
Εγχειρίδιο Εκμάθησης της Βιβλιοθήκης OpenGL

John Crabs

Αθήνα 2018



Περιεχόμενα

1	Εισαγωγή	5
1.1	Γενικά Στοιχεία για το Εγχειρίδιο	5
1.2	Εγκατάσταση των Απαραίτητων Βιβλιοθηκών	6
1.3	Άδεια	6
2	Μία πρώτη ματιά στον κόσμο της OpenGL	7
2.1	Βασική Θεωρία	7
2.1.1	Τι είναι η OpenGL	7
2.1.2	Η OpenGL ως μηχανή κατάστασης	7
2.2	Δημιουργία Παραθύρου με Χρήση της GLUT	8
2.2.1	Ανάλυση των Συναρτήσεων	9
2.2.1.A'	display(void)	9
2.2.1.B'	init(void)	10
2.2.1.Γ'	winCreate(void)	10
2.2.1.Δ'	OpenGL/GLUT Functions	10
2.3	Δημιουργία Παραθύρου με Χρήση του QtCreator	12
2.3.1	Ανάλυση των Συναρτήσεων	19
2.3.1.A'	initializeGL(void)	19
2.3.1.B'	paintGL(void)	19
2.3.1.Γ'	resizeGL(int w, int h)	19
2.3.1.Δ'	OpenGL Functions	20
3	Απεικόνιση Απλής Κίνησης	21
3.1	Βασική Θεωρία	21
3.1.1	Ο Μηχανισμός της Κίνησης	21
3.1.2	Ανανέωση με Παύσεις	22
3.2	Απεικόνιση Κίνησης σε Περιβάλλον GLUT	23
3.2.1	Ανάλυση των Συναρτήσεων	25
3.2.1.A'	reshape(int w, int h)	25
3.2.1.B'	mouse(int button, int state, int x, int y)	25
3.2.1.Γ'	OpenGL/GLUT Functions	25
3.3	Συνοπτική Περιγραφή του Αλγορίθμου Κίνησης	26
3.4	Απεικόνιση Κίνησης σε Περιβάλλον QtCreator	27
4	Tic-Tac-Toe	33
4.1	Βασική Θεωρία	33
4.1.1	Ενίσχυση της μέχρι τώρα θεωρίας της OpenGL	33
4.1.1.A'	Σύνταξη των συναρτήσεων OpenGL	33
4.1.1.B'	Τα ορίσματα της συνάρτησης glBegin(...)	34
4.1.1.Γ'	Οι διεπαφές της συνάρτησης glClear(...)	36

4.1.2	Δομή Προγράμματος - Αλγόριθμος	37
4.2	Tic-Tac-Toe σε περιβάλλον GLUT	38
4.3	Tic-Tac-Toe σε περιβάλλον QtCreator	44

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικά Στοιχεία για το Εγχειρίδιο

Το παρόν εγχειρίδιο προσπαθεί να αποτελέσει έναν εύκολα προσβάσιμο κι εύκολα κατανοητό οδηγό εκμάθησης της βιβλιοθήκης γραφικών OpenGL. Κύριο μέλημά του είναι η εξοικείωση του αναγνώστη, τόσο με τις συναρτήσεις/εντολές της βιβλιοθήκης, όσο και με τη φιλοσοφία που κρύβεται πίσω από αυτές (προγραμματιστική λογική, μαθηματικά μοντέλα...).

Προκειμένου να επιτευχθεί ο παραπάνω στόχος, ο οδηγός χωρίζεται σε ενότητες με παραδείγματα. Κάθε παράδειγμα στηρίζεται σε κάποια βασική θεωρεία, στην οποία θα εξηγούνται τόσο οι συναρτήσεις που χρησιμοποιούνται στο παράδειγμα, όσο και η λογική σειρά με την οποία αυτές πρέπει να εκτελεστούν. Τέλος για την καλύτερη κατανόηση, αλλά και για τη δημιουργία χρήσιμων παραδειγμάτων κώδικα, κάθε παράδειγμα θα εκφράζεται με δύο διαφορετικούς τρόπους.

Ο πρώτος τρόπος έκφρασης είναι η χρήση της βιβλιοθήκης GLUT, για τη δημιουργία παραθύρου. Αυτός ο τρόπος αποτελεί μία εύκολη μέθοδο για την εκμάθηση και καλύτερη κατανόηση της βιβλιοθήκης OpenGL. Είναι ο τρόπος υπόδειξης των παραδειγμάτων του οδηγού OpenGL Programming Guide 2nd Edition, από τον οποίο έχουν παρθεί και τροποποιηθεί αρκετά από τα παραδείγματα, ώστε να ικανοποιούν τις ανάγκες του παρόντος εγχειρίδιου.

Ο δεύτερος τρόπος έκφρασης των παραδειγμάτων, είναι με τη χρήση του περιβάλλοντος QtCreator για τη διαχείριση παραθύρων. Αν και ο κώδικας είναι ελαφρώς πιο σύνθετος, συγκριτικά με τον πρώτο τρόπο, τα αρχεία κώδικα που δημιουργούνται αποτελούν χρήσιμο υλικό για τη δημιουργία εφαρμογών αργότερα. Επιπλέον, για να μη δημιουργηθεί σύγχυση, όπου κρίνεται απαραίτητο θα αναλύονται και οι συναρτήσεις/εντολές των βιβλιοθηκών του Qt.

Σε αυτό το σημείο κρίνεται σκόπιμο να ειπωθεί, πως το παρόν εγχειρίδιο θεωρεί δεδομένη τη βασική εξοικείωση του χρήστη με τη γλώσσα προγραμματισμού C/C++.

Τέλος, μπορεί κανείς να βρει τα παραδείγματα που περιγράφονται στο παρόν εγχειρίδιο, μαζί με τον οδηγό, καθώς κι άλλο χρήσιμο υλικό, στη σελίδα μου στο github.

<https://github.com/JohnCrabs>

1.2 Εγκατάσταση των Απαραίτητων Βιβλιοθηκών

Προκειμένου να μην υπάρξουν τεχνικές δυσκολίες, είναι αναγκαίο να εγκατασταθούν οι παρακάτω βιβλιοθήκες και προγράμματα:

- gl.h
- glu.h
- glut.h
- glew.h
- QtCreator

Οι τέσσερις βιβλιοθήκες που σχετίζονται με την OpenGL είναι απαραίτητες να υπάρχουν στο σύστημα. Η εγκατάσταση του QtCreator δεν είναι υποχρεωτική, ωστόσο η μη εγκατάσταση του θέτει αδύνατη τη χρήση των παραδειγμάτων που ακολουθούν το δεύτερο τρόπο έκφρασης.

Ο τρόπος εγκατάστασης των παραπάνω βιβλιοθηκών, διαφέρει ανάλογα το λειτουργικό σύστημα και επειδή υπάρχει μπόλικο υλικό στο διαδίκτυο σχετικά με το πως μπορούν να εγκατασταθούν, δε θα αναφερθεί κάποια μέθοδος εγκατάστασης.

1.3 Άδεια



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Κεφάλαιο 2

Μία πρώτη ματιά στον κόσμο της OpenGL

2.1 Βασική Θεωρεία

2.1.1 Τι είναι η OpenGL

Η OpenGL αποτελεί τη διεπαφή μεταξύ ενός λογισμικού και του υλικού συστήματος γραφικών του υπολογιστή. Η διεπαφή αυτή αποτελείται από περίπου 150 διακριτές εντολές, οι οποίες χρησιμοποιούνται για τον προσδιορισμό ενός αντικειμένου και των διαδικασιών που απαιτούνται ώστε να δημιουργηθούν διαδραστικές τριδιάστατες εφαρμογές.

Η OpenGL λειτουργεί ανεξαρτήτως υλικού συστήματος και είναι συμβατή με πολλές πλατφόρμες. Για να πραγματοποιηθεί αυτό, η OpenGL δε διαθέτει καμία εντολή διαχείρισης παραθύρων ή αλληλεπίδρασης του χρήστη με το πρόγραμμα, αλλά αντιθέτως μπορεί να συνεργαστεί με το οποιοδήποτε σύστημα διαχείρισης παραθύρων και περιφερειακού υλικού (στο παρόν εγχειρίδιο τα συστήματα αυτά είναι η βιβλιοθήκη GLUT και το λογισμικό QCreator). Τέλος η OpenGL δεν παρέχει υψηλού επιπέδου εντολές για την περιγραφή μοντέλων ή τριδιάστατων αντικειμένων. Η δημιουργία τέτοιων μοντέλων πραγματοποιείται με τη χρήση ενός μικρού σετ από γεωμετρικά πρότυπα (σημεία, ευθείες και πολύγωνα), τα οποία μπορούν να χρησιμοποιηθούν για να περιγράψουν οποιοδήποτε σύνθετο μοντέλο.

2.1.2 Η OpenGL ως μηχανή κατάστασης

Η OpenGL είναι μία μηχανή κατάστασης. Αυτό σημαίνει ότι κατά τη δημιουργία ενός προγράμματος, ο προγραμματιστής ορίζει διάφορες καταστάσεις, οι οποίες παραμένουν ενεργές έως ότου τροποποιηθούν. Ένα παράδειγμα είναι ο ορισμός του χρώματος σχεδιασμού ως άσπρο, κόκκινο κλπ. Όταν οριστεί ένα χρώμα, τότε κάθε φορά που σχεδιάζεται στην οθόνη ένα μοντέλο, θα εμφανιστεί με αυτό το χρώμα έως ότου οριστεί ένα νέο. Ένα άλλο χαρακτηριστικό παράδειγμα είναι τα προβολικά συστήματα και οι μετασχηματισμοί, οι οποίοι καθορίζουν το πως θα απεικονιστούν τα σχεδιασμένα αντικείμενα στην οθόνη. Σε επόμενα παραδείγματα θα γίνει περαιτέρω ανάλυση αυτών των καταστάσεων.

