Προγραμματιστική Άσκηση 1Γ

Γραφικά Υπολογιστών Και Συστήματα Αλληλεπίδρασης

Αβανίδη Ιωάννα-Μαρία 4977

Παπαδοπούλου Μαρία 4920

4/12/2024

Περιεχόμενα

Περιγραφή της Εργασίας

i. Δημιουργία Παραθύρου

Ξεκινάμε για την δημιουργία του παραθύρου αλλάζοντας τις διαστάσεις που προϋπάρχουν στον κώδικα στις κατάλληλες (950 x 950) και συμπληρώνοντας τον τίτλο που μας ζητείται (Άσκηση 1Β – 2024).

```
window = glfwCreateWindow(950, 950, u8"Άσκηση 1Β – 2024", NULL, NULL);
```

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

```
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_SPACE', ώστε να τερματίζει η εφαρμογή με το πλήκτρο 'Q'.

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

ii. Δημιουργία Λαβυρίνθου και Χαρακτήρα Α

- 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 (0,0,0) του επιπέδου. Κάθε τετράγωνο έχει πλευρά μήκους 1.

Θεωρούμε ότι δουλεύουμε στο 3Δ χώρο

Περιγραφή Αλγορίθμου

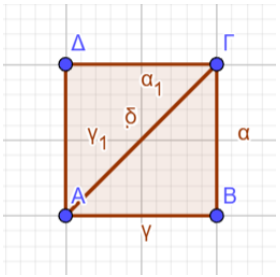
- Ορισμός του πίνακα του λαβύρινθου.

```
//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 τον πρώτο κύβο του λαβυρίνθου
  - Πρώτα ορίζουμε το κάτω τετράγωνο του κύβου. Οι συντεταγμένες στο 'cube[]' αναπαριστούν δύο τρίγωνα που σχηματίζουν το κάτω τετράγωνο του κύβου, (A-B-C) και (A-D-C).

```
//down square
-5.0f, 4.0f, 0.0f,//A
-4.0f, 4.0f, 0.0f,//B
-4.0f, 5.0f, 0.0f,//C

-5.0f, 4.0f, 0.0f,//A
-5.0f, 5.0f, 0.0f,//D
-4.0f, 5.0f, 0.0f,//C
```



A = (-5, 4, 0)

B = (-4, 4, 0)

Γ = (-4, 5, 0)

Δ = (-5, 5, 0)

- Στην συνέχεια ορίζουμε το πάνω τετράγωνο του κύβου, αντίστοιχα με τα τρίγωνα (A'-B'-C') και (A'-D'-C'). (Προσθέτουμε την διάσταση z και της δίνουμε την τιμή '1'.)

```
//up square
-5.0f, 4.0f, 1.0f,//A'
-4.0f, 4.0f, 1.0f,//B'
-4.0f, 5.0f, 1.0f,//C'

-5.0f, 4.0f, 1.0f,//A'
-5.0f, 5.0f, 1.0f,//D'
-4.0f, 5.0f, 1.0f,//C'
```

$A' = (-5, 4, 1)$

$B' = (-4, 4, 1)$

$C' = (-4, 5, 1)$

$D' = (-5, 5, 1)$

- Πλέον έχουμε ορίσει όλες τις κορυφές του κύβου. Για να ορίσουμε και τις υπόλοιπες πλευρές του κύβου ενώνουμε κατάλληλα τις κορυφές.
- Για το δεξί τετράγωνο θα χρειαστεί να ενώσουμε τις κορυφές B, C, B' και C', δημιουργώντας τα τρίγωνα (B'-B-C) και (B'-C'-C).

```
//Right Square
-4.0f, 4.0f, 1.0f,//B'
-4.0f, 4.0f, 0.0f,//B
-4.0f, 5.0f, 0.0f,//C

-4.0f, 4.0f, 1.0f,//B'
-4.0f, 5.0f, 1.0f,//C'
-4.0f, 5.0f, 0.0f,//C
```

- Για το αριστερό τετράγωνο θα χρειαστεί να ενώσουμε τις κορυφές A, D, A' και D', δημιουργώντας τα τρίγωνα (A-A'-D') και (A-D-D').

```
//Left Square
-5.0f, 4.0f, 0.0f,//A
-5.0f, 4.0f, 1.0f,//A'
-5.0f, 5.0f, 1.0f,//D'

-5.0f, 4.0f, 0.0f,//A
-5.0f, 5.0f, 0.0f,//D
-5.0f, 5.0f, 1.0f,//D'
```

