Εργαστηριακή Εργασία 2

Γενικά Στοιχεία

Μάθημα

Γραφικά Υπολογιστών και Συστήματα Αλληλεπίδρασης

Στοιχεία ομάδας

Α.Μ: 5108
Ονοματεπώνυμο: Κουτσονικολής Νικόλαος

Α.Μ: 4937
Ονοματεπώνυμο: Κωστόπουλος Αλέξανδρος

Ημερομηνία
15 Νοεμβρίου 2024

Σημείωση
Η παρούσα αναφορά χρησιμοποίησε ως template το readme της πρώτης εργασίας, μιας και οι διαφορές έρχονται σε τεχνικά θέματα τα οποία αναλύουμε σε αυτό εδώ το readme.

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) (5%) Φτιάξτε ένα πρόγραμμα που θα ανοίγει ένα βασικό παράθυρο **950x950**. Το background του παραθύρου στην περιοχή εργασίας να είναι μαύρο. Το παράθυρο θα έχει τίτλο «Άσκηση 1Β - 2024» (με ελληνικούς χαρακτήρες – όχι greeklish). Με το πλήκτρο **SPACE** η εφαρμογή τερματίζει.

Καλούμαστε να φτιάξουμε πρόγραμμα στην **OpenGL** το οποίο θα ανοίγει παράθυρο με διαστάσεις **950x950**, θα έχει **μαύρο background**, θα φέρει τίτλο "**Άσκηση 1Β - 2024**" και θα εκτελεί έξοδο με το πάτημα του πλήκτρου **SPACE**.

Παρατίθεται ο κώδικας:

```
window = glfwCreateWindow(950, 950, "Άσκηση 1Β - 2024", NULL, NULL);
```

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

```
} // Check if the SPACE key was pressed or the window was closed
while (glfwGetKey(window, GLFW_KEY_SPACE) != GLFW_PRESS && glfwWindowShouldClose(window) == 0);
```

Ερώτημα (β)

(ii) (25%) 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 κύβους 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, ενώ η τιμή "0" αντιστοιχεί σε μονοπάτι. Το πρόγραμμά σας ζωγραφίζει τους τοίχους του λαβύρινθου είτε ζωγραφίζοντας έναν-έναν τους κύβους, είτε φτιάχνοντας παραλληλεπίπεδα (=ομαδοποίηση κύβων) χρησιμοποιώντας κατάλληλα τρίγωνα. Η κάτω πλευρά του λαβύρινθου «κάθεται» πάνω στο επίπεδο xy όπως φαίνεται στην Εικόνα 2 (δηλαδή όπου z=0). Κάθε κύβος έχει πλευρά μήκους 1.

Πρέπει να προσδιορίστε τις συντεταγμένες των σημείων των τριγώνων που σχηματίζουν τους μπλε κύβους (τοίχους) και να τις αποθηκεύσετε σε κατάλληλο πίνακα μέσα στον κώδικά σας. Ο προσδιορισμός των συντεταγμένων να δοθεί αναλυτικά, μαζί με σχέδιο στο readme.

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 σε αυτόν.

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 παρατεταγμένους μπλε **κύβους** με πλευρές μήκους 1, καθένας από τους οποίους κατασκευάζεται από 12 τρίγωνα των οποίων τα σημεία καλούμαστε να βρούμε.

Υποσημείωση: Εργαζόμαστε στον 3D χώρο αυτη τη φορά! Τα vertices που "ακουμπάνε" στο xy δάπεδο έχουν συνιστώσα z = 0, και τα υπόλοιπα z = 1(αφού μήκος πλευράς κύβων τοιχώματος = 1).

Παραθέτουμε τον κώδικα των shaders, για να συμπεριλάβουμε τα χρώματα του λαβύρινθου και του κύβου. Ο vertex shader δεχεται σαν attribute το χρώμα, το οποίο δεν αλλάζει και απλώς περνάει στο επόμενο στάδιο του pipeline, δηλαδή στον fragment shader. Τελικά ο fragment shader εφαρμόζει το χρώμα στα pixels.

P1BVertexShader.vertexshader

```
You, 5 days ago | 1 author (You)
#version 330 core       You, last week • #1 readme and #2 file
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec4 vertexColor;

out vec4 fragmentColor;

uniform mat4 MVP;

void main() {
    gl_Position =  MVP * vec4(vertexPosition_modelspace,1);
    fragmentColor = vertexColor;
}
```

το location = 1 είναι οδηγία για το που να περιμένει ο shader να βρει τα δεδομένα του χρώματος

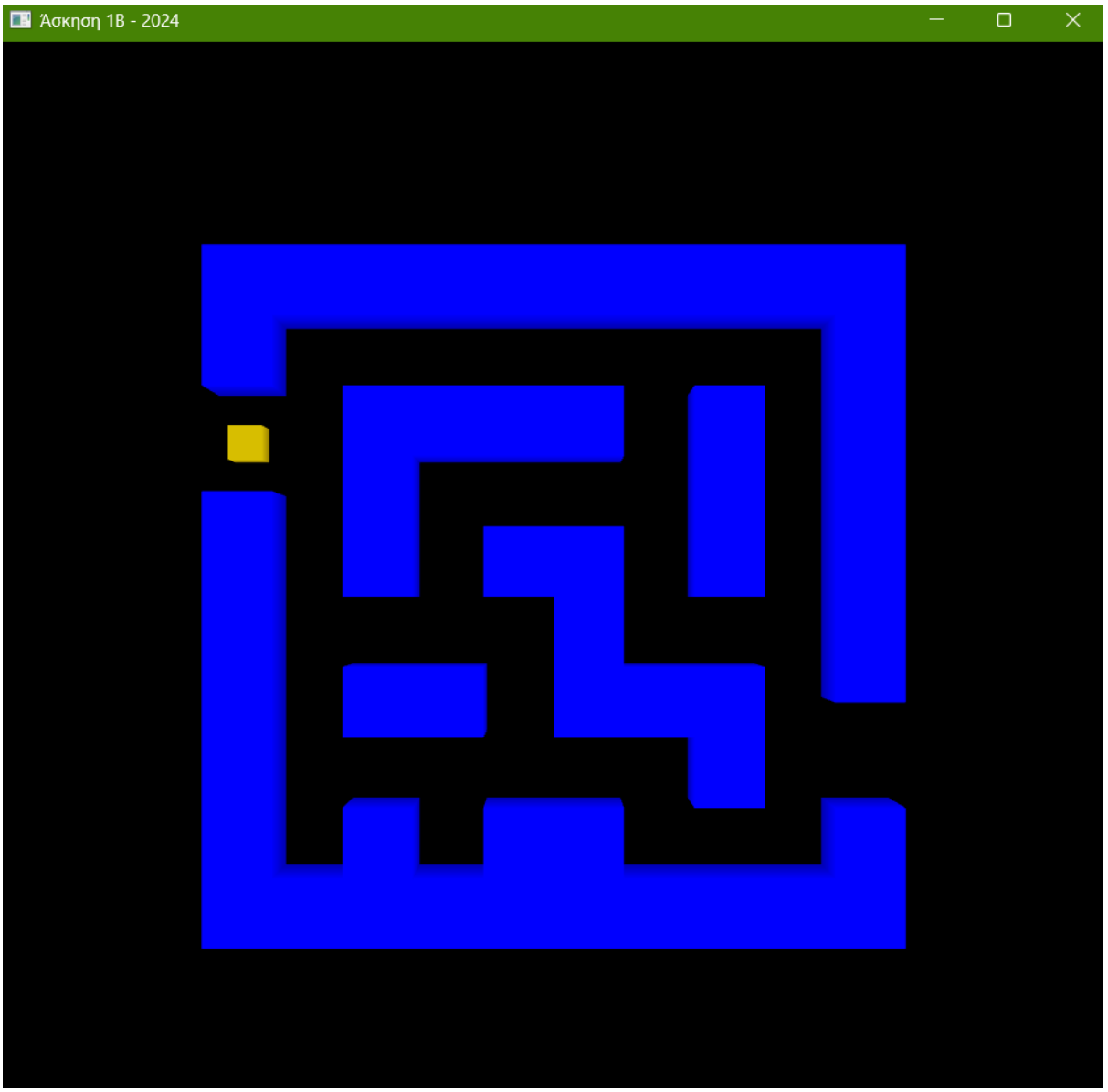
P1BFragmentShader.fragmentshader

```
You, last week | 1 author (You)
#version 330 core
in vec4 fragmentColor;

out vec4 color;

void main()
{
    color = fragmentColor;
}       You, last week • #1 rea
```