2.2 Δημιουργία Παραθύρου με Χρήση της GLUT

```
main.cpp 1

#include <GL/gl.h>
#include <GL/glut.h>

#define MY_WIN_WIDTH 250
#define MY_WIN_HEIGHT 250
#define MY_WIN_POS_X 100
#define MY_WIN_POS_Y 100
#define MY_WIN_TITLE "hello"

void display(void) {
    //clear all pixels
    glClear(GL_COLOR_BUFFER_BIT);

    /* draw white polygon (rectangle) with corners at
     * (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)
     */
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();

    /* don't wait!
     * start processing buffered OpenGL routines
     */
    glFlush();
}

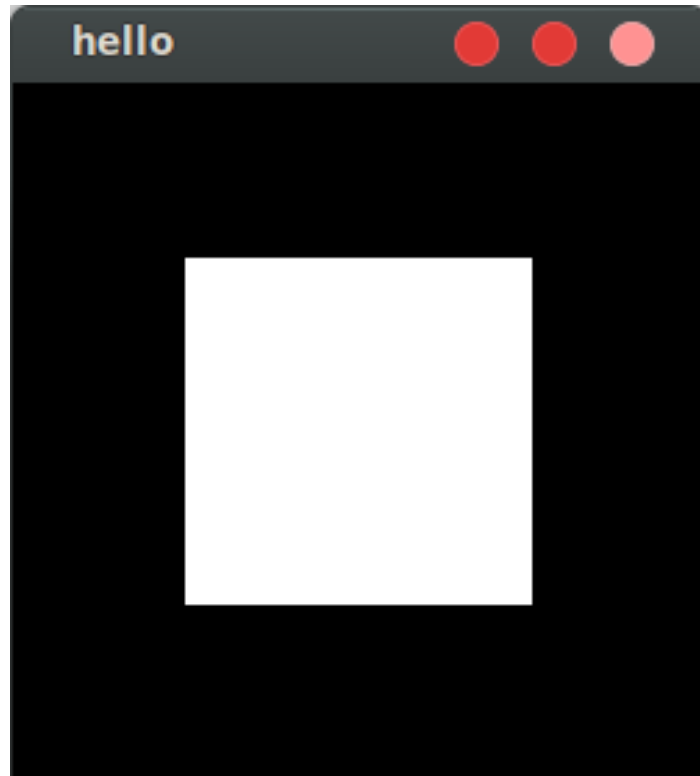
void init(void) {
    /* select clearing (background) color */
    glClearColor(0.0, 0.0, 0.0, 0.0);

    /* initialize viewing values */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

void winCreate(void) {
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(MY_WIN_WIDTH, MY_WIN_HEIGHT);
    glutInitWindowPosition(MY_WIN_POS_X, MY_WIN_POS_Y);
    glutCreateWindow(MY_WIN_TITLE);
}

/*
 * Declares initial window size, position and display mode
 * (single buffer and RGB). Open window with "hello"
 * in its title bar. Call initialization routines.
 * Register callback function to display graphics.
 * Enter main loop and process events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    winCreate();
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```


Στην προηγούμενη σελίδα φαίνεται ο κώδικας ενός απλού προγράμματος, που ανοίγει ένα παράθυρο και ζωγραφίζει σε αυτό ένα άσπρο ορθογώνιο, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 2.1: Δημιουργία Παραθύρου με χρήση της GLUT

2.2.1 Ανάλυση των Συναρτήσεων

Με μία πρώτη ματιά στον παραπάνω κώδικα παρατηρούνται τρεις βασικές συναρτήσεις.

- `display(void)`
- `init(void)`
- `winCreate(void)`

2.2.1.A' `display(void)`

Η συνάρτηση αυτή αποτελεί μία από τις τρεις βασικές και απαραίτητες συναρτήσεις που πρέπει να έχει ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη OpenGL. Οι άλλες δύο συναρτήσεις είναι οι:

- `init(void)`
- `resize(int w, int h)`

Η συνάρτηση `display(void)` διαχειρίζεται το τι πρέπει να σχεδιαστεί στην οθόνη και με ποια σειρά. Αυτό επιτυγχάνεται με την εντολή `glutDisplayFunc(display)` που καλείται στη συνάρτηση `main(...)`.

2.2.1.B' `init(void)`

Η συνάρτηση αυτή αποτελεί μία από τις τρεις βασικές και απαραίτητες συναρτήσεις που πρέπει να έχει ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη OpenGL. Οι άλλες δύο συναρτήσεις είναι οι:

- `display(void)`
- `resize(int w, int h)`

Η συνάρτηση `init(void)` αρχικοποιεί τις βασικές καταστάσεις της OpenGL, όπως είναι το χρώμα με το οποίο θα καθαρίζεται φόντο, χρησιμοποιώντας την εντολή `glClearColor(...)`. Επιπλέον η συνάρτηση αυτή περιέχει και το τι είδους προσανατολισμούς και μαθηματικά μοντέλα θα χρησιμοποιηθούν για την απεικόνιση.

Στο συγκεκριμένο παράδειγμα αυτά που αρχικοποιεί η συνάρτηση αυτή είναι ελάχιστα, σε πιο προχωρημένα παραδείγματα, ωστόσο, η συνάρτηση θα αρχικοποιεί παραπάνω παραμέτρους, όπως είναι το βάθος, η φωτεινότητα και άλλες χρήσιμες καταστάσεις, ώστε να δημιουργηθεί ένα πιο ρεαλιστικό γραφικό περιβάλλον.

2.2.1.Γ' `winCreate(void)`

Η συνάρτηση αυτή περιέχει τις βασικές συναρτήσεις για τη δημιουργία ενός παραθύρου GLUT. Οι συναρτήσεις αυτές θα μπορούσαν κάλλιστα να βρίσκονταν απευθείας στη `main(...)`, ωστόσο, είναι καλύτερα προγραμματιστικά εάν η συνάρτηση `main(...)` είναι μικρή σε έκταση. Οι συναρτήσεις της βιβλιοθήκης GLUT που χρησιμοποιούνται εντός της συνάρτησης είναι οι εξής:

- **`glutInitDisplayMode(unsigned int mode)`:** Η συνάρτηση αρχικοποίησης λειτουργιών εμφάνισης χρησιμοποιείται όταν δημιουργούνται παράθυρα, δευτερεύοντα παράθυρα και επικαλύψεις ανωτέρου επιπέδου, για να καθορίζεται η λειτουργία απεικόνισης της OpenGL, για το παράθυρο ή την επικάλυψη που πρόκειται να δημιουργηθεί.
- **`glutInitWindowSize(int width, int height)`:** Καθορίζει τις αρχικές διαστάσεις του παραθύρου.
- **`glutInitWindowPosition(int x, int y)`:** Καθορίζει την αρχική θέση του παραθύρου στην οθόνη.
- **`glutCreateWindow(char* title)`:** Δημιουργεί ένα παράθυρο υψηλού επιπέδου με τίτλο την εισαχθείσα ροή.

2.2.1.Δ' `OpenGL/GLUT Functions`

- **`glClear(GLbitfield mask)`:** Καθαρίζει τα pixel της οθόνης με τις προκαθορισμένες τιμές που έχουν οριστεί πρωτίτερα με την αρχικοποίηση των αντίστοιχων καταστάσεων.
- **`glColor3f(GLfloat red, GLfloat green, GLfloat blue)`:** Είναι μία από τις πολλές παραλλαγές της συνάρτησης που καθορίζει την κατάσταση του χρώματος σχεδίασης.
- **`glBegin(GLenum mode)`, `glEnd()`:** Οτιδήποτε βρίσκεται εντός των δύο αυτών συναρτήσεων θα σχεδιαστεί στην οθόνη. Η συνάρτηση `glBegin(...)` δέχεται σαν όρισμα την πρωτεύουσα γεωμετρική δομή σχεδίασης.
- **`glVertex3f(GLfloat x, GLfloat y, GLfloat z)`:** Είναι μία από τις πολλές παραλλαγές της συνάρτησης που ορίζει τη θέση ενός κόμβου σχεδίασης.

- **glFlush(void)**: Αδειάζει τα buffer, προκειμένου να δημιουργήσει χώρο και να επιταχύνει τη διαδικασία σχεδίασης σε πεπερασμένο χρόνο.
- **glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)**: Προσδιορίζει το ποσοστό συμμετοχής του κόκκινου, πράσινου, μπλε και αδιαφάνειας με το οποίο θα καθαρίζεται ο ρυθμιστής χρώματος. Οι προκαθορισμένες τιμές των παραμέτρων είναι 0.
- **glMatrixMode(GLenum mode)**: Καθορίζει ποια στοίβας μήτρας είναι ο στόχος για τις επόμενες λειτουργίες μήτρας. Οι επιτρεπόμενες τιμές είναι οι GL_PROJECTION, GL_MODELVIEW, GL_TEXTURE. Η προκαθορισμένη τιμή είναι GL_MODELVIEW.
- **glLoadIdentity(void)**: Αντικαθιστά την υφιστάμενη μήτρα με τη μήτρα που καθορίστηκε από τη συνάρτηση glMatrixMode(...).
- **glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal)**: Καθορίζει τις συντεταγμένες του αριστερού και δεξιού κατακόρυφων επιπέδων, του κάτω και πάνω οριζόντιων επιπέδων και των επιπέδων του βάθους.
- **glutInit(int *argc, char **argv)**: Αρχικοποιεί τα συστήματα και υποσυστήματα της βιβλιοθήκης GLUT.
- **glutMainLoop(void)**: Εισέρχεται στον κύριο ατέρμονο βρόγχο κι εκτελεί όλες τις παραπάνω συναρτήσεις. Δίχως αυτή τη συνάρτηση, δεν εμφανίζεται τίποτα στην οθόνη.

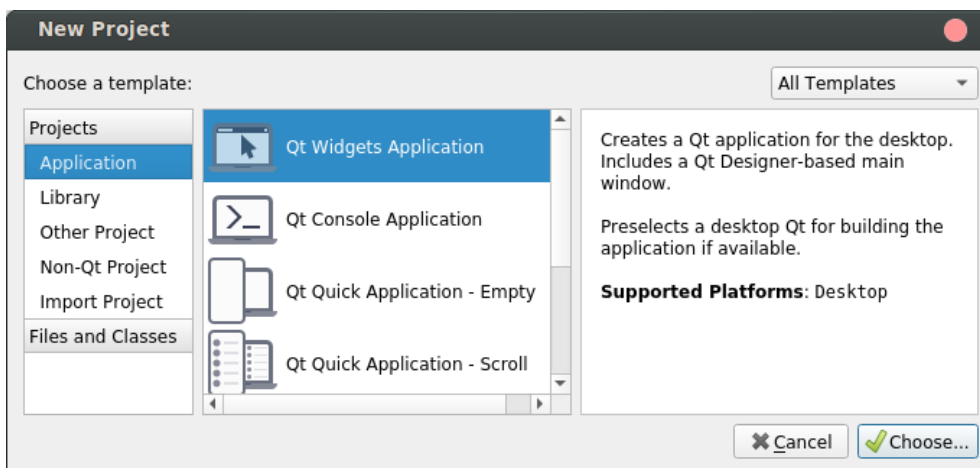
Σε αυτό το σημείο αξίζει να σημειωθεί, ότι μπορούν να βρεθούν εκτενέστερες περιγραφές για τις παραπάνω συναρτήσεις της OpenGL και GLUT στους παρακάτω ιστοχώρους.

- <https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/>
- <https://www.opengl.org/resources/libraries/glut/spec3/node1.html>

2.3 Δημιουργία Παραθύρου με Χρήση του QtCreator

Η Δημιουργία παραθύρου με χρήση του περιβάλλοντος QtCreator, απαιτεί κάποια στάδια, τα οποία διευκολύνουν τη διαδικασία. Τα στάδια είναι τα εξής:

New Project → Application → Qt Widgets Application



Σχήμα 2.2: Δημιουργία Qt Widgets Application

Στη συνέχεια πραγματοποιούνται οι κατάλληλες επιλογές και δημιουργούνται τα αρχεία **main.cpp**, **mainwindow.h**, **mainwindow.cpp** και **mainwindow.ui**.

Πριν ωστόσο αρχίσει το βασικό κομμάτι του προγραμματισμού, πρέπει να πραγματοποιηθούν δύο ακόμα στάδια.

Το πρώτο στάδιο είναι δημιουργία της κλάσης **GLWidget**, η οποία πρέπει να συνοδεύεται από τη βιβλιοθήκη **QGLWidget**. Η δημιουργία της κλάσης δημιουργεί τα αρχεία **glwidget.h** και **glwidget.cpp**.

Το δεύτερο στάδιο είναι η τροποποίηση του αρχείου **ProjectName.pro**, ώστε να περιέχει τους φακέλους και τις κατάλληλες σημαίες, για να αναγνωρίζει τις συναρτήσεις της βιβλιοθήκης OpenGL και τη διαδικασία του **compile**.

Τέλος χρειάζεται να σημειωθεί ότι για στο αρχείο **mainwindow.ui** πρέπει να δημιουργηθεί ένα **Widget**, το οποίο στη συνέχεια θα συνδεθεί με την κλάση **GLWidget**, χρησιμοποιώντας της επιλογής **Promote to**. Όσον αφορά την αλλαγή του ονόματος του παραθύρου, στο αρχείο **mainwindow.ui** και στις ιδιότητες του **MainWindow** αρχεί να αλλάξει το όνομα, όπως φαίνεται στο παρακάτω σχήμα.

contextMenuPolicy	DefaultContextMenu
acceptDrops	<input type="checkbox"/>
▶ windowTitle	Hello
▶ windowIcon	
windowOpacity	1.000000

Σχήμα 2.3: Αλλαγή του Ονόματος Παραθύρου

Στις επόμενες σελίδες φαίνονται τα αρχεία κώδικα.

```
#-----  
#  
# Project created by QtCreator 2018-02-05T17:43:03  
#  
#-----  
  
QT      += core gui opengl  
  
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  
  
TARGET = 03_Qt_version  
TEMPLATE = app  
  
# The following define makes your compiler emit warnings if you use  
# any feature of Qt which has been marked as deprecated (the exact warnings  
# depend on your compiler). Please consult the documentation of the  
# deprecated API in order to know how to port your code away from it.  
DEFINES += QT_DEPRECATED_WARNINGS  
  
# You can also make your code fail to compile if you use deprecated APIs.  
# In order to do so, uncomment the following line.  
# You can also select to disable deprecated APIs only up to a certain version of Qt.  
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before  
Qt 6.0.0  
  
SOURCES += \  
    main.cpp \  
    mainwindow.cpp \  
    glwidget.cpp  
  
HEADERS += \  
    mainwindow.h \  
    glwidget.h  
  
FORMS += \  
    mainwindow.ui  
  
LIBS += -lGL -lGLEW -lglut -lGLU
```

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```



```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <QGLWidget>
#include <QTimer>

class GLWidget : public QGLWidget
{
    Q_OBJECT
public:
    explicit GLWidget(QWidget *parent = nullptr);

    void initializeGL(void);

    void paintGL(void);

    void resizeGL(int w, int h);

private:
    QTimer GLtimer;
};

#endif // GLWIDGET_H
```

```
#include "glwidget.h"
#include <GL/glu.h>

GLWidget::GLWidget(QWidget *parent) : QGLWidget(parent)
{
    //Update Window every 30ms
    connect(&GLtimer, SIGNAL(timeout()), this, SLOT(updateGL()));
    GLtimer.start(30);
}

/* Same as init(void) */
void GLWidget::initializeGL(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0); //Set Clear Color State
}