- Για το μπροστά τετράγωνο θα χρειαστεί να ενώσουμε τις κορυφές A, B, A' και B', δημιουργώντας τα τρίγωνα (A-B-B') και (A-A'-B')

```
//Forward Square
-5.0f, 4.0f, 0.0f,//A
-4.0f, 4.0f, 0.0f,//B
-4.0f, 4.0f, 1.0f,//B'

-5.0f, 4.0f, 0.0f,//A
-5.0f, 4.0f, 1.0f,//A'
-4.0f, 4.0f, 1.0f,//B'
```

- Για το πίσω τετράγωνο θα χρειαστεί να ενώσουμε τις κορυφές C, D, C’ και D’, δημιουργώντας τα τρίγωνα (D’-C’-C) και (D’-D-C).

```
//Backward Square
-5.0f, 5.0f, 1.0f,//D'
-4.0f, 5.0f, 1.0f,//C'
-4.0f, 5.0f, 0.0f,//C

-5.0f, 5.0f, 1.0f,//D'
-5.0f, 5.0f, 0.0f,//D
-4.0f, 5.0f, 0.0f,//C
```

- 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 ‘cube’, ως αρχικοποίηση των συντεταγμένων για όλα τα τετράγωνα.

```
GLfloat temp_shape_buffer[108]; //temporate array

//copy the vertices
for (int k = 0; k < 108; k++) {
    temp_shape_buffer[k] = cube[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, και η i αφαιρείται από τις y συντεταγμένες για τη γραμμή.

```
//μετακίνηση των συντεταγμενων
for (int k1 = 0; k1 < 108; k1 += 3) {
    temp_shape_buffer[k1] += j;
    temp_shape_buffer[k1 + 1] -= i;
}
```

Έχουμε αντιληφθεί πως μετακινούνται οι κύβοι από την εργασία 1Α, η οποία αφορούσε τις 2 διαστάσεις του χώρου.

- 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 κύβο τοίχου (55 κύβοι) που έχει αποθηκευτεί στους buffer των VBOs για τον λαβύρινθο.
- Η ‘glDrawArrays(GL_TRIANGLES, 0, 36)’ σχεδιάζει έξι τρίγωνα (6 κορυφές/τετράγωνη πλευρά) για τον κύβο τοίχου.

```
//Labyrinth
for (int i = 0; i < 55; i++) {
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER,vbo[i]);
    glVertexAttribPointer(
        0,
        3,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );


    // 2nd attribute buffer : colors
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
    glVertexAttribPointer(
        1,
        4,                              // size
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );

    // Draw triangles
    glDrawArrays(GL_TRIANGLES, 0, 36);

    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.

```
// Cleanup VBO

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

- Χρωματίζουμε τον λαβύρινθο μπλε μέσω του buffer 'color[]'.

```
GLfloat a=0.4f;
static const GLfloat color[] = {
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
    0.0f,  0.0f,  1.0f,a,
};

GLuint colorbuffer;
glGenBuffers(1, &colorbuffer);
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(color), color, GL_STATIC_DRAW);
```

- Δημιουργία Χαρακτήρα A

Ο χαρακτήρας A αντιπροσωπεύεται ως μπλε κύβος με πλευρά 0.5 και κέντρο του είναι το κέντρο του κύβου στο οποίο βρίσκεται.

Περιγραφή Αλγορίθμου:

- Ο κύβος σχεδιάζεται με 6 τετραγωνικές πλευρές, όπου η κάθε πλευρά αποτελείται από δύο τρίγωνα, τα οποία προσδιορίζονται από έξι κορυφές και οι συντεταγμένες τους αποθηκεύονται σε έναν πίνακα buffer, 'square_buffer'.

  Ακολουθούμε την νοοτροπία που είδαμε στην δημιουργία του πρώτου κύβου του λαβύρινθου, με κορυφές A, B, C, D, A', B', C' και D'.

  A = (-4.75, 2.25, 0.25)

  B = (-4.25, 2.25, 0.25)

  C = (-4.25, 2.75, 0.25)

  D = (-4.75, 2.75, 0.25)

  A' = (-4.75, 2.25, 0.75)

  B' = (-4.25, 2.25, 0.75)

  C' = (-4.25, 2.75, 0.75)

  D' = (-4.75, 2.75, 0.75)

```
GLfloat square_buffer[] = {

    //down square
    -4.75f, 2.25f, 0.25f,//A
    -4.25f, 2.25f, 0.25f,//B
    -4.25f, 2.75f, 0.25f,//C

    -4.75f, 2.25f, 0.25f,//A
    -4.75f, 2.75f, 0.25f,//D
    -4.25f, 2.75f, 0.25f,//C

    //up square
    -4.75f, 2.25f, 0.75f,//A'
    -4.25f, 2.25f, 0.75f,//B'
    -4.25f, 2.75f, 0.75f,//C'

