**Computer Graphics (UCS505)**

**Project on**

# Car Driving Game


**Submitted By**

| | |
|---|---|
| Arun Gautam | 102103811 |
| Ojaswani | 102103417 |


**3CO-15**

**B.E. Third Year – CSE**

**Submitted To:**

**Dr. Jyoti Rani**



**Computer Science and Engineering Department**

**Thapar Institute of Engineering and Technology**

**Patiala – 147001**

## Table of Contents

# INTRODUCTION

Welcome to our project on making a car driving game using OpenGL! In this project, we're diving into the exciting world of computer graphics, where we'll create a cool game that lets you drive cars in a virtual environment.

We're using OpenGL, which is like a toolbox for making awesome graphics on computers. It helps us make things look realistic, like the cars, the roads, and the scenery around them. With OpenGL, we can make our game work on different types of computers, which is really handy.

But making a game isn't just about making things look pretty – it's also about making it fun to play! So, we'll be working on making the cars handle realistically, so they feel like real cars when you drive them. We'll also add cool features like challenging roads to drive on and maybe even some other cars to race against or obstacles to avoid.

We want our game to be exciting and immersive, so we're putting a lot of effort into making it feel like a real adventure. Whether you're cruising down a highway or tackling a tough off-road course, we want you to feel like you're right there in the driver's seat, experiencing all the thrills and challenges of the road.

So, get ready to hit the gas and join us on this journey into the world of OpenGL car driving game development – where the possibilities are endless, and the fun never stops.

## Computer Graphics Concept Used

1. **OpenGL:** The code uses OpenGL for rendering 2D graphics. OpenGL is a widely-used graphics library that provides functions for rendering geometric shapes, applying transformations, and managing the rendering pipeline.
2. **GLUT (OpenGL Utility Toolkit):** GLUT is used to create a window, handle user input, and manage events in the OpenGL application. It simplifies the process of setting up a graphical user interface.
3. **Geometric Transformation:** Geometric transformations like translation and scaling are used to position and resize objects in the scene. The glTranslatef() function is used to translate objects, while scaling is achieved by adjusting the vertex coordinates.
4. **Primitive Shapes:** Basic primitive shapes like polygons and rectangles are used to construct objects such as houses, trees, and the river. These shapes serve as building blocks for creating more complex structures.
5. **Color:** Different colors are used to paint objects in the scene. Colors are specified using RGB values with the glColor3ub() function.
6. **Animation:** Animation is achieved by updating the position of certain elements over time. For example, the clouds and the boat are animated by changing their position in each frame using the glTranslatef() function.
7. **Interactivity:** The scene is made interactive by allowing the user to switch between Restarting the game using keyboard input. This is accomplished by registering a keyboard callback function with GLUT and updating the scene based on user input.
8. **Rendering Optimization:** Basic rendering optimizations are employed, such as clearing the color buffer before rendering each frame (glClear()), and using glutPostRedisplay() to trigger redrawing only when necessary

## User Defined Function

1. **Anim**: This function is likely the main animation loop of the program. It controls the overall animation of the game, coordinating the movement of various elements such as the car, obstacles, power-ups, and background.

2. **animatePowerup**s: This function handles the animation of power-up items in the game. It may control their movement, appearance, and disappearance based on game logic or player interactions.

3. **animateObstacles**: Similar to `animatePowerups`, this function manages the animation of obstacles in the game. It likely controls their movement and behavior, such as obstacles moving towards the player or appearing at certain intervals.

4. **animateRegisteredData**: This function seems to animate some registered data based on the ticks passed. It might handle animations related to game data or state changes that occur over time.

5. **animateBackground**: This function is responsible for animating the background of the game. It could involve scrolling backgrounds, changing scenery, or other effects to create a dynamic visual experience.

6. **updateScore**: This function updates the score display based on the player's performance or game events. It likely calculates the score and then updates the score display on the screen.

7. **drawTheCa**r: This function draws the player's car on the screen. It handles rendering the car model or sprite and placing it at the correct position on the screen.

8. **drawObstacles**: This function draws obstacles on the screen. It likely iterates through a list of obstacles and renders each one at its current position.

9. **drawPowerups**: Similar to `drawObstacles`, this function draws power-up items on the screen. It renders each power-up at its current position, possibly with different visual effects depending on the type of power-up.

10. **drawStatusPart**: This function likely draws various status indicators or UI elements on the screen, such as health bars, score displays, and acceleration bars.

11. **moveCar**: This function handles the movement of the player's car in response to input. It likely adjusts the car's position based on user input, such as arrow key presses or touchscreen gestures.

12. **processSpecialKeys** and **processNormalKeys**: These functions likely handle keyboard or input device events. They might interpret key presses or mouse clicks and trigger appropriate actions in the game.

13. **initConfig** and **initValues**: These functions likely initialize configuration settings and game variables, respectively, at the start of the game.

Line Drawing Algorithms:

Bresenham's Line Algorithm: This algorithm efficiently determines which points on a grid should be plotted to form a straight line between two given points. It is widely used for rendering lines in computer graphics due to its simplicity and efficiency.

Digital Differential Analyzer (DDA): Another algorithm for drawing lines, DDA calculates the incremental changes in x and y coordinates to plot pixels along the line. While simpler than Bresenham's algorithm, DDA may be less efficient due to floating-point calculations.

Circle Drawing Algorithms:

Midpoint Circle Algorithm: This algorithm efficiently draws circles by plotting points along the circumference based on a midpoint criterion. It is faster than using trigonometric functions for circle drawing.

Bresenham's Circle Algorithm: Similar to Bresenham's Line Algorithm, this approach plots points along the circumference of a circle using integer arithmetic, making it efficient for raster displays.

Polygon Filling Algorithms:

Scanline Fill Algorithm: This algorithm fills polygons by scanning horizontal lines across the polygon and determining intersections with its edges. It then fills the enclosed regions between pairs of intersections.

Flood Fill Algorithm: Flood fill is a recursive algorithm used for filling bounded areas with a given color. It starts at a seed point and recursively fills neighboring pixels until a boundary is reached.

Transformation Algorithms:

Translation: Moving objects from one position to another can be achieved using simple translation matrices. OpenGL provides functions like glTranslatef() to perform translations.

Rotation: Rotating objects around a point involves applying rotation matrices. Functions like glRotatef() in OpenGL can be used to rotate objects around specified axes.