/* Same as display(void) */
void GLWidget::paintGL(void) {
    glClear(GL_COLOR_BUFFER_BIT); //Clear Window Background

    glColor3f(1.0, 1.0, 1.0); //Set Draw Color to WHITE

    //Draw rectangle
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();
}

/* Same as reshape(int w, int h) */
void GLWidget::resizeGL(int w, int h) {
    glViewport(0, 0, (GLint)w, (GLint)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
```

Αυτά που πρέπει να σημειωθούν είναι ότι το αρχείο `ProjectName.pro` πρέπει να είναι όπως φαίνεται παραπάνω, ενώ τα αρχεία `main.cpp`, `mainwindow.h` και `mainwindow.cpp` όσον αφορά τη διαχείριση της OpenGL μένουν ως έχουν. Τα αρχεία `mainwindow.h` και `mainwindow.cpp` αποτελούν το συνδετικό κρίκο όλων των διαδικασιών που πραγματοποιούνται από την εφαρμογή, ωστόσο αυτό έχει να κάνει καθαρά με τη διαχείριση Qt παραθύρων κι έτσι δε θα πραγματοποιηθούν εκτενείς αναλύσεις (στο παρόν εγχειρίδιο θα αναλύονται μόνον ότι έχει να κάνει με την OpenGL και το πως αυτή μπορεί να συνεργαστεί με το Qt).

2.3.1 Ανάλυση των Συναρτήσεων

Αυτό που πρέπει να επισημανθεί σε αυτή την ενότητα είναι ότι στο περιβάλλον Qt, εφόσον στο αρχείο `glwidget.h` περιέχεται η βιβλιοθήκη **QGLWidget**, είναι πως δε χρειάζεται να γίνει τίποτε περισσότερο από τη δημιουργία των τριών βασικών συναρτήσεων της βιβλιοθήκης OpenGL με συγκεκριμένο όνομα. Επιπλέον χρειάζεται να δημιουργηθεί κι ένα χρονόμετρο `QTimer`, το οποίο θα ανανεώνει ανά τακτά χρονικά διαστήματα το παράθυρο.

Αυτό επιτυγχάνεται με τον εξής τρόπο:

- Στην κλάση `GLWidget` δημιουργείται μία ιδιωτική μεταβλητή `QTimer GLtimer`.
- Στο constructor της κλάσης στο αρχείο `glwidget.cpp` προστίθενται οι εξής γραμμές:
`connect(&GLtimer, SIGNAL(timeout()), this, SLOT(UpdateGL()));`
`GLtimer.start(30);`

Τέλος οι τρεις βασικές συναρτήσεις που πρέπει να περιέχονται στην κλάση είναι οι εξής:

- `initializeGL(void)`
- `paintGL(void)`
- `resizeGL(int w, int h)`

2.3.1.A' `initializeGL(void)`

Η συνάρτηση αυτή είναι η ίδια με την `init(void)` που χρησιμοποιείται στη δημιουργία παραθύρου μέσω GLUT. Αυτό που πρέπει να παρατηρηθεί είναι πως στη συνάρτηση αυτή τοποθετούνται μόνον οι συναρτήσεις που δίνουν ορίζουν αρχικές τιμές. Οι συναρτήσεις που διαχειρίζονται τους προσανατολισμούς και γενικά το πως θα προβληθούν στην οθόνη τα γεωμετρικά στοιχεία, τοποθετούνται στη συνάρτηση `resizeGL(int w, int h)`.

2.3.1.B' `paintGL(void)`

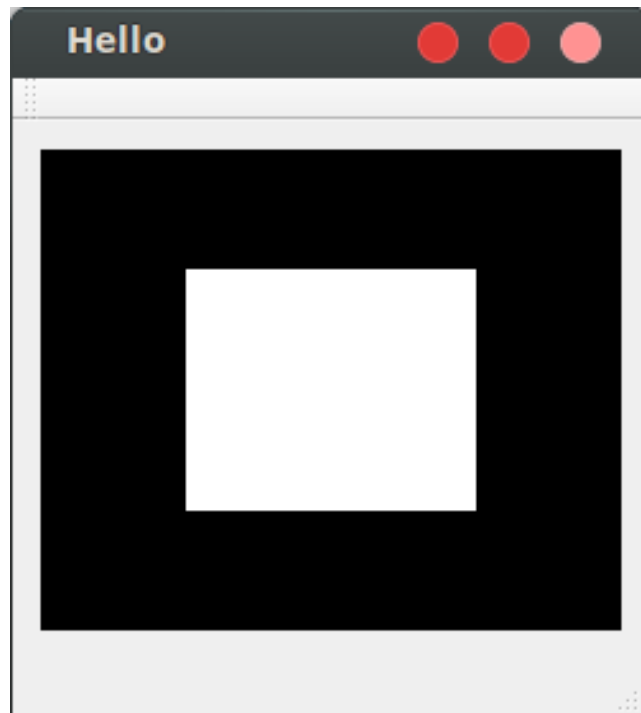
Η συνάρτηση αυτή είναι η ίδια με την `display(void)` που χρησιμοποιείται στη δημιουργία παραθύρου μέσω GLUT. Οτιδήποτε χρειάζεται να σχεδιαστεί στην οθόνη, τοποθετείται σε αυτή τη συνάρτηση.

2.3.1.Γ' `resizeGL(int w, int h)`

Η συνάρτηση αυτή αποτελεί στην τρίτη βασική συνάρτηση που απαιτείται να έχει ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη OpenGL και αντιστοιχεί στη συνάρτηση `reshape(int w, int h)` που θα χρησιμοποιηθεί στο επόμενο παράδειγμα. Η συνάρτηση αυτή περιέχει όλα εκείνα τα στοιχεία που απαιτούνται ώστε να προσδιοριστεί το πως θα απεικονιστούν τα γεωμετρικά στοιχεία στην οθόνη.

2.3.1.Δ' OpenGL Functions

- **glViewport**(GLint x, GLint y, GLsizei width, GLsizei height): Καθορίζει τα στοιχεία του αφινικού μετασχηματισμού των x και y από κανονικοποιημένες συντεταγμένες σε συντεταγμένες παραθύρου.



Σχήμα 2.4: Το απεικονιζόμενο παράθυρο με χρήση του QtCreator

Κεφάλαιο 3

Απεικόνιση Απλής Κίνησης

3.1 Βασική Θεωρεία

3.1.1 Ο Μηχανισμός της Κίνησης

Ένα από τα πιο ενδιαφέροντα πράγματα που μπορούν να συμβούν με τα γραφικά υπολογιστών είναι ο σχεδιασμός κινούμενων εικόνων. Το αντικείμενο αυτό έχει ευρεία χρήση και μπορεί να απασχολήσει, είτε μηχανικούς που προσπαθούν να μελετήσουν όλες τις πλευρές ενός μηχανικού αντικειμένου, είτε από πιλότους που εκπαιδεύονται μέσω ενός προγράμματος προσομοίωσης, είτε απλά από προγραμματιστές βιντεοπαιχνιδιών. Γενικότερα η κίνηση είναι ένα από τα σημαντικότερα ζητήματα στα γραφικά υπολογιστών.

Σε μία κινηματογραφική ταινία, η κίνηση πραγματοποιείται από τη μία διαδοχική προβολή εικόνων, οι οποίες προβάλλονται με ταχύτητα 24 καρέ/δευτερόλεπτο στην οθόνη. Κάθε καρέ μετακινείται πίσω από το φακό και το κλείστρο της μηχανής ανοίγει ώστε να προβληθεί το καρέ. Στη συνέχεια το κλείστο ξανακλείνει καθώς η ταινία μεταβάλλεται στο επόμενο καρέ και η διαδικασία αυτή επαναλαμβάνεται. Αν και οι θεατές βλέπουν 24 διαφορετικά καρέ/δευτερόλεπτο, το μυαλό του ανθρώπου τα αναμειγνύει, δημιουργώντας κατά αυτό τον τρόπο την αίσθηση μίας απλής κίνησης.

Τα σύγχρονα συστήματα προβολής προβάλλουν κάθε καρέ δύο φορές με ταχύτητα 48 καρέ/δευτερόλεπτο προκειμένου να μειώσουν το θόρυβο που προκαλείται από τη γρήγορη αλληλουχία των εικόνων που δίνουν την αίσθηση πως οι εικόνες τρεμοπαίζουν στην οθόνη. Οι οθόνες γραφικών των υπολογιστών τυπικά ανανεώνονται (επανασχεδιάζουν την εικόνα) με ταχύτητα που προσεγγίζει τα 60 καρέ/δευτερόλεπτο έως 76 καρέ/δευτερόλεπτο, ενώ υπάρχουν και περιπτώσεις που μπορούν να φτάσουν έως και ταχύτητα 120 καρέ/δευτερόλεπτο. Είναι φανερό πως ταχύτητα 60 καρέ/δευτερόλεπτο είναι πιο ομαλή από ταχύτητα 30 καρέ/δευτερόλεπτο και ταχύτητα 120 καρέ/δευτερόλεπτο είναι ελαφρώς καλύτερη από ταχύτητα 60 καρέ/δευτερόλεπτο. Ωστόσο δεν έχει νόημα να γίνεται συζήτηση για ταχύτητα πάνω από 120 καρέ/δευτερόλεπτο, αφού αυτή είναι η μέγιστη αντιληπτή από τον άνθρωπο ταχύτητα.

Προκειμένου η OpenGL να πετύχει τα παραπάνω αποτελέσματα, χρησιμοποιεί διπλές διεπαφές. Η διεπαφή D_1 προβάλλεται στην οθόνη και είναι αυτό που φαίνεται κάποια χρονική στιγμή t_0 , ενώ η διεπαφή D_2 βρίσκεται στην προσωρινή μνήμη και λειτουργεί ως καμβάς σχεδίασης της επόμενης εικόνας. Έτσι τη χρονική στιγμή $t_1 = t_0 + dt$ η διεπαφή D_2 είναι αυτή που προβάλλεται στην οθόνη, ενώ η διεπαφή D_1 λειτουργεί ως καμβάς σχεδίασης και η διαδικασία αυτή επαναλαμβάνεται έως ότου τερματιστεί η εφαρμογή. Το βήμα εναλλαγής dt των καρέ ισούται με το αντίστροφο της ταχύτητας εναλλαγής ($\text{fps} = \text{frame per second}$) των εικόνων, δηλαδή $dt = 1/\text{fps}$.

3.1.2 Ανανέωση με Παύσεις

Σε μερικές εκτελέσεις της OpenGL, εκτός από την απλή αλλαγή της εικονιζόμενης και της σχεδιαστικής διεπαφής, της ρουτίνας `swap_the_buffers()`, το σύστημα περιμένει μέχρι ο υφιστάμενος χρόνος ανανέωσης της οθόνης τελειώσει, ώστε η προηγούμενη διεπαφή να προβληθεί πλήρως. Η ρουτίνα αυτή επιπλέον επιτρέπει στη νέα διεπαφή να προβληθεί ολοκληρωμένη, ξεκινώντας από την αρχή.

Για παράδειγμα, ας γίνει η υπόθεση ότι ένα σύστημα ανανεώνεται 60 φορές/δευτερόλεπτο, αυτό σημαίνει ότι η μέγιστη ταχύτητα που μπορεί να επιτευχθεί είναι 60fps και εάν όλα τα καρέ καθαρίζονται κι επανασχεδιάζονται σε λιγότερο από $\frac{1}{60}sec$, τότε η κίνηση θα προβάλεται ομαλά σε αυτό το ρυθμό. Αυτό όμως που συμβαίνει σε ένα πραγματικό σύστημα είναι ότι τα καρέ είναι αρκετά πολύπλοκα ώστε να σχεδιάζονται σε χρόνο $\frac{1}{60}sec$, με αποτέλεσμα κάθε καρέ να προβάλεται περισσότερες από μία φορές. Εάν για παράδειγμα, το κάθε καρέ χρειάζεται $\frac{1}{45}sec$ για να σχεδιαστεί, τότε η ταχύτητα μειώνεται στα 30fps και τα γραφικά προβάλλονται ιδανικά για χρόνο $\frac{1}{30} - \frac{1}{45} = \frac{1}{90}spf$ (second per frame), ή αλλιώς το $\frac{1}{3}$ του χρόνου.

Επιπλέον, ο χρόνος ανανέωσης ενός βίντεο είναι σταθερός, το οποίο μπορεί να έχει προβλήματα στην απόδοση. Για παράδειγμα με $\frac{1}{60}spr$ (second per refresh) monitor και ένα σταθερό χρόνο εναλλαγής των καρέ, το σύστημα μπορεί να τρέχει με ταχύτητες 60fps, 30fps, 20fps, 15fps, 12fps και λοιπά ($\frac{60}{1}, \frac{60}{2}, \frac{60}{3}, \frac{60}{4}, \frac{60}{5}, \dots, \frac{60}{n}$). Αυτό σημαίνει πως όταν προγραμματίζεται μία εφαρμογή και προστίθενται σταδιακά διάφορα χαρακτηριστικά, αρχικά κάθε χαρακτηριστικό που προστίθεται δεν έχει κανένα ίχνος χρονικής καθυστέρησης ως προς τη συνολική απόδοση, ενώ κάποια στιγμή, με την προσθήκη ενός ακόμα χαρακτηριστικού, το σύστημα δε μπορεί να σχεδιάσει τη σκηνή του καρέ σε χρόνο $\frac{1}{60}sec$ κι έτσι η κίνηση μειώνεται από 60fps στα 30fps, επειδή χάνει το πρώτο χρονικό περιθώριο της πρώτης εναλλαγής της διεπαφής. Αντίστοιχα συμβαίνει και όταν ο χρόνος σχεδίασης αυξάνεται περισσότερο από $\frac{1}{30}sec$ (η κίνηση μειώνεται από 30έν fps στα 20fps).

Στην περίπτωση όπου η χρονική καθυστέρηση μεταξύ των καρέ είναι οριακή, δηλαδή έστω ότι το ένα καρέ μεταβάλλεται με ταχύτητα 60fps, ενώ το επόμενο με ταχύτητα 30fps και μετά πάλι μεταβολή με ταχύτητα 60fps, τότε το αποτέλεσμα είναι ενοχλητικό και συνίσταται η προσθήκη μίας επιπλέον καθυστέρησης, ώστε όλα τα καρέ να μεταβάλλονται σε χρόνο $\frac{1}{30}sec$, ώστε το τελικό αποτέλεσμα να φαίνεται ομαλότερο. Εάν η κατάσταση τείνει να είναι πιο πολύπλοκη, τότε χρειάζεται μία πιο σοφιστική μέθοδος προσέγγισης.

Γενικά αυτό που χρειάζεται να έχει υπόψιν ο προγραμματιστής είναι πως ο χρόνος μεταβολής της κίνησης υπολογίζεται από την παρακάτω σχέση:

$$Motion = Redraw_{scene} + Swap_{buffers} \quad (3.1)$$

3.2 Απεικόνιση Κίνησης σε Περιβάλλον GLUT