    -4.75f, 2.25f, 0.75f,//A'
    -4.75f, 2.75f, 0.75f,//D'
    -4.25f, 2.75f, 0.75f,//C'

    //right square
    -4.25f, 2.25f, 0.75f,//B'
    -4.25f, 2.25f, 0.25f,//B
    -4.25f, 2.75f, 0.25f,//C

    -4.25f, 2.25f, 0.75f,//B'
    -4.25f, 2.75f, 0.75f,//C'
    -4.25f, 2.75f, 0.25f,//C

    //left square
    -4.75f, 2.25f, 0.25f,//A
    -4.75f, 2.25f, 0.75f,//A'
    -4.75f, 2.75f, 0.75f,//D'

    -4.75f, 2.25f, 0.25f,//A
    -4.75f, 2.75f, 0.25f,//D
    -4.75f, 2.75f, 0.75f,//D'

    //forward square
    -4.75f, 2.25f, 0.25f,//A
    -4.25f, 2.25f, 0.25f,//B
    -4.25f, 2.25f, 0.75f,//B'

    -4.75f, 2.25f, 0.25f,//A
    -4.75f, 2.25f, 0.75f,//A'
    -4.25f, 2.25f, 0.75f,//B'

    //Backward Square
    -4.75f, 2.75f, 0.75f,//D'
    -4.25f, 2.75f, 0.75f,//C'
    -4.25f, 2.75f, 0.25f,//C

    -4.75f, 2.75f, 0.75f,//D'
    -4.75f, 2.75f, 0.25f,//D
    -4.25f, 2.75f, 0.25f,//C
};
```

- 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, 12 * 3)' σχεδιάζει έξι τρίγωνα (6 κορυφές/τετράγωνη πλευρά) για τον κύβο 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
);

// 2nd attribute buffer : colors
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, colorbufferA);
glVertexAttribPointer(
    1,
    4,                                // size
    GL_FLOAT,
    GL_FALSE,
    0,
    (void*)0
);

// Draw the triangle !
glDrawArrays(GL_TRIANGLES, 0, 12 * 3); // 12 * 3 indices starting at 0 -> 2 triangle
glDisableVertexAttribArray(0);

// Swap buffers
glfwSwapBuffers(window);
glfwPollEvents();
```

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

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

- Χρωματίζουμε τον κύβο του χαρακτήρα Α σε κίτρινο μέσω του buffer 'colorA[]'.

```
GLfloat aA = 0.4f;
static const GLfloat colorA[] = {
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
    1.0f,  1.0f,  0.0f,aA,
};

GLuint colorbufferA;
glGenBuffers(1, &colorbufferA);
glBindBuffer(GL_ARRAY_BUFFER, colorbufferA);
glBufferData(GL_ARRAY_BUFFER, sizeof(colorA), colorA, GL_STATIC_DRAW);
```

iii. Εμφάνιση Θησαυρού

Η εμφάνιση του χαρακτήρα B θέλουμε να γίνεται περιοδικά και για ορισμένο χρονικό διάστημα σε μια τυχαία επιτρεπόμενη θέση μέσα στον λαβύρινθο. Ως επιτρεπόμενη θέση ορίζουμε κάθε θέση που δεν είναι τοίχος του λαβυρίνθου και στην οποία δεν βρίσκεται ο κινούμενος χαρακτήρας A την δεδομένη χρονική στιγμή.

Για την δημιουργία των τυχαίων θέσεων ορίζουμε μια νέα συνάρτηση (getRandomPosition), με είσοδο τη γραμμή και τη στήλη ενός πίνακα καθώς και έναν πίνακα labyrinth 10 γραμμών και 10 θέσεων και έξοδο ένα διάνυσμα 2 θέσεων. Μέσα στη συνάρτηση ορίζουμε έναν πίνακα 2 θέσεων που σε κάθε μια από τις θέσεις του αναθέτουμε έναν τυχαίο αριθμό από το 0 ως το 9. Με τον βρόχο do while εξασφαλίζουμε ότι η επιλεγμένη θέση είναι επιτρεπόμενη. Τέλος ορίζουμε και ένα διάνυσμα (το οποίο είναι και αυτό που επιστρέφουμε) και αποθηκεύουμε σε αυτό τις τυχαίες επιτρεπόμενες τιμές.

```
glm:: vec2 getRandomPosition(int row, int column, int labyrinth[10][10]) {
    int pos[2];
    do {
        pos[0] = rand() % 10;
        pos[1] = rand() % 10;
    } while ((labyrinth[pos[0]][pos[1]]==1) || (pos[0]==row && pos[1]==column));
    glm::vec2 position;
    position.x = pos[1];
    position.y = pos[0];
    return position;
}
```