Scaling: Scaling objects involves resizing them along different axes. OpenGL provides glScalef() for scaling objects uniformly or non-uniformly along x, y, and z axes.

Clipping Algorithms:
Cohen-Sutherland Algorithm: This algorithm is used for line clipping against a rectangular clipping window. It classifies line segments into different regions and clips them against the window boundaries efficiently.
Sutherland-Hodgman Algorithm: Used for polygon clipping against an arbitrary clipping polygon, this algorithm clips polygons against each edge of the clipping window successively.

# Code

```
#include "pch.h"
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <iostream>
#include <windows.h>
#include <mmsystem.h>
using namespace std;
#include <GL/glut.h>
#include "GameObject.h"

/* Structs Definitions */
struct Bg {
        bool  on;
        float r = 1.0f;
        float g = 0.1f;
        float b = 0.1f;
        float acc_r = 0.001f;
        float acc_b = 0.001f;
        float acc_g = 0.001f;
        float delta_r = 0.01;
        float delta_b = 0.02;
        float gradient_rate = 1 / 40;
} bg;

struct Car {
        double x_pos;
        double y_pos;
        double height;
        double width;
        double hp;
        bool alive;
        int score;
        float acceleration;
        float registered_acc;
        double registered_hp;
        double registered_x;
        double registered_theta;
} car;

/*      Methods Signatures */
void Display(void);
void Anim(void);
void animatePowerups(void);
```

```
void animateObstacles(void);
void animateRegisteredData(int ticks_passed);
void animateBackground(void);
void updateScore(void);
// --- Drawers
void drawTheCar(void);
void drawObstacles(void);
void drawWoodObstacle(Obstacle*);
void drawPowerups(void);
void drawPowerUpStrike(Powerup*);
void drawPowerUpBottle(Powerup*);
void drawStatusPart(void);
void drawHpBar(void);
void drawScoreBar(void);
void drawAccBar(void);
void drawRoadLine(bool, double);
void drawRoadLines(void);
void drawRoadPlane(void);
void drawBackground(void);
void drawCircle(double x, double y, float r);
void print(int, int, char *);
// --- Collision Detectors
bool carHit(Obstacle*);
bool carHit(Powerup*);
// --- Key Processors
void processSpecialKeys(int, int, int);
void processNormalKeys(unsigned char, int, int);
// --- Movers
void moveCar(int delta_x, int delta_y);
void moveCarRegistered(int ticks_passed);
// --- Debugging Functions
void debugScreenLines(int x_splits, int y_splits);
void debugObstaclesOutOfRange(void);
// --- Other Helper Functions
double getLane(int);
void initConfig(void);
void initValues(void);
int getRandomValue(int m_v, int u_v);
//.....
```

```
/*      Constants & Intial Values */
int WIN_W = 1080;
int WIN_H = 720;
float GLOBAL_SLOPE = -5;
double ROAD_START_Y_INIT = 20;
double BACKGROUND_START_Y_INIT = WIN_H;
double SLIGHTLY_OFF_TOP_SCREEN = 5 * WIN_H / 4;
```

```
//const double SLIGHTLY_OFF_BOTTOM_SCREEN = - (WIN_H / 2);
bool DEBUGING_MODE = false;
bool VERBOSE = false;
// Speed Constraints
float DELTA_VELOCITY = 0.05;
float MAX_ACC = 1.5f;
float MIN_ACC = 0.5f;
float MAX_HP = 100;
// Global variables
double road_start_y;
double background_start_y;
int num_of_on_obstacles;
int num_of_on_powerups;
time_t past_time_scored;
int past_tick;
//.....

void Display(void)
{
	//glClearColor(0.1, 1, 1, 0.4f); // update the background color
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glPushMatrix(); // marking the initial state of the transformations matrix
	/* Drawing the background */
	drawBackground();
	/* Drawing the track */
	drawRoadPlane();
	drawRoadLines();
	/* Drawing Power-ups */
	drawPowerups();
	/* Drawing rock obstacles */
	drawObstacles();
	/* Drawing the CAR OUTLINE*/
	drawTheCar();
	/* Draw GAME-STATUS PRINTS*/
	drawStatusPart();
	drawHpBar();
	drawScoreBar();
	drawAccBar();
	if(DEBUGING_MODE)
		debugScreenLines(22, 22);

	glPopMatrix(); //	returning to the initial state of the transformations matrix
	//glutSwapBuffers();
	glFlush();
}

void Anim()
```

```
{
	int t = glutGet(GLUT_ELAPSED_TIME);
	int passed_ticks = t - past_tick;
	past_tick = t;

	if (!car.alive) {
		sndPlaySound(TEXT("C:\\Sound Effects\\car_explosion.wav"), SND_FILENAME |
SND_ASYNC);

		glutIdleFunc(NULL);

		glutPostRedisplay();

		return;
	}

	if (past_tick > 1) {
		if (road_start_y > -9 * WIN_H / 16)
		{
			road_start_y -= car.acceleration;
		}
		else
		{
			road_start_y = ROAD_START_Y_INIT;
		}


		//debugObstaclesOutOfRange();

		animateBackground();
		animatePowerups();
		animateObstacles();
		animateRegisteredData(passed_ticks);
		updateScore();


		// Generations
		Powerup pu = powerups[powerups_counter];
		Obstacle ob = obstacles[obstacles_counter];

		if (pu.current_down_y < WIN_H && ob.current_down_y < WIN_H)
		{
			if (2 * num_of_on_powerups >= num_of_on_obstacles)
			{
				generateRandomObstacle();
			}
			else{
```

```
                        generateRandomPowerup();
                        //generateRandomObject();
                    }
                }
            }
            glutPostRedisplay();

