Πανεπιστήμιο Ιωαννίνων
Τμήμα Μηχ. Η/Υ και Πληροφορικής Χειμερινό Εξάμηνο 2024

ΜΥΥ702

Γραφικά Υπολογιστών & Συστήματα Αλληλεπίδρασης

Διδάσκων: Ιωάννης Φούντος

Υπεύθυνη εργαστηριακού μέρους μαθήματος: Βασιλική Σταμάτη

Προγραμματιστική Άσκηση 1-Β

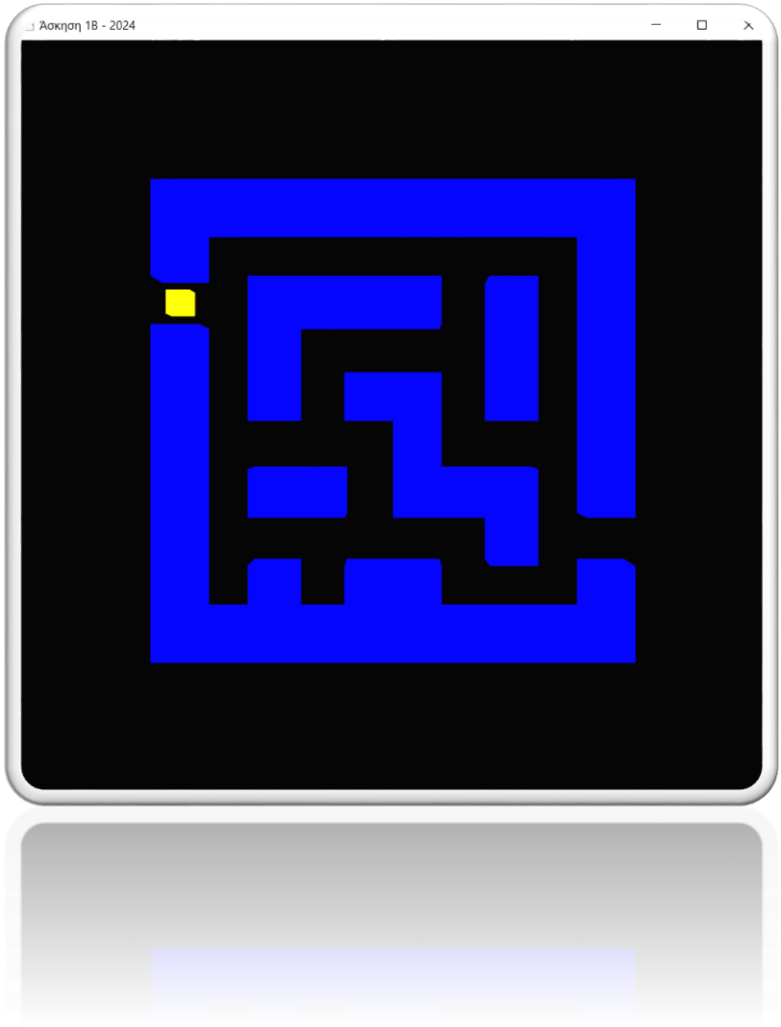
OpenGL 2024-2025

Αναφορά

Παναγιώτης Παρασκευόπουλος

AM:2905

Περιεχόμενα



Περιγραφή της εργασίας

Η συγκεκριμένη αναφορά βασίζεται στην δεύτερη προγραμματιστική άσκηση OpenGL. Σκοπός αυτής της άσκησης είναι να εξοικειωθούμε με την χρήση βασικών βιβλιοθηκών στοιχειωδών γραφικών της Opengl οι οποίες υποστηρίζουν 2Δ και 3Δ γραφικά. Στην άσκηση αυτή θα δημιουργήσουμε ένα παράθυρο στο οποίο θα ζωγραφίζουμε έναν 3Δ λαβύρινθο με χρώμα μπλε και στη συνέχεια έναν 3Δ χαρακτήρα Α με χρώμα κίτρινο, ο οποίος θα κινείται μέσα στον λαβύρινθο. Η κίνησή του θα ελέγχεται από το πληκτρολόγιο του χρήστη. Επιπλέον, θα υλοποιήσουμε και την λειτουργία της κάμερας η οποία επίσης θα ελέγχεται από το πληκτρολόγιο του χρήστη.

Η άσκηση αυτή έγινε από ένα άτομο. Η υλοποίηση της είναι βασισμένη στην πρώτη προγραμματιστική άσκηση. Στην αναφορά χρησιμοποιώ α' πληθυντικό για καλύτερη ανάγνωση.

Ερώτημα (i)

Φτιάχνουμε το πρόγραμμα μας να ανοίγει ένα βασικό παράθυρο με μέγεθος 950 x 950 pixels και να έχει τίτλο "Άσκηση 1Β - 2024".

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

Το background του παραθύρου πρέπει να είναι μαύρο, οπότε αλλάζουμε όλες τις τιμές σε 0.0.

```
260     // background color black
261     glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
```

Βάζουμε την εφαρμογή μας να τερματίζει οποιαδήποτε στιγμή πατώντας το πλήκτρο SPACE.

```
690     while (glfwGetKey(window, GLFW_KEY_SPACE) != GLFW_PRESS &&
691         glfwWindowShouldClose(window) == 0);
```

Ερώτημα (ii)

Το πρόγραμμα ξεκινάει ζωγραφίζοντας έναν λαβύρινθο. Ο λαβύρινθος σχηματίζεται ζωγραφίζοντας κύβους μπλε χρώματος, που αντιστοιχούν στα τοιχώματα του λαβύρινθου.

Οπότε αρχικά, μέσα στη main, ορίζουμε τον maze ως έναν 2Δ πίνακα, που αναπαριστά το λαβύρινθο, ως πλέγμα 10x10 που περιέχει τιμές 0 ή 1, όπως στην εικόνα 2(δεξιά πάνω) της εκφώνησης. Η τιμή "1" αντιπροσωπεύει τοίχους και η τιμή "0" κενά (μονοπάτι).

```
288     const int maze[10][10] = {
289         {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
290         {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
291         {0, 0, 1, 1, 1, 1, 0, 1, 0, 1},
292         {1, 0, 1, 0, 0, 0, 0, 1, 0, 1},
293         {1, 0, 1, 0, 1, 1, 0, 1, 0, 1},
294         {1, 0, 0, 0, 0, 1, 0, 0, 0, 1},
295         {1, 0, 1, 1, 0, 1, 1, 1, 0, 1},
296         {1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
297         {1, 0, 1, 0, 1, 1, 0, 0, 0, 1},
298         {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
299     };
```

Με το
```
221         std::vector<GLfloat> maze_vertices;
```
αποθηκεύουμε τις κορυφές (vertices) των τοίχων του λαβύρινθου, που θα χρησιμοποιηθούν για την σχεδίαση τους.