Επειδή θέλουμε οι τυχαίες τιμές να δημιουργούντα ανεξάρτητα με το τι συμβαίνει μέσα στο σύστημα, δημιουργούμε ένα thread, για να το πετύχουμε. Το thread, περιέχει ένα βρόχο που τρέχει επ'αοριστον και έναν βρόχο που τρέχει όταν το flag που δημιουργήσαμε γίνεται 1(Το flag γίνεται 1 όποτε ο χαρακτήρας Α ακουμπάει τον θησαυρό). Μέσα στον 2ο βρόχο, καλείται η getRandomPosition για να παράξει τις συντεταγμένες και στη συνέχεια η λειτουργία παύει για 5 δευτερόλεπτα.

```
void changePositions() {
    while (1) {
        while (touchEffect == 1) { /*wait until touch effect finish*/ }
        positions = getRandomPosition(row, column, labyrinth);
        Sleep(5000);
    }
}
```

Για να δημιουργήσουμε τον χαρακτήρα Β μετασχηματίζουμε το αρχικό κυβάκι του λαβύρινθου κατά 0.2, με τη γνωστή διαδικασία(Εύρεση κέντρου του κύβου, μεταφορά του κέντρου στο κέντρο των αξόνων, κλιμάκωση στο συγκεκριμένο σημείο, μεταφορά ξανά στην αρχική του θέση).

```
//Treasure
//scale translation
for (int k = 0; k < 108; k++) { treasure_buffer[k] = cube[k]; } //copy here

//step 1 translate to center
glm::vec3 center(-4.5f, 4.5f, 0.5f);
glm::mat4 TranslateCenterMatrix = glm::translate(glm::mat4(1.0f), -center);

//step 2 scaling
glm::mat4 ScalingMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(0.8f,0.8f,0.8f));

//step 3 move back
glm::mat4 TranslateBackMatrix = glm::translate(glm::mat4(1.0f), center);

glm::mat4 FinalMatrix = TranslateBackMatrix * ScalingMatrix * TranslateCenterMatrix;

for (int k = 0; k < 108; k += 3) {

    glm::vec4 treasure_point(treasure_buffer[k], treasure_buffer[k + 1], treasure_buffer[k + 2], 1.0f); //take the point here
    glm::vec4 transformation = FinalMatrix * treasure_point;

    //copy the vertices to new buffer
    treasure_buffer[k] = transformation.x;
    treasure_buffer[k + 1] = transformation.y;
    treasure_buffer[k + 2] = transformation.z;

}
glGenBuffers(1, &tr);
glBindBuffer(GL_ARRAY_BUFFER, tr);
glBufferData(GL_ARRAY_BUFFER, sizeof(treasure_buffer), treasure_buffer, GL_STATIC_DRAW);
```

Για την εφαρμογή της υφής στο κουτάκι αρχικά δημιουργούμε buffer στο οποίο τοποθετούμε ανά 2 τις uv συντεταγμένες, με βάσει τις συντεταγμένες του κύβου, με την τεχνική uv mapping.

```
//texture treasure
static const GLfloat uv_buffer[] = {

    //down square
    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 1.0f,

    //up square
    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 1.0f,

    //left square
    1.0f, 0.0f,
    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    0.0f, 1.0f,
```

```
    //right square
    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 1.0f,

    //front square
    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f,
    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 1.0f,

    //back square
    1.0f, 1.0f,
    0.0f, 1.0f,
    0.0f, 0.0f,
    1.0f, 1.0f,
    1.0f, 0.0f,
    0.0f, 0.0f
};
```

Δημιουργούμε και φορτώνουμε του Buffer για OpenGL όπως μας έχουν δοθεί από την θεωρία.

```
GLuint uvVBO;
glGenBuffers(1, &uvVBO);
glBindBuffer(GL_ARRAY_BUFFER, uvVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(uv_buffer), uv_buffer, GL_STATIC_DRAW);
```

Δημιουργούμε τις απαραίτητες μεταβλητές και προχωράμε στη φόρτωση της υφής

```
int width, height, nrChannels;
unsigned char* data = stbi_load("\\coins.jpg", &width, &height, &nrChannels, 0);
```

«Ζωγραφίζουμε» τον χαρακτήρα Β και κάνουμε επίσης ρυθμίσεις για τη μεγέθυνση/σμίκρυνση της υφής.

```
//TREASURE
glUniform1i(glGetUniformLocation(programID, "useTexture"), 1);
glEnableVertexAttribArray(0);

glBindBuffer(GL_ARRAY_BUFFER, tr);
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
);



glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);


//bind the uv buffer (texture cordinates)
glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, uvVBO);
glVertexAttribPointer(
    2,
    2,
    GL_FLOAT,
    GL_FALSE,
    0,
    (void*)0

);

// Draw the triangle !
glDrawArrays(GL_TRIANGLES, 0, 12 * 3); // 12 * 3 indices starting at 0 -> 2 triangle
glDisableVertexAttribArray(2);
glDisableVertexAttribArray(0);
```