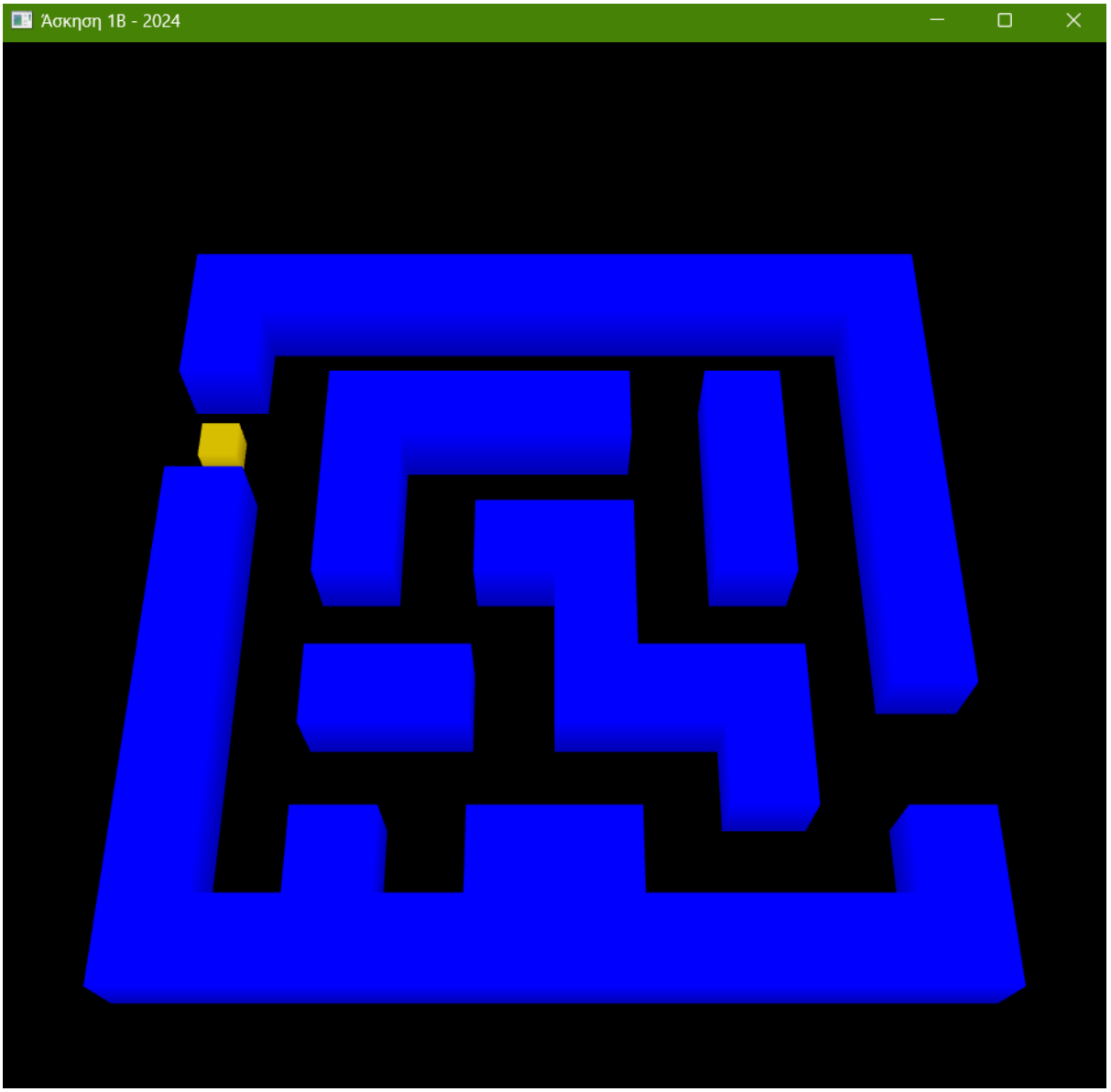
Τελικά ο fragment shader καθορίζει το χρώμα των pixels

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

Τελική αναπαράσταση με χρώμα

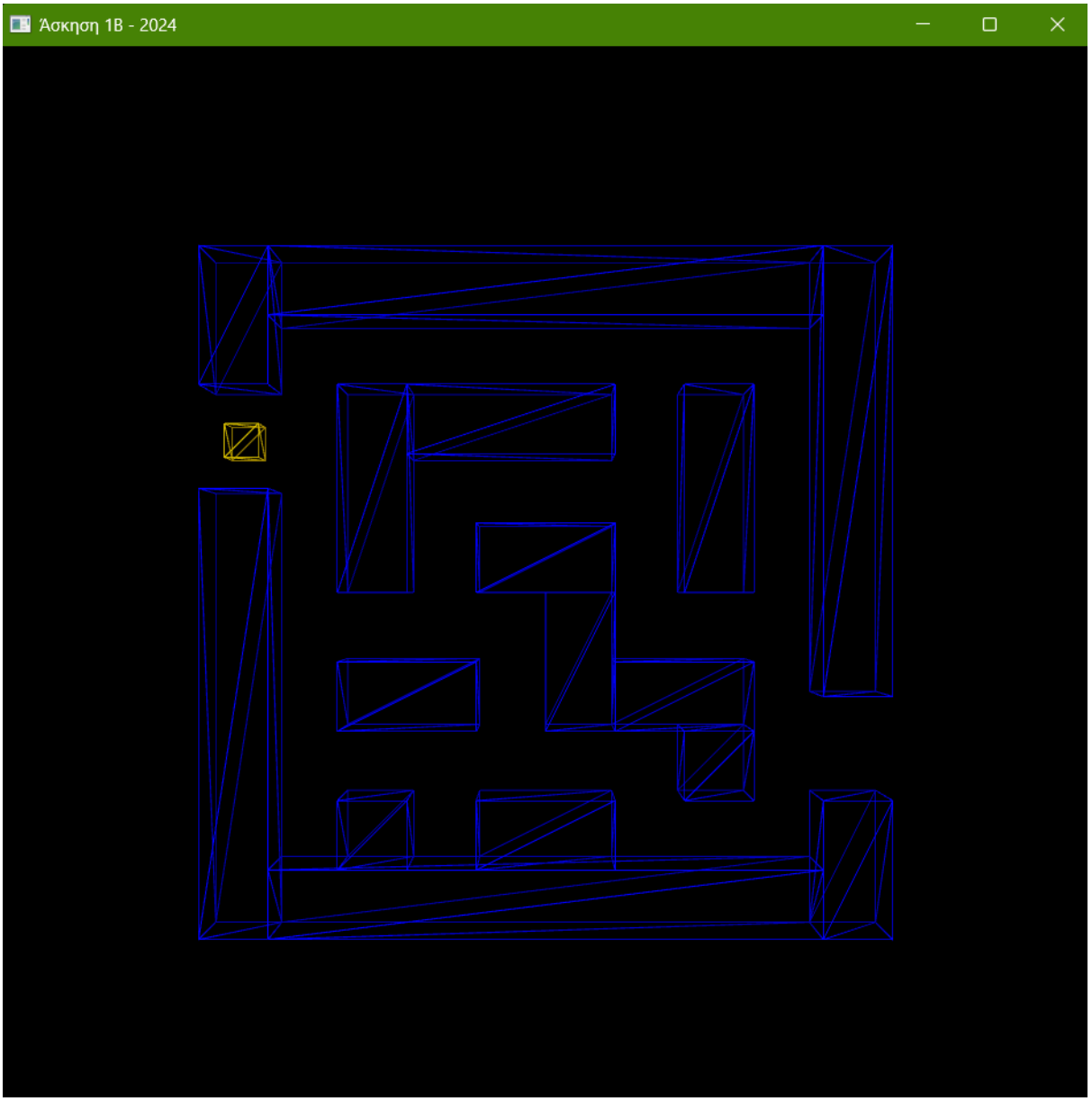


Από δοσμένη γωνία θέασης της κάμερας (βλέπε **ερώτημα δ**)