```
main.cpp 1

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>

#define MY_WIN_WIDTH 250
#define MY_WIN_HEIGHT 250
#define MY_WIN_POS_X 100
#define MY_WIN_POS_Y 100
#define MY_WIN_TITLE "Motion"

static GLfloat spin = 0.0;

void init(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glRectf(-25.0, -25.0, 25.0, 25.0);
    glPopMatrix();
    glutSwapBuffers();
}

void reshape(int w, int h) {
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void spinDisplay(void) {
    spin = spin + 2.0;
    if(spin > 360)
        spin -= 360;
    glutPostRedisplay();
}

void mouse(int button, int state, int x, int y) {
    switch(button) {
        case GLUT_LEFT_BUTTON: {
            if(state == GLUT_DOWN)
                glutIdleFunc(spinDisplay);
            break;
        } case GLUT_MIDDLE_BUTTON: {
            if(state == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
        } default:
            break;
    }
}

void initializeGL(void) {
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(MY_WIN_WIDTH, MY_WIN_HEIGHT);
    glutInitWindowPosition(MY_WIN_POS_X, MY_WIN_POS_Y);
    glutCreateWindow(MY_WIN_TITLE);
}
```

```
/*
 * Request double buffer display mode.
 * Register mouse input callback functions.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    initializeGL();

    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutMainLoop();

    return 0;
}
```


Το παράδειγμα αυτό είναι συνέχεια του προηγούμενου. Ο κώδικας που παρουσιάζεται παραπάνω ανοίγει ένα παράθυρο, στο κέντρο του οποίου εμφανίζεται ένα άσπρο ορθογώνιο. Τα νέα χαρακτηριστικά που εισάγει ο κώδικας είναι τα εξής:

- Τη συνάρτηση **reshape**(int w, int h), η οποία όπως ειπώθηκε στην προηγούμενη ενότητα διαχειρίζεται τους πίνακες προσανατολισμού και γενικότερα όλες εκείνες τις εξισώσεις που προσδιορίζουν την απεικόνιση των γεωμετρικών στοιχείων στην οθόνη. Η συνάρτηση αυτή εκτελείται κάθε φορά που αλλάζει διαστάσεις το παράθυρο.
- Απλή αλληλεπίδραση του χρήστη με το περιεχόμενο του παραθύρου, με τη χρήση του ποντικιού.
- Απλή στροφική κίνηση του ορθογώνιου, γύρω από το κέντρο μάζας του.

3.2.1 Ανάλυση των Συναρτήσεων

Σε αυτό το παράδειγμα εισήχθησαν οι εξής συναρτήσεις:

- **reshape**(int w, int h)
- **mouse**(int button, int state, int x, int y)

3.2.1.A' reshape(int w, int h)

Περιέχει τις συναρτήσεις της OpenGL, οι οποίες προσδιορίζουν το πως θα προβληθούν τα γεωμετρικά χαρακτηριστικά στην οθόνη. Η συνάρτηση αυτή αποτελεί την τρίτη βασική συνάρτηση που πρέπει να έχει ένα πρόγραμμα OpenGL, για να δημιουργεί χρήσιμες εφαρμογές. Καλείται από τη συνάρτηση **glutReshapeFunc(...)** κι εκτελείται κάθε φορά που το παράθυρο αλλάζει διαστάσεις.

3.2.1.B' mouse(int button, int state, int x, int y)

Διαχειρίζεται τα σήματα που δέχεται το πρόγραμμα από το ποντίκι. Τα ορίσματά της πρέπει αυστηρά να είναι τα παραπάνω, διότι καλείται αργότερα από τη συνάρτηση **glutMouseFunc(...)**, ώστε να γίνει η διεπαφή μεταξύ της εφαρμογής και του ποντικιού (σύνδεση μεταξύ hardware-software).

3.2.1.Γ' OpenGL/GLUT Functions

- **glShadeModel**(GLenum mode): Τα γεωμετρικά πρωτεύοντα μπορούν να έχουν είτε επίπεδο, είτε ομαλό τρόπο σκίασης (shading). Η ομαλή σκίαση, το προκαθορισμένο, επιβάλει στον υπολογισμό του χρώματος των κόμβων την παρεμβολή κατά τη διαδικασία ραστεροποίησης, τυπικά καθορίζει ένα διαφορετικό χρώμα για κάθε ένα pixel. Η επίπεδη σκίαση επιλέγει το υπολογιζόμενο χρώμα ενός κόμβου και με αυτό χρωματίζει όλα τα pixel κατά τη ραστεροποίηση ως εννιαίο πρωτεύον.

Οι τιμές αυτές επιλέγονται με τη χρήση των σημαιών **GL_SMOOTH** και **GL_FLAT**.

- **glPushMatrix**(void): Υπάρχει μία στοίβα από πίνακες για κάθε σύστημα πινάκων. Στο **GL_MODELVIEW** σύστημα, το μέγεθος της στοίβας είναι τουλάχιστον 32. Στα άλλα συστήματα **GL_COLOR**, **GL_PROJECTION** και **GL_TEXTURE**, το μέγεθος της στοίβας είναι τουλάχιστον 2. Ο υφιστάμενος πίνακας σε οποιοδήποτε σύστημα είναι ο πίνακας που βρίσκεται στην κορυφή της στοίβας του συστήματος.

Η συνάρτηση αυτή λοιπόν, προωθεί τον υφιστάμενο πίνακα της στοίβας, έναν πίνακα κάτω, διπλασιάζοντας τον υφιστάμενο πίνακα. Έτσι λοιπόν, μετά το κάλεσμα της συνάρτησης `glPushMatrix(void)`, ο πίνακας στην κορυφή της στοίβας είναι ίδιος με τον πίνακα που βρίσκεται ακριβώς από κάτω του.

- **glutPopMatrix(void)**: Η συνάρτηση αυτή αντικαθιστά τον υφιστάμενο πίνακα που βρίσκεται στην κορυφή της στοίβας με τον ακριβώς από κάτω του πίνακα.
- **glRectf(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2)**: Υποστηρίζει έναν ιδανικό τρόπο για τη δημιουργία ενός τετραγώνου, προσδιορίζοντας τα σημεία δύο αντιδιαμετρικών γωνιών. Το τελικό ορθογώνιο ορίζεται πάνω στο επίπεδο με $z = 0$.
- **glutSwapBuffers(void)**: Πραγματοποιεί την αλλαγή των διεπαφών, εάν το παράθυρο έχει οριστεί ως παράθυρο διπλής διεπαφής (double buffered window).
- **glutPostRedisplay(void)**: Μαρκάρει το υφιστάμενο παράθυρο έτσι όπως χρειάζεται ώστε να επαναπαρουσιαστεί.
- **glutIdleFunc(void (*func)(void))**: Θέτει ένα καθολικό κάλεσμα, ώστε το πρόγραμμα που χρησιμοποιεί διαχείριση παραθύρων GLUT να πραγματοποιεί επεξεργασία παρασκήνιου ή να συνεχίζει την κίνηση όταν η δραστηριότητα στο σύστημα παραθύρου δεν έχει ακόμα ληφθεί.

3.3 Συνοπτική Περιγραφή του Αλγορίθμου Κίνησης

Όπως φαίνεται παραπάνω, ο αλγόριθμος ακολουθεί μερικά απλά βήματα. Συνοπτικά τα βήματα είναι τα εξής:

- **Δημιουργία-Άνοιγμα Παραθύρου**: Σε αυτό το στάδιο συμπεριλαμβάνονται οι συναρτήσεις αρχικοποίησης και προσδιορισμού των γεωμετρικών και ποιοτικών χαρακτηριστικών του παραθύρου.
- **Επιλογή Προβολικού Συστήματος**: Συμπεριλαμβάνεται η συνάρτηση `reshape(int w, int h)`, η οποία προσδιορίζει το πως θα προβληθούν τα γεωμετρικά στοιχεία. Το βήμα αυτό εκτελείται κάθε φορά που αλλάζουν οι διαστάσεις του παραθύρου.
- **Υπολογισμός των Συντεταγμένων**: Σε κάθε επανάληψη επαναπροσδιορίζονται οι συντεταγμένες των γεωμετρικών στοιχείων. Εάν οι συντεταγμένες είναι ίδιες, τότε το σημείο θεωρείται σταθερό. Εάν αλλάζουν, τότε το αντικείμενο κινείται. Σημειώνεται πως ως κίνηση μπορεί να θεωρηθεί και η αλλαγή θέσης της κάμερας. Το πως θα πραγματοποιηθεί η κίνηση σε μία εφαρμογή, εξαρτάται καθαρά από την προβαλλόμενη σκηνή και από την αίσθηση που πρέπει να δοθεί.
- **Προβολή της Σκηνής (Εναλλαγή των Διεπαφών)**: Σε αυτό το στάδιο προβάλεται η σκηνή στην οθόνη, εναλλάσσοντας τις διεπαφές.
- **Επανάληψη των δύο τελευταίων σταδίων**.

3.4 Απεικόνιση Κίνησης σε Περιβάλλον QtCreator

```
mainwindow.h 1

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_SpinIt_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_SpinIt_clicked() //When Button Clicked
{
    ui->glWidget->changeSpinState();
}
```

```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <QGLWidget>
#include <QMouseEvent>
#include <QTimer>

class GLWidget : public QGLWidget
{
public:
    GLWidget(QWidget *parent);

    void initializeGL(void); //init(void)

    void resizeGL(int w, int h); //reshape(int h, int w)

    void paintGL(void); //display(void)

    void mousePressEvent(QMouseEvent* event); //Check Mouse Events

    void changeSpinState();

private:
    QTimer glTimer;
    GLdouble spin;
    bool spinState;

    void spinDisplay(void);
};

#endif // GLWIDGET_H
```

```

#include "glwidget.h"

GLWidget::GLWidget(QWidget* parent) :
    QGLWidget(parent)
{
    connect(&glTimer, SIGNAL(timeout()), this, SLOT(updateGL()));
    glTimer.start(30);

    spin = 0.0;
    spinState = false;
}

/* Same as
 * void init(void){...}
 * function.
 */
void GLWidget::initializeGL(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
}

/* Same as
 * void reshape(int w, int h){...}
 * function. There isn't any need to call
 * glutReshapeFunc(...)
 * to run this function.
 */
void GLWidget::resizeGL(int w, int h) {
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

/* Same as
 * void display(void){...}
 * function. There isn't any need to call
 * glutDisplayFunc(...)
 * to run this function.
 */
void GLWidget::paintGL(void) {
    if(spinState)
        spinDisplay();
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(spin, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glRectf(-25.0, -25.0, 25.0, 25.0);
    glPopMatrix();
}

//Mouse Event Handling on QT
void GLWidget::mousePressEvent(QMouseEvent* event) {
    if(event->button() == Qt::LeftButton) {
        spinState = true;
    } else if (event->button() == Qt::MiddleButton) {
        spinState = false;
    } else {
        ;
    }
}

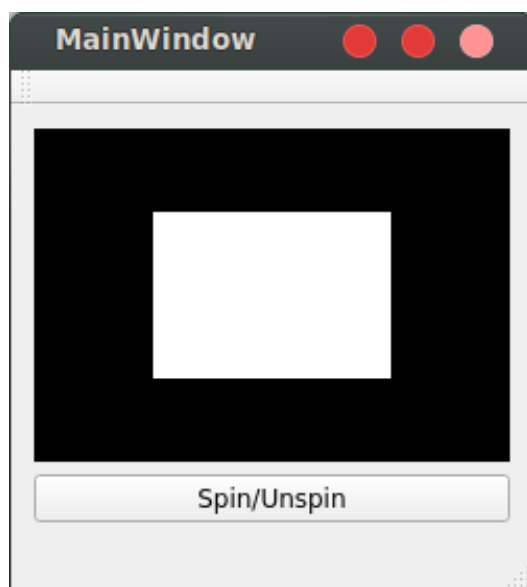
```

```
void GLWidget::spinDisplay(void) {  
    spin += 2.0;  
    if(spin > 360)  
        spin -= 360;  
}  
  