Τέλος, συνδέουμε τους δυο shaders(Vertex, Fragment) με το κύριο πρόγραμμά μας. Αναλυτικότερα:

- Ο vertex shader επεξεργάζεται κάθε κορυφή ενός αντικειμένου ξεχωριστά και καθορίζει την τελική θέση της στην οθόνη. Δέχεται ως είσοδο τις συντεταγμένες της κορυφής στο τοπικό σύστημα συντεταγμένων του αντικειμένου, το χρώμα που έχει καθοριστεί για κάθε κορυφή και τις uv συντεταγμένες του αντικειμένου. Μετασχηματίζει τη θέση κάθε κορυφής από το σύστημα συντεταγμένων στην οθόνη, μεταφέρει τις uv συντεταγμένες στον fragment shader και καθορίζει αν θα χρησιμοποιηθεί υφή(μέσω του useTextureCheck)

```
#version 330 core

layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec4 vertexColor;
layout(location = 2) in vec2 VertexUV;

out vec4 fragmentColor;
out vec2 UV;

flat out int useTextureCheck;

uniform mat4 MVP;

uniform int useTexture;

void main(){

        gl_Position =  MVP * vec4(vertexPosition_modelspace,1);
    fragmentColor = vertexColor;
        UV = VertexUV;

        useTextureCheck = useTexture;
}
```

- Ο Fragment Shader είναι υπεύθυνος για τον υπολογισμό του τελικού χρώματος κάθε pixel που συνδέεται με την κορυφή. Δέχεται ως είσοδο το χρώμα της κορυφής από τον vertex shader, τις uv συντεταγμένες του αντικειμένου και το flag που δηλώνει αν θα χρησιμοποιηθεί ή όχι η υφή. Αν το flag είναι 1, τότε υπολογίζει το χρώμα του pixel της κορυφή με βάσει τις uv συντεταγμένες, αλλιώς χρησιμοποιεί το χρώμα της κορυφής

```
#version 330 core
in vec4 fragmentColor;

in vec2 UV;

out vec4 color;

flat in int useTextureCheck;
uniform sampler2D textureSampler;

void main()
{
    if (useTextureCheck==1){
        color = texture(textureSampler,UV);
    }else{
        color = fragmentColor;
    }
}
```

iv. Κίνηση Χαρακτήρα A

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.

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.

Θεωρούμε ότι δουλεύουμε σε 3Δ χώρο.

Περιγραφή Αλγορίθμου:

- Ορίζουμε την τρέχουσα θέση του κύβου 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).

```
//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 κύβου ενημερώνονται και το buffer ανανεώνεται.
  Όταν ο χαρακτήρας A βρίσκεται στην έξοδο του λαβύρινθου, δηλαδή θέση [7,9] του πίνακα του λαβύρινθου 'labyrinth', και πατηθεί το πλήκτρο L, μεταφέρεται στην θέση [2,0] του ίδιου πίνακα, δηλαδή στην θέση εκκίνησης. Αν κατά τη κίνηση προς τα δεξιά ο χαρακτήρας A ακουμπησει τον χαρακτηρα B τότε το flag touchEffect γινεται 1.

```
if (glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS && !key_pressed) {
    if ((row == 7) && (column == 9)) {
        row = 2;
        column = 0;

        for (int k1 = 0; k1 < 108; k1 += 3) {
            square_buffer[k1] = square_buffer[k1] - 9;
            square_buffer[k1 + 1] = square_buffer[k1 + 1] + 5;
        }

        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
        continue;
    }
    if ((labyrinth[row][column + 1] == 0) && (column + 1 < 10)) {
        if ((row == positions.y) && ((column + 1) == positions.x)) {
            touchEffect = 1;
        }
        else {
            column++;
            for (int k1 = 0; k1 < 108; 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.
  Όταν ο χαρακτήρας A βρίσκεται στην εκκίνηση του λαβύρινθου, δηλαδή θέση [2,0] του πίνακα του λαβύρινθου 'labyrinth', και πατηθεί το πλήκτρο J, μεταφέρεται στην θέση [7,9] του ίδιου πίνακα, δηλαδή στην έξοδο. Αν κατά τη κίνηση προς τα αριστερά ο χαρακτήρας A ακουμπήσει τον χαρακτήρα B τότε το flag touchEffect γίνεται 1.

```
if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS && !key_pressed) {
    if ((row == 2) && (column == 0)) {
        row = 7;
        column = 9;

        for (int k1 = 0; k1 < 108; k1 += 3) {
            square_buffer[k1] = square_buffer[k1] + 9;
            square_buffer[k1 + 1] = square_buffer[k1 + 1] - 5;
        }

        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
        continue;
    }
    if ((labyrinth[row][column - 1] == 0) && (column - 1 >= 0)) {
        if ((row == positions.y) && ((column - 1) == positions.x)) {
            touchEffect = 1;
        }
        else {
            column--;
            for (int k2 = 0; k2 < 108; 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. Αν κατά τη κίνηση προς τα κάτω ο χαρακτήρας A ακουμπήσει τον χαρακτήρα B τότε το flag touchEffect γίνεται 1.