}

void animateRegisteredData(int ticks_passed) {

            moveCarRegistered(ticks_passed);

            if (car.registered_hp != 0)
            {
                if (car.hp <= 0.1)
                {
                    car.hp = 0;
                    car.registered_hp = 0;
                    car.alive = false;
                }
                float delta_hp = 0.2;
                car.hp += (car.registered_hp > 0) ? delta_hp : -delta_hp;
                car.registered_hp += (car.registered_hp > 0) ? -delta_hp : delta_hp;

                if (car.hp >= MAX_HP)
                {
                    car.hp = MAX_HP;
                    car.registered_hp = 0;
                }

                if (car.hp <= 0)
                {
                    car.hp = 0;
                    car.registered_hp = 0;
                    car.alive = false;
                }



            }
            //printf("car hp = %f\n", car.hp);
}

void moveCarRegistered(int ticks_passed) {

            //printf("ticks_elapsed = %d\n", ticks_passed);
```

```
	double car_road_margin = WIN_W / 32;
	double car_x_pos_tmp = car.x_pos;

	if (car.registered_x == 0) {
		if (car.registered_theta < 1 && car.registered_theta > -1) {
			car.registered_theta = 0;
		}

		if (car.registered_theta > 0)
			car.registered_theta--;
		else if (car.registered_theta < 0) {
			car.registered_theta++;
		}
		return;
	}

	//car.x_pos += car.registered_x;
	//car.registered_x = 0;
	if (ticks_passed != 0) {
		float delta_x = car.registered_x* (1 - (1 / ticks_passed))*0.4;
		car.x_pos += delta_x;
		car.registered_x -= delta_x;
		if (car.registered_x < 1 && car.registered_x > -1) {
			car.registered_x = 0;
		}
	}

	if (car.x_pos < (getLane(1) - car_road_margin) ||
		car.x_pos >(getLane(3) + car_road_margin)) {
		car.x_pos = car_x_pos_tmp;
		car.registered_x = 0;
	}
	//printf("\n x_pos = %f, getLane(1) - car_road_margin = %f", car_x_pos, (getLane(3) +
car_road_margin));

}

void animateBackground() {

	//bg.r += car.acceleration/40;
	bg.r -= car.acceleration*(bg.acc_r);
	if (bg.r >= 0.7 || bg.r <= 0.1) {
		bg.acc_r = -bg.acc_r;
	}

	bg.g += car.acceleration*(bg.acc_g);
```

```
	if (bg.g >= 0.4 || bg.g <= 0.1) {
		bg.acc_g = -bg.acc_g;
	}

	bg.b += car.acceleration*(bg.acc_b);
	if (bg.b >= 0.5 || bg.b <= 0.1) {
		bg.acc_b = -bg.acc_b;
	}

}

void accelerateCar(float amount) {
	car.acceleration += amount;
	if (car.acceleration < MIN_ACC) {
		car.acceleration = MIN_ACC;
	}
	if (car.acceleration > MAX_ACC) {
		car.acceleration = MAX_ACC;
	}
}

void animatePowerups() {
	double SLIGHTLY_OFF_BOTTOM_SCREEN = -(WIN_H / 2);
	//printf("\n\nSLIGHTLY_OFF_BOTTOM_SCREEN = %f",
SLIGHTLY_OFF_BOTTOM_SCREEN);
	//printf("\n\nSLIGHTLY_OFF_TOP_SCREEN = %f\n", SLIGHTLY_OFF_TOP_SCREEN);

	Powerup* pu;
	for (int pu_num = POWERUPS_MAX_INDEX; pu_num >= 0; pu_num--) {
		pu = &powerups[pu_num];

		if (pu->on && ((pu->current_down_y) > SLIGHTLY_OFF_BOTTOM_SCREEN))
		{
			pu->current_down_y -= car.acceleration;
			pu->theta++;
			if (pu->theta > 360) {
				pu->theta = 0;
			}
			if (carHit(pu)) {
				//sndPlaySound(TEXT("C:\\Sound Effects\\hockey_sound.wav"),
SND_FILENAME | SND_ASYNC);

				pu->on = false;
				if (pu->type == 1)
				{
					//printf("acc before = %f\n", car.registered_acc);
					accelerateCar((pu->hit)*MAX_ACC);
```

```
                                        //printf("acc after = %f\n", car.registered_acc);
                                }
                                else
                                {
                                        car.registered_hp += (pu->hit)*MAX_HP;
                                }
                        }


                }
                else
                {
                        //destroyPowerup(pu);
                        pu->on = false;
                        pu->id = -1;
                        num_of_on_powerups--;
                }
        }
}


void animateObstacles() {
        double SLIGHTLY_OFF_BOTTOM_SCREEN = -(WIN_H / 2);
        //printf("\n\nSLIGHTLY_OFF_BOTTOM_SCREEN                =                %f",
SLIGHTLY_OFF_BOTTOM_SCREEN);
        //printf("\n\nSLIGHTLY_OFF_TOP_SCREEN = %f\n", SLIGHTLY_OFF_TOP_SCREEN);

        Obstacle* ob;
        for (int ob_num = OBSTACLES_MAX_INDEX; ob_num >= 0; ob_num--) {
                ob = &obstacles[ob_num];

                if (ob->on && ((ob->current_down_y) > SLIGHTLY_OFF_BOTTOM_SCREEN))
                {
                        ob->current_down_y -= car.acceleration;
                        if (carHit(ob)) {
                                //sndPlaySound(TEXT("C:\\Sound      Effects\\hitting_metal_sound.wav"),
SND_FILENAME | SND_ASYNC);

                                ob->on = false;
                                //printf("acc before = %f\n", car.registered_acc);
                                car.registered_hp -= (ob->hit)*MAX_HP;
                                accelerateCar(-(ob->hit)*MAX_ACC);
                                //printf("acc after = %f\n", car.registered_acc);

                                        {

                }
                else