Το μήκος κάθε πλευράς του κύβου να είναι 1.0.
```
303     float size = 1.0f;
```

Στη συνέχεια με ένα loop διατρέχουμε τα στοιχεία του πίνακα maze, δηλαδή τις 10 γραμμές και 10 στήλες.
```
306     for (int i = 0; i < 10; i++) {
307         for (int j = 0; j < 10; j++) {
```

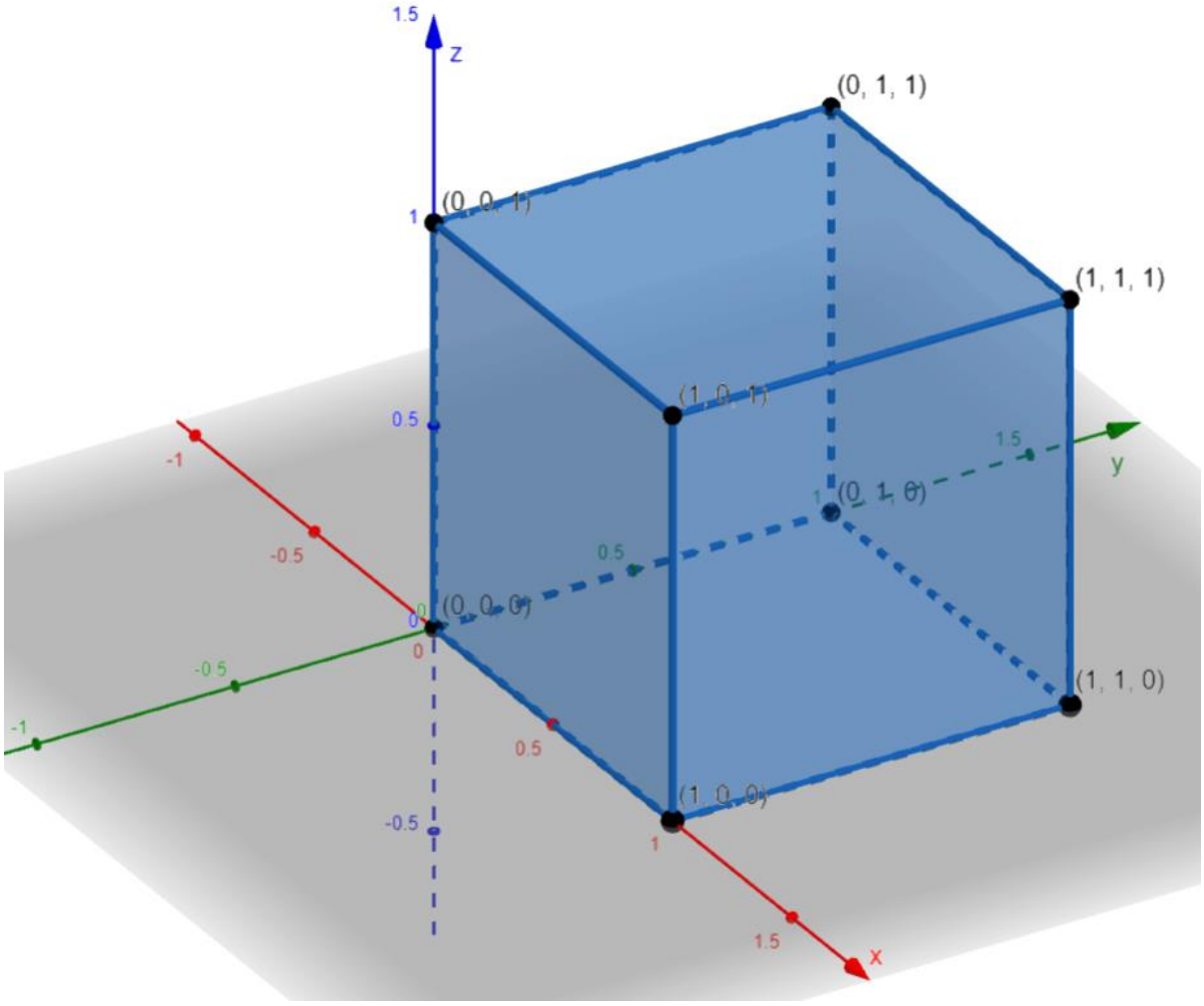
Ελέγχουμε αν το στοιχείο είναι “1” (δηλαδή τοίχος).
```
308             if (maze[i][j] == 1) {
```

Αν είναι, προχωράμε στην δημιουργία των κορυφών. Για κάθε τοίχο του λαβύρινθου δημιουργούμε έναν κύβο. Ο κύβος έχει 6 πλευρές και κάθε πλευρά σχηματίζεται από δύο τρίγωνα. Άρα δημιουργούμε τα 12 τρίγωνα που σχηματίζουν τον κύβο βάζοντας τις κατάλληλες συντεταγμένες για κάθε κορυφή των τριγώνων.
```
317             // Πρόσωπο μπροστά
318             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
319             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
320             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
321
322             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
323             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
324             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
325
326             // Πρόσωπο πίσω
327             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z);
328             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z);
329             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z);
330
331             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z);
332             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z);
333             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z);
334
335             // Πρόσωπο αριστερά
336             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z);
337             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
338             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
339
340             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z);
341             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
342             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z);
343
344             // Πρόσωπο δεξιά
345             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z);
346             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
347             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
348
349             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z);
350             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
351             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z);
352
353             // Πρόσωπο πάνω
354             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
355             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
356             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z);
357
358             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z + size);
359             maze_vertices.push_back(x + size); maze_vertices.push_back(y + size); maze_vertices.push_back(z);
360             maze_vertices.push_back(x);        maze_vertices.push_back(y + size); maze_vertices.push_back(z);
361
362             // Πρόσωπο κάτω
363             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
364             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
365             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z);
366
367             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z + size);
368             maze_vertices.push_back(x + size); maze_vertices.push_back(y);        maze_vertices.push_back(z);
369             maze_vertices.push_back(x);        maze_vertices.push_back(y);        maze_vertices.push_back(z);
```

Παίρνουμε για παράδειγμα την μπροστινή(πάνω) πλευρά του κύβου. Κάθε πλευρά αντιστοιχεί σε 6 κορυφές (2 τρίγωνα) που προστίθενται στο maze_vertices. Το πρώτο τρίγωνο σχηματίζεται από τις τρεις πρώτες κορυφές:
Κορυφή 1: (x, y) (κάτω αριστερά)
Κορυφή 2: (x + size, y) (κάτω δεξιά)
Κορυφή 3: (x + size, y + size) (πάνω δεξιά)