```
if (glfwGetKey(window, GLFW_KEY_K) == GLFW_PRESS && !key_pressed) {
    if ((labyrinth[row + 1][column] == 0) && (row + 1 < 10)) {
        if (((row+1) == positions.y) && (column  == positions.x)) {
            touchEffect = 1;
        }
        else {
            row++;
            for (int k3 = 0; k3 < 108; 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. Αν κατά τη κίνηση προς τα πάνω ο χαρακτήρας A ακουμπήσει τον χαρακτήρα B τότε το flag touchEffect γίνεται 1.

```
if (glfwGetKey(window, GLFW_KEY_I) == GLFW_PRESS && !key_pressed) {
    if ((labyrinth[row - 1][column] == 0) && (row - 1 >= 0)) {
        if (((row-1) == positions.y) && (column == positions.x)) {
            touchEffect = 1;
        }
        else {
            row--;
            for (int k4 = 0; k4 < 108; 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
    }
}
```

Όταν το touchEffect γινει 1, τοτε μετασχηματίζουμε το κυβάακι κατά 0.4 με τη γνωστή διαδικασία(Εύρεση κέντρου του κύβου, μεταφορά του κέντρου στο κέντρο των αξόνων, κλιμάκωση στο συγκεκριμένο σημείο, μεταφορά ξανά στην αρχική του θέση), βάζουμε για 2 δευτερόλεπτα σε αναμονή τη συνάρτηση και θέτουμε τις συντεταγμένες θέσης ίσες με 0(προκειμένου να το “εξαφανίσουμε”). Τέλος, θετουμε το flag touchEffect ισο με 0, για να συνεχιστει η υλοποίση.

```
if (touchEffect == 1) {
    GLfloat temp_buffer[108];
    //scale translation
    for (int k = 0; k < 108; k++) { temp_buffer[k] = cube[k]; } //copy here

    //step 1 translate to center
    glm::vec3 center(-4.5f, 4.5f, 0.5f);
    glm::mat4 TranslateCenterMatrix = glm::translate(glm::mat4(1.0f), -center);

    //step 2 scaling
    glm::mat4 ScalingMatrix = glm::scale(glm::mat4(1.0f), glm::vec3(0.4f, 0.4f, 0.4f));

    //step 3 move back
    glm::mat4 TranslateBackMatrix = glm::translate(glm::mat4(1.0f), center);

    glm::mat4 FinalMatrix = TranslateBackMatrix * ScalingMatrix * TranslateCenterMatrix;

    for (int k = 0; k < 108; k += 3) {

        glm::vec4 treasure_point(temp_buffer[k], temp_buffer[k + 1], temp_buffer[k + 2], 1.0f); //take the point here
        glm::vec4 transformation = FinalMatrix * treasure_point;

        //copy the vertices to new buffer
        temp_buffer[k] = transformation.x;
        temp_buffer[k + 1] = transformation.y;
        temp_buffer[k + 2] = transformation.z;

    }

    for (int k1 = 0; k1 < 108; k1 += 3) {
        temp_buffer[k1] += moveX; //move x to the right
        temp_buffer[k1 + 1] -= moveY; //move y down
    }

    glGenBuffers(1, &tr);
    glBindBuffer(GL_ARRAY_BUFFER, tr);
    glBufferData(GL_ARRAY_BUFFER, sizeof(temp_buffer), temp_buffer, GL_STATIC_DRAW);
```

```
if (touchEffect == 1) {
    Sleep(2000);
    positions.x = 0;
    positions.y = 0;
    touchEffect = 0;
}
```

v. Κίνηση Κάμερας

Τα πλήκτρα ‘w’, ‘x’, ‘q’, ‘z’ κινούν την κάμερα αριστερόστροφα ή δεξιόστροφα, γύρω από τον άξονα ‘x’ και ‘y’ αντίστοιχα. Επιπλέον, τα πλήκτρα ‘+’ και ‘-’ κάνουν την κίνηση zoom in και zoom out στο κέντρο του λαβυρίνθου.