void GLWidget::changeSpinState() {  
    spinState = (spinState == true) ? false : true;  
}
```

Από αυτό το παράδειγμα και μετά, στην ενότητα αυτή θα προστίθενται μόνο τα τμήματα κώδικα που διαφοροποιούνται από το αρχικό. Δηλαδή το αρχείο κώδικα **main.cpp**, είναι το ίδιο με το πρώτο παράδειγμα κώδικα, οπότε δεν υπάρχει λόγος υπόδειξής του. Μία δεύτερη επισήμανση που πρέπει να γίνει σε αυτό το σημείο, είναι ότι ο κώδικας στο περιβάλλον QtCreator θα έχει επιπλέον προσθήκες, προκειμένου να προσεγγίζει όσο το δυνατόν γίνεται μία εμφανίσιμη εφαρμογή. Στο συγκεκριμένο παράδειγμα, εκτός από την αλληλεπίδραση του χρήστη με το παράθυρο μέσω του ποντικιού, προστίθεται κι ένα QPushButton (κουμπί), το οποίο κάνει ακριβώς το ίδιο, αλλά με πιο κομψό τρόπο.

Οι διαφορές του περιβάλλοντος QtCreator με το περιβάλλον GLUT είναι οι εξής:

- **mousePressEvent**(QMouseEvent* event): Η αντίστοιχη συνάρτηση που διαχειρίζεται τα σήματα του ποντικιού.
- **changeSpinState**(void): Προκειμένου να ελεγχθεί το αν πρέπει η όχι να πραγματοποιηθεί η κίνηση, ορίζεται μία μεταβλητή **bool**, η οποία έχει τιμή **true** όταν πρέπει να γίνει κίνηση και τιμή **false** όταν δεν πρέπει να γίνει κίνηση. Η συνάρτηση αυτή χρησιμοποιείται από το QPushButton.
- **on_SpinIt_Clicked**(void): Είναι ένα Slot του QtCreator, το οποίο σημαίνει πως όταν γίνει κλικ στο QPushButton με όνομα SpinIt, θα εκτελεστεί το αντίστοιχο τμήμα κώδικα που υπάρχει στο αρχείο mainwindow.cpp και αντιστοιχεί στη συγκεκριμένη συνάρτηση.



Σχήμα 3.1: Το απεικονιζόμενο παράθυρο με χρήση του QtCreator. Εάν παρηθεί το κουμπί Spin/Unspin το ορθογώνιο αρχίζει να στρέφεται ή σταματά την κίνηση. Το τετράγωνο κινείται επίσης με αριστερό κλικ στο παράθυρο και σταματάει πιέζοντας τη ροδέλα.

Κεφάλαιο 4

Tic-Tac-Toe

Σε αυτό το κεφάλαιο δε θα εισαχθεί κάποια νέα συνάρτηση της OpenGL. Για την ακρίβεια ότι έχει ειπωθεί έως τώρα είναι ότι χρειάζεται κάποιος για να δημιουργήσει απλές ολοκληρωμένες εφαρμογές. Οι πιο σύνθετες εφαρμογές, χρησιμοποιούν τις παραπάνω έννοιες, εμπλουτίζοντάς τις με μαθηματικά μοντέλα που διαχειρίζονται χαρακτηριστικά μεγέθη, όπως είναι η φωτεινότητα του χώρου, το υλικό ενός αντικειμένου, το χρώμα ενός αντικειμένου, η σκιά ενός αντικειμένου πάνω σε μία ή πολλές επιφάνειες/ες και γενικότερα οτιδήποτε θεωρεί ο προγραμματιστής απαραίτητο, ώστε να δημιουργήσει τη σκηνή που φαντάζεται.

Γι' αυτό το λόγο, κρίνεται σκόπιμο σε αυτό το σημείο η δημιουργία ενός ολοκληρωμένου προγράμματος, κυρίως για δύο λόγους:

- Ολοκλήρωση της θεωρείας που έχει αναφερθεί παραπάνω, εμπλουτίζοντας παράλληλα και τις περιγραφές κάποιων συναρτήσεων, με στοιχεία που έχουν παραληφθεί.
- Σύνδεση των εντολών της OpenGL που έχουν αναφερθεί έως τώρα με στοιχειώδη προγραμματιστική λογική που θεωρείται δεδομένη στο παρόν εγχειρίδιο, για τη δημιουργία μίας χρήσιμης εφαρμογής.

Τέλος σε αυτή την ενότητα θα φανεί και η μεγάλη των δυνατοτήτων και της σημαντικότητας επιλογής του περιβάλλοντος διαχείρισης παραθύρων. Ενώ το πρόγραμμα σε γενικές γραμμές θα είναι το ίδιο, θα υπάρχουν διαφοροποιήσεις ως προς την οργάνωση του και την απόδοση του μεταξύ των δύο περιβάλλοντων διαχείρισης παραθύρων.

4.1 Βασική Θεωρεία

4.1.1 Ενίσχυση της μέχρι τώρα θεωρείας της OpenGL

Στη μέχρι τώρα θεωρεία, έχουν σκοπίμως παραληφθεί κάποιες μικρές λεπτομέρειες, οι οποίες αναφέρονται παρακάτω και αποτελούν τη θεματολογία αυτής της ενότητας.

- Σύνταξη των συναρτήσεων της OpenGL.
- Τα ορίσματα της συνάρτησης `glBegin(...)`.
- Οι διεπαφές της συνάρτησης `glClear(...)`.

4.1.1.A' Σύνταξη των συναρτήσεων OpenGL

Οι εντολές της βιβλιοθήκης OpenGL έχουν συνταχθεί με μία συγκεκριμένη τυποποίηση. Για παράδειγμα η συνάρτηση με την οποία δημιουργείται ένα σημείο έχει όνομα `glVertex3f(...)` ή

glVertex2i(...). Παρατηρώντας τις παραπάνω συναρτήσεις, μπορεί εύκολα κάποιος να διακρίνει τα εξής χαρακτηριστικά:

- Κάθε εντολή της βιβλιοθήκης OpenGL αρχίζει με την προσφώνηση `gl`. Αυτό συμβαίνει, επειδή υπάρχουν και άλλες βοηθητικές βιβλιοθήκες συμβατές με την OpenGL, οι οποίες έχουν άλλες προσφωνήσεις, όπως π.χ. `glut` για τη βιβλιοθήκη OpenGL Utility Toolkit, την οποία χρησιμοποιεί το παρόν εγχειρίδιο για τη δημιουργία παραθύρων.
- Το δεύτερο τμήμα της εντολής υποδεικνύει το τι κάνει η εντολή. Στα παραπάνω παραδείγματα το τμήμα αυτό αποτελεί η λέξη `Vertex` και δημιουργεί ένα κόμβο - σημείο. Αντίστοιχα η εντολή προσδιορισμού χρώματος, θα είχε τη λέξη `Color`.
- Το τρίτο τμήμα κάθε εντολής, εάν υπάρχει, είναι το πλήθος των ορισμάτων της εντολής. Για παράδειγμα η εντολή `glVertex3f(...)` δέχεται τρία ορίσματα, ενώ η εντολή `glVertex2f(...)` δέχεται δύο ορίσματα.
- Τέλος ο αριθμός ακολουθείται από τουλάχιστον ένα γράμμα, το οποίο προσδιορίζει τον τύπο των δεδομένων που δέχεται (π.χ. `f` για floating point μορφοποίηση)

Στον παρακάτω πίνακα φαίνονται οι διάφοροι τύποι μορφοποιήσεων που μπορεί να δέχεται μία εντολή και τα γράμματα με τα οποία υποδεικνύεται αυτό στις συναρτήσεις.

Κατάληξη	Τύπος Δεδομένων	Μορφοποίηση στη C	Όρισμος μορφοποίησης στην OpenGL
b	8-bit ακέραιος	signed char	GLbyte
s	16-bit ακέραιος	short	GLshort
i	32-bit ακέραιος	int ή long	GLint, GLsizei
f	32-bit δεκαδικός	float	GLfloat, GLclampf
d	64-bit δεκαδικός	double	GLdouble, GLclampd
ub	8-bit μη προσημασμένος ακέραιος	unsigned char	GLubyte, GLboolean
us	16-bit μη προσημασμένος ακέραιος	unsigned short	GLushort
ui	32-bit μη προσημασμένος ακέραιος	unsigned int ή unsigned long	GLuint, GLenum, GLbitfield

Σχήμα 4.1: Μορφοποιήσεις Εντολών OpenGL

Τέλος κάποιες από τις εντολές μπορεί στο τέλος να έχουν το γράμμα `n`, το οποίο υποδεικνύει πως αντί για μια σειρά ορισμάτων, απαιτείται ένας μόνο πίνακας μεγέθους όσο ο αριθμός των ορισμάτων και αντίστοιχης μορφοποίησης. Για παράδειγμα η εντολή `glColor3fv(...)`, δέχεται σαν όρισμα έναν πίνακα τριών διαστάσεων δεκαδικής μορφοποίησης (`GLfloat array[3]`).

4.1.1.B' Τα ορίσματα της συνάρτησης `glBegin(...)`

Σε προηγούμενο κεφάλαιο πραγματοποιήθηκε ανάλυση της συνάρτησης `glBegin(...)`. Όπως ειπώθηκε, οτιδήποτε βρίσκεται μεταξύ της των συναρτήσεων `glBegin(...)` και `glEnd()` θα σχεδιαστεί στην οθόνη. Αυτά που δεν ειπώθηκαν είναι πως η συνάρτηση `glBegin(...)` δέχεται ένα

συγκεκριμένο πλήθος ορισμάτων/σημαίων, τα οποία χρησιμοποιούνται για το σχεδιασμό διαφορετικών πρωτεύοντων, καθώς επίσης πως οι συναρτήσεις, οι οποίες μπορούν να χρησιμοποιηθούν μεταξύ των `glBegin(...)` και `glEnd()` είναι περιορισμένες.

Αυτή η ενότητα αποσκοπεί, στο να καλύψει αυτά τα κενά. Παρακάτω φαίνονται οι λίστες των συναρτήσεων που χρησιμοποιούνται εντός των `glBegin(...)` και `glEnd()`, καθώς και τα επιτρεπόμενα ορίσματα της `glBegin(...)`.

Λίστα Συναρτήσεων:

- **glVertex*(...)**: Ορίζει τις συντεταγμένες του κόμβου.
- **glColor*(...)**: Ορίζει το χρώμα σχεδίασης.
- **glIndex*(...)**: Θέτει δείκτη για το υφιστάμενο χρώμα.
- **glNormal*(...)**: Ορίζει τις συντεταγμένες ενός κανονικού διανύσματος.
- **glTexCoord*(...)**: Θέτει τις συντεταγμένες ταπετσαρίας (texture).
- **glEdgeFlag*(...)**: Ελέγχει τις ακμές του σχεδίου.
- **glMaterial*(...)**: Ορίζει τις ιδιότητες των υλικών.
- **glArrayElement(...)**: Αποσυμπίεζει διανυσματικά δεδομένα πίνακα.
- **glEvalCoord*(...), glEvalPoint*()**: Παράγει συντεταγμένες.
- **glCallList(...), glCallLists(...)**: Εκτελεί λίστα/ες προβολής.

Όπου υπάρχει το σύμβολο *, η συνάρτηση συντάσσεται με παραπάνω από ένα τρόπο.

Λίστα Ορισμάτων της `glBegin(...)`:

- **GL_POINTS**: Σχεδιάζει ένα σημείο για κάθε κόμβο.
- **GL_LINES**: Σχεδιάζει μία σειρά από μη ενωμένες ευθείες. Κάθε ευθεία σχεδιάζεται μεταξύ δύο κόμβων. Έστω ότι ορίζονται οι κόμβοι v_1, v_2, v_3 και v_4 , θα σχεδιαστούν δύο ευθείες μεταξύ των κόμβων v_1 και v_2 (e_1) και v_3 και v_4 (e_2). Εάν ο αριθμός των κόμβων είναι περιττός (δε διαιρείται με το 2, τότε ο κόμβος που περισσεύει αγνοείται).
- **GL_LINE_STRIP**: Σχεδιάζει μία σειρά ευθειών που ενώνει τους κόμβους, με τη σειρά που ορίζονται. Έστω ότι ορίζονται οι κόμβοι $v_1, v_2, v_3, \dots, v_{n-1}$. Το τελικό σχήμα θα σχηματίζεται από τις ευθείες, $v_1 - v_2, v_2 - v_3, \dots, v_{n-2} - v_{n-1}$. Τίποτα δε σχεδιάζεται εάν δεν ισχύει $n > 1$.
- **GL_LINE_LOOP**: Εκτελεί το ίδιο με το όρισμα **GL_LINE_STRIP**, με τη διαφορά ότι η το σχήμα κλείνει, ενώνοντας τους κόμβους $v_{n-1} - v_1$, σχηματίζοντας έτσι ένα βρόγχο.
- **GL_TRIANGLES**: Σχεδιάζει μία σειρά από τρίγωνα, με την ίδια λογική που σχεδιάζει μία σειρά ευθειών χρησιμοποιώντας το όρισμα **GL_LINES**. Η διαφορά εδώ είναι πως οι κόμβοι πρέπει να είναι πολλαπλάσιοι του τρία, ενώ εάν περισσεύουν ένας ή δύο, τότε αυτοί αγνοούνται.

- **GL_TRIANGLE_STRIP**: Σχεδιάζει μία σειρά από τρίγωνα που μοιράζονται μία πλευρά, για παράδειγμα $v_0 - v_1 - v_2$, $v_2 - v_1 - v_3$ (προσοχή στη σειρά σχεδίασης, δηλαδή τα επόμενα δύο τρίγωνα θα είναι $v_2 - v_3 - v_4$ και $v_4 - v_3 - v_5$). Η σειρά αυτή ορίστηκε, ώστε τα τρίγωνα να σχεδιάζονται όλα με συγκεκριμένη φορά, ώστε το τελικό σχήμα να προσδιορίζει σωστά μία επιφάνεια. Για να σχεδιαστεί κάτι, πρέπει να υπάρχουν τουλάχιστον τρεις κόμβοι.
- **GL_TRIANGLE_FAN**: Έχει ακριβώς την ίδια φιλοσοφία με το προαναφερθέν όρισμα **GL_TRIANGLE_STRIP**, με τη διαφορά ότι τα τρίγωνα που σχηματίζονται είναι τα $v_0 - v_1 - v_2$, $v_0 - v_2 - v_3$, $v_0 - v_3 - v_4$ και λοιπά.
- **GL_QUADS**: Σχεδιάζει μία σειρά από τετράπλευρα, με την ίδια λογική που σχεδιάζει μία σειρά από ευθείες, χρησιμοποιώντας το όρισμα **GL_LINE**, δηλαδή για κόμβους $v_1, v_2, v_3, v_4, v_5, v_6, v_7$ και v_8 , θα σχεδιάσει δύο τετράπλευρά ενώνοντας τους κόμβους $v_1 - v_2 - v_3 - v_4$ (q_1) και $v_5 - v_6 - v_7 - v_8$ (q_2). Το ελάχιστο πλήθος των κόμβων που απαιτείται για να σχεδιάσει κάτι πρέπει να είναι 4, ενώ για κάθε πολλαπλάσιο του 4 σχεδιάζει κι από ένα τετράπλευρο, εάν περισσεύουν ένα, δύο ή τρεις κόμβοι, τότε αυτοί αγνοούνται.
- **GL_QUAD_STRIP**: Σχεδιάζει μία σειρά από τετράπλευρα που μοιράζονται μία κοινή πλευρά. Για να σχεδιάσει κάτι στην οθόνη απαιτούνται τουλάχιστον 4 κόμβοι, ενώ ο αριθμός των κόμβων πρέπει να είναι άρτιος. Εάν ο αριθμός είναι περιττός, ο κόμβος που περισσεύει (τελευταίος κατά σειρά κόμβος) αγνοείται.
- **GL_POLYGON**: Σχεδιάζει ένα κλειστό συμπαγές πολύγωνο χρησιμοποιώντας τους κόμβους που δίνονται. Το ελάχιστο πλήθος κόμβων για να σχεδιάσει κάτι είναι 3, ενώ χρησιμοποιούνται όλοι οι κόμβοι που δίνονται με τη σειρά, για το σχηματισμό του πολυγώνου. Επιπλέον το πολύγωνο δεν πρέπει να τέμνει τον εαυτό του και πρέπει να είναι κυρτό. Εάν δεν πληρούνται αυτές οι προδιαγραφές, τότε το αποτέλεσμα θα είναι απροσδόκητο.

4.1.1.Γ' Οι διεπαφές της συνάρτησης `glClear(...)`

Η βιβλιοθήκη OpenGL διαθέτει τέσσερις διεπαφές, για τις οποίες μπορούν να καθοριστούν αρχικές τιμές, οι οποίες τίθενται σε εφαρμογή κάθε φορά που εκτελείται η συνάρτηση `glClear(...)`. Προκειμένου να εφαρμοστεί όμως η αρχική τιμή σε μία συγκεκριμένη διεπαφή, αυτή πρέπει να έχει οριστεί στη συνάρτηση `glClear(...)`. Παρακάτω φαίνονται οι σημαίες που μπορεί να δεχτεί η συνάρτηση `glClear(...)`.

Διεπαφές Καθαρισμού

- **GL_COLOR_BUFFER_BIT**: Διεπαφή χρώματος παρασκήνιου.
- **GL_DEPTH_BUFFER_BIT**: Διεπαφή βάθους.
- **GL_ACCUMULATION_BUFFER_BIT**: Διεπαφή συσσώρευσης.
- **GL_STENCIL_BUFFER_BIT**: Διεπαφή πολυγράφου.

Ο τρόπος με τον οποίο ορίζονται παραπάνω από μία διεπαφή, είναι η χρήση του λογικού συμβόλου `or`, το οποίο στο πληκτρολόγιο ορίζεται ως `|`. Δηλαδή:

```
glClearColor(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

4.1.2 Δομή Προγράμματος - Αλγόριθμος

Η δημιουργία του Tic-Tac-Toe, δηλαδή της κλασσικής τρίλιζας είναι ένα από τα πιο απλά παιχνίδια που μπορεί να προγραμματίσει κανείς. Η λογική είναι η εξής:

- Δημιουργία ενός πίνακα 3×3 , ο οποίος θα αποθηκεύει τις επιλογές των παιχτών, ώστε να γίνεται ο έλεγχος νίκης και ισοπαλίας. Στην περίπτωση των γραφικών, ο πίνακας θα έχει αποθηκευμένες και τις συντεταγμένες των σημείων που περιγράφουν το αντίστοιχο κελί στην οθόνη.
- Ορισμός μεταβλητών ελέγχου νίκης και ισοπαλίας, οι οποίες θα δέχονται λογικά ορίσματα (true ή false) και θα τερματίζουν την εφαρμογή, εμφανίζοντας το κατάλληλο μήνυμα.
- Πρέπει να υπάρχει μία συνάρτηση που θα διαχειρίζεται το πως θα επιλέγουν οι παίκτες το κελί που θέλουν. Στο συγκεκριμένο παράδειγμα, η επιλογή αυτή πραγματοποιείται με αριστερό κλικ στο κελί. Το πρόγραμμα αναγνωρίζει τις συντεταγμένες σε pixel παραθύρου και πραγματοποιεί αλλαγή κλίμακας, ώστε να μετατραπούν οι συντεταγμένες παραθύρου σε συντεταγμένες πίνακα.
- Ακόμα υπάρχουν άλλες δύο λογικές μεταβλητές. Η μία μεταβλητή ελέγχει εάν το κελί είναι ελεύθερο ή όχι, ενώ η δεύτερη εάν πρέπει να σχεδιαστεί κάποιο σύμβολο ή όχι. Ένας εναλλακτικός τρόπος για την αντικατάσταση αυτών των μεταβλητών, είναι να οριστεί μία αρχική τιμή για κάθε κελί και το κελί να σχεδιάζεται, όταν η τιμή του είναι διαφορετική από την αρχική.
- Τέλος στα κελιά που έχουν επιλεγεί σχεδιάζεται το σύμβολο του παίχτη που τα επέλεξε και η διαδικασία επαναλαμβάνεται έως ότου κάποιος από τους δύο παίκτες κερδίσει ή συμπληρωθούν όλα τα κελιά του πίνακα (συνθήκη ισοπαλίας).

Σε αυτό το παράδειγμα, δεν θα υπάρξει επεξήγηση του κώδικα. Οι παραπάνω περιγραφές και η θεώρηση ότι ο αναγνώστης έχει βασικές γνώσεις σχετικά με τη γλώσσα προγραμματισμού C/C++, είναι ότι χρειάζεται κανείς για να κατανοήσει τους κώδικες. Επιπλέον σε αυτό το σημείο θα δοθεί και η πρώτη άσκηση.

Άσκηση: Να τροποποιηθεί ο κώδικας του παιχνιδιού Tic-Tac-Toe, ώστε να υποστηρίζει την επιλογή παιχνιδιού ενάντια σε υπολογιστή. Ποιές πρέπει να είναι οι επιλογές και με ποια σειρά που πρέπει να εκτελέσει ένας υπολογιστής, ώστε το καλύτερο δυνατό σενάριο που μπορεί να φέρει ένας παίχτης εναντίον του να είναι ισοπαλία;

4.2 Tic-Tac-Toe σε περιβάλλον GLUT

```

main.cpp 1

#include <GL/gl.h>
#include <GL/glut.h>
#include <math.h>
#include <iostream>

#define MY_WIN_WIDTH 200
#define MY_WIN_HEIGHT 200
#define MY_WIN_POS_X 100
#define MY_WIN_POS_Y 100
#define MY_WIN_TITLE "Tic-Tac-Toe"

#define PLAYER_X 'X'
#define PLAYER_O 'O'

/*
 * This is ta GameBoard Tiles Structure.
 * Every classic Tic-Tac-Toe has a 3x3 board.
 * The win objective is for a player to score
 * 3 of his symbol horrizondal or vertical or diagonal.
 */
struct Tiles {
    float X_min;
    float Y_min;

    float X_max;
    float Y_max;

    char ID;
    bool isFree;
    bool isDraw;
};

/*
 * Global Variables
 */
Tiles tiles[9]; //Tiles
bool O_turn = true; //Check whose turn is
float width; //keep the current width of the window
float height; //keep the current height of the window
const float DEG2RAD = 3.14159265/180; //transform DEGREE to RAD
bool winGame = false; //set the winGame bool value
bool drawGame = false; //set the drawGame bool value
char winPlayer = '\0'; //set win Player symbol

/*
 * Set the values for a tile
 */
Tiles setTiles(float x_min, float y_min, float x_max, float y_max, bool freeState, bool
drawState, char id = '\0') {
    Tiles t;

    t.X_min = x_min;
    t.Y_min = y_min;
    t.X_max = x_max;
    t.Y_max = y_max;
    t.isFree = freeState;
    t.isDraw = drawState;
    t.ID = id;

    return t;
}

```

```

/*
 * Initialize the starting tiles values. When the game needs to be restart this function must
 * be called
 */
*
* 6 | 7 | 8
* ---|---|---
* 3 | 4 | 5
* ---|---|---
* 0 | 1 | 2
*/
void initializeTiles(void) {
    tiles[0] = setTiles(0.0, 0.0, 1.0, 1.0, true, false);
    tiles[1] = setTiles(1.0, 0.0, 2.0, 1.0, true, false);
    tiles[2] = setTiles(2.0, 0.0, 3.0, 1.0, true, false);

    tiles[3] = setTiles(0.0, 1.0, 1.0, 2.0, true, false);
    tiles[4] = setTiles(1.0, 1.0, 2.0, 2.0, true, false);
    tiles[5] = setTiles(2.0, 1.0, 3.0, 2.0, true, false);

    tiles[6] = setTiles(0.0, 2.0, 1.0, 3.0, true, false);
    tiles[7] = setTiles(1.0, 2.0, 2.0, 3.0, true, false);
    tiles[8] = setTiles(2.0, 2.0, 3.0, 3.0, true, false);
}

/*
 * The init(void) function.
 * This function runs when the program start and set the
 * default clearing values.
 */
void init(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glShadeModel(GL_FLAT);
}

/*
 * The reshape(int w, int h) function.
 * This function runs when the window is reshaped.
 */
void reshape(int w, int h) {
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 3.0, 0.0, 3.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    width = w;
    height = h;
}

void grid(void) {
    glLineWidth(5.0);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
        glVertex3f(0.0, 1.0, 0.0);
        glVertex3f(3.0, 1.0, 0.0);

        glVertex3f(0.0, 2.0, 0.0);
        glVertex3f(3.0, 2.0, 0.0);

        glVertex3f(1.0, 0.0, 0.0);
        glVertex3f(1.0, 3.0, 0.0);

        glVertex3f(2.0, 0.0, 0.0);
        glVertex3f(2.0, 3.0, 0.0);
    glEnd();
}

```

```

/*
 * draw the symbol for X player.
 */
void drawX(float x_min, float y_min, float x_max, float y_max) {
    glLineWidth(3.0);
    glColor3f(0.5, 0.0, 0.0);
    glBegin(GL_LINES);
        glVertex3f(x_min + 0.2, y_min + 0.2, 0.0);
        glVertex3f(x_max - 0.2, y_max - 0.2, 0.0);

        glVertex3f(x_min + 0.2, y_max - 0.2, 0.0);
        glVertex3f(x_max - 0.2, y_min + 0.2, 0.0);
    glEnd();
}

/*
 * draw the symbol for O player.
 */
void drawO(float x_min, float y_min, float x_max, float y_max) {

    glLineWidth(2.0);
    glColor3f(0.0, 0.0, 0.5);
    float x_circle = x_min + (x_max-x_min)/2;
    float y_circle = y_min + (y_max-y_min)/2;
    glBegin(GL_LINE_LOOP);

        for (int i=0; i < 360; i++)
        {
            float degInRad = i*DEG2RAD;
            glVertex2f(x_circle + cos(degInRad)*0.4, y_circle + sin(degInRad)*0.4);
        }

    glEnd();
}

/*
 * Check if three char values are the same.
 */
bool checkSameChar(char c1, char c2, char c3) {
    if ((c1 != '\0' || c2 != '\0' || c3 != '\0') && c1 == c2 && c2 == c3) {
        winPlayer = c1;
        return true;
    }
    return false;
}

/*
 * Check if someone Wonned
 */
bool checkWinCondition(void) {
    if(checkSameChar(tiles[0].ID, tiles[1].ID, tiles[2].ID) || checkSameChar(tiles[0].ID,
tiles[3].ID, tiles[6].ID)
        || checkSameChar(tiles[0].ID, tiles[4].ID, tiles[8].ID) ||
checkSameChar(tiles[1].ID, tiles[4].ID, tiles[7].ID)
        || checkSameChar(tiles[2].ID, tiles[5].ID, tiles[8].ID) ||
checkSameChar(tiles[2].ID, tiles[4].ID, tiles[6].ID)
        || checkSameChar(tiles[3].ID, tiles[4].ID, tiles[5].ID) ||
checkSameChar(tiles[6].ID, tiles[7].ID, tiles[8].ID))
        return true;
    return false;
}