Το δεύτερο τρίγωνο σχηματίζεται από τις τρεις επόμενες κορυφές:
Κορυφή 4: (x, y) (κάτω αριστερά)
Κορυφή 5: (x + size, y + size) (πάνω δεξιά)
Κορυφή 6: (x, y + size) (πάνω αριστερά)

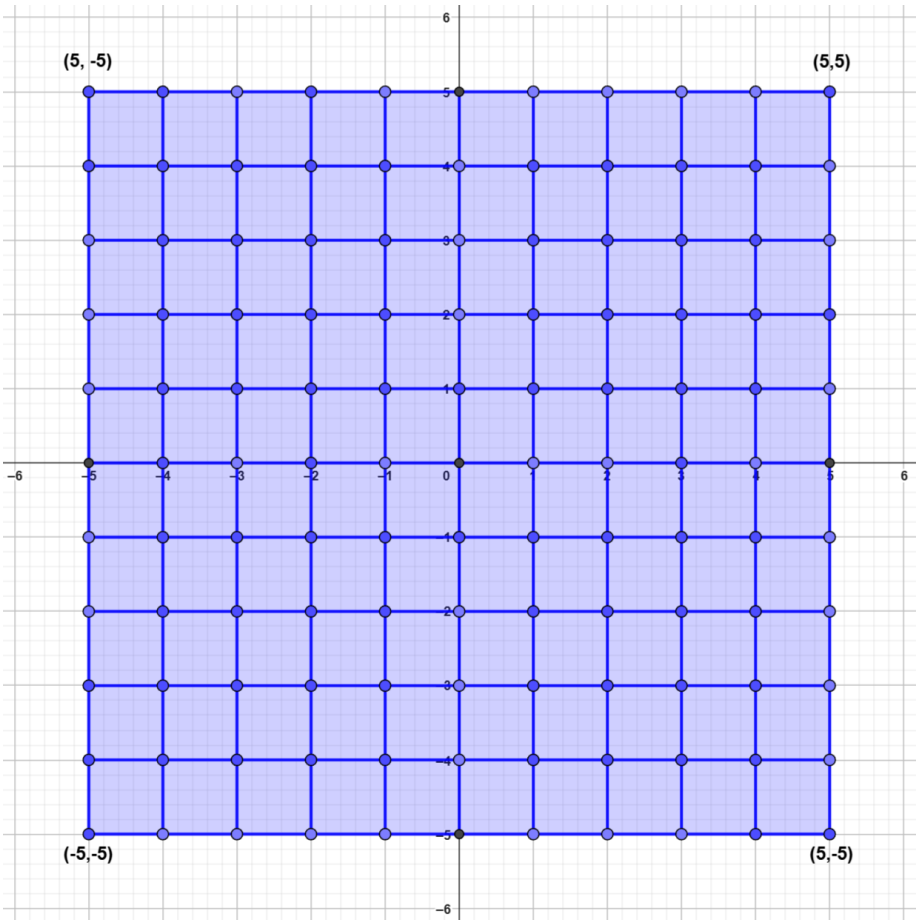
Από την στιγμή που η πίσω(κάτω) πλευρά του λαβύρινθου «κάθεται» πάνω στο επίπεδο xy (δηλαδή όπου z=0), και κάθε πλευρά έχει μήκος 1, η μπροστινή(πάνω) πλευρά θα βρίσκεται στο z=1. Γι αυτό όλες οι κορυφές των δύο τριγώνων της μπροστινής πλευράς θα είναι (z + size).
Με την ίδια λογική θα σχεδιάσουμε και τις άλλες 5 πλευρές του κύβου τοποθετώντας τις κορυφές στις κατάλληλες θέσεις των σωστών αξόνων.



Μετατοπίζουμε τις συντεταγμένες έτσι ώστε ο λαβύρινθος να κεντραριστεί γύρω από το σημείο (0,0,0).

```
310             float x = (j - 5) * size; // Με
311             float y = ((9 - i) - 5) * size;
312             float z = 0.0f; // Εδώ μπορούμε
```

Με το (j – 5) μετατοπίζουμε τον τοίχο κατά 5 μονάδες προς τα αριστερά ή δεξιά. Με το (i – 5) μετατοπίζουμε τον τοίχο κατά 5 μονάδες προς τα πάνω ή κάτω. Επειδή ο λαβύρινθος εμφανιζόταν ανεστραμμένος , χρειάστηκε να βάλουμε (9 – i) για να αντιστρέψουμε την κατεύθυνση του άξονα y, καθώς οι γραμμές του πίνακα διατρέχουν από πάνω προς τα κάτω.



Δημιουργούμε και τα buffer για την αποθήκευση των γεωμετρικών δεδομένων του λαβύρινθου.

```
374     GLuint vertexbuffer;
375     glGenBuffers(1, &vertexbuffer);
376     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
377     glBufferData(GL_ARRAY_BUFFER, maze_vertices.size() * sizeof(GLfloat), maze_vertices.data(), GL_STATIC_DRAW);
```

Στη συνέχεια πρέπει να ορίσουμε και το χρώμα του λαβύρινθου να είναι μπλε. Ορίζουμε ένα `382  std::vector<GLfloat> maze_colors;`. Αυτός ο πίνακας θα χρησιμοποιηθεί για να αποθηκεύσει τα χρώματα για τα τρίγωνα που αποτελούν τον κύβο. Δημιουργούμε έναν πίνακα

```
385     GLfloat blueColor[] = { 0.0f, 0.0f, 1.0f, 0.0};
```

με 4 στοιχεία που προσδιορίζουν το μπλε χρώμα και τη διαφάνεια (0.0). Μετά, με μια for περνάμε αυτά τα στοιχεία του χρώματος μπλε στις κορυφές των τριγώνων ώστε να

χρωματιστεί ο λαβύρινθος.

```
388     for (int i = 0; i < maze_vertices.size() / 3; ++i) {
389         maze_colors.push_back(blueColor[0]);
390         maze_colors.push_back(blueColor[1]);
391         maze_colors.push_back(blueColor[2]);
392         maze_colors.push_back(blueColor[3]);
393     }
```

Δημιουργούμε και τα buffer για τα χρώματα του λαβύρινθου.

```
396     GLuint colorbuffer;
397     glGenBuffers(1, &colorbuffer);
398     glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
399     glBufferData(GL_ARRAY_BUFFER, maze_colors.size() * sizeof(GLfloat), maze_colors.data(), GL_STATIC_DRAW);
```