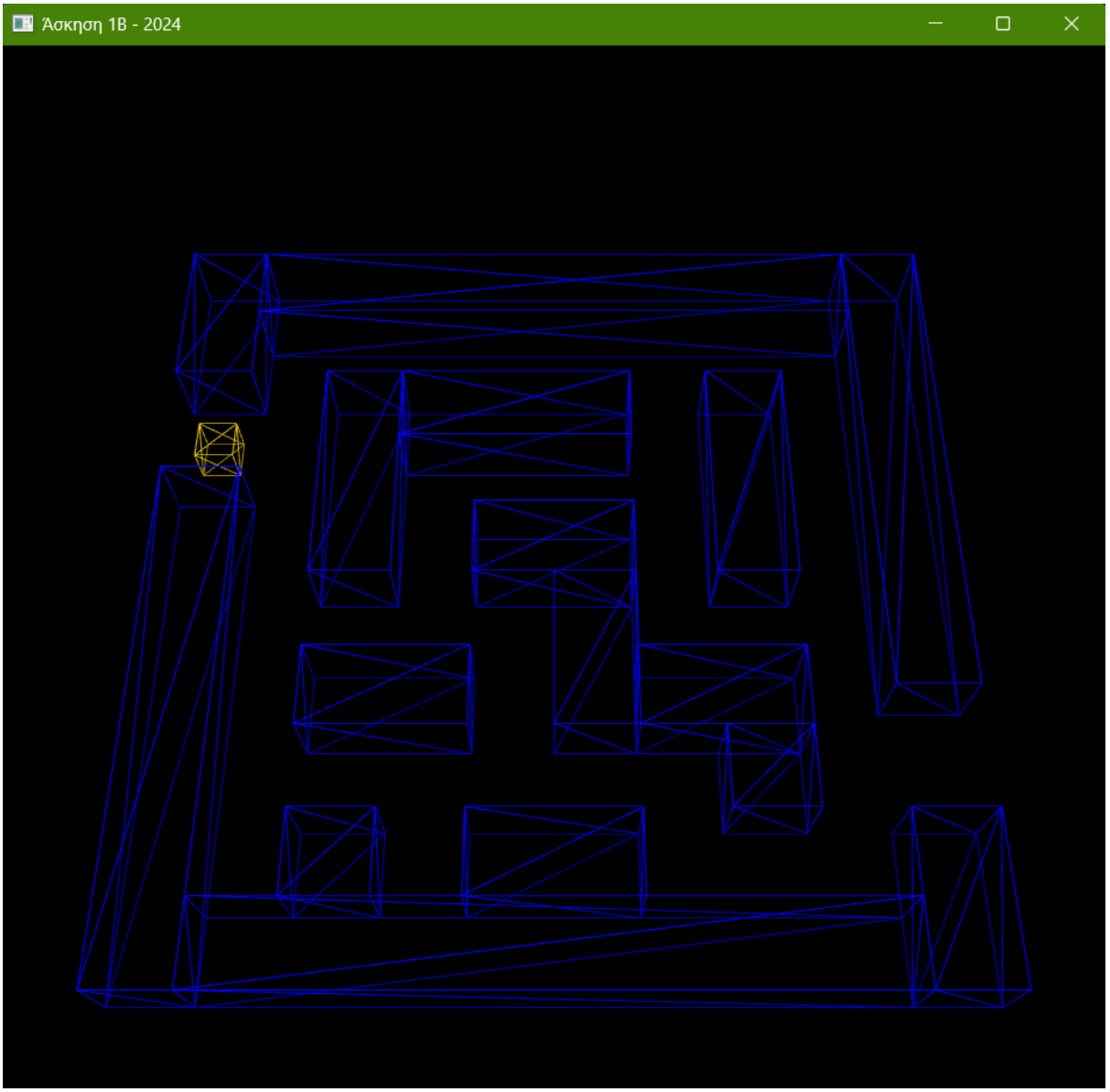


Υπό γωνία

Αναπαράσταση εσωτερικών τριγώνων



Από δοσμένη γωνία θέασης της κάμερας (βλέπε **ερώτημα iv**)



Υπό γωνία

(Σύνολο 192 τρίγωνα του λαβυρίνθου)

Κάθε τοίχωμα φτιάχνεται από 12 τρίγωνα. Σύνολο έχουμε 16 ξεχωριστά τοιχώματα στην υλοποίηση μας και έτσι βγαίνει ο συνολικός αριθμός των τριγώνων. Επίσης επισημαίνουμε ότι 192 τρίγωνα συνεπάγονται την ύπαρξη 576 σημείων(vertices).

