The Ministry of Education and Science of the Republic of Kazakhstan

Al-Farabi Kazakh National University

Faculty of Mechanics and Mathematics

Department of Mathematical and Computer Modeling

GRADUATE WORK

Theme: «Interactive graphics and game modeling in OpenGL»

by specialty 5B070500 – «Mathematical and Computer Modeling»

Done by _____ Kakibay A.K.
(sign)
Scientific adviser
Professor _____ Khadjieva L.A.
(sign)

Approved to protection
Protocol № ___ 2021 year
Head of department _____ Issakhov A.A.
(sign and stamp)
Compliance supervisor _____ Ayatbek A.S.
(sign)

Almaty 2021

ABSTRACT

The thesis, written on the topic "Interactive graphics and game modeling in OpenGL", consists of 59 pages, 36 illustrations and 18 sources used.

List of keywords: OpenGL, graphics, games, animation, conditions, operators, model.

Objectives of the thesis:

- Simulation of the game in the OpenGL environment and its practical application on individual examples with the development of program code.
- Animation. Program code.
- Write the conditions for the movement. Program code.
- Use the example to show mouse and keyboard control. Program code.
- Texture and light overlay. Program code.

This thesis is dedicated to show the main features of OpenGL for scene visualization and its application in game modeling. Also, the creation of objects, cameras, and object management, such as mouse, keyboard, are considered.

РЕФЕРАТ

Дипломная работа, написанная на тему «Интерактивная графика и моделирование игр в среде OpenGL», состоит из 59 страниц, 36 иллюстраций и 18 использованных источников.

Перечень ключевых слов: OpenGL, графика, игры, анимация, условья, операторы, модель.

Цели дипломной работы:

- Моделирование игры в среде OpenGL и ее практическое применение на отдельных примерах с разработкой программного кода.
- Анимация. Программный код.
- Написать условья для движения. Программный код.
- На примере показать управление мышью, клавиатурой. Программный код.
- Наложение текстуры и света. Программный код.

Данная дипломная работа посвящена, чтобы показать основные возможности OpenGL для визуализации сцены и его применение в моделировании игр. Также, рассматривается создание объектов, камеры, и управление объектом, такие как мышь, клавиатура.

РЕФЕРАТ

"OpenGL ортасында интерактивті графиканы және ойындарды модельдеу" тақырыбында жазылған дипломдық жұмыс 59 беттен, 36 иллюстрациядан және 18 пайдаланылған дереккөздерден тұрады.
Кілт сөздер тізімі: OpenGL, графика, ойындар, анимация, шарттар, операторлар, модель.

Дипломдық жұмыстың мақсаттары:

- OpenGL ортасында ойынды модельдеу және оны бағдарламалық кодты әзірлеумен жеке мысалдарда практикалық қолдану.
- Анимация. Бағдарламалық код.
- Қозғалыс шарттарын жазу. Бағдарламалық код.
- Мысалында көрсету басқармасы тінтуір, пернетақта бар. Бағдарламалық код.
- Текстураның және жарықтың қабаттасуы. Бағдарламалық код.

Бұл тезис OpenGL-дің сахнаны визуализациялаудың негізгі мүмкіндіктерін және оны ойындарды модельдеуде қолдануды көрсетуге арналған. Сондай-ақ, осындай тінтуір, пернетақта ретінде объектілерді құру, камера қарастырылды.

CONTENT

INTRODUCTION

Computer graphics – the use of computer technology to create graphic images, display them by various means and manipulate them. [1]

Today, computer graphics are used in most engineering and scientific disciplines to convey information, its visual perception. Now it is the norm to use it in the preparation of demo slides, the development of websites on any topic is also not complete without the use of computer graphics, in medicine, three-dimensional images are widely used-computed tomography. Do not forget about cartography, geophysics, printing, nuclear physics and many other fields and spheres of life. Various branches of the entertainment industry and television constantly use animated means of computer graphics — computer games and movies. In the training of pilots and professionals from other fields, the practice of computer modeling and various simulators is common. Now knowledge of the basics of computer graphics is necessary for both an engineer and a scientist. [2]

It is noticeable that in computer science, computer graphics is one of the large areas where there are three main sections: the illustrative direction, the research direction and the self-developing direction. Let’s partially reveal each direction:

1) illustrative direction - it is the most widely covered direction in computer science, as it considers problems for simple data visualization and is even used for creating animated films and other types of animation.