Θεωρούμε ότι δουλεύουμε σε 3Δ χώρο.

Περιγραφή Αλγορίθμου:

- Ορίζουμε τις γωνίες περιστροφής της κάμερα γύρω από τους άξονες x, y (σε ακτίνια) καθώς και την απόσταση της κάμερας από τον στόχο. Οι αρχίκες τιμές τους καθορίζονται από τις τιμές που μας δίνονται για την κάμερα.

```
//camera rotation
float angleX = glm::radians(0.0f);
float angleY = glm::radians(90.0f);
float radius = 20.0f;
```

- Ορίζουμε τις συντεταγμένες της θέσης της κάμερας στον 3Δ χώρο. (Αρχικά βλέπει στο 0,0,20).

```
//initial values
float x_camera_pos = 0.0f;
float y_camera_pos = 0.0f;
float z_camera_pos = 20.0f;
```

- Επιπλέον, ορίζουμε τις x, y συντεταγμένες του στόχου της κάμερας

```
float x_look = 0.0f;
float y_look = 0.0f;
```

- Στην προαναφερόμενη if για τον έλεγχο των εισόδων του χρήστη προσθέτουμε τα επιπλέον πλήκτρα για το zoom και την κίνηση γύρω από τους άξονες.

```
//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) &&
    (glfwGetKey(window, GLFW_KEY_W) == GLFW_RELEASE) && (glfwGetKey(window, GLFW_KEY_X) == GLFW_RELEASE) &&
    (glfwGetKey(window, GLFW_KEY_Q) == GLFW_RELEASE) && (glfwGetKey(window, GLFW_KEY_Z) == GLFW_RELEASE)  &&
    (glfwGetKey(window, GLFW_KEY_KP_ADD) == GLFW_RELEASE) && (glfwGetKey(window, GLFW_KEY_KP_SUBTRACT) == GLFW_RELEASE)){
    key_pressed = 0;
}
```

- Μετά από αναζήτηση, καταλήξαμε στο ότι όταν θέλουμε να ορίσουμε ένα σημείο σε έναν τρισδιάστατο χώρο, χρησιμοποιούμε τις σφαιρικές συντεταγμένες. Οι σφαιρικές συντεταγμένες ορίζονται ως εξής:
  - Ακτίνα r: Η απόσταση ενός σημείου από το κέντρο περιστροφής(0,0,0)
  - Γωνία φ: Η γωνία περιστροφής γύρω από τον άξονα y (οριζόντια περιστροφή)
  - Γωνία θ: Η γωνία περιστροφής γύρω από τον άξονα x (κάθετη περιστροφή)

  Με βάση αυτές τις συντεταγμένες ορίζουμε τις εξισώσεις για τα x, y, z, και τις χρησιμοποιούμε για τις ζητούμενες περιστροφές/μεγεθύνσεις.

```
//change camera coordinates
x_camera_pos = radius * cos(angleY) * cos(angleX);
y_camera_pos = radius * sin(angleX);
z_camera_pos = radius * sin(angleY) * cos(angleX);
```

- Πλήκτρα W, X
  - Πλήκτρο W: Για αριστερόστροφη κίνηση, γύρω από τον άξονα x, αυξάνουμε τη γωνία περιστροφής x (angleX), και ενημερώνουμε τις συντεταγμένες της κάμερας με βάσει τις εξισώσεις που αναφέραμε προηγουμένως.
  - Πλήκτρο X : Για δεξιόστροφη κίνηση γύρω από τον άξονα y, μειώνουμε την γωνία περιστροφής x (angleX), και ενημερώνουμε τις συντεταγμένες της κάμερας με βάσει τις εξισώσεις που αναφέραμε προηγουμένως.

```
if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS && !key_pressed) {
    angleX += glm::radians(5.0f);

    //change camera coordinates
    x_camera_pos = radius * cos(angleY) * cos(angleX);
    y_camera_pos = radius * sin(angleX);
    z_camera_pos = radius * sin(angleY) * cos(angleX);

    key_pressed = 1;//flag true

}

if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS && !key_pressed) {
    angleX -= glm::radians(5.0f);

    //change camera coordinates
    x_camera_pos = radius * cos(angleY) * cos(angleX);
    y_camera_pos = radius * sin(angleX);
    z_camera_pos = radius * sin(angleY) * cos(angleX);

    key_pressed = 1;//flag true
}
```