Πίνακας συντεταγμένων του λαβυρίνθου

```
// amount of vetrices(every triangle has 3)
    int number_of_maze_vertices = 576; // 192
triangles * 3 vertices

    // vertex data of maze
    GLfloat maze_vertex_buffer_data[] = {
        /*1st rectangle - top-left border*/
        // z = 0
        -5.0f, 5.0f, 0.0f,
        -5.0f, 3.0f, 0.0f,
        -4.0f, 5.0f, 0.0f,
        -4.0f, 3.0f, 0.0f,
        // z = 1
        -5.0f, 5.0f, 1.0f,
        -5.0f, 3.0f, 1.0f,
        -4.0f, 5.0f, 1.0f,
        -4.0f, 3.0f, 1.0f,
        ////////////////////////
        /*2nd rectangle - bottom-left border*/
        // z = 0
        -5.0f, 1.5f, 0.0f,
        -5.0f, -5.0f, 0.0f,
        -4.0f, 1.5f, 0.0f,
        -4.0f, -5.0f, 0.0f,
        // z = 1
        -5.0f, 1.5f, 1.0f,
        -5.0f, -5.0f, 1.0f,
        -4.0f, 1.5f, 1.0f,
```

```
        -4.0f, -5.0f, 1.0f,
        //////////////////////
        /*3nd rectangle - top border*/
        // z = 0
        -4.0f, 5.0f, 0.0f,
        -4.0f, 4.0f, 0.0f,
        4.0f, 5.0f, 0.0f,
        4.0f, 4.0f, 0.0f,
        // z = 1
        -4.0f, 5.0f, 1.0f,
        -4.0f, 4.0f, 1.0f,
        4.0f, 5.0f, 1.0f,
        4.0f, 4.0f, 1.0f,
        //////////////////////
        /*4th rectangle - top-right border*/
        // z = 0
        4.0f, 5.0f, 0.0f,
        4.0f, -1.5f, 0.0f,
        5.0f, 5.0f, 0.0f,
        5.0f, -1.5f, 0.0f,
        // z = 1
        4.0f, 5.0f, 1.0f,
        4.0f, -1.5f, 1.0f,
        5.0f, 5.0f, 1.0f,
        5.0f, -1.5f, 1.0f,
        //////////////////////
        /*5th rectangle - bottom-right border*/
        // z = 0
        4.0f, -3.0f, 0.0f,
        4.0f, -5.0f, 0.0f,
```

```
        5.0f, -3.0f, 0.0f,
        5.0f, -5.0f, 0.0f,
        // z = 1
        4.0f, -3.0f, 1.0f,
        4.0f, -5.0f, 1.0f,
        5.0f, -3.0f, 1.0f,
        5.0f, -5.0f, 1.0f,
        ////////////////////////
        /*6th rectangle - bottom border*/
        // z = 0
        -4.0f, -4.0f, 0.0f,
        -4.0f, -5.0f, 0.0f,
        4.0f, -4.0f, 0.0f,
        4.0f, -5.0f, 0.0f,
        // z = 1
        -4.0f, -4.0f, 1.0f,
        -4.0f, -5.0f, 1.0f,
        4.0f, -4.0f, 1.0f,
        4.0f, -5.0f, 1.0f,
        ////////////////////////
        /*7th rectangle - bottom bump #1*/
        // z = 0
        -3.0f, -3.0f, 0.0f,
        -3.0f, -4.0f, 0.0f,
        -2.0f, -3.0f, 0.0f,
        -2.0f, -4.0f, 0.0f,
        // z = 1
        -3.0f, -3.0f, 1.0f,
        -3.0f, -4.0f, 1.0f,
        -2.0f, -3.0f, 1.0f,
```

```
        -2.0f, -4.0f, 1.0f,
        //////////////////////////
        /*8th rectangle - bottom bump #2*/
        // z = 0
        -1.0f, -3.0f, 0.0f,
        -1.0f, -4.0f, 0.0f,
        1.0f, -3.0f, 0.0f,
        1.0f, -4.0f, 0.0f,
        // z = 1
        -1.0f, -3.0f, 1.0f,
        -1.0f, -4.0f, 1.0f,
        1.0f, -3.0f, 1.0f,
        1.0f, -4.0f, 1.0f,
        //////////////////////////
        /*9th rectangle - horizontal mid-left
wall*/
        // z = 0
        -3.0f, -1.0f, 0.0f,
        -3.0f, -2.0f, 0.0f,
        -1.0f, -1.0f, 0.0f,
        -1.0f, -2.0f, 0.0f,
        // z = 1
        -3.0f, -1.0f, 1.0f,
        -3.0f, -2.0f, 1.0f,
        -1.0f, -1.0f, 1.0f,
        -1.0f, -2.0f, 1.0f,
        //////////////////////////
        /*10th & 11th rectangles - 90deg mid-top
walls*/
        // z = 0
```

```
        -3.0f, 3.0f, 0.0f,
        -3.0f, 0.0f, 0.0f,
        -2.0f, 3.0f, 0.0f,
        -2.0f, 0.0f, 0.0f,
        // z = 1
        -3.0f, 3.0f, 1.0f,
        -3.0f, 0.0f, 1.0f,
        -2.0f, 3.0f, 1.0f,
        -2.0f, 0.0f, 1.0f,
        //////
        // z = 0
        -2.0f, 3.0f, 0.0f,
        -2.0f, 2.0f, 0.0f,
        1.0f, 3.0f, 0.0f,
        1.0f, 2.0f, 0.0f,
        // z = 1
        -2.0f, 3.0f, 1.0f,
        -2.0f, 2.0f, 1.0f,
        1.0f, 3.0f, 1.0f,
        1.0f, 2.0f, 1.0f,
        ////////////////////////
        /*12th to 15th rectangles - zig zag
walls*/
        // z = 0
        -1.0f, 1.0f, 0.0f,
        -1.0f, 0.0f, 0.0f,
        1.0f, 1.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        // z = 1
        -1.0f, 1.0f, 1.0f,
```

```
        -1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 1.0f,
        1.0f, 0.0f, 1.0f,
        /////
        // z = 0
        0.0f, 0.0f, 0.0f,
        0.0f, -2.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        1.0f, -2.0f, 0.0f,
        // z = 1
        0.0f, 0.0f, 1.0f,
        0.0f, -2.0f, 1.0f,
        1.0f, 0.0f, 1.0f,
        1.0f, -2.0f, 1.0f,
        /////
        // z = 0
        1.0f, -1.0f, 0.0f,
        1.0f, -2.0f, 0.0f,
        3.0f, -1.0f, 0.0f,
        3.0f, -2.0f, 0.0f,
        // z = 1
        1.0f, -1.0f, 1.0f,
        1.0f, -2.0f, 1.0f,
        3.0f, -1.0f, 1.0f,
        3.0f, -2.0f, 1.0f,
        /////
        // z = 0
        2.0f, -2.0f, 0.0f,
        2.0f, -3.0f, 0.0f,
        3.0f, -2.0f, 0.0f,
```

```
        3.0f, -3.0f, 0.0f,
        // z = 1
        2.0f, -2.0f, 1.0f,
        2.0f, -3.0f, 1.0f,
        3.0f, -2.0f, 1.0f,
        3.0f, -3.0f, 1.0f,
        ////////////////////////
        /*16th rectangle - vertical top-right
wall*/
        // z = 0
        2.0f, 3.0f, 0.0f,
        2.0f, 0.0f, 0.0f,
        3.0f, 3.0f, 0.0f,
        3.0f, 0.0f, 0.0f,
        // z = 1
        2.0f, 3.0f, 1.0f,
        2.0f, 0.0f, 1.0f,
        3.0f, 3.0f, 1.0f,
        3.0f, 0.0f, 1.0f,
};
```

Οι συντεταγμένες αποφασίστηκαν για τους παρακάτω λόγους:

- Ο λαβύρινθος θα είναι κεντραρισμένος στο (0,0) του XY επιπέδου, δηλαδή την αρχή των αξόνων. Οι συντεταγμένες τόσο στον άξονα των X όσο και στον άξονα των Y θα κυμαίνονται από το -5 έως το 5.

- Η συντεταγμένη Z χρησιμοποιείται για τη δημιουργία της τρίτης διάστασης, δηλαδή για να δώσουμε αίσθηση βάθους. Κυμαίνεται από το 0 έως το 1, με σκοπό να δημιουργήσουμε ύψος ίσο με 1 μονάδα για τους τοίχους.

Ο λαβύρινθος μας αποτελείται από πολλαπλά ορθογώνια, επεκτεινόμενα στην τρίτη διάσταση, τα οποία έχουν τοποθετηθεί καταλλήλως για να φτιάξουν τα τοιχώματα.

Αναφέρουμε ορθογώνια, καθώς στην υλοποίηση μας δεν χρησιμοποιήσαμε μόνο μοναδιαίους κύβους, με σκοπό να γίνει ακόμα πιο αποδοτικό το πρόγραμμα, δηλαδή να περιέχει λιγότερα τρίγωνα που θα πρεπει να ζωγραφιστούν. Δείτε την αναπαράσταση των πολυγώνων πάνω.

Κάθε τοίχος ορίζεται από 8 κορυφές (4 για τη βάση και 4 για την κορυφή) και κατασκευάζεται ορίζοντας κορυφές για δύο τιμές Z, δημιουργώντας ένα τρισδιάστατο εφέ.