2) the direction of research-it involves the use of computer graphics aimed at creating an image of some problem that does not have a physical analogue, to show the model for calculation and to track the change in parameters and correct them when an error occurs.

3) the self-developing direction is computer graphics, the subject matter and possibilities of which are truly limitless, allows you to expand and improve the skills of humanity and helps to improve the obtaining of visas. [3]

OpenGL quickly became the industry-leading real-time graphics API, as it was basically the only one available on multiple platforms. [4]

This work is devoted to the development of software code for game modeling, as well as its visualization using the open graphics library OpenGL.

Approbation of the work. The results of the study were presented in the form of abstracts and were presented at the International Scientific Conference of Students and Young Scientists "Farabi Alemi" (Almaty, April 6-9, 2021);

1 GRAPHIC EDITOR IN OPENGL

1.1 OpenGL Feature

At a basic level, OpenGL is just a document which is describing a set of functions with their exact behavior which can be implemented to something. It is known that the hardware manufacturers use OpenGL to create implementations-libraries of functions that correspond to the set of functions in the specification.

Also, the capabilities of the hardware are used by the implementation in order to use it where it can be possible to implement, if it cannot be possible to use then it must be emulated in program. As one of implementation of OpenGL can be given as an example of testing something in manufactures in order to cut the outcomes for it. For software developers is enough to learn how to use the functions described in the specification and use it in optimized way, leaving the effective implementation of the latter to hardware developers.

OpenGL focuses on the following two tasks:

- Hide the complexity of adapting different 3D accelerators by providing the developer with a single API.
- Hide differences in the capabilities of hardware platforms, requiring the implementation of missing functionality using software emulation. [5]

1.2 Key features of OpenGL

- **Geometric and raster primitives.** All objects are built on the basis of geometric and raster primitives. From geometric primitives, the library provides: points, lines, polygons. From bitmaps: bitmap and image)
- **Use of B-splines.** B-splines are used to draw curves along reference points.
- **View and model transformations.** With these transformations, it can be position objects in space, rotate them, change their shape, and change the position of the camera from which the surveillance is conducted.
- **Working with color.** OpenGL provides the programmer with the ability to work with color in RGBA mode (red-green-blue-alpha) or using index mode, where the color is selected from the palette.
- **Remove invisible lines and surfaces. Z-buffering.**
- **Double buffering.** OpenGL provides both single and double buffering.

Double buffering is used to eliminate flickering during animation, i.e. the image of each frame is first drawn in the second (invisible) buffer, and then, when the frame is completely drawn, the entire buffer is displayed on the screen.

- **Texture overlay.** Allows to give objects a realistic appearance. An object, such as a ball, is overlaid with a texture (just some image), as a result of which our object now looks not just like a ball, but like a multi-colored ball.
- **Anti-aliasing.** Anti-aliasing allows to hide the gradation inherent in bitmap displays. Anti-aliasing changes the intensity and color of the pixels near the line, while the line looks on the screen without any zigzags.
- **Lighting.** Allows to set the light sources, their location, intensity, etc.
- **Atmospheric effects.** For example, fog, smoke. All this also allows to give objects or scenes a realistic feel, as well as" feel " the depth of the scene.
- **Transparency of objects.**
- **Using image lists.** [6]

2 MODELING ANIMATION OF OBJECT IN OPENGL

2.1 Scene visualization and projection

It is known that if do not define the space output is not declared then it will be taken automatically as a cube and the start of coordinates will be the center of cube.

So, let's describe two types of projection:

1) Parallel
2) Perspective

Parallel projection is defined by orthogonal projection, which means that the objects are located in the parallelepiped will be in output. This projection is needed to be written as function glOrtho(), gluOrtho2D(), as well as the arguments number for these to function are different. For example for glOrtho(left, right, bottom, top, near, far), and for gluOrtho2D(left, right, bottom, top).

Next is perspective projection, which is used for visualization of 3D picture with perspective. Transformation can be written as function gluPerspective() and this function defines truncated visibility pyramid. It is noticeable that clipping surface is parallel to z surface which is equal to zero. So the argument of this function is gluPerspective(fovy, aspects, near, far), where fovy is the angle of view, aspect is the relationship of Window width and height, near is the destination from the center of projection to the nearest cutting surface, and last argument is far, which is destination from the center of projection to the further cutting surface. One notion is that it is not needed to be symmetric pyramid.

Also, can be used transformation of resizing, rotation, motion of initial unit matrix and do matrix project formation transformation by each element.