```



```

/*
 * Check if game is Drawn
 */
bool checkDrawCondition(void) {
    int counter = 0;
    for(int i = 0; i < 9; i++) {
        if(!tiles[i].isFree) {
            counter++;
        }
    }
    if(counter == 9)
        return true;
    return false;
}

/*
 * The display(void) function.
 * This function draw the graphics to screen.
 */
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    grid();

    for(int i = 0; i < 9; i++) {
        if(tiles[i].isDraw) {
            if(tiles[i].ID == PLAYER_O) {
                drawO(tiles[i].X_min, tiles[i].Y_min, tiles[i].X_max, tiles[i].Y_max);
            } else if(tiles[i].ID == PLAYER_X) {
                drawX(tiles[i].X_min, tiles[i].Y_min, tiles[i].X_max, tiles[i].Y_max);
            }
        }
    }

    glutSwapBuffers();

    if(winGame) {
        std::cout << "The player " << winPlayer << " won the game" << std::endl;
    } else if(drawGame) {
        std::cout << "The game is draw! Well done to both of you!" << std::endl;
    }
}

/*
 * Initialize GLUT window features.
 */
void initializeGL(void) {
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(MY_WIN_WIDTH, MY_WIN_HEIGHT);
    glutInitWindowPosition(MY_WIN_POS_X, MY_WIN_POS_Y);
    glutCreateWindow(MY_WIN_TITLE);
}

/*
 * Check if a number is between a min and a max value.
 */
bool NumBetweenEqual(float num, float min, float max) {
    if(min <= num && max >= num)
        return true;
    return false;
}

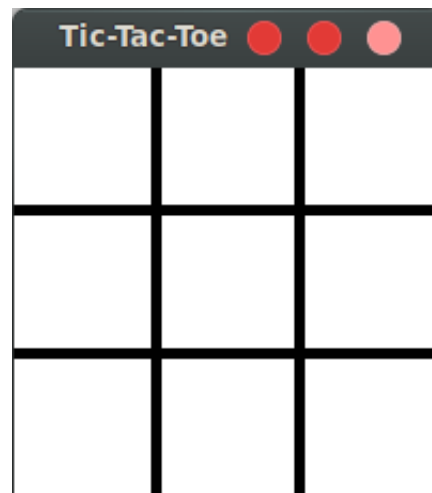
```

```
/*
 * Check which tiles must be drawn and set tiles symbol.
 */
void checkTile(float x, float y) {
    for(int i = 0; i < 9; i++) {
        if(NumBetweenEqual(x*3.0, tiles[i].X_min, tiles[i].X_max) && NumBetweenEqual(3-y*3.0,
tiles[i].Y_min, tiles[i].Y_max)) {
            if(tiles[i].isFree) {
                tiles[i].isFree = false;
                tiles[i].isDraw = true;
                if(O_turn) {
                    tiles[i].ID = PLAYER_O;
                    O_turn = false;
                } else {
                    tiles[i].ID = PLAYER_X;
                    O_turn = true;
                }
            }
            winGame = checkWinCondition();
            if(!winGame)
                drawGame = checkDrawCondition();
            glutPostRedisplay();
            break;
        }
    }
}

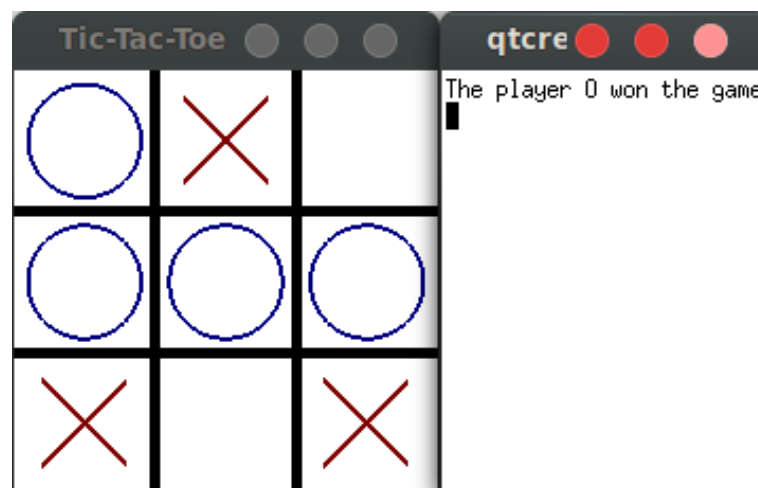
/*
 * Check mouse input events.
 */
void mouse(int button, int state, int x, int y) {
    switch(button) {
        case GLUT_LEFT_BUTTON: {
            if(state == GLUT_DOWN)
                if(!winGame && !drawGame)
                    checkTile((float)x/width, (float)y/height);
            break;
        } default:
            break;
    }
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    initializeGL();
    initializeTiles();

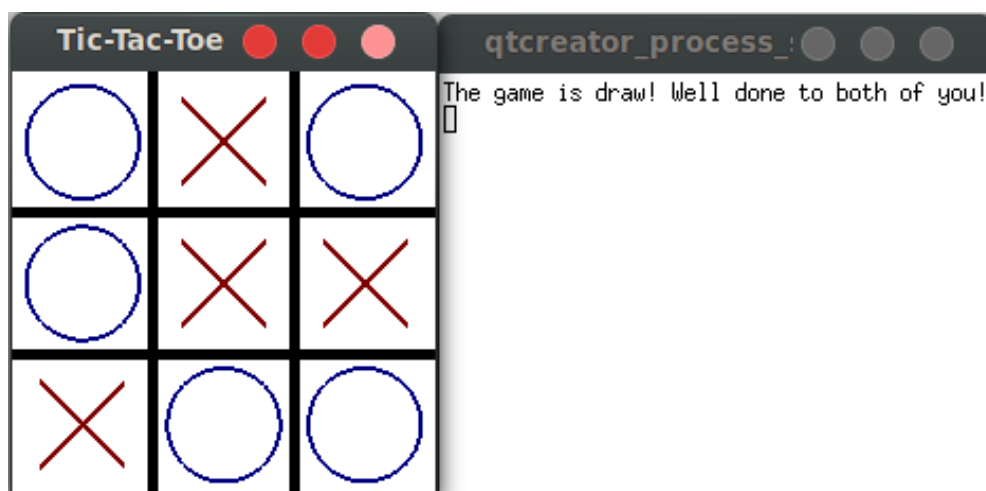
    init();
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}
```



Σχήμα 4.2: Αρχικός Πίνακας



Σχήμα 4.3: Σενάριο Νίκης



Σχήμα 4.4: Σενάριο Ισοπαλίας

4.3 Tic-Tac-Toe σε περιβάλλον QtCreator

```
mainwindow.h 1

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStatusBar>
#include <QTimer>
#include <QLCDNumber>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void checkWinDrawConditions();

    void on_resetBoard_clicked();

private:
    Ui::MainWindow *ui;

    QTimer timer;
    int WinCounter_0;
    int WinCounter_X;
    bool isOn;
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include "tictactoe.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->StatusBar->showMessage("Player 0 plays First", 2000);

    connect(&timer, SIGNAL(timeout()), this, SLOT(checkWinDrawConditions()));
    timer.start(100);

    WinCounter_0 = 0;
    WinCounter_X = 0;

    ui->OWins_LCD->display(WinCounter_0);
    ui->OWins_LCD->setPalette(Qt::blue);

    ui->XWins_LCD->display(WinCounter_X);
    ui->XWins_LCD->setPalette(Qt::red);

    isOn = true;
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::checkWinDrawConditions() {
    if(ui->TicTacToeWidget->isWin()) {
        if(ui->TicTacToeWidget->WinPlayer() == PLAYER_0 && isOn) {
            WinCounter_0++;
            ui->StatusBar->showMessage("Player 0 win!", 2000);
            ui->OWins_LCD->display(WinCounter_0);
            isOn = false;
        } else if(ui->TicTacToeWidget->WinPlayer() == PLAYER_X && isOn) {
            WinCounter_X++;
            ui->StatusBar->showMessage("Player X win!", 2000);
            ui->XWins_LCD->display(WinCounter_X);
            isOn = false;
        }
    } else if(ui->TicTacToeWidget->isDraw() && isOn) {
        ui->StatusBar->showMessage("Congratulations both of you! The game is DRAWN!", 2000);
        isOn = false;
    }
}

void MainWindow::on_resetBoard_clicked()
{
    ui->TicTacToeWidget->resetGame();
    ui->StatusBar->showMessage("Player 0 plays First", 2000);
    isOn = true;
}
```

```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <QGLWidget>
#include <QMouseEvent>
#include <QTimer>

#include "tictactoe.h"

class GLWidget : public QGLWidget
{
public:
    GLWidget(QWidget *parent);

    void initializeGL(void); //init(void)

    void resizeGL(int w, int h); //reshape(int h, int w)

    void paintGL(void); //display(void)

    void mousePressEvent(QMouseEvent* event); //Check Mouse Events

    void resetGame(void) {game.resetBoard();}

    bool isWin(void) {return game.IsWin();}

    bool isDraw(void) {return game.IsDraw();}

    char WinPlayer(void) {return game.WinPlayer();}

private:
    QTimer glTimer;
    TicTacToe game;
};

#endif // GLWIDGET_H
```

```

#include "glwidget.h"

GLWidget::GLWidget(QWidget* parent) :
    QGLWidget(parent)
{
    connect(&glTimer, SIGNAL(timeout()), this, SLOT(updateGL()));
    glTimer.start(30);

    resetGame();
}

/* Same as
 * void init(void){...}
 * function.
 */
void GLWidget::initializeGL(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
}

/* Same as
 * void reshape(int w, int h){...}
 * function. There isn't any need to call
 * glutReshapeFunc(...)
 * to run this function.
 */
void GLWidget::resizeGL(int w, int h) {
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, game.boardWidth(), 0.0, game.boardHeight(), -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

/* Same as
 * void display(void){...}
 * function. There isn't any need to call
 * glutDisplayFunc(...)
 * to run this function.
 */
void GLWidget::paintGL(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    game.drawGrid();
    game.drawTiles();
}

//Mouse Event Handling on QT
void GLWidget::mousePressEvent(QMouseEvent* event) {
    switch (event->button()) {
        case Qt::LeftButton:
            if(!game.IsWin() && !game.IsDraw())
                game.checkTileToDraw((float)event->x()/width(), (float)(height() - event->y())/
height());
            break;
        default:
            break;
    }
}

```

```

#ifndef TICTACTOE_H
#define TICTACTOE_H

#define PLAYER_X 'X'
#define PLAYER_O 'O'
#define DEFAULT_SYMBOL '\0'

class Tile
{
public:
    void setTile(float x_min, float y_min, float x_max, float y_max, char sym =
DEFAULT_SYMBOL);

    inline float minX(void) {return Xmin;}
    inline float minY(void) {return Ymin;}
    inline float maxX(void) {return Xmax;}
    inline float maxY(void) {return Ymax;}
    inline char Symbol(void) {return symbol;}
    inline void setSymbol(char sym) {symbol = sym;}

private:
    float Xmin; //minimum X tile coordinate
    float Ymin; //minimum Y tile coordinate
    float Xmax; //maximum X tile coordinate
    float Ymax; //maximum Y tile coordinate

    char symbol; //the symbol of the player 'O' or 'X' if not free or '\0' if free.
};

class TicTacToe
{
public:
    TicTacToe(void);

    void resetBoard(void);
    void drawGrid(void);
    void drawTiles(void);
    void drawSymbolX(float x_min, float y_min, float x_max, float y_max);
    void drawSymbolO(float x_min, float y_min, float x_max, float y_max);

    float boardWidth(void) {return maxBoardWidth;}
    float boardHeight(void) {return maxBoardHeight;}

    inline bool IsWin(void) {return isWin;}
    inline bool IsDraw(void) {return isDraw;}
    inline char WinPlayer(void) {return winPlayer;}

    void checkTileToDraw(float x, float y);

private:
    Tile boardTiles[9];

    float maxBoardWidth;
    float maxBoardHeight;

    bool isWin;
    bool isDraw;
    bool O_plays;

    char winPlayer;

    bool checkSameChar(char c1, char c2, char c3);
    bool checkWinCondition(void);
    bool checkDrawCondition(void);
    bool numBetweenOrEqualToMin(float num, float min, float max);
};

#endif // TICTACTOE_H

```



```

#include "tictactoe.h"
#include "GL/gl.h"

#include "math.h"

const float DEG2RAD = 3.14159265/180;

void Tile::setTile(float x_min, float y_min, float x_max, float y_max, char sym) {
    Xmin = x_min;
    Ymin = y_min;
    Xmax = x_max;
    Ymax = y_max;
    symbol = sym;
}

TicTacToe::TicTacToe(void)
{
    maxBoardWidth = 3.0;
    maxBoardHeight = 3.0;

    isWin = false;
    isDraw = false;
    O_plays = true;
}

/*
 * Reset the board tiles when the game needs to be called.
 *
 * 6 | 7 | 8
 * ---|---|---
 * 3 | 4 | 5
 * ---|---|---
 * 0 | 1 | 2
 *
 */
void TicTacToe::resetBoard(void) {

    /*
     * boardTiles[0].setTile(0.0, 0.0, 1.0, 1.0);
     * boardTiles[1].setTile(1.0, 0.0, 2.0, 1.0);
     * boardTiles[2].setTile(2.0, 0.0, 3.0, 1.0);
     *
     * boardTiles[3].setTile(0.0, 1.0, 1.0, 2.0);
     * boardTiles[4].setTile(1.0, 1.0, 2.0, 2.0);
     * boardTiles[5].setTile(2.0, 1.0, 3.0, 2.0);
     *
     * boardTiles[6].setTile(0.0, 2.0, 1.0, 3.0);
     * boardTiles[7].setTile(1.0, 2.0, 2.0, 3.0);
     * boardTiles[8].setTile(2.0, 2.0, 3.0, 3.0);
     */
    int index = 0;
    for(int y = 0; y < maxBoardHeight; y++)
        for(int x = 0; x < maxBoardWidth; x++)
            boardTiles[index++].setTile(x, y, x+1, y+1);

    isWin = false;
    isDraw = false;
    O_plays = true;
}

/*
 * Draw GameBoard Grid
 */
void TicTacToe::drawGrid(void) {
    glLineWidth(5.0);
    glColor3f(0.9, 0.9, 0.9);