Πίνακας χρωμάτων του λαβυρίνθου

```
GLfloat a = 0.8f;
static const GLfloat maze_color[] = {
        0.0f, 0.0f, 0.7f, a, // χρώμα κατώ μέρους
        0.0f, 0.0f, 0.7f, a, // πιο σκούρο για
        0.0f, 0.0f, 0.7f, a, // βάθος
        0.0f, 0.0f, 0.7f, a,

        0.0f, 0.0f, 1.0f, a, // χρώμα πάνω μέρους
        0.0f, 0.0f, 1.0f, a, // πιο έντονο μπλε
        0.0f, 0.0f, 1.0f, a,
        0.0f, 0.0f, 1.0f, a,
        … // επαναλαμβάνεται το ίδιο για όλα τα
ορθογώνια
}
```

Αρχικοποίηση των buffers, VAOs, EBOs και σχεδιασμός

```
// init vao, ebo, buffers for character and maze
GLuint mazevertexbuffer, mazeVAO, mazecolorbuffer;
GLuint charvertexbuffer, charVAO, charcolorbuffer;
unsigned int mazeEBO, charEBO;

glGenVertexArrays(1, &charVAO);
glGenVertexArrays(1, &mazeVAO);
glGenBuffers(1, &charvertexbuffer);
glGenBuffers(1, &mazevertexbuffer);
glGenBuffers(1, &charcolorbuffer);
glGenBuffers(1, &mazecolorbuffer);
glGenBuffers(1, &charEBO);
glGenBuffers(1, &mazeEBO);
```

Ρύθμιση παραμέτρων πριν τη σχεδίαση

```
// setup maze
    glBindVertexArray(mazeVAO);
    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,
mazeEBO);
```

```
    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);

    // setup maze color
    glBindBuffer(GL_ARRAY_BUFFER,
mazecolorbuffer);
    glBufferData(GL_ARRAY_BUFFER,
sizeof(maze_color), maze_color, GL_STATIC_DRAW);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 4, GL_FLOAT,
GL_FALSE, 0, (void*)0);
```

Ακολουθούν τα βήματα που εκτελούνται:

1. **Bind Vertex Array Object (VAO)**:
    - `glBindVertexArray(mazeVAO);`: Ορίζει ότι θα χρησιμοποιηθεί το `mazeVAO`, 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 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, mazeEBO);`: 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.

Με όμοιο τρόπο φτιάχνουμε και τον color buffer, ο οποίος περιέχει τα δεδομένα για το χρώμα του λαβύρινθου.

```
    // setup maze color

    glBindBuffer(GL_ARRAY_BUFFER,
mazecolorbuffer);

    glBufferData(GL_ARRAY_BUFFER,
sizeof(maze_color), maze_color, GL_STATIC_DRAW);

    glEnableVertexAttribArray(1);

    glVertexAttribPointer(1, 4, GL_FLOAT,
GL_FALSE, 0, (void*)0);
```

Παρατηρούμε ότι τώρα στην `glVertexAttribPointer` περνάμε την τιμή 1, αντί για 0. Αυτό γιατί όπως είπαμε το location πρεπει να είναι 1, ώστε ο shader να ξέρει που θα βρει τα δεδομένα.

Έτσι, ορίζουμε και φορτώνουμε τα δεδομένα του λαβύρινθου στους αντίστοιχους buffers, και πλέον είναι έτοιμος για απεικόνιση στην οθόνη.

```
// draw maze

        glBindVertexArray(mazeVAO);

        glDrawElements(GL_TRIANGLES, 576,
GL_UNSIGNED_INT, 0);
```

Τελικά, μέσα στο render loop γίνεται η σχεδίαση/απεικόνιση του λαβυρίνθου.

Παρατηρούμε 576 vertices, γιατί έχουμε 192 τρίγωνα.

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) (15%) Ο κινούμενος χαρακτήρας **A**, που διασχίζει τον λαβύρινθο, είναι ένας κύβος με μήκος πλευράς 0.5 και κίτρινο χρώμα. Το κέντρο του **A** συμπίπτει με το κέντρο του κύβου του λαβύρινθου στο οποίο είναι τοποθετημένο. Προσδιορίσετε τις συντεταγμένες των κορυφών των τριγώνων του A και αποθηκεύστε τις κατάλληλα.

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

Πίνακας συντεταγμένων χαρακτήρα

```
    // vertex data of character
    GLfloat char_vertex_buffer_data[] = {
        // Bottom face(laying on xy-plane as
z=0.0f)
        -4.75f, 2.5f, 0.0f,
        -4.75f, 2.0f, 0.0f,
        -4.25f, 2.5f, 0.0f,
        -4.25f, 2.0f, 0.0f,

        // Top face(z=0.5f)
        -4.75f, 2.5f, 0.5f,
        -4.75f, 2.0f, 0.5f,
        -4.25f, 2.5f, 0.5f,
        -4.25f, 2.0f, 0.5f,
};
```

```
unsigned int char_indices[] = {
        // bottom face
        0, 1, 2,
        1, 2, 3,

        // top face
        4, 5, 6,
        5, 6, 7,

        // back face
        1, 3, 5,
        3, 5, 7,

        // front face
        0, 2, 4,
        2, 4, 6,

        // left face
        0, 1, 4,
        1, 4, 5,

        // right face
        2, 3, 6,
        3, 6, 7,
};
```

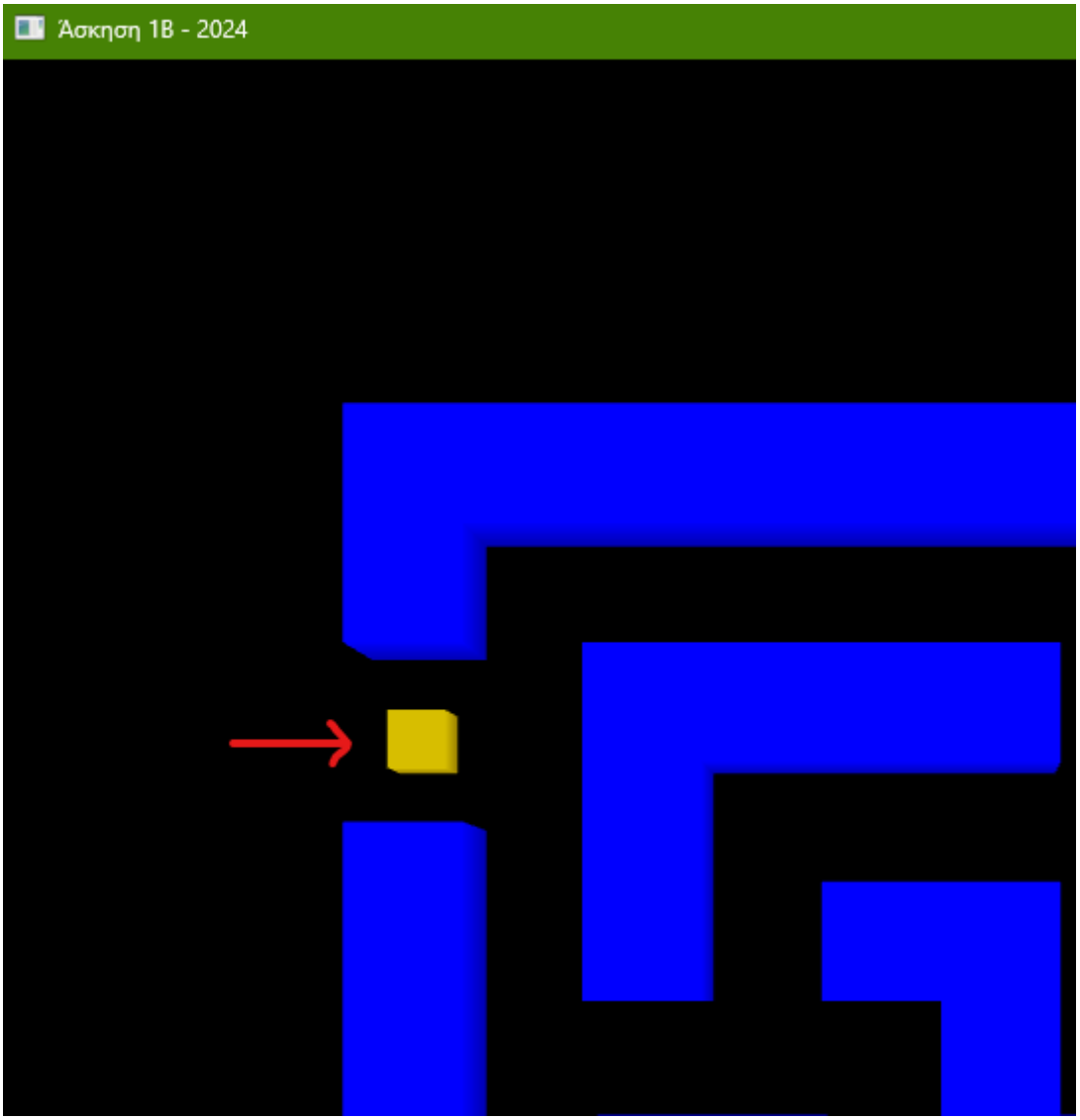
(τα 6 τετράγωνα και 12 τρίγωνα που τον αποτελούν)