In the beginning it is important to select the matrix projection work mode before calling projection function. And first load unit matrix and then transform it by gluPerspective().

Example of forming a visibility pyramid:

```
glMatrixMode(GL_PROJECTION);// Projection Matrix
glLoadIdentity;// setting the unit matrix
gluPerspective(60.0, Width / Height, 0.1, 70.0); // perspective setting function
glViewport(0, 0, ClientWidth, ClientHeight); // Setting the output area.
glMatrixMode(GL_MODELVIEW); // Switching to the model matrix.
InvalidateRect(Handle, nil, False);// Redrawing the window
```

2.2 Modeling of animation as an example of Solar System motion

If the contents of the frame buffer change while the image is being redrawn, the viewer may see completely undesirable effects, such as flickering of the changing

image. This problem is solved using double buffering, a standard technology for organizing computer animation.

In this case, have two buffers at disposal, which are usually called front and background. The front buffer is the one from which the screen image is sent, and in the background buffer, the image is generated by the OpenGL driver.

Switching buffers is performed by the glutSwapBuffers() function. When using double buffering, first it is needed to clear the working buffer by calling the glClear(GL_BUFFER_BIT) command, and then call the buffer switching function glutSwapBuffers() with the last statement.

```
void drawLogoScene(void){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, logTexture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
    glBegin(GL_POLYGON);
    glTexCoord2f(0.0, 0.0); glVertex3f(-100, -100, -100);
    glTexCoord2f(1.0, 0.0); glVertex3f(100, -100, -100);
    glTexCoord2f(1.0, 1.0); glVertex3f(100, 100, -100);
    glTexCoord2f(0.0, 1.0); glVertex3f(-100, 100, -100);
    glEnd();
    glutSwapBuffers();}
```

See the result in figure 2.1.

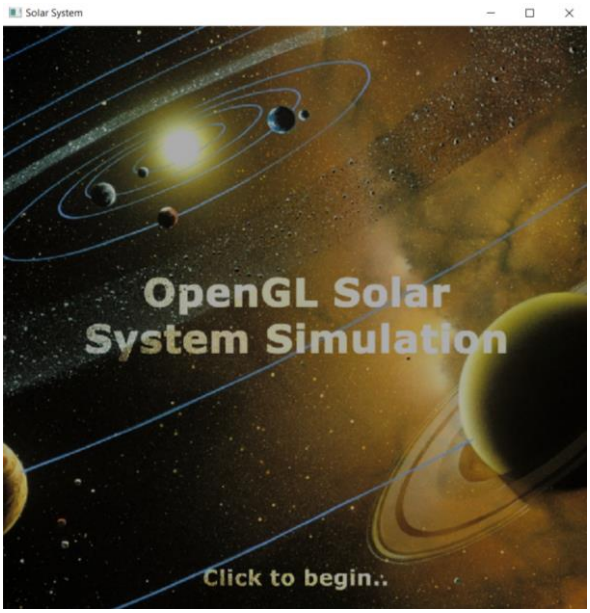


Figure 2.1 Example for glutSwapBuffers();

The creation of moving objects is implemented using affine transformations. To do this, instead of specifying the specific numeric values of the arguments, it must be specified variables or functions whose values change in the system timer. Result is in figure 2.2.

```
void animate(int n){
    if (isAnimate){
        mer.orbit += mer.orbitSpeed;
        ven.orbit += ven.orbitSpeed;
        ear.orbit += ear.orbitSpeed;
        mar.orbit += mar.orbitSpeed;
        jup.orbit += jup.orbitSpeed;
        sat.orbit += sat.orbitSpeed;
        ura.orbit += ura.orbitSpeed;
        nep.orbit += nep.orbitSpeed;
        plu.orbit += plu.orbitSpeed;
        lun.orbit += lun.orbitSpeed;
        pho.orbit += pho.orbitSpeed;
        dei.orbit += dei.orbitSpeed;
        eur.orbit += eur.orbitSpeed;
        gan.orbit += gan.orbitSpeed;
        cal.orbit += cal.orbitSpeed;
        tit.orbit += tit.orbitSpeed;
        nix.orbit += nix.orbitSpeed;
        puc.orbit += puc.orbitSpeed;
        tri.orbit += tri.orbitSpeed;
        if (mer, ven, ear, mar, jup, sat, ura, nep, plu, lun, pho, dei, eur, gan, cal,
tit, nix, puc, tri.orbit > 360.0){
            mer, ven, ear, mar, jup, sat, ura, nep, plu, lun, pho, dei, eur, gan,
cal, tit, nix, puc, tri.orbit -= 360.0;
        }
        mer.axisAni += 10.0;
        ven.axisAni += 10.0;
        ear.axisAni += 10.0;
        mar.axisAni += 10.0;
        jup.axisAni += 10.0;
        sat.axisAni += 10.0;
        ura.axisAni += 10.0;
        nep.axisAni += 10.0;
        plu.axisAni += 10.0;
        if (mer, ven, ear, mar, jup, sat, ura, nep, plu.axisAni > 360.0){
            mer, ven, ear, mar, jup, sat, ura, nep, plu.axisAni -= 360.0;
```

```
            }
            glutPostRedisplay();
            glutTimerFunc(30, animate, 1);
        }
}
```