```

```
}


ob->on = false;
```

```
ob->id = -1; num_of_on_obstacles--;
                }
        }

 }

 void updateScore() {
        time_t current_time;
        time(&current_time);  /* get current time; same as: timer = time(NULL)  */
        double seconds = difftime(current_time, past_time_scored);

        if (seconds >= 3) {
                past_time_scored = current_time;
                car.score += (seconds / 3) * 5;
        }
 }

 /* Collision Detectors*/
 bool carHit(Powerup *ob)
 {
        double x = ob->current_down_x;
        double y = ob->current_down_y;
        double h = ob->height;
        double w = ob->width; double
        r_w = ob->real_width;

        float margin = h / 5;
        double ob_down_y = ob->current_down_y + margin;
        double ob_h = h;
        double ob_x_left = x - r_w / 2;
        double ob_x_right = x + r_w / 2;


        //For Debugging
        if (DEBUGING_MODE && VERBOSE) {
                char ob_id = ob->id;
                printf("\n\nPowerup ::\nx_id = %d\n", ob_id);
                printf("y axis ==== ob_down_y = %f, car.y_pos + car.height = %f \n", ob_down_y,
 car.y_pos + car.height);
                printf("y axis ==== ob_down_y + ob_h = %f, car_y_pos = %f \n", ob_down_y + ob_h,
 car.y_pos);
                printf("x axis ==== ob_x_left = %f, car.x_pos - car.width / 2 = %f \n", ob_x_left, car.x_pos
 - car.width / 2);
                printf("x axis ==== ob_x_right = %f, car.x_pos + car.width / 2 = %f \n", ob_x_right,
 car.x_pos + car.width / 2);
                printf("some vals ==== ob_x_left = %f, ob_x_right = %f, ob_current_width = %f\n",
```

```
                ob_x_left, ob_x_right, ob->real_width);
            printf("some vals ==== car.x_pos = %f, car.width = %f\n\n", car.x_pos, car.width);
        }


        if (ob_down_y <= car.y_pos + car.height &&
            ob_h + ob_down_y >= car.y_pos)
        {
            double car_x_left = car.x_pos - car.width / 2;
            double car_x_right = car.x_pos + car.width / 2;

            if ((car_x_left <= ob_x_left && car_x_right >= ob_x_left) ||
                (car_x_left <= ob_x_right && car_x_right >= ob_x_left)) {

                return true;
            }
        }

        return false;
}

bool carHit(Obstacle *ob)
{
        float margin = ob->height / 5;
        double ob_down_y = ob->current_down_y + margin;
        double ob_h = ob->height;
        double ob_x_left = ob->current_down_x - ob->real_width / 2;
        double ob_x_right = ob->current_down_x + ob->real_width / 2;

        //For Debugging
        /*
        if (DEBUGING_MODE && VERBOSE) {
            char ob_id = ob->id;
            printf("\n\nObstacle ::\nx_id = %d\n", ob_id);
            printf("y axis ==== ob_down_y = %f, car.y_pos + car.height = %f \n", ob_down_y,
car.y_pos + car.height);
            printf("y axis ==== ob_down_y + ob_h = %f, car_y_pos = %f \n", ob_down_y + ob_h,
car.y_pos);
            printf("x axis ==== ob_x_left = %f, car.x_pos - car.width / 2 = %f \n", ob_x_left, car.x_pos
- car.width / 2);
            printf("x axis ==== ob_x_right = %f, car.x_pos + car.width / 2 = %f \n", ob_x_right,
car.x_pos + car.width / 2);
            printf("some vals ==== ob_x_left = %f, ob_x_right = %f, ob_current_width = %f\n",
                ob_x_left, ob_x_right, ob->real_width);
            printf("some vals ==== car.x_pos = %f, car.width = %f\n\n", car.x_pos, car.width);
        }
        */
```

```
	if (ob_down_y <= car.y_pos + car.height &&
		ob_h + ob_down_y >= car.y_pos)
	{
		double car_x_left = car.x_pos - car.width / 2;
		double car_x_right = car.x_pos + car.width / 2;

		if ((car_x_left <= ob_x_left && car_x_right >= ob_x_left) ||
			(car_x_left <= ob_x_right && car_x_right >= ob_x_left)) {

			return true;
		}
	}

	return false;
}

/* Game Drawers */
void drawTheCar()
{
	glPushMatrix();
	glTranslated(car.x_pos, car.y_pos, 0);
	glRotated(car.registered_theta, 0, 0, 1);
	float x_scale = 6, y_scale = 6;
	glScaled(x_scale, y_scale, 1);
	car.height = 22 * y_scale;
	car.width = 22 * x_scale;

	glBegin(GL_QUAD_STRIP);
	float x, y;
	float r = 1.0f, g = 0.0f, b = 0.5f;
	glColor3f(r, g, b);
	x = 7.0f;
	for (y = 20; y >= 0; y -= 1) {
		if (int(y) % 5 == 0) {
			x = (x == 7.0f || x == 8.0f) ? 5.0f : 8.0f;
		}
		r -= 0.03;
		glColor3f(r, g, b);
		glVertex3f(x, y, 0.0f);
		glVertex3f(-x, y, 0.0f);
	}
	glEnd();

	glBegin(GL_QUADS);
	//glColor3f(0, 0, 0);
	glVertex2d(-10, 17);
```

```
	glVertex2d(-10, 9);
	glVertex2d(-6, 9);
	glVertex2d(-6, 17);

	glVertex2d(10, 17);
	glVertex2d(10, 9);
	glVertex2d(6, 9);
	glVertex2d(6, 17);

	glVertex2d(-11, 7);
	glVertex2d(-11, -3);
	glVertex2d(-6, -3);
	glVertex2d(-6, 7);

	glVertex2d(11, 7);
	glVertex2d(11, -3);
	glVertex2d(6, -3);
	glVertex2d(6, 7);

	glEnd();

	glPopMatrix();
}

// Obstacles Drawers
void drawObstacles() {
	double x_pos, y_pos;
	Obstacle *ob;
	for (int ob_num = OBSTACLES_MAX_INDEX; ob_num >= 0; ob_num--) {
		ob = &obstacles[ob_num];
		if (ob->on)
		{
			x_pos = ob->current_down_x;
			y_pos = ob->current_down_y;
			drawWoodObstacle(ob);
		}
	}
}