Με παρόμοιο τρόπο όπως στο προηγούμενο ερώτημα, κατασκευάζουμε τους buffers, 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 character
    glBindVertexArray(charVAO);
    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,
charEBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
sizeof(char_indices), char_indices,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT,
GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

// setup character color
```

```
    glBindBuffer(GL_ARRAY_BUFFER,
charcolorbuffer);
    glBufferData(GL_ARRAY_BUFFER,
sizeof(char_color), char_color, GL_STATIC_DRAW);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 4, GL_FLOAT,
GL_FALSE, 0, (void*)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). Δεν θέλουμε κίνηση στον z άξονα, επομένως ισχύουν τα ίδια με τη 2Δ μορφή του παιχνιδιού.



(Αρχική θέση του χαρακτήρα)

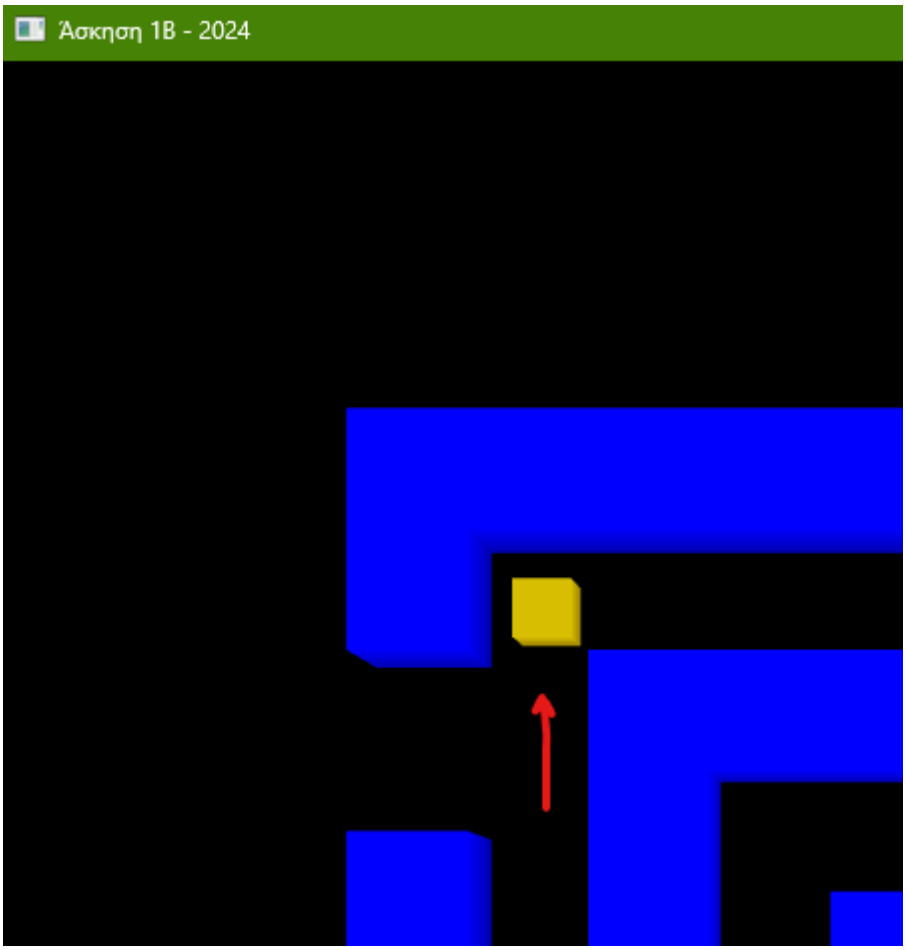
Έπειτα καλούμαστε να υλοποιήσουμε αλγορίθμους για την κίνηση του χαρακτήρα και την ανίχνευση συγκρούσεων με τα τοιχώματα του λαβύρινθου, με σκοπό στην κίνηση του μόνο μέσα σε αυτόν και όχι μέσα από τα τοιχώματα.

Ερώτημα (ε)

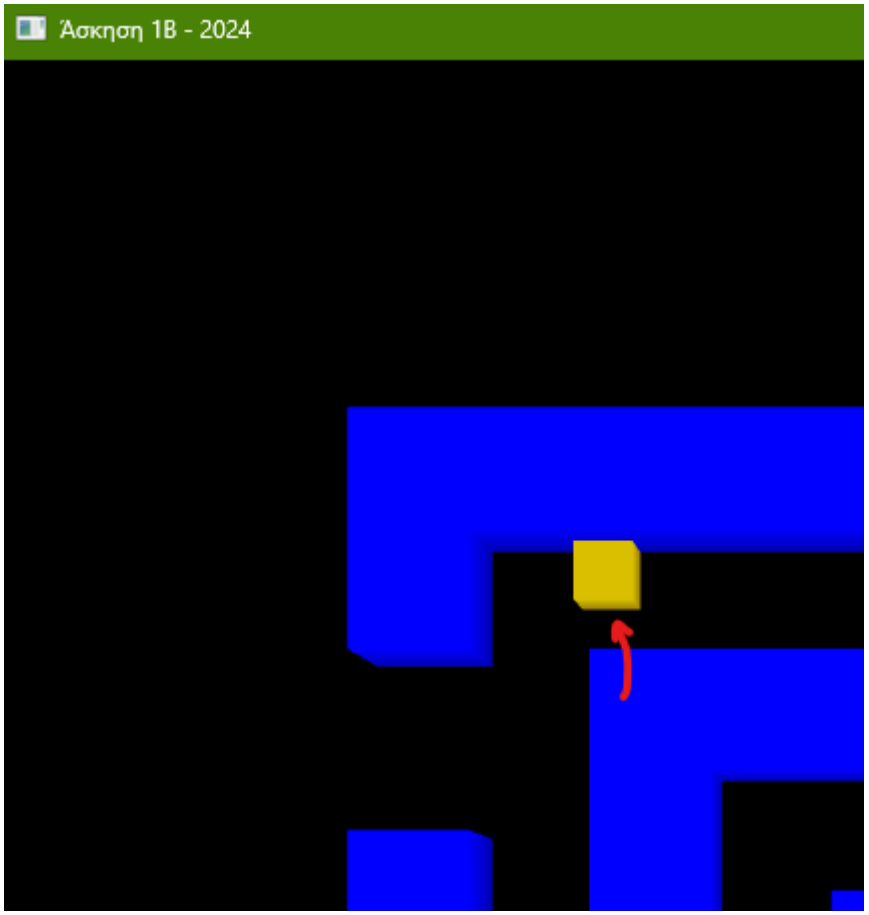
(ν) (20%) Υλοποιήστε την λειτουργία της κίνησης του Α. Η κίνησή του ελέγχεται από το πληκτρολόγιο, και συγκεκριμένα:

- 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.

Αν στην έξοδο του λαβύρινθου πατηθεί το πλήκτρο L, τότε ο χαρακτήρας **A** εμφανίζεται στην θέση εκκίνησης του λαβύρινθου. Αντίστοιχα, αν στο σημείο εκκίνησης πατηθεί το πλήκτρο **J**, τότε ο **A** εμφανίζεται στην έξοδο του λαβύρινθου. Ο χαρακτήρας **A** δεν μπορεί να περάσει μέσα από τοίχο.