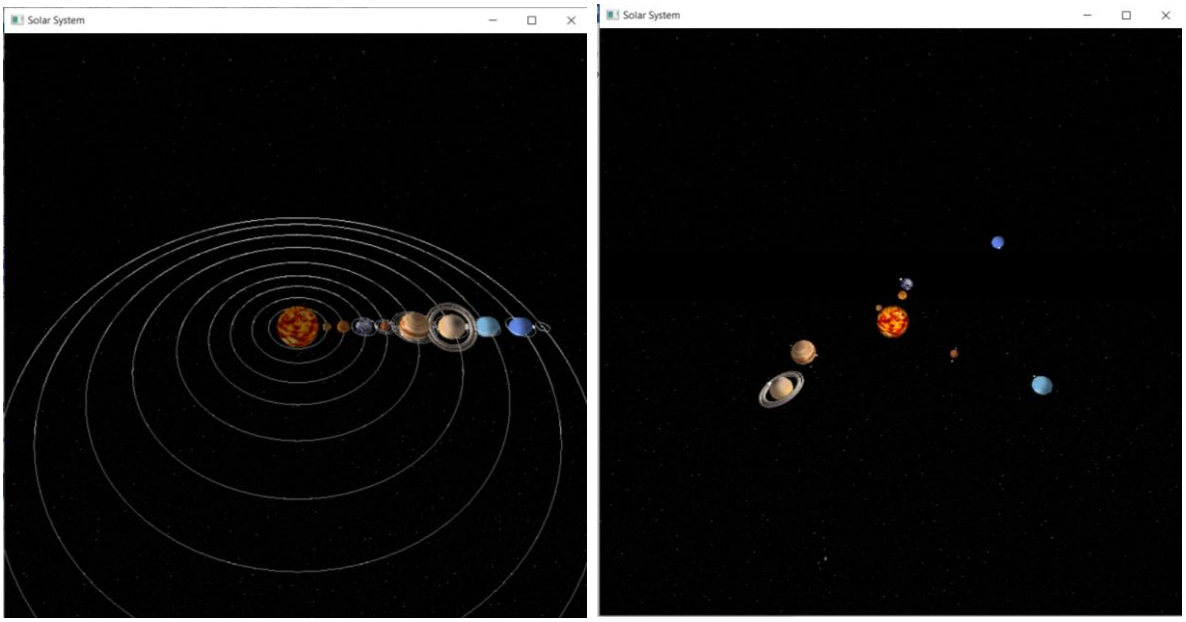


Figure 2.2 Animation of the movement of the planets

2.3 Modeling of light sources

Without a light source, the image is not visible. To initialize the source and enable the handler for calculating the impact of the source on objects, just run the following commands:
glEnable(gl_lighting);// enable lighting analysis mode
glEnable(gl_light0);//include a specific (null) source in the scene, with its characteristics

To disable the source, use the Disable () function. Initially the light always is located by x in 0, and by y in 0 and by z it is in infinity, so the lights source can be created at anywhere of taken space.

The OpenGL library supports four types of light sources:

- ambient lighting;
- point sources;
- spotlights;
- distant light.

Each light source has its own set of characteristics.

To set the vector parameters, use the glLightfv(source, parameter, pointer_to_array) function.

There are four vector parameters that determine the position and direction of the source rays and the color composition of its components – background, diffusion, and mirror.