Μέσα στη do κάνουμε bind τα buffer μας για τις συντεταγμένες των κορυφών(attribute(0)) και των χρωμάτων των κορυφών(attribute(1)) και ενημερώνουμε την glDrawArrays για τον σχεδιασμό και χρωματισμό των τριγώνων του λαβύρινθου στην οθόνη.

```
621         // 1rst attribute buffer : vertices for maze
622         glEnableVertexAttribArray(0);
623         glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
624         glVertexAttribPointer(
625             0,
626             3,
627             GL_FLOAT,
628             GL_FALSE,
629             0,
630             (void*)0
631         );
632
633         // 2nd attribute buffer : colors for maze
634         glEnableVertexAttribArray(1);
635         glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
636         glVertexAttribPointer(
637             1,
638             4,                                  // size
639             GL_FLOAT,
640             GL_FALSE,
641             0,
642             (void*)0
643         );
644
645         // Draw triangles
646         glDrawArrays(GL_TRIANGLES, 0, maze_vertices.size() / 3);
647         glDisableVertexAttribArray(0);
```

Ερώτημα (iii)

Θέλουμε να δημιουργήσουμε ένα μικρότερο κίτρινο κύβο (εμείς εδώ το λέμε παίκτη), αντιστοιχεί σε έναν κινούμενο χαρακτήρα που θα διασχίζει τον λαβύρινθο.
Αρχικά ορίζουμε την θέση του παίκτη στον λαβύρινθο.

```
402     int player_x = 0;
403     int player_y = 2;
```

Οι μεταβλητές player_x και player_y αρχικοποιούνται με τιμές 0 και 2, αντίστοιχα, γιατί ο

παίκτης θέλουμε να ξεκινάει από το κελί της τρίτης γραμμής και πρώτης στήλης του λαβύρινθου.

Φτιάχνουμε τη συνάρτηση update_square_A_vertices, η οποία υπολογίζει τις κορυφές ενός κύβου βασισμένο στη θέση του παίκτη.

```
406     auto update_cube_A_vertices = [&](int x, int y) -> std::vector<GLfloat> {
407         float cell_size = 1.0f;
408         float center_x = (x - 5) * cell_size + 0.25f;
409         float center_y = ((9 - y) - 5) * cell_size + 0.25f;
410         float center_z = 0.25f;
411 
412         // Το μέγεθος του κύβου
413         float size = 0.5f;  // Μισό του μήκους της ακμής του κύβου (κύβος 0.5x0.5x0.5)
```

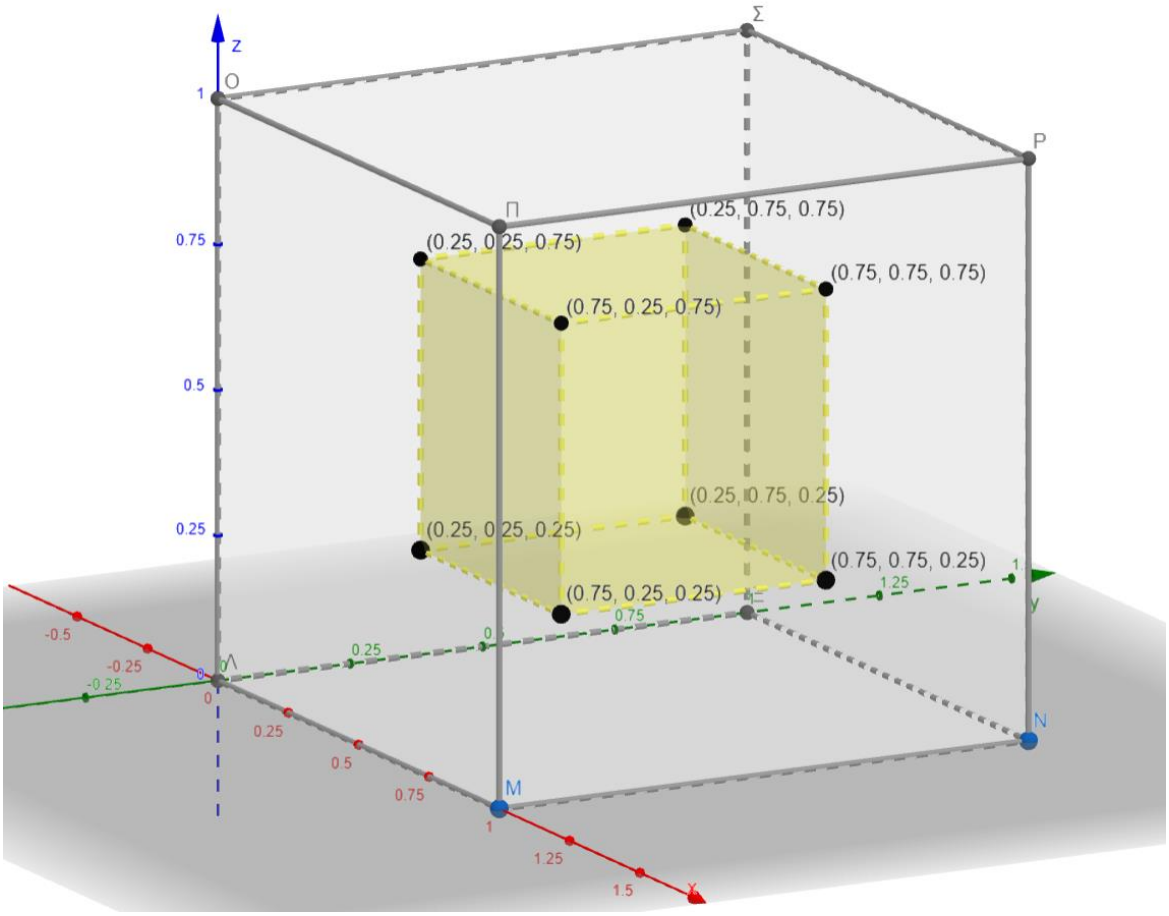
Ορίζουμε το cell_size ως 1.0f και αντιπροσωπεύει το μέγεθος κάθε κελιού στο grid. Ορίζουμε το size ως 0.5f και αντιπροσωπεύει το μήκος κάθε πλευράς του κύβου(χαρακτήρα A). Ορίζουμε τα center_x, center_y και center_z τα οποία υπολογίζουν τη θέση του κέντρου του κελιού με βάση τις συντεταγμένες (x, y, z).