```

```

    glBegin(GL_LINES);
    for(int y = 1; y < maxBoardHeight; y++) {
        glVertex2f(0.0, y);
        glVertex2f(maxBoardWidth, y);
    }
    for(int x = 1; x < maxBoardWidth; x++) {
        glVertex2f(x, 0.0);
        glVertex2f(x, maxBoardHeight);
    }
    glEnd();
}

void TicTacToe::drawTiles(void) {
    for(int i = 0; i < maxBoardWidth*maxBoardHeight; i++) {
        if(boardTiles[i].Symbol() != DEFAULT_SYMBOL) {
            if(boardTiles[i].Symbol() == PLAYER_0)
                drawSymbolO(boardTiles[i].minX(), boardTiles[i].minY(), boardTiles[i].maxX(),
boardTiles[i].maxY());
            else if(boardTiles[i].Symbol() == PLAYER_X)
                drawSymbolX(boardTiles[i].minX(), boardTiles[i].minY(), boardTiles[i].maxX(),
boardTiles[i].maxY());
        }
    }
}

/*
 * Draw 'O' Symbol
 */
void TicTacToe::drawSymbolO(float x_min, float y_min, float x_max, float y_max) {
    glLineWidth(2.0);
    glColor3f(0.0, 0.0, 1.0);
    float x_circle= x_min + (x_max - x_min)/2;
    float y_circle= y_min + (y_max - y_min)/2;

    glBegin(GL_LINE_LOOP);
    for(int i = 0; i < 360; i++) {
        float degInRad = i * DEG2RAD;
        glVertex2f(x_circle + cos(degInRad) * 0.4, y_circle + sin(degInRad) * 0.4);
    }
    glEnd();
}

/*
 * Draw 'X' Symbol
 */
void TicTacToe::drawSymbolX(float x_min, float y_min, float x_max, float y_max) {
    glLineWidth(3.0);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
        glVertex2f(x_min + 0.2, y_min + 0.2);
        glVertex2f(x_max - 0.2, y_max - 0.2);

        glVertex2f(x_min + 0.2, y_max - 0.2);
        glVertex2f(x_max - 0.2, y_min + 0.2);
    glEnd();
}

/*
 * Check if three char values are the same.
 */
bool TicTacToe::checkSameChar(char c1, char c2, char c3) {
    if((c1 != DEFAULT_SYMBOL && c2 != DEFAULT_SYMBOL && c3 != DEFAULT_SYMBOL) && (c1 == c2 &&
c2 == c3)) {
        winPlayer = c1;
        return true;
    }
    return false;
}

```

```

}

/*
 * Check if the game has been won.
 */
bool TicTacToe::checkWinCondition(void) {
    if(checkSameChar(boardTiles[0].Symbol(), boardTiles[1].Symbol(), boardTiles[2].Symbol())
    || checkSameChar(boardTiles[0].Symbol(), boardTiles[4].Symbol(), boardTiles[8].Symbol())
    || checkSameChar(boardTiles[0].Symbol(), boardTiles[3].Symbol(), boardTiles[6].Symbol())
    || checkSameChar(boardTiles[1].Symbol(), boardTiles[4].Symbol(), boardTiles[7].Symbol())
    || checkSameChar(boardTiles[2].Symbol(), boardTiles[4].Symbol(), boardTiles[6].Symbol())
    || checkSameChar(boardTiles[3].Symbol(), boardTiles[4].Symbol(), boardTiles[5].Symbol())
    || checkSameChar(boardTiles[6].Symbol(), boardTiles[7].Symbol(), boardTiles[8].Symbol()))
        return true;
    return false;
}

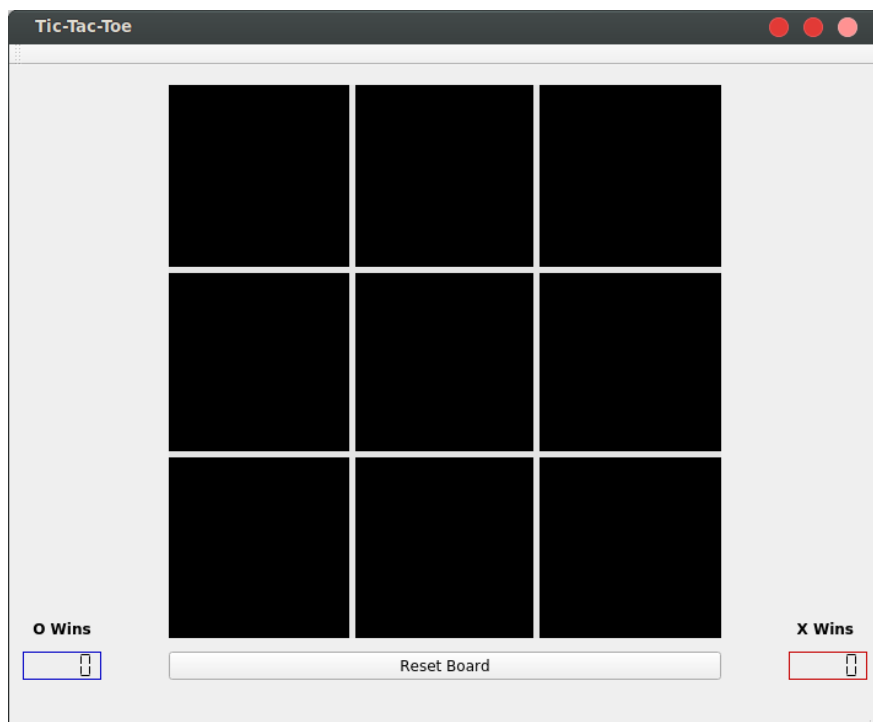
/*
 * Check if the game has been drawn.
 */
bool TicTacToe::checkDrawCondition(void) {
    for(int i = 0; i < maxBoardWidth*maxBoardHeight; i++) {
        if(boardTiles[i].Symbol() == DEFAULT_SYMBOL) {
            return false;
        }
    }
    return true;
}

/*
 * Check if a Number is higher or equal than a min value and lower than a max value
 */
bool TicTacToe::numBetweenOrEqualToMin(float num, float min, float max) {
    if(min <= num && max > num)
        return true;
    return false;
}

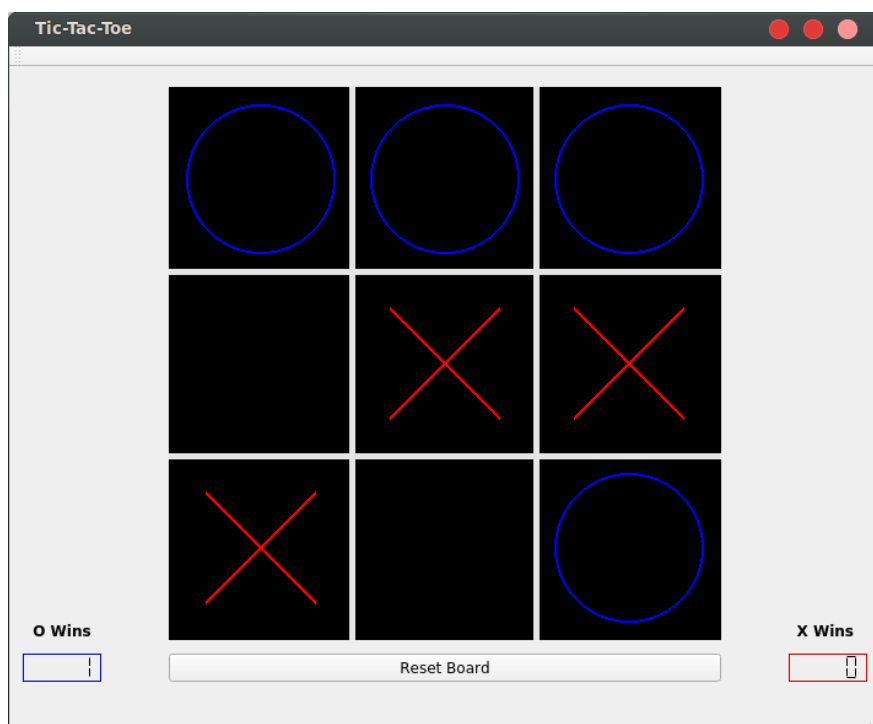
/*
 * Check which Tile must be drawn and set tile symbol.
 */
void TicTacToe::checkTileToDraw(float x, float y) {
    for(int i = 0; i < maxBoardWidth*maxBoardHeight; i++) {
        if(numBetweenOrEqualToMin(x * 3.0, boardTiles[i].minX(), boardTiles[i].maxX())
        && numBetweenOrEqualToMin(y * 3.0, boardTiles[i].minY(),
        boardTiles[i].maxY())) {

            if(boardTiles[i].Symbol() == DEFAULT_SYMBOL) {
                if(O_plays) {
                    boardTiles[i].setSymbol(PAYER_O);
                    O_plays = false;
                } else {
                    boardTiles[i].setSymbol(PAYER_X);
                    O_plays = true;
                }
            }
            break;
        }
    }
    isWin = checkWinCondition();
    if(!isWin)
        isDraw = checkDrawCondition();
}

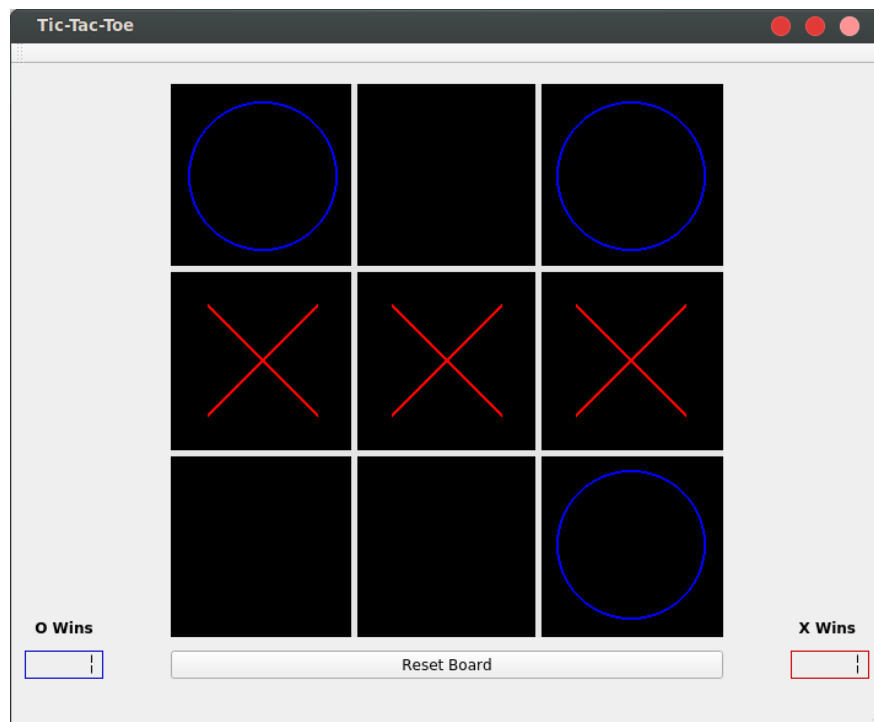
```



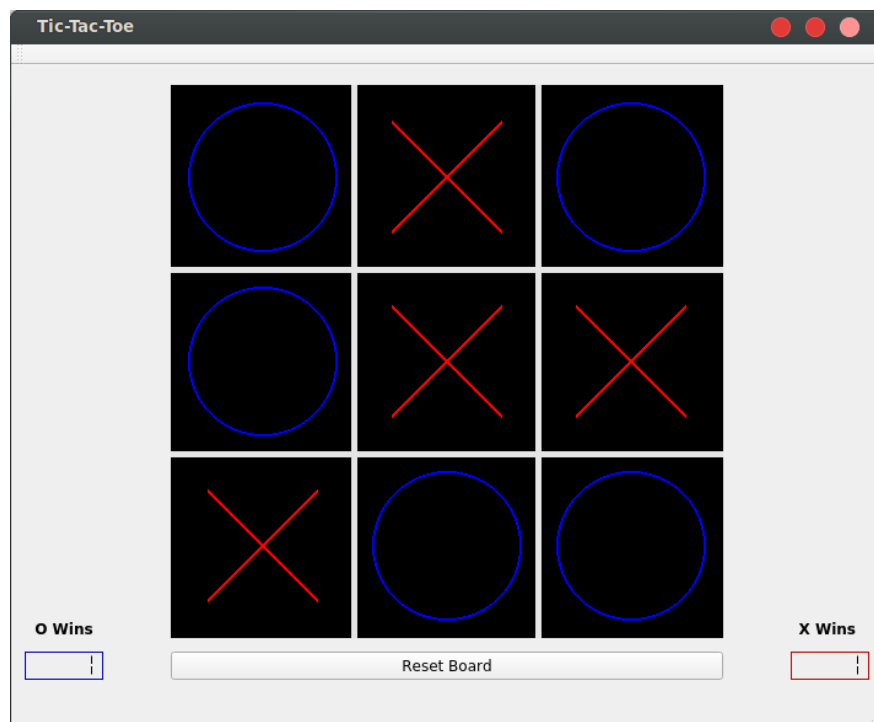
Σχήμα 4.5: Αρχικός Πίνακας



Σχήμα 4.6: Σενάριο Νίκης για τον Παίκτη O



Σχήμα 4.7: Σενάριο Νίκης για τον Παίχτη X



Σχήμα 4.8: Σενάριο Ισοπαλίας