void drawWoodObstacle(Obstacle *ob)
{

	double x = ob->current_down_x;
	double y = ob->current_down_y;
	float w = ob->width;
	float h = ob->height;
```

```
glPushMatrix();

float depth_ratio = 5 * (1 - (y / WIN_H));
float base_width = w;
double depth_change = base_width / depth_ratio;
float slope = GLOBAL_SLOPE - (depth_change / abs(GLOBAL_SLOPE));

double symbolic_height = (WIN_H / depth_ratio);

float x_scale = 6, y_scale = 6;


//base_width = base_width * ((WIN_H - y) / WIN_H);

if (x <= getLane(1) || x >= getLane(3))
{
	slope = (x >= getLane(3)) ? slope : -slope;
	x = (symbolic_height + (slope*x)) / slope;
	//up_right_x = ((up_y - down_y) + (slope*down_right_x)) / slope;

}

glTranslatef(x, y, 0);
glScalef(x_scale, y_scale, 1);

float width = base_width;
float minor_width = width/3;
float height = h;

float delta_height;
float up_y;

//double down_right_x, up_right_x, down_left_x, up_left_x;

for (float i = -width/2; i < width/2; i+=minor_width) {

	delta_height = height/5;
	up_y = (height - delta_height);

	glBegin(GL_POLYGON);
	glColor3f(0.8, 0.6, 0.2);

	glVertex2f(i, up_y);
	glVertex2f(i, 0);
	glVertex2f(i + minor_width, 0);
	glVertex2f(i + minor_width, up_y);
```

```
                glEnd();

                glBegin(GL_TRIANGLES);
                glVertex2f(i, up_y);
                glVertex2f(i + (minor_width / 2), height);
                glVertex2f(i + minor_width, up_y);
                glEnd();

                glPushMatrix();
                glPointSize(100);
                glBegin(GL_LINE_STRIP);
                glColor3f(0.8, 0.2, 0.2);
                glVertex2f(i, up_y);
                glVertex2f(i, 0);
                glVertex2f(i + minor_width, 0);
                glVertex2f(i + minor_width, up_y);
                glEnd();
                glPopMatrix();
        }

        ob->real_width = base_width*x_scale;
        ob->real_height = height*y_scale;

        glPopMatrix();

        // For Debugging
        if (DEBUGING_MODE) {
                char* id[20];
                sprintf_s((char *)id, 20, "%d", ob->id);
                print(x - width / 2, y, (char*)id);
        }
}

// Powerups Drawers
void drawPowerups() {
        double x_pos, y_pos;

        Powerup *pu;
        for (int pu_num = POWERUPS_MAX_INDEX; pu_num >= 0; pu_num--) {
                pu = &powerups[pu_num];
                if (pu->on)
                {
                        x_pos = pu->current_down_x;
                        y_pos = pu->current_down_y;


                        if (pu->type == 0) {
```

```
                        drawPowerUpBottle(pu);
                    }
                    else
                    {
                        drawPowerUpStrike(pu);
                    }
                }
            }
        }


void drawPowerUpStrike(Powerup *pu)
{
        double x = pu->current_down_x;
        double y = pu->current_down_y;
        float w = pu->width;
        float  h  =  pu->height;
        double theta = pu->theta;

        if (x < -2) {
                printf("\n @drawer_start: Powerup Strike::\n id= %d,\nx= %f, y= %f,\nw= %f, h= %f\n",
                        pu->id, x, y, w, h);
                if (x < -5)
                        glutIdleFunc(NULL);
        }

        glPushMatrix();

        float depth_ratio = 5 * (1 - (y / WIN_H));
        float base_width = w;
        double depth_change = base_width / depth_ratio;
        float slope = GLOBAL_SLOPE - (depth_change / abs(GLOBAL_SLOPE));

        double symbolic_height = (WIN_H / depth_ratio);

        float x_scale = 6, y_scale = 6;

        if (x <= getLane(1) || x >= getLane(3))
        {
                slope = (x >= getLane(3)) ? slope : -slope;
                x = (symbolic_height + (slope*x)) / slope;
        }

        glTranslated(x, y, 0);
        glScaled(x_scale, y_scale, 1);
        glRotated(-3 + theta, 0, 0, 1);
```

```
	float width = base_width;
	float minor_width = width / 3;
	float height = h;


	glBegin(GL_TRIANGLES);
	glColor3f(1, 1, 0.1);
	glVertex2f(2.8, 9);
	glVertex2f(-2.5, 5);
	glVertex2f(1, 3);
	glVertex2f(0, 5);
	glVertex2f(2.8, 3);
	glVertex2f(-1, 0);
	glEnd();
	glPopMatrix();

	pu->real_width = 5 * x_scale;
	pu->real_height = 10 * y_scale;

	glPopMatrix();

	// For Debugging
	if (DEBUGING_MODE) {
		glColor3f(0, 0, 0);
		char* id[20];
		sprintf_s((char *)id, 20, "%d", pu->id);
		print(x - pu->real_width / 2, y, (char*)id);
	}
}

void drawPowerUpBottle(Powerup *pu)
{
	double x = pu->current_down_x;
	double y = pu->current_down_y;
	float w = pu->width;
	float h = pu->height;
	double theta = pu->theta;

	if (x < -2) {
		printf("\n @drawer_start: Powerup bottle::\n id= %d,\nx= %f, y= %f,\nw= %f, h= %f\n",
			pu->id, x, y, w, h);
		if(x<-5)
			glutIdleFunc(NULL);
	}

	glPushMatrix();
```

```
float depth_ratio = 5 * (1 - (y / WIN_H));
float base_width = w;
double depth_change = base_width / depth_ratio;
float slope = GLOBAL_SLOPE - (depth_change / abs(GLOBAL_SLOPE));

double symbolic_height = (WIN_H / depth_ratio);

float x_scale = 6, y_scale = 6;

if (x <= getLane(1) || x >= getLane(3))
{