Η συνάρτηση επιστρέφει έναν std::vector<GLfloat>, ο οποίος περιέχει τις συντεταγμένες για τις 36 κορυφές του κύβου. Είναι ίδια λογική με την σχεδίαση των κύβων για τα κελιά του λαβύρινθου με την διαφορά ότι στον χαρακτήρα A το μήκος των πλευρών του κύβου είναι 0.5.

```
415         return {
416             // Εμπρόσθια πλευρά (6 κορυφές, 2 τρίγωνα)
417             center_x,        center_y,        center_z + size,  // κάτω αριστερά
418             center_x + size, center_y,        center_z + size,  // κάτω δεξιά
419             center_x + size, center_y + size, center_z + size,  // πάνω δεξιά
420 
421             center_x,        center_y,        center_z + size,  // κάτω αριστερά
422             center_x + size, center_y + size, center_z + size,  // πάνω δεξιά
423             center_x,        center_y + size, center_z + size,  // πάνω αριστερά
424 
425             // Πίσω πλευρά (6 κορυφές, 2 τρίγωνα)
426             center_x,        center_y,        center_z,         // κάτω αριστερά
427             center_x + size, center_y,        center_z,         // κάτω δεξιά
428             center_x + size, center_y + size, center_z,         // πάνω δεξιά
429 
430             center_x,        center_y,        center_z,         // κάτω αριστερά
431             center_x + size, center_y + size, center_z,         // πάνω δεξιά
432             center_x,        center_y + size, center_z,         // πάνω αριστερά
433 
434             // Δεξιά πλευρά (6 κορυφές, 2 τρίγωνα)
435             center_x + size, center_y,        center_z + size,  // εμπρόσθιο κάτω δεξιά
436             center_x + size, center_y + size, center_z + size,  // εμπρόσθιο πάνω δεξιά
437             center_x + size, center_y + size, center_z,         // πίσω πάνω δεξιά
438 
439             center_x + size, center_y,        center_z + size,  // εμπρόσθιο κάτω δεξιά
440             center_x + size, center_y + size, center_z,         // πίσω πάνω δεξιά
441             center_x + size, center_y,        center_z,         // πίσω κάτω δεξιά
442 
443             // Αριστερή πλευρά (6 κορυφές, 2 τρίγωνα)
444             center_x,        center_y,        center_z + size,  // εμπρόσθιο κάτω αριστερά
445             center_x,        center_y + size, center_z + size,  // εμπρόσθιο πάνω αριστερά
446             center_x,        center_y + size, center_z,         // πίσω πάνω αριστερά
447 
448             center_x,        center_y,        center_z + size,  // εμπρόσθιο κάτω αριστερά
449             center_x,        center_y + size, center_z,         // πίσω πάνω αριστερά
450             center_x,        center_y,        center_z,         // πίσω κάτω αριστερά
451 
452             // Άνω πλευρά (6 κορυφές, 2 τρίγωνα)
453             center_x,        center_y + size, center_z + size,  // εμπρόσθιο πάνω αριστερά
454             center_x + size, center_y + size, center_z + size,  // εμπρόσθιο πάνω δεξιά
455             center_x + size, center_y + size, center_z,         // πίσω πάνω δεξιά
456 
457             center_x,        center_y + size, center_z + size,  // εμπρόσθιο πάνω αριστερά
458             center_x + size, center_y + size, center_z,         // πίσω πάνω δεξιά
459             center_x,        center_y + size, center_z,         // πίσω πάνω αριστερά
460 
461             // Κάτω πλευρά (6 κορυφές, 2 τρίγωνα)
462             center_x,        center_y,        center_z + size,  // εμπρόσθιο κάτω αριστερά
463             center_x + size, center_y,        center_z + size,  // εμπρόσθιο κάτω δεξιά
464             center_x + size, center_y,        center_z,         // πίσω κάτω δεξιά
465 
466             center_x,        center_y,        center_z + size,  // εμπρόσθιο κάτω αριστερά
467             center_x + size, center_y,        center_z,         // πίσω κάτω δεξιά
468             center_x,        center_y,        center_z          // πίσω κάτω αριστερά
469         };
```

Παράδειγμα μπροστινής πλευράς κύβου:
Πρώτο τρίγωνο: κάτω αριστερή, κάτω δεξιά, πάνω δεξιά κορυφή.
Δεύτερο τρίγωνο: κάτω αριστερή, πάνω δεξιά, πάνω αριστερή κορυφή .

Χρειάστηκε να προσθέσουμε 0.25 στα center_x, center_y και center_z για να κεντραριστεί ο κύβος με μήκος πλευράς 0.5 στο κέντρο του κελιού που έχει πλευρά με μήκος 1.0.



Η cube_A_vertices παίρνει την τιμή που επιστρέφει η update_cube_A_vertices (player_x, player_y), δηλαδή τις αρχικές κορυφές του κύβου που αντιστοιχούν στη θέση του παίκτη.

```
474        std::vector<GLfloat> cube_A_vertices = update_cube_A_vertices(player_x, player_y);
```

Δημιουργούμε και τα buffers της OpenGL για την αποθήκευση των γεωμετρικών δεδομένων του κύβου.

```
525        GLuint cube_vertexbuffer;
526        glGenBuffers(1, &cube_vertexbuffer);
527        glBindBuffer(GL_ARRAY_BUFFER, cube_vertexbuffer);
528        glBufferData(GL_ARRAY_BUFFER, cube_A_vertices.size() * sizeof(GLfloat), cube_A_vertices.data(), GL_STATIC_DRAW);
```

Ορίζουμε και τον πίνακα με τα χρώματα των κορυφών το κύβου(χαρακτήρα Α) με το χρώμα κίτρινο και διαφάνεια 0.0.

```cpp
std::vector<GLfloat> cube_A_colors = {
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,

    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f,
    1.0f, 1.0f, 0.0f, 0.0f
};
```

Όπως και στην περίπτωση του λαβύρινθου, μέσα στη do κάνουμε bind τα buffer μας για τις συντεταγμένες των κορυφών(attribute(0)) και των χρωμάτων των κορυφών(attribute(1)) και ενημερώνουμε την glDrawArrays για τον σχεδιασμό και χρωματισμό των τριγώνων του κύβου(χαρακτήρα A) στην οθόνη.

```cpp
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, cube_vertexbuffer);
glVertexAttribPointer(
    0,
    3,
    GL_FLOAT,
    GL_FALSE,
    0,
    (void*)0
);

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

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

glDisableVertexAttribArray(0);
```

Επίσης, ενημερώνουμε τις κορυφές του κύβου και τον buffer με τις νέες κορυφές, όταν αυτό αλλάξει θέση (κελί).

```
616         // Ενημέρωση κορυφών του κυβου A
617         cube_A_vertices = update_cube_A_vertices(player_x, player_y);
618
619         // Ενημέρωση του buffer με τις νέες κορυφές
620         glBindBuffer(GL_ARRAY_BUFFER, cube_vertexbuffer);
621         glBufferData(GL_ARRAY_BUFFER, cube_A_vertices.size() * sizeof(GLfloat), cube_A_vertices.data(), GL_STATIC_DRAW);
```