- Πλήκτρα Q, Z
  - Πλήκτρο Q: Για δεξιόστροφη κίνηση γύρω από τον άξονα y, αυξάνουμε τη γωνία περιστροφής y (angleY), και ενημερώνουμε τις συντεταγμένες της κάμερας με βάσει τις εξισώσεις που αναφέραμε προηγουμένως.
  - Πλήκτρο Z: Για αριστερόστροφη κίνηση γύρω από τον άξονα y, αυξάνουμε τη γωνία περιστροφής y (angleY), και ενημερώνουμε τις συντεταγμένες της κάμερας με βάσει τις εξισώσεις που αναφέραμε προηγουμένως.

```
if (glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS && !key_pressed) {
    angleY += glm::radians(5.0f);

    //change camera coordinates
    x_camera_pos = radius * cos(angleY) * cos(angleX);
    y_camera_pos = radius * sin(angleX);
    z_camera_pos = radius * sin(angleY) * cos(angleX);

    key_pressed = 1;//flag true
}

if (glfwGetKey(window, GLFW_KEY_Z) == GLFW_PRESS && !key_pressed) {
    angleY -= glm::radians(5.0f);

    //change camera coordinates
    x_camera_pos = radius * cos(angleY) * cos(angleX);
    y_camera_pos = radius * sin(angleX);
    z_camera_pos = radius * sin(angleY) * cos(angleX);

    key_pressed = 1;//flag true
}
```

- Πλήκτρα +, -
  - Πλήκτρο +: Για μεγέθυνση προς το κέντρο του λαβυρίνθου, αυξάνουμε την ακτίνα (radius) και ενημερώνουμε τις συντεταγμένες της κάμερας με βάσει τις εξισώσεις που αναφέραμε προηγουμένως.
  - Πλήκτρο -: Για σμίκρυνση προς το κέντρο του λαβυρίνθου, μειώνουμε την ακτίνα (radius) και ενημερώνουμε τις συντεταγμένες της κάμερας με βάσει τις εξισώσεις που αναφέραμε προηγουμένως.

```
if (glfwGetKey(window, GLFW_KEY_KP_SUBTRACT) == GLFW_PRESS && !key_pressed) {
    radius += 5.0f;

    //change camera coordinates
    x_camera_pos = radius * cos(angleY) * cos(angleX);
    y_camera_pos = radius * sin(angleX);
    z_camera_pos = radius * sin(angleY) * cos(angleX);

    key_pressed = 1;//flag true
}

if (glfwGetKey(window, GLFW_KEY_KP_ADD) == GLFW_PRESS && !key_pressed) {
    if (radius>0.0f){
        radius -= 5.0f;
    }

    //change camera coordinates
    x_camera_pos = radius * cos(angleY) * cos(angleX);
    y_camera_pos = radius * sin(angleX);
    z_camera_pos = radius * sin(angleY) * cos(angleX);

    key_pressed = 1;//flag true
}
```

- Πλήκτρα G, H
  - Πλήκτρο G: Για αριστερή μετακίνηση πάνω στον άξονα x (panning), αυξάνουμε τη x συντεταγμένη της κάμερας καθως και την x συνιστώσα του στόχου της κάμερας.
  - Πλήκτρο T: Για δεξιά μετακίνηση πάνω στον άξονα x (panning), μειώνουμε τη x συντεταγμένη της κάμερας καθως και την x συνιστώσα του στόχου της κάμερας.

```
if (glfwGetKey(window, GLFW_KEY_G) == GLFW_PRESS && !key_pressed) {
    x_camera_pos++;
    x_look++;
    key_pressed = 1;
}

if (glfwGetKey(window, GLFW_KEY_H) == GLFW_PRESS && !key_pressed) {
    x_camera_pos--;
    x_look--;
    key_pressed = 1;
}
```

- Πλήκτρα T, B
  - Πλήκτρο T: Για προς τα πάνω κίνηση στον άξονα y (panning), μειώνουμε τη y συντεταγμένη της κάμερας καθως και την y συνιστώσα του στόχου της κάμερας.
  - Πλήκτρο B: Για προς τα κάτω κίνηση στον άξονα y (panning), αυξάνουμε τη y συντεταγμένη της κάμερας καθως και την y συνιστώσα του στόχου της κάμερας.

```
if (glfwGetKey(window, GLFW_KEY_T) == GLFW_PRESS && !key_pressed) {
    y_camera_pos--;
    y_look--;
    key_pressed = 1;
}

if (glfwGetKey(window, GLFW_KEY_B) == GLFW_PRESS && !key_pressed) {
    y_camera_pos++;
    y_look++;
    key_pressed = 1;
}
```

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 η αναφορά ήταν μια αρκετά χρονοβόρα διαδικασία. Αναγκαστικά λειτουργήσαμε ταυτόχρονα καταμερίζοντας το φόρτο εργασίας ανάμεσα σε κώδικα και αναφορά. 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.

Αναφορές-Πηγές

-