(Ο χαρακτήρας κινείται)



(Ο χαρακτήρας δεν περνάει από τον τοίχο)

Δημιουργία bounding boxes για τους τοίχους

```
    // Create bounding boxes for the maze walls
    // will be used for collision detection later
on
    std::vector<Rectangle> mazeWalls = {
        createRectangle(maze_vertex_buffer_data,
0),
        createRectangle(maze_vertex_buffer_data,
24),
```

```
        createRectangle(maze_vertex_buffer_data,
48),
        createRectangle(maze_vertex_buffer_data,
72),
        createRectangle(maze_vertex_buffer_data,
96),
        createRectangle(maze_vertex_buffer_data,
120),
        createRectangle(maze_vertex_buffer_data,
144),
        createRectangle(maze_vertex_buffer_data,
180),
        createRectangle(maze_vertex_buffer_data,
204),
        createRectangle(maze_vertex_buffer_data,
228),
        createRectangle(maze_vertex_buffer_data,
252),
        createRectangle(maze_vertex_buffer_data,
276),
        createRectangle(maze_vertex_buffer_data,
300),
        createRectangle(maze_vertex_buffer_data,
324),
        createRectangle(maze_vertex_buffer_data,
348),
        createRectangle(maze_vertex_buffer_data,
372),
    };
```

Καλείται μία φορά γιατί οι τοίχοι δεν μετακινούνται, επομένως δεν υπάρχει λόγος να βρίσκεται μέσα στο loop

Επεξήγηση κώδικα της **processInput**

```
// this function has all the movement logic
// checking for key press and updating the
coordinates of the moveable character
void processInput(GLfloat
*char_vertex_buffer_data, std::vector<Rectangle>
mazeWalls) {
    float moveX, moveY = 0.0f;
    GLfloat new_char_vertex_buffer_data[] = { //
temporary array with character coordinates so we
don't directly update the original
        // Bottom face(laying on xy-plane as
z=0.25f)
        0.0f, 0.0f, 0.0f,
        0.0f, 0.0f, 0.0f,
        0.0f, 0.0f, 0.0f,
        0.0f, 0.0f, 0.0f,

        // Top face(z=0.5f)
        0.0f, 0.0f, 0.50f,
        0.0f, 0.0f, 0.50f,
        0.0f, 0.0f, 0.50f,
        0.0f, 0.0f, 0.50f,
    };
```

```
    GLfloat char_starting_vertex_buffer_data[] = {
// maze start character coordinates
        // Bottom face(laying on xy-plane as
z=0.0f)
        -4.75f, 2.5f, 0.0f,
        -4.75f, 2.0f, 0.0f,
        -4.25f, 2.5f, 0.0f,
        -4.25f, 2.0f, 0.0f,

        // Top face(z=0.50f)
        -4.75f, 2.5f, 0.50f,
        -4.75f, 2.0f, 0.50f,
        -4.25f, 2.5f, 0.50f,
        -4.25f, 2.0f, 0.50f,
    };
    GLfloat char_ending_vertex_buffer_data[] = {
// maze end character coordinates
        // Bottom face(laying on xy-plane as
z=0.0f)
        4.25f, -2.0f, 0.0f,
        4.25f, -2.5f, 0.0f,
        4.75f, -2.0f, 0.0f,
        4.75f, -2.5f, 0.0f,

        // Top face(z=0.50f)
        4.25f, -2.0f, 0.50f,
        4.25f, -2.5f, 0.50f,
        4.75f, -2.0f, 0.50f,
        4.75f, -2.5f, 0.50f,
    };
```

```
    if (glfwGetKey(window, GLFW_KEY_I) ==
GLFW_PRESS)
        moveY = 0.002f;
    if (glfwGetKey(window, GLFW_KEY_K) ==
GLFW_PRESS)
        moveY = -0.002f;
    if (glfwGetKey(window, GLFW_KEY_J) ==
GLFW_PRESS)
        moveX = -0.002f;
    if (glfwGetKey(window, GLFW_KEY_L) ==
GLFW_PRESS)
        moveX = 0.002f;

    // calculate new char pos and store them
seperately
    for (int i = 0; i < 24; i += 3) {
        new_char_vertex_buffer_data[i] =
char_vertex_buffer_data[i] + moveX;
        new_char_vertex_buffer_data[i + 1] =
char_vertex_buffer_data[i + 1] + moveY;
    }

    // create bounding box for the character
    Rectangle charRect =
createRectangle(new_char_vertex_buffer_data, 0);

    // check if the character surpassed the start
or the end
```

```
    bool surpassedStart =
checkIfSurpassedStart(charRect);
    bool surpassedEnd =
checkIfSurpassedEnd(charRect);

    // check for 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 < 24; i++) {
            char_vertex_buffer_data[i] =
new_char_vertex_buffer_data[i];
        }
    }

    // check if surpassed start or end of maze to
reposition character
    if(surpassedStart) {
        for (int i = 0; i < 24; i++) {
            char_vertex_buffer_data[i] =
char_ending_vertex_buffer_data[i];
        }
    } else if(surpassedEnd) {
```

```
        for (int i = 0; i < 24; i++) {
            char_vertex_buffer_data[i] =
char_starting_vertex_buffer_data[i];
        }
    }}
```

Σημαντικό: Αυτή η έκδοση της **processInput** είναι μία πιο εξελιγμένη μορφή αυτής που χρησιμοποιήθηκε στην πρώτη άσκηση.

Ας σπάσουμε σε βήματα την εξήγηση της, ώστε να μην μπερδευτούμε!

1. Παίρνει σαν όρισμα τα δεδομένα του χαρακτήρα και τα bounding boxes των τοίχων.
2. Τα ορίσματα αυτά θα χρησιμοποιηθούν για να ανανεώνουμε τη θέση του χαρακτήρα και να κάνουμε έλεγχο συγκρούσεων με τους τοίχους.
3. Η ανανέωση των συντεταγμένων του χαρακτήρα δεν γίνεται απευθείας στο **char_vertex_buffer_data**, καθώς στην περίπτωση σύγκρουσης με τοίχο θέλουμε να μην προχωρήσει μέσα του ο χαρακτήρας μας. Γι’αυτο χρησιμοποιούμε έναν δεύτερο πίνακα, ο οποίος υπολογίζει, ανάλογα με το πάτημα του κουμπιού κίνησης, τη νέα θέση και την αποθηκεύουμε προσωρινά σε αυτόν.
4. Έπειτα με βάσει αυτή τη θέση, φτιάχνουμε το bounding box του χαρακτήρα μας, **charRect**
5. Ελέγχουμε για πιθανή σύγκρουση και κρατάμε την κατάσταση σε ένα **collision flag**

6. Αν δεν υπάρχει σύγκρουση, ενημερώνουμε τον πραγματικό πίνακα με τα δεδομένα του χαρακτήρα
7. Ελέγχουμε αν πέρασε από την αρχή ή το τέλος με τις **checkIfSurpassedStart** και **checkIfSurpassedEnd**, αν ναι μεταφέρουμε το χαρακτήρα στις συντεταγμένες των **char_start_vertex_buffer_data** ή **char_end_vertex_buffer_data**. Αυτό γιατί θέλουμε όταν περνάει ο χαρακτήρας από την αρχή να μεταφέρεται στο τέλος, και το αντίθετο.

Παρατίθεται ο κώδικας του **Rectangle**, των **createRectangle**, **checkRectCollision,** **checkIfSurpassedStart**, **checkIfSurpassedEnd**.

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

Το struct θα περιέχει τις τιμές των κορυφών του κάθε rectangle και αποτελεί το περιβάλλον ορθογώνιο. Στην πραγματικότητα, επειδή δεν υπάρχει κίνηση στην τρίτη διάσταση, κρατήσαμε το ίδιο **struct Rectangle**, με την προηγούμενη άσκηση. Επομένως ναι μεν το Rectangle δημιουργεί bounding boxes, αλλά τυπικά το κάνει μόνο για z=0. Σε περίπτωση που θέλαμε και κίνηση στον z άξονα, θα έπρεπε να προσαρμόσουμε το Rectangle ώστε να παίρνει

και τη z συντεταγμένη, καθώς και να προσαρμόσουμε την **checkRectCollision**.