Ερώτημα (iv)

Ορίζουμε την κάμερα και την τοποθετούμε 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).

```
48      glm::vec3 cameraPosition = glm::vec3(0.0f, 0.0f, 20.0f); // Camera in World Space
49      glm::vec3 cameraCenterPoint = glm::vec3(0.0f, 0.0f, 0.25f); // and looks at the origin
50      glm::vec3 upVector = glm::vec3(0.0f, 1.0f, 0.0f); // Head is up
51      glm::mat4 newView; // Initialize a view model matrix
```

```
276         // Camera matrix
277         glm::mat4 View = glm::lookAt(
278             cameraPosition,     // Camera is initially at (0.0, 0.0, 20.0) in world space
279             cameraCenterPoint,  // Camera looks at (0.0, 0.0, 0.25)
280             upVector            // Camera is using an up vector of (0.0, 1.0, 0.0) - head looks at z axis
281         );
```

Ερώτημα (v)

Τώρα θέλουμε ο χαρακτήρας A να κινείται μέσα στον λαβύρινθο. Η κίνησή του ελέγχεται από το πληκτρολόγιο του χρήστη, 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.

Ορίζουμε έξω από την do τις λογικές μεταβλητές που αποθηκεύουν την κατάσταση του κάθε πλήκτρου. Αν μια μεταβλητή είναι true, σημαίνει ότι το αντίστοιχο πλήκτρο είναι πατημένο. Αυτή η κατάσταση αποτρέπει την επανειλημμένη κίνηση σε κάθε ανανέωση του προγράμματος όσο το πλήκτρο παραμένει πατημένο.

```
540     bool key_l_pressed = false;
541     bool key_j_pressed = false;
542     bool key_k_pressed = false;
543     bool key_i_pressed = false;
```

Δημιουργούμε μέσα στην do οκτώ βασικές if (δύο για κάθε πλήκτρο), και με την συνάρτηση glfwGetKey (όπως είχαμε χρησιμοποιήσει για να τερματίσουμε το πρόγραμμα με το πλήκτρο SPACE) ελέγχουμε αν το πλήκτρο είναι πατημένο ή όχι.

Αναλύουμε για το πλήκτρο “L”

```
564     if (glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS && !key_l_pressed) {
565         if (player_x + 1 < 10 && maze[player_y][player_x + 1] == 0) { // κίνηση δεξιά
566             player_x++;
567         }
568         else if (player_x + 1 >= 10) {
569                 player_x = 0;
570                 player_y = 2;
571             }
572
573         key_l_pressed = true; // Σημείωσε ότι το πλήκτρο είναι τώρα πατημένο
574     }
575     if (glfwGetKey(window, GLFW_KEY_L) == GLFW_RELEASE) {
576         key_l_pressed = false; // Επαναφορά όταν το πλήκτρο απελευθερωθεί
577     }
```

Στο πρώτο if ελέγχουμε αν το πλήκτρο L είναι πατημένο και η μεταβλητή key_l_pressed είναι false (δηλαδή, δεν είχε προηγουμένως καταγραφεί ότι το πλήκτρο είναι πατημένο), τότε:
Ελέγχουμε αν ο παίκτης μπορεί να κινηθεί δεξιά χωρίς να βγει από τα όρια του λαβύρινθου (player_x + 1 < 10), και αν η θέση προς τα δεξιά είναι κενή (maze[player_y][player_x + 1] == 0). Αν οι συνθήκες ισχύουν, αυξάνει τη συντεταγμένη x του παίκτη (player_x++), μετακινώντας τον μία θέση δεξιά. Μετά την κίνηση, ορίζουμε την μεταβλητή key_l_pressed ως true για να σημειώσει ότι το πλήκτρο έχει καταγραφεί ως πατημένο, αποτρέποντας την επανειλημμένη κίνηση μέχρι να απελευθερωθεί.
Στο δεύτερο if ελέγχουμε αν το πλήκτρο L έχει απελευθερωθεί (GLFW_RELEASE) και τότε η key_l_pressed γίνεται false, επιτρέποντας έτσι μια νέα κίνηση με επόμενο πάτημα.

Το αντίστοιχο γίνεται και για τα υπόλοιπα πλήκτρα με τις διαφορές ότι:

Για το πλήκτρο J, ελέγχουμε αν ο παίκτης μπορεί να κινηθεί αριστερά. Αν μπορεί μειώνουμε την συντεταγμένη x κατά 1 για να μετακινηθεί μια θέση αριστερά.

Για το πλήκτρο K, ελέγχουμε αν ο παίκτης μπορεί να κινηθεί κάτω. Αν μπορεί αυξάνουμε

την συντεταγμένη y κατά 1 για να μετακινηθεί μια θέση κάτω.

Για το πλήκτρο I, ελέγχουμε αν ο παίκτης μπορεί να κινηθεί πάνω. Αν μπορεί μειώνουμε την συντεταγμένη y κατά 1 για να μετακινηθεί μια θέση πάνω.

```
579         if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS && !key_j_pressed) {
580             if (player_x - 1 >= 0 && maze[player_y][player_x - 1] == 0) { // κίνηση αριστερά
581                 player_x--;
582             }
583             else if (player_x - 1 < 0) {
584                     player_x = 9;
585                     player_y = 7;
586                 }
587
588             key_j_pressed = true;
589         }
590         if (glfwGetKey(window, GLFW_KEY_J) == GLFW_RELEASE) {
591             key_j_pressed = false;
592         }
593
594         if (glfwGetKey(window, GLFW_KEY_K) == GLFW_PRESS && !key_k_pressed) {
595             if (player_y + 1 < 10 && maze[player_y + 1][player_x] == 0) { // κίνηση κάτω
596                 player_y++;
597             }
598             key_k_pressed = true;
599         }
600         if (glfwGetKey(window, GLFW_KEY_K) == GLFW_RELEASE) {
601             key_k_pressed = false;
602         }
603
604         if (glfwGetKey(window, GLFW_KEY_I) == GLFW_PRESS && !key_i_pressed) {
605             if (player_y - 1 >= 0 && maze[player_y - 1][player_x] == 0) { // κίνηση πάνω
606                 player_y--;
607             }
608             key_i_pressed = true;
609         }
610         if (glfwGetKey(window, GLFW_KEY_I) == GLFW_RELEASE) {
611             key_i_pressed = false;
612         }
```

Μέσα στον έλεγχο για το πάτημα των πλήκτρων 'L' και 'J' βάζουμε και μία else if για να ελέγξουμε αν η κίνηση του παίκτη είναι εκτός των ορίων του λαβύρινθου και αν ισχύει τοποθετούμε τον χαρακτήρα A στην είσοδο(κελί [2,0]) ή στην έξοδο(κελί[7,9]) αντίστοιχα του λαβύρινθου.