        slope = (x >= getLane(3)) ? slope : -slope;
        x = (symbolic_height + (slope*x)) / slope;
}



glTranslated(x, y, 0);
glScaled(x_scale, y_scale, 1);
glRotated(30 + theta, 0, 0, 1);

float width = base_width;
float minor_width = width / 3;
float height = h;

//float delta_height;
//float up_y;

glBegin(GL_POLYGON);
glColor3f(1, 0.5, 0.1);
glColor3f(0, 0, 0);
glVertex2f(-1, 6);
glVertex2f(-1.5, 6);
glVertex2f(-1.5, 6.5);
glVertex2f(1.5, 6.5);
glVertex2f(1.5, 6);
glVertex2f(1, 6);
glEnd();

glBegin(GL_POLYGON);
glColor3f(1, 0.5, 0.1);
glVertex2f(1, 6);
glVertex2f(1, 5);
glVertex2f(-1, 5);
glVertex2f(-1, 6);
glEnd();
```

```
	glBegin(GL_POLYGON);
	glColor3f(1, 0.1, 0.1);
	glVertex2f(1, 5);
	glVertex2f(2, 5);
	glVertex2f(2, 0);
	glVertex2f(-2, 0);
	glVertex2f(-2, 5);
	glVertex2f(-1, 5);
	glEnd();

	pu->real_width = 4 * x_scale;
	pu->real_height = 13 * y_scale;
	glPopMatrix();

	// For Debugging
	if (DEBUGING_MODE) {
		glColor3f(0, 0, 0);
		char* id[20];
		sprintf_s((char *)id, 20, "%d", pu->id);
		print(x - pu->real_width / 2, y, (char*)id);
	}
}

// Status Drawers
void drawStatusPart() {
	glPushMatrix();

	float r = 0.3, g = 0.1, b = 0.3;
	glTranslated(WIN_W / 2, 0, 0);

	double width = WIN_W;
	double original_width = width;
	double height = WIN_H / 8;
	double down_y = (WIN_H - WIN_H / 8);
	double y_point_of_change = height / 3 + down_y;

	glBegin(GL_QUAD_STRIP);
	int y;
	for (y = WIN_H; y >= down_y; y -= height / 12) {
		glColor3f(r, g, b);
		if (y < y_point_of_change) {
			width += 5 * GLOBAL_SLOPE;

			r += 0.1;
		}
		else
		{
```

```
                r -= 0.005;
                b += 0.005;
            }

            glVertex2f(-width / 2, y);
            glVertex2f(width / 2, y);
        }
        glEnd();

        // return y to previous value to map the outline
        y += height / 12;

        glBegin(GL_LINE_LOOP);
        glPointSize(50);
        glColor3f(1, 1, 1); // WHITE
        glVertex2f(-width / 2, y);
        glVertex2f(width / 2, y);
        width = original_width;
        glVertex2f(width / 2, y_point_of_change);
        glVertex2f(width / 2, WIN_H);
        glVertex2f(-width / 2, WIN_H);
        glVertex2f(-width / 2, y_point_of_change);
        glEnd();

        glPopMatrix();

}

void drawHpBar() {

        glPushMatrix();

        double x_pos, y_pos;
        double width = 0.3f* WIN_W;
        double height = 0.03f * WIN_H;

        x_pos = 0.1 * WIN_W;
        y_pos = 0.92f * WIN_H;
        glTranslated(x_pos, y_pos, 0);
        double margin = 50;


        double INIT_CAR_HP = 100;
        double hp_width_fill = width * (car.hp / INIT_CAR_HP);

        glBegin(GL_POLYGON);
        glColor3f(0.9, 0.2, 0.3); // RED
```

```
		glVertex2f(0, height);
		glVertex2f(0, 0);
		glVertex2f(hp_width_fill, 0);
		glVertex2f(hp_width_fill, height);
		glEnd();

		glPushMatrix();

		glBegin(GL_LINE_LOOP);
		glPointSize(50);
		glScaled(1.5, 1.5, 1);
		glColor3f(1, 1, 1); // WHITE
		glVertex2f(0, height);
		glVertex2f(0, 0);
		glVertex2f(width, 0);
		glVertex2f(width, height);
		glEnd();

		glColor3f(1, 1, 1);
		char* id[20];
		sprintf_s((char *)id, 20, "HP:");
		print(-margin, margin / 4, (char*)id);

		glPopMatrix();

		glPopMatrix();
}

void drawScoreBar() {
		glPushMatrix();

		double x_pos, y_pos;
		//double width = 0.3f* WIN_W;
		//double height = 0.03f * WIN_H;

		x_pos = WIN_W - (0.25 * WIN_W);
		y_pos = 0.92f * WIN_H;
		glTranslated(x_pos, y_pos, 0);
		double margin = 50;

		glPointSize(50);
		glColor3f(1, 1, 1);
		char* id[20];
		sprintf_s((char *)id, 20, "Score: %d", car.score);
		print(0, 0, (char*)id);

		glPopMatrix();
```

```
}

void drawAccBar() {
	glPushMatrix();

	double x_pos, y_pos;
	//double width = 0.3f* WIN_W;
	//double height = 0.03f * WIN_H;

	x_pos = WIN_W - (0.45 * WIN_W);
	y_pos = 0.92f * WIN_H;
	glTranslated(x_pos, y_pos, 0);
	double margin = 50;

	glPointSize(50);
	glColor3f(1, 1, 1);
	char* id[20];
	sprintf_s((char *)id, 20, "Acc: %f", car.acceleration);
	print(0, 0, (char*)id);

	glPopMatrix();
}

// Road Drawers
void drawRoadLine(bool left, double down_y)
{

	float r = 0.2, g = 0.3, b = 0.7;
	r = 0.9, g = 0.8, b = 1;
	glColor3f(r, g, b);

	float depth_ratio = 5 * (1 + (down_y / WIN_H));
	double height = (WIN_H/depth_ratio);
	float base_width = (WIN_W / 18);
	double depth_change = base_width / depth_ratio;
	float slope = GLOBAL_SLOPE - (depth_change / abs(GLOBAL_SLOPE));
	double up_y = height + down_y;
	double down_right_x, up_right_x, down_left_x, up_left_x;

	//height = height * (1 - down_y / up_y);
	base_width = base_width * (((WIN_H - down_y) /WIN_H));


	if (left)
	{
		slope = -slope;
		down_right_x = (3 * (WIN_W / 8));
```

```
			up_right_x = down_right_x;//((up_y - down_y) + (slope * down_right_x)) / slope -
depth_change;
			down_left_x = down_right_x - base_width;
			up_left_x = ((up_y - down_y) + (slope*down_left_x)) / slope;

		}
		else
		{
			down_left_x = (5 * (WIN_W / 8));
			up_left_x = down_left_x;//((up_y - down_y) + (slope * down_left_x)) / slope +
depth_change;
			down_right_x = down_left_x + base_width;
