...
struct Rectangle {
    float minX, maxX, minY, maxY;
};
```

Το παραπάνω struct θα περιέχει τις τιμές των κορυφών του κάθε τετραγώνου και αποτελεί το περιβάλλον ορθογώνιο.

- **minX**: Η μικρότερη x-συνιστώσα τη δεδομένη στιγμή
- **maxX**: Η μεγαλύτερη x-συνιστώσα τη δεδομένη στιγμή
- **minY**: Η μικρότερη y-συνιστώσα τη δεδομένη στιγμή
- **maxX**: Η μεγαλύτερη y-συνιστώσα τη δεδομένη στιγμή

```
// draw moveable character + deal with movement
glBindVertexArray(VAOs[1]);
glBindBuffer(GL_ARRAY_BUFFER, charvertexbuffer);
processInput(window, char_vertex_buffer_data, maze_vertex_buffer_data);
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(char_vertex_buffer_data), char_vertex_buffer_data);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

Τέλος, στο loop καλούμε την **processInput** και με τη χρήση της **glBufferSubData** ανανεώνουμε τα δεδομένα των συντεταγμένων του χαρακτήρα μας. Έπειτα καλούμε ξανά την **glDrawElements** και σχεδιάζουμε τον χαρακτήρα στην οθόνη.

Πληροφορίες συστήματος

Για την εκπόνηση της συγκεκριμένης εργασίας έγινε χρήση των προσωπικών μας υπολογιστών με λειτουργικό **Windows 11**. Ο κώδικας γράφτηκε στο **VSCode** και **όχι στο Visual Studio** λόγω συνήθειας και portability, καθώς και επειδή είναι αρκετά πιο βαρύ το Visual Studio. Ωστόσο, αν και λίγο **δύσκολο** το **setup** της **OpenGL στο VSCode**, δουλεύει κανονικά και η εργασία μπόρεσε να ολοκληρωθεί δίχως άλλες δυσκολίες/ιδιαιτερότητες.

Αξιολόγηση Ομάδας

Η ομάδα μας συνεργάστηκε με πολύ καλό τρόπο στην εκπόνηση της εργασίας. Και οι δύο μαζί είδαμε την εκφώνηση και τις απαιτήσεις του προγράμματος και αποφασίσαμε να δουλέψουμε από κοινού παρέχοντας ιδέες στην υλοποίηση του κώδικα. Κάναμε αρκετές αλλαγές μέχρι να καταλήξουμε στο τελικό αποτέλεσμα που είναι και το καλύτερο και λειτουργικό. Τέλος, ο ένας εκ των δύο ασχολήθηκε με την επιβεβαίωση και επιδιόρθωση προβληματικού/επαναλαμβανόμενου κώδικα και την συγγραφή σχολίων, ενώ ο άλλος με την συγγραφή του readme.

Αναφορές/Πηγές

[Learn OpenGL, extensive tutorial resource for learning Modern OpenGL]

[GLFW: Getting started]

[docs.gl]

και μερικά λήμματα από stack overflow:

[opengl - understanding glVertexAttribPointer? - Stack Overflow]

[c++ - OpenGL -- Multiple glDrawArrays() calls only showing results from first? - Stack Overflow]