Ερώτημα (vi)

Τώρα θα υλοποιήσουμε 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.

Αρχικοποιούμε ένα view model matrix το οποίο στη συνέχεια θα χρησιμοποιηθεί για τον υπολογισμό του MVP `glm::mat4 newView;`.

Υλοποιούμε τη συνάρτηση camera_function.

```
void camera_function()
{
    glm::vec3 worldCenterPoint = glm::vec3(0.0f, 0.0f, 0.0f);
    float zoomingStep = 0.01f;

    float rotationAngle = 0.01f;
```

Ορίζουμε το κέντρο του world space μας στο σημείο (0, 0, 0).

Ορίζουμε το βήμα του ζουμ σε 0.1, δηλαδή πόσο μετακινείται η κάμερα με κάθε κίνηση ζουμ.

Ορίζουμε και το βήμα της γωνίας περιστροφής σε 0.1.

Για να πετύχουμε περιστροφή γύρω από τον άξονα x σημαίνει ότι θα έχουμε αλλαγές στις συντεταγμένες των επιπέδων yz. Αντίστοιχα για να πετύχουμε περιστροφή γύρω από τον άξονα y σημαίνει ότι θα έχουμε αλλαγές στις συντεταγμένες των επιπέδων xz

Rotation Matrices in 3-Dimensions

The Rotation Process

x–axis

To rotate the vector $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ counterclockwise through an angle θ around the x–axis to a new position $\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$,

perform the matrix multiplication,

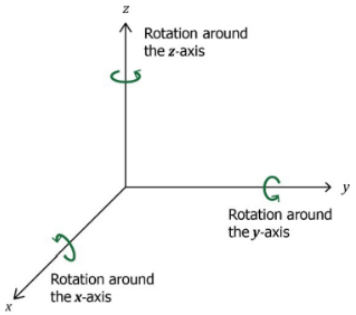
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

y–axis

To rotate the vector $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ counterclockwise through an angle θ around the y–axis to a new position $\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$,

perform the matrix multiplication,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



Άρα ορίζουμε τα rotationMatrixX και rotationMatrixY ως εξής

```
// Rotate on x axis means that the coords y,z of the yz plane change while x coord stays the same
glm::mat3 rotationMatrixX = glm::mat3(
    1, 0, 0,
    0, cos(rotationAngle), -sin(rotationAngle),
    0, sin(rotationAngle), cos(rotationAngle)
);

// Rotate on y axis means that the coords x,z of the xz plane change while y coord stays the same
glm::mat3 rotationMatrixY = glm::mat3(
    cos(rotationAngle), 0, sin(rotationAngle),
    0, 1, 0,
    -sin(rotationAngle), 0, cos(rotationAngle)
);
```

Υπολογίζουμε το κανονικοποιημένο διάνυσμα από τη θέση της κάμερας προς το κέντρο του κόσμου για τον έλεγχο ζουμ.

```
glm::vec3 normalizedVector = glm::normalize(cameraPosition - worldCenterPoint);
```

Στη συνέχεια υλοποιούμε τους ελέγχους για το πάτημα των πλήκτρων. Οι λειτουργίες του κάθε ελέγχου τροποποιούν το cameraPosition που έχουμε ορίσει με τον δικό τους τρόπο ανάλογα με το πλήκτρο που πατήθηκε.

```
76      // Zoom in
77      if (glfwGetKey(window, GLFW_KEY_KP_ADD) == GLFW_PRESS) {
78          cameraPosition -= zoomingStep * normalizedVector;
79      }
80      // Zoom out
81      if (glfwGetKey(window, GLFW_KEY_KP_SUBTRACT) == GLFW_PRESS) {
82          cameraPosition += zoomingStep * normalizedVector;
83      }
```

Αν έχει πατηθεί το πλήκτρο '+' μειώνεται η τιμή του cameraPosition και κάνει zoom in.
Αν έχει πατηθεί το πλήκτρο '-' αυξάνεται η τιμή του cameraPosition και κάνει zoom out

```
85      // Περιστροφή γύρω από τον άξονα X (counterclockwise)
86      if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
87          cameraPosition = rotationMatrixX * cameraPosition;
88          upVector = rotationMatrixX * upVector; // Ανανέωση του upVector
89      }
90
91      // Περιστροφή γύρω από τον άξονα X (clockwise)
92      if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS) {
93          cameraPosition = cameraPosition * rotationMatrixX;
94          upVector = upVector * rotationMatrixX; // Ανανέωση του upVector
95          }
96
97      // Περιστροφή γύρω από τον άξονα Y (counterclockwise)
98      if (glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) {
99          cameraPosition = rotationMatrixY * cameraPosition;
100         upVector = rotationMatrixY * upVector; // Ανανέωση του upVector
101     }
102
103     // Περιστροφή γύρω από τον άξονα Y (clockwise)
104     if (glfwGetKey(window, GLFW_KEY_Z) == GLFW_PRESS) {
105         cameraPosition = cameraPosition * rotationMatrixY;
106         upVector = upVector * rotationMatrixY; // Ανανέωση του upVector
107     }
```

Ελέγχουμε αν έχει πατηθεί το πλήκτρο 'w':
Για να κάνουμε περιστροφή counterclockwise στον x άξονα, κάνουμε αριστερό πολλαπλασιασμό του πίνακα rotationMatrixX που δημιουργήσαμε με το cameraPosition.
Ελέγχουμε αν έχει πατηθεί το πλήκτρο 'x':
Για να κάνουμε περιστροφή clockwise στον x άξονα, κάνουμε δεξιό πολλαπλασιασμό του πίνακα rotationMatrixX που δημιουργήσαμε με το cameraPosition.

Υλοποιούμε αντίστοιχα για τον y άξονα.
Ελέγχουμε αν έχει πατηθεί το πλήκτρο 'q':
Για να κάνουμε περιστροφή counterclockwise στον y άξονα, κάνουμε αριστερό πολλαπλασιασμό του πίνακα rotationMatrixY που δημιουργήσαμε με το cameraPosition.

Ελέγχουμε αν έχει πατηθεί το πλήκτρο ’z’:
Για να κάνουμε περιστροφή clockwise στον y άξονα, κάνουμε δεξιό πολλαπλασιασμό του πίνακα rotationMatrixY που δημιουργήσαμε με το cameraPosition.

Μετά από κάθε περιστροφή, ενημερώνουμε και τον upVector για να διατηρείται σωστά ο προσανατολισμός της κάμερας.