To set scalar parameters in OpenGL, use glLightf(source, parameter, value). For example, suppose it is needed to include a GL_LIGHT0 source in the scene, which should be located at (1.0, 2.0, 3.0). The position of the source is stored in the program as a point in uniform coordinates:
GLfloat light0_pos[]={1.0, 2.0, 3.0, 1.0};

If the fourth component of this point is zero, then the point source turns into a remote one, for which only the direction of the rays is important:
GLfloat light0_dir[]={1.0, 2.0, 3.0, 0.0};

Next, the color composition of the background, diffusion, and mirror components of the source is determined. If in the example under consideration the source has a mirror component of white color, and the background and diffusion components should be red, then the program fragment that forms the source looks like this:

```
GLfloat diffise0[]= {1.0, 0.0, 0.0, 1.0};
GLfloat ambient0[]={1.0, 0.0, 0.0, 1.0};
GLfloat specular0[]={1.0, 1.0, 1.0, 1.0};
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
```

It can also include global background lighting in the scene, which is not associated with any individual light source. If, for example, it is wanted to weakly highlight all the objects of the scene in white, the program should include this code snippet:

```
GLfloat global_ambient[]={0.1, 0.1, 0.1, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
```

To convert a point source to a spotlight, it is needed to set the direction of the spotlight beam (GL_SPOT_DIRECTION), the intensity distribution function (GL_SPOT_EXPONENT), and the beam scattering angle (GL_SPOT_CUTTOF). These parameters are set using the glLightf() and glLightfv () functions. Results of lighting is in figure 2.3.

```
//LIGHTING SETUP
glEnable(GL_LIGHTING);
float lightAmb[] = { 0.0, 0.0, 0.0, 1.0 };
float lightDifAndSpec[] = { 1.0, 1.0, 1.0, 1.0 };
float globAmb[] = { 0.5, 0.5, 0.5, 1.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDifAndSpec);
glLightfv(GL_LIGHT0, GL_SPECULAR, lightDifAndSpec);
glEnable(GL_LIGHT0);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globAmb);
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, spotAngle);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDirection);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, spotExponent);
```

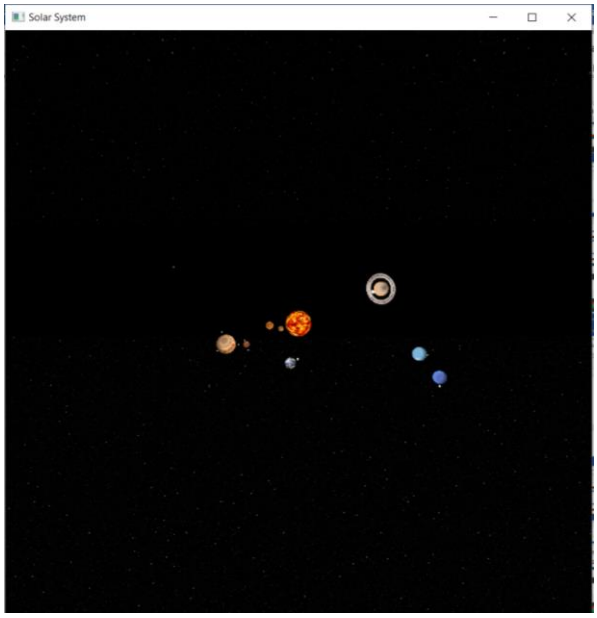


Figure 2.3 Illumination of the planets by the sun

2.4 Texture mapping

When creating a graphic application, to make objects realistic it is used to overlay textures in order to make objects colorful and good in vision. And in cases it is most used in three-dimensional objects and in each side overlay the defined texture. Interesting fact is that it is possible to use one texture for array of objects. For example,

in following chapters will be presented the use of bricks, and it is possible to upload only one texture and define it for all bricks, in order to make easier the programming process as it will take time to define each brick with texture.

Next, let's see the application of texturing on object by steps. To apply a texture to on object, it is needed to first load the image file into memory after it is needed to create a texture ID name and make it active. Next action is to create in memory the needed texture. After all of these action work on settings, it means set parameters of texture and then define objects with texture parameters in order to use it in program code. Do not forget to link the coordinate of texture.

So, after locating the texture in the memory it is important to define with dimension of the image, it means choose one-dimensional or two-dimensional.

In order to avoid the error of ladder it is important to take all elements of taken array and after need-to-know what elements are influencing to taken array. Next thing is needed to know is to find a way to avoid such error, for it is needed to be taken four corner points of taken element in array of image, after those corner points will be connected and will be shaped as a rectangle or square. If some element will be in this area, then will be taken sum of all elements in this area as well as weighting it with other elements, in order to be sure with proportion of the taken element.