			up_right_x = ((up_y - down_y) + (slope*down_right_x)) / slope;

		}


		glBegin(GL_POLYGON);
		glVertex2f(down_left_x, down_y);
		glVertex2f(down_right_x, down_y);
		glVertex2f(up_right_x, up_y);
		glVertex2f(up_left_x, up_y);
		glEnd();
}

void drawRoadLines(void)
{
		for (int y = road_start_y; y < (WIN_H - WIN_H/10); y += 3*WIN_H / 10) {
			drawRoadLine(false, y);
			drawRoadLine(true, y);
		}
}

void drawRoadPlane(void)
{
		double x_pos, y_pos;
		glPushMatrix();

		float r = 0.7f, g = 0.66f, b = 1.0f;
		glColor3f(r, g, b);

		x_pos = (WIN_W / 2);
		y_pos = 0;
		glTranslated(x_pos, y_pos, 0);

		float slope = GLOBAL_SLOPE;
		double down_right_x = (3 * (WIN_W / 4)) / 2;
```

```
	double up_right_x = (WIN_H + (slope*down_right_x)) / slope;

	glBegin(GL_POLYGON);
	glVertex2f(down_right_x, 0);
	glVertex2f(up_right_x, WIN_H);
	glVertex2f(-up_right_x, WIN_H);
	glVertex2f(-down_right_x, 0);
	glEnd();
	glPopMatrix();
}

void drawBackground(void) {

	double width = WIN_W;
	double original_width = width;
	double height = WIN_H;
	float r = bg.r, b = bg.b, g = bg.g;
	float delta_r = bg.delta_r;
	float delta_b = bg.delta_b;
	glPushMatrix();
	glTranslated(WIN_W / 2, 0, 0);
	//printf("\n rgb = %f, %f, %f\n", bg.r, bg.g, bg.b);

	glBegin(GL_QUAD_STRIP);
	int y;
	for (y = WIN_H; y >= 0; y -= height / 40) {
		r -= delta_r;
		b += delta_b;

		glColor3f(r, g, b);

		glVertex2f(-width / 2, y);
		glVertex2f(width / 2, y);
	}
	glEnd();

	glPopMatrix();
}

// draws a circle using OpenGL's gluDisk, given (x,y) of its center and tis radius
void drawCircle(double x, double y, float r) {
	glPushMatrix();
	glTranslatef(x, y, 0);
	GLUquadric *quadObj = gluNewQuadric();
	gluDisk(quadObj, 0, r, 50, 50);
	glPopMatrix();
```

```
}
// Text Drawers
void print(int x, int y, char *string)
{
	int len, i;

	//set the position of the text in the window using the x and y coordinates
	glRasterPos2f(x, y);
	//printf("String to print = '%s'", string);
	//get the length of the string to display
	len = (int)strlen(string);

	//loop to display character by character
	for (i = 0; i < len; i++)
	{
		glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
	}
}

/* Movers */
void moveCar(int delta_x, int delta_y) {

	car.registered_x += delta_x;
	car.registered_theta += (delta_x/ (GLOBAL_SLOPE + 1));


	if (car.registered_theta > 35) {
		car.registered_theta = 35;
	}
	else if (car.registered_theta < -35) {
		car.registered_theta = -35;
	}


	//double car_road_margin = WIN_W / 32;
	//double car_x_pos_tmp = car.x_pos;
	//car.x_pos += delta_x;
	/*
	if (car.x_pos < (getLane(1) - car_road_margin) ||
		car.x_pos > (getLane(3) + car_road_margin)) {
		car.x_pos = car_x_pos_tmp;
	}
	*/
	//printf("\n x_pos = %f, getLane(1) - car_road_margin = %f", car_x_pos, (getLane(3) +
car_road_margin));
}
```

```
/Prepherals Processors/
void processSpecialKeys(int k, int x, int y)
{
	if (car.alive) {
		if (k == GLUT_KEY_RIGHT)
			moveCar(35, 0);
		if (k == GLUT_KEY_LEFT)
			moveCar(-35, 0);
	}

	if (k == GLUT_KEY_DOWN) {
		initValues(); // Reset game values
		glutIdleFunc(Anim);
	}

	glutPostRedisplay();
}

void processSpecialUpKeys(int k, int x, int y) {
	if (car.alive) {
		if (k == GLUT_KEY_RIGHT)
			moveCar(-10, 0);
		if (k == GLUT_KEY_LEFT)
			moveCar(10, 0);
		glutPostRedisplay();

	}

}

void passiveMotion(int x, int y)
{
	double car_road_margin = WIN_W / 32;
	double car_x_pos_tmp = car.x_pos;

	car.x_pos = x;
	/*
	if (x > car.x_pos)
	{
		moveCar(x - car.x_pos, 0);
	}
	else
	{
		moveCar(car.x_pos - x, 0);
	}
	*/
```

```
	if (car.x_pos < (getLane(1) - car_road_margin) ||
		car.x_pos >(getLane(3) + car_road_margin)) {
		car.x_pos = car_x_pos_tmp;
	}

	glutPostRedisplay();
}

/* Debugging Functions */
void debugScreenLines(int width_minor, int height_minor) {
	glColor3f(1, 0, 0);
	for (int i = 0; i < WIN_W; i += WIN_W / width_minor) {
		glBegin(GL_LINES);
		glVertex2f(i, 0);
		glVertex2f(i, WIN_H);
		glEnd();
	}
	for (int j = 0; j < WIN_H; j += WIN_H / height_minor) {
		glBegin(GL_LINES);
		glVertex2f(0, j);
		glVertex2f(WIN_W, j);
		glEnd();
	}
}

void debugObstaclesOutOfRange() {
	Powerup* pu;
	for (int pu_num = POWERUPS_MAX_INDEX; pu_num >= 0; pu_num--) {
		pu = &powerups[pu_num];
		if (pu->on)
		{
			double x = pu->current_down_x;
			double y = pu->current_down_y;
			float w = pu->width;
			float h = pu->height;

			if (x < -2) {
				printf("\n @debugger: Powerup bottle::\n id= %d,\nx= %f, y= %f,\nw= %f,
h= %f\n",
					pu->id, x, y, w, h);
				if (x < -5)
					glutIdleFunc(NULL);
			}
		}
	}
}
```

```
/* Objects Generators */

bool generateRandomObject() {

	int obstacle_powerup = getRandomValue(1, 10);
	if (obstacle_powerup >= 6) {
		if (obstacles[obstacles_counter + 1].on)
			return false;
		generateRandomObstacle();
	}
	else {
		if (powerups[powerups_counter + 1].on)
			return false;
		generateRandomPowerup();
	}

	return true;
}

void generateRandomObstacle() {
	int lane_num = getRandomValue(1, 3);
	if (DEBUGING_MODE)
		printf("\nGenerating Obstacle:: lane_num = %d\n", lane_num);
	int obstacle_type = (getRandomValue(1, 10) >= 5) ? 0 : 1;
	// TODO, add obstacle_type to the called method