Τέλος, πρέπει να υπολογίσουμε τον ανανεωμένο view matrix, δηλαδή να τροποποιήσουμε το πού κοιτάει η κάμερα. Χρησιμοποιούμε τη lookAt() και της δίνουμε τα κατάλληλα ορίσματα: το τροποποιημένο cameraPosition, το όρισμα cameraCenterPoint το οποίο δεν έχει αλλάξει και τον ανανεωμένο upVector.

```
110     glm::mat4 view = glm::lookAt(
111         cameraPosition, // Νέα θέ
112         cameraCenterPoint, // Το 
113         upVector // Ανανέωση του 
114     );
115
116
117
118     newView = view;
```

Μέσα στην do while καλούμε την συνάρτηση camera_function μαζί με το νέο MVP.

```
555         camera_function();
556
557         glm::mat4 Model = glm::mat4(1.0f);
558
559         glm::mat4 MVP = Projection * newView * Model;
```

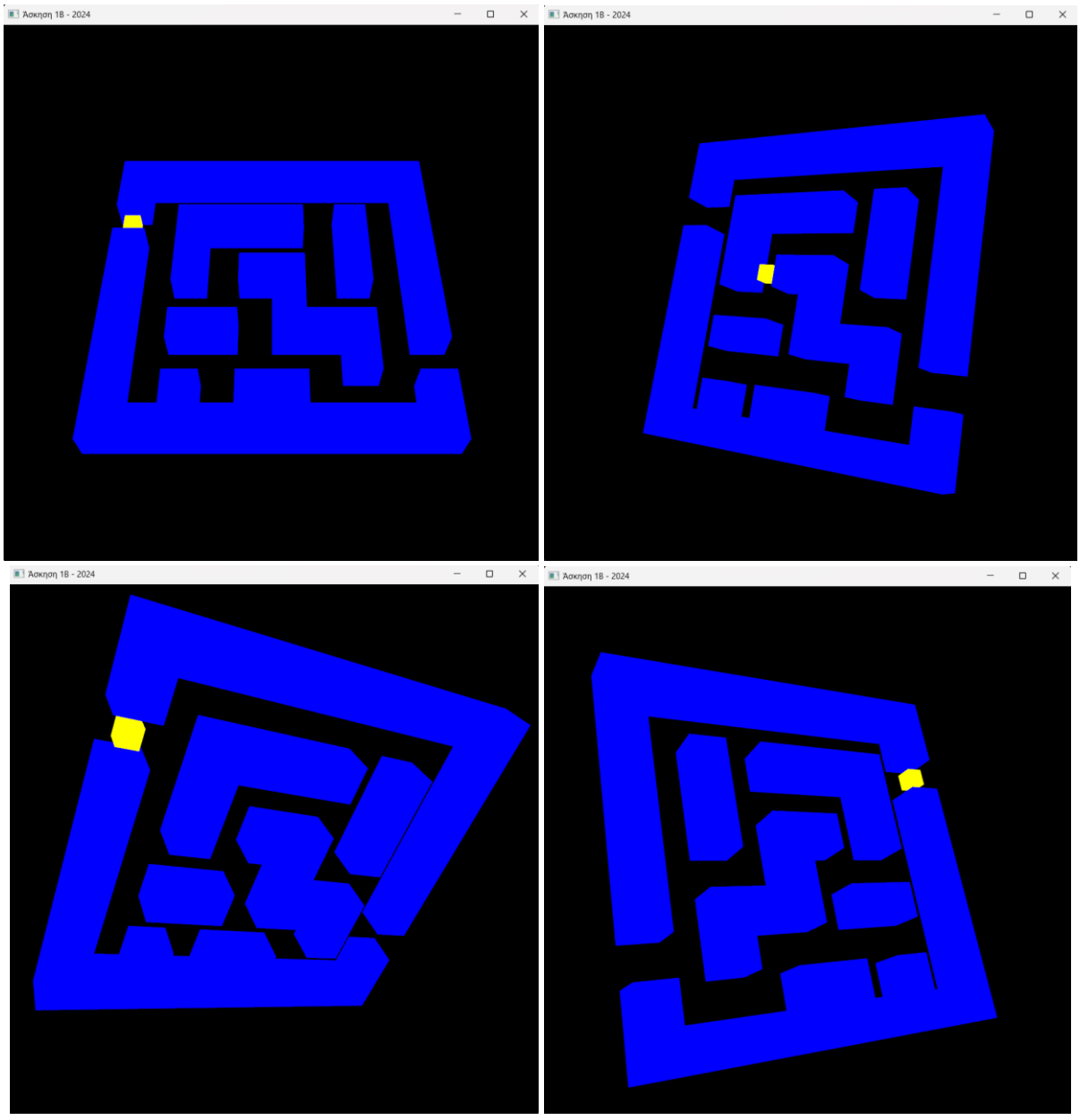
Περιγραφή δυσκολιών υλοποίησης -προβλήματα που συναντήθηκαν

Έχοντας υλοποιήσει την πρώτη εργαστηριακή άσκηση και βασισμένοι πάνω σε αυτήν, δεν συναντήθηκαν ιδιαίτερες δυσκολίες στην κατασκευή του σχήματος και στην κίνηση του παίκτη. Επίσης ο χρωματισμός βγήκε σχετικά εύκολα εις πέρας με την βοήθεια των βίντεο και των παραδειγμάτων των εργαστηρίων.
Η μεγαλύτερη δυσκολία ήταν στην κατανόηση των MVP και στην υλοποίηση της κάμερας, δηλαδή τι αλλαγές και προσθήκες θα έπρεπε να υλοποιήσουμε για την περιστροφή γύρω από τους άξονες και για το zoom in/zoom out, χωρίς να υπάρχουν προβλήματα.

Πληροφορίες σχετικά με την υλοποίηση

Λειτουργικό σύστημα: Windows 11 Pro Version 23H2
Περιβάλλον: Microsoft Visual Studio Community 2022 (64-bit) - Current Version 17.11.5

Στιγμιότυπα



Αναφορές – Πηγές που χρησιμοποιήθηκαν κατά την εκπόνηση της εργασίας.

LearnOpenGL - Hello Triangle
c++ - How to create a grid in OpenGL and drawing it with lines - Stack Overflow
GLFW: Input reference
unicode - UTF-8 Compatibility in C++ - Stack Overflow
stackoverflow-is-there-a-function-to-calculate-this-unit-vector-in-glm
https://www.opengl-tutorial.org/beginners-tutorials/tutorial-4-a-colored-cube/
https://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

Ευχαριστώ για την ανάγνωση.

Παναγιώτης Παρασκευόπουλος
ΑΜ: 2905