Πεδία του struct

- **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

```
// 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]
    };
};
```

Η **createRectangle** κατασκευάζει τα bounding boxes.

```
// check if two rectangles overlap/collide
bool checkRectCollision(Rectangle border, Rectangle character) {
    return ((
        (character.minX <= -5.25f || character.maxX >= 4.75f) ||
        (character.minX < border.maxX &&
        character.maxX > border.minX &&
        character.minY < border.maxY &&
        character.maxY > border.minY)
    ));
}
```

checkRectCollision: Ελέγχει αν το ορθογώνιο του χαρακτήρα τέμνεται με κάποιο από τα ορθογώνια των τοίχων του λαβύρινθου. Αν εντοπιστεί σύγκρουση, η θέση του χαρακτήρα δεν ενημερώνεται.

```
// check if we have surpassed the start of the maze
bool checkIfSurpassedStart(Rectangle character) {
    if(character.minX <= -5.25f) {
        return true;
    }
    return false;
}
```

```
// check if we have surpassed the end of the maze
bool checkIfSurpassedEnd(Rectangle character) {
    if(character.maxX >= 4.75f) {
        return true;
    }
    return false;
}
```

Και τελικά, μετά από αυτά τα απλά βήματα…. ;)

Στο render 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.

```
// draw character + movement
glBindVertexArray(charVAO);
glBindBuffer(GL_ARRAY_BUFFER, charvertexbuffer);
processInput(char_vertex_buffer_data, mazeWalls);
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(char_vertex_buffer_data), char_vertex_buffer_data);
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
```

Ξεχάσαμε όμως κάτι σημαντικό, και αυτός είναι ο χειρισμός της κάμερας, που στην προηγούμενη άσκηση δεν μας απασχόλησε.

Ας δούμε πως το κάναμε ρίχνοντας μια ματιά στα υποερωτήματα **δ** και **στ**.

Ερωτήματα (δ και στ)

(iv) (5%) Τοποθετείστε την κάμερα αρχικά στο σημείο (0.0, 0.0, 20.0) ώστε να κοιτάει προς το σημείο (0,0,0.25) με ανιόν διάνυσμα (up vector) το (0.0, 1.0, 0.0).

(vi) (20%) Να υλοποιήσετε μια κάμερα που θα ελέγχεται μόνο με τα πλήκτρα του πληκτρολογίου (*να γίνεται έλεγχος μόνο για key press*).
Η κάμερα θα κινείται στους άξονες του παγκόσμιου συστήματος συντεταγμένων με τους εξής τρόπους:
- γύρω από τον άξονα x με τα πλήκτρα <**w**> και <**x**>
- γύρω από τον άξονα y με τα πλήκτρα <**q**> και <**z**>
- θα κάνει zoom in/zoom out με κατεύθυνση το κέντρο του λαβύρινθου με τα πλήκτρα <+> και <-> του numerical keypad του πληκτρολογίου

(Σημείωση: αφού ορίσετε τιμή για το FOV (field of view), αυτή δεν θα πρέπει να αλλάζει κατά την διάρκεια εκτέλεσης του προγράμματος).

Σημείωση: Λόγω του τελευταίου υποερωτήματος, το οποίο μας ζητάει τη δημιουργία μιας δυναμικής κάμερας, δηλαδή μιας κάμερας που θα μετακινούμε εμείς στο χώρο με τα πλήκτρα και επομένως δεν θα είναι στατική, ο κώδικας του **View matrix** μπαίνει μέσα στο render loop, ενώ στην αρχή του προγράμματος φτιάχνουμε τις παρακάτω μεταβλητές:

```
float cam_x = 0.0f;
float cam_y = 0.0f;
float cam_z = 20.0f;
```

Είναι αρχικοποιημένες σε αυτές τις τιμές καθώς αυτή θα είναι η αρχική θέση της κάμερας μας.

Στο render loop περνάμε αυτές τις τιμές στο όρισμα της lookAt:

```
// Camera matrix
        glm::mat4 View = glm::lookAt(
             glm::vec3(cam_x, cam_y, cam_z), // cam
position coordinates
             glm::vec3(0.0f, 0.0f, 0.25f),
             glm::vec3(0.0f, 1.0f, 0.0f));
```

Τα **cam_x**, **cam_y**, **cam_z** ορίζουν τη θέση της κάμερας, το αμέσως επόμενο διάνυσμα **(0,0,0.25)** ορίζει το σημείο στο οποίο κοιτάζει η κάμερα και το τελευταίο αποτελεί το ανιόν διάνυσμα **(0,1,0)**.

Παρατηρούμε ότι το **Projection matrix** έχει φτιαχτεί εκτός του render loop. Αυτό γιατί δεν θέλουμε να αλλάζουμε συνεχώς το FOV(πρώτο όρισμα).

```
glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 4.0f / 4.0f, 0.1f, 100.0f);
```

Φτάνοντας στο τελευταίο ερώτημα, καταλαβαίνουμε πως μόνο με το να έχουμε μεταβλητές αντί για hardcoded τιμές ή/και να κατασκευάζουμε το **View matrix** μέσα στο render loop, δεν επαρκούν για να κάνουν την κάμερα μας να μετακινείται. Για τον λόγο αυτό χρειαζόμαστε μία συνάρτηση(το **camera_function**), η οποία αναλαμβάνει αυτό το ρόλο.

```
// camera function for applying camera movement
// W/X -> changing x coordinate
// Q/Z -> changing y coordinate
// =/- or +/-(numpad) -> zoom in/out(changing z
coordinate)
void camera_function() {
    if (glfwGetKey(window, GLFW_KEY_W) ==
GLFW_PRESS) {
        cam_x += 0.01f;
```

```
    } else if(glfwGetKey(window, GLFW_KEY_X) ==
GLFW_PRESS) {
        cam_x -= 0.01f;
    } else if(glfwGetKey(window, GLFW_KEY_Q) ==
GLFW_PRESS) {
        cam_y += 0.01f;
    } else if(glfwGetKey(window, GLFW_KEY_Z) ==
GLFW_PRESS) {
        cam_y -= 0.01f;
    } else if((glfwGetKey(window, GLFW_KEY_MINUS)
== GLFW_PRESS) || (glfwGetKey(window,
GLFW_KEY_KP_SUBTRACT) == GLFW_PRESS)) {
        cam_z += 0.01f; // incrementing z coord
moves the camera further away from the maze
    } else if((glfwGetKey(window, GLFW_KEY_EQUAL)
== GLFW_PRESS) || (glfwGetKey(window,
GLFW_KEY_KP_ADD) == GLFW_PRESS)) {
        cam_z -= 0.01f; // decrementing the z
coord moves the camera closer to the maze
    }
}
```

- Με το πάτημα των πλήκτρων **W** και **X**:
    - Μετακίνηση της θέσης της κάμερας πάνω στον άξονα των x
- Με το πάτημα των πλήκτρων **Q** και **Z**:
    - Μετακίνηση της θέσης της κάμερας πάνω στον άξονα των y

- Με το πάτημα των πλήκτρων **+** και **-**:
    - Μετακίνηση της θέσης της κάμερας πάνω στον άξονα των z(δηλαδή zoom in/out)

Καλούμε την camera_function μέσα στο render loop και παρατηρούμε πως συμπεριφέρεται όπως περιμένουμε.

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 και σε αυτό το μέρος της εργασίας με άψογο τρόπο. Η δουλειά ήταν μοιρασμένη εξίσου και στα δύο άτομα που απαρτίζουν την ομάδα και δεν υπήρξαν προβλήματα κατά τη διάρκεια της συνεργασίας.

Αναφορές/Πηγές

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

[GLFW: Getting started]

[docs.gl]