	makeObstacle(getLane(lane_num), SLIGHTLY_OFF_TOP_SCREEN, 0);

	if (getRandomValue(1, 100) > 30) {
		int new_lane_num;
		if (getRandomValue(50, 100 >= 70)) {

			new_lane_num = lane_num + 1;
			if (new_lane_num > 3)
				new_lane_num = 1;
			if (DEBUGING_MODE)
				printf("\nGenerating Obstacle:: lane_num = %d\n", new_lane_num);
			makeObstacle(getLane(lane_num), SLIGHTLY_OFF_TOP_SCREEN, 0);
		}
		else {
			new_lane_num = lane_num - 1;
			if (new_lane_num < 1)
				new_lane_num = 3;
			if (DEBUGING_MODE)
				printf("\nGenerating Obstacle:: lane_num = %d\n", new_lane_num);

			makeObstacle(getLane(lane_num), SLIGHTLY_OFF_TOP_SCREEN, 0);
		}
```

```
	}

}

void generateRandomPowerup() {
	int lane_num = getRandomValue(1, 3);
	if (DEBUGING_MODE)
		printf("\nGenerating Powerup:: lane_num = %d\n", lane_num);
	int powerup_type = (getRandomValue(1, 10) >= 5) ? 0 : 1;
	makePowerup(getLane(lane_num), SLIGHTLY_OFF_TOP_SCREEN, powerup_type);
}

/* Structs Creators */
Powerup* makePowerup(double x, double y, int t, float w, float h) {

	if (++powerups_counter > POWERUPS_MAX_INDEX) {
		powerups_counter = 0;
	}

	Powerup* pu = &powerups[powerups_counter];
	pu->on = true;
	pu->current_down_y  =  y;
	pu->current_down_x  =  x;
	pu->type = (t <= 0) ? 0 : 1;
	pu->hit = 0.2;
	pu->width = (w > 0) ? w : pu->width;
	pu->height = (h > 0) ? w : pu->height;
	pu->real_width = pu->width;
	pu->real_height = pu->height;
	pu->id = powerups_counter;

	x = pu->current_down_x;
	y = pu->current_down_y;
	w = pu->width;
	h = pu->height;
	if (x < 0) {
		printf("\n @makePowerup: Powerup bottle::\n id= %d,\nx= %f, y= %f,\nw= %f, h= %f\n",
			pu->id, x, y, w, h);
		glutIdleFunc(NULL);
	}

	num_of_on_powerups++;
	return pu;
}

Powerup* makePowerup(double x, double y, int t)
```

```
{
	return makePowerup(x, y, t, -1, -1);
}

Obstacle* makeObstacle(double x, double y, int t, float w, float h) {

	if (++obstacles_counter > OBSTACLES_MAX_INDEX) {
		obstacles_counter = 0;
	}

	Obstacle* ob = &obstacles[obstacles_counter];
	ob->on = true;
	ob->current_down_y  =  y;
	ob->current_down_x  =  x;
	ob->type = (t <= 0) ? 0 : 1;
	ob->hit = (t <= 0) ? 0.2 : 0.4;
	ob->width = (w > 0) ? w : ob->width;
	ob->height = (h > 0) ? w : ob->height;
	ob->real_width = ob->width;
	ob->real_height = ob->height;
	ob->id = obstacles_counter;

	num_of_on_obstacles++;
	return ob;
}

Obstacle* makeObstacle(double x, double y, int t)
{
	return makeObstacle(x, y, t, -1, -1);
}


/* Other Helpfull Functions */
double getLane(int lane_num)
{
	if (lane_num > 0 && lane_num <= 3) {
		//	return (lane_num * 2) *(WIN_W / 8);
	}

	switch (lane_num) {
	case(1): return 5 * WIN_W / 22;
	case(2): return WIN_W / 2;
	case(3): return 17 * WIN_W / 22;
	}

	return 0;
}
```

```
int getRandomValue(int minimum, int maximum) {
	/* initialize random seed: */
	srand(time(NULL));
	/* generate secret number between 1 and maximum: */
	return (rand() % maximum + minimum);
}

void initConfig() {

	glutInitWindowSize(WIN_W, WIN_H);
	glutInitWindowPosition(50, 50);
	glutCreateWindow("MadCar - DMET");
	glutDisplayFunc(Display);
	glutIdleFunc(Anim);
	glClearColor(0.1f, 0.1f, 0.2f, 0.0f);

	//glutFullScreen();		// making the window full screen

	//glutPassiveMotionFunc(passiveMotion);
	glutSpecialFunc(processSpecialKeys);
	glutSpecialUpFunc(processSpecialUpKeys);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	gluOrtho2D(0.0, WIN_W, 0.0, WIN_H);

	GLint m_viewport[4];
	glGetIntegerv(GL_VIEWPORT, m_viewport);
	WIN_W = m_viewport[2];
	WIN_H = m_viewport[3];
	//printf("%d, %d, %d, %d", WIN_W, WIN_H);

}

void initValues() {

	if(car.alive = false)
		sndPlaySound(TEXT("C:\\Sound Effects\\game_sound.wav"), SND_FILENAME |
SND_ASYNC | SND_LOOP);


	//Intialize Car values
	car.x_pos = getLane(2);
	car.y_pos = WIN_H / 10;
	car.alive = true;
	car.hp = MAX_HP;
	car.acceleration = MIN_ACC;
	car.registered_acc = 0;
```

```
	car.registered_x = 0;
	car.registered_hp = 0;
	car.score = 0;

	//Intialize Obstacles values
	for (int i = 0; i <= OBSTACLES_MAX_INDEX; i++) {
		Obstacle *ob = &obstacles[i];
		ob->on = false;
		ob->height = 8;
		ob->width = 16;
		ob->id = -1;
		ob->type = 0; // TODO just for now until I make the other type
	}

	//Intialize Powerups values
	for (int i = 0; i <= POWERUPS_MAX_INDEX; i++) {
		Powerup *pu = &powerups[i];
		pu->on = false;
		pu->id = -1;
		pu->type = -1; // TODO just for now until I make the other type
		pu->theta = 0;
	}

	generateRandomObject();

	road_start_y = ROAD_START_Y_INIT;
	background_start_y = BACKGROUND_START_Y_INIT;
	num_of_on_obstacles = 0;
	num_of_on_powerups = 0;

	time(&past_time_scored);
	int past_tick = glutGet(GLUT_ELAPSED_TIME);
}

int main(int argc, char** argr)
{

	glutInit(&argc, argr);
	// Init OpenGL Configs
	sndPlaySound(TEXT("C:\\Sound Effects\\game_sound.wav"), SND_FILENAME |
SND_ASYNC | SND_LOOP);

	initConfig();

	// - the correct code
	// Intializations of variables
```

```
	initValues();

	glutMainLoop();
	return 0;
}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu
// Debug program: F5 or Debug > Start Debugging menu
```

# Screen shots