So, now move on the texture parameter and its command and command parameters. First, let's introduce the command of texture parameters, it is void glTexParameter [i , f, v](target, pname, param). Here as it was, need to concentrate on parameters of this command, so, the first one is target, which is defines to what object is needed to overlay the texture on one-dimensional or two-dimensional, and it can be written as well as it was said with GL_TEXTURE_(1D/2D). Next parameters is pname, which is indicates the symbolic name of the texture parameter, which will be discusses widely below. And the last one is param, which is the definer of pname, it means it gives the meaning to parameter of pname. Also, it will be discussed below.

So, let's start from pname parameter. For this parameter is defined several functions of it, like: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S.

Let's discuss them one by one. Fist is GL_TEXTURE_MIN_FILTER, it is function of reduction of the texture, which is used when the pixel area is much bigger than the area of overlaying of texture on element. And the vise versa of it is the function GL_TEXTURE_MAG_FILTER, which is said to overlaying of texture on element if the pixel area is less than it. So, such kind of regulations are used in order to overlay texture in right way. Also, there are used param parameters with GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, like GL_NEAREST, GL_LINEAR. Here GL_NEAREST helps to return the nearest value from the center of pixel of texture, and GL_LINEAR helps to return arithmetic average value of four elements of texture, which are located in the center of pixel.

The last one is GL_TEXTURE_WRAP_S, which helps to avoid the artificial wrapping when relay next texture on object and creates a repeating stencil. Also, it works only with GL_CLAMP or GL_REPEAT as well.

Only for two-dimensional image can be used linear filtration for four near elements of texture and for one-dimensional image can be used linear filtration for two near elements of texture. Also it is important to know that any image can be created in (s, t) system of coordinates, then the next thing to be needed to pay attention is the function glTexCoord2d(), which defines to map the coordinate texture in the vertices of the rectangle or square. Remember that by default left-lower angle has coordinate 0 in x as well as in y, and right-upper angle has coordinate, which is 1 in x same as in y.

So, below can be found the example of how to bind the rectangle of vertices to the texture coordinates:

```
glBegin(GL_QUADS);
glTexCoord2d(0,0); glVertex3d(-5,-5, -0.1);
glTexCoord2d(0,1); glVertex3d(-5, 5, -0.1);
glTexCoord2d(1,1); glVertex3d( 5, 5, -0.1);
glTexCoord2d(1,0); glVertex3d( 5,-5, -0.1);
glEnd();
glDisable(GL_TEXTURE_2D);
```

Example:

```
GLuint loadTexture(Image* image) {
	GLuint textureId;
	glGenTextures(1, &textureId);
	glBindTexture(GL_TEXTURE_2D, textureId);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height, 0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
	return textureId;
}
GLuint sunTexture, merTexture, venTexture, earTexture, marTexture, jupTexture, satTexture, uraTexture, nepTexture, pluTexture, staTexture, logTexture;
//TEXUTRING SETUP
	glEnable(GL_NORMALIZE);
	glEnable(GL_COLOR_MATERIAL);
	Image* sta = loadBMP("stars.bmp");		staTexture = loadTexture(sta);
	delete sta;
	Image* sun = loadBMP("sun.bmp");		sunTexture = loadTexture(sun);
	delete sun;
	Image* mer = loadBMP("mercury.bmp");	merTexture = loadTexture(mer);
		delete mer;
	Image* ven = loadBMP("venus.bmp");		venTexture = loadTexture(ven);		delete ven;
```

```
        Image* ear = loadBMP("earth.bmp");          earTexture = loadTexture(ear);
        delete ear;
        Image* mar = loadBMP("mars.bmp");                marTexture =
loadTexture(mar);          delete mar;
        Image* jup = loadBMP("jupiter.bmp");        jupTexture = loadTexture(jup);
        delete jup;
        Image* sat = loadBMP("saturn.bmp");              satTexture =
loadTexture(sat);          delete sat;
        Image* ura = loadBMP("uranus.bmp");              uraTexture =
loadTexture(ura);          delete ura;
        Image* nep = loadBMP("neptune.bmp");     nepTexture = loadTexture(nep);
        delete nep;
        Image* plu = loadBMP("pluto.bmp");          pluTexture = loadTexture(plu);
        delete plu;
        Image* log = loadBMP("logo.bmp");           logTexture = loadTexture(log);
        delete log;
```

The used bmp files are in figure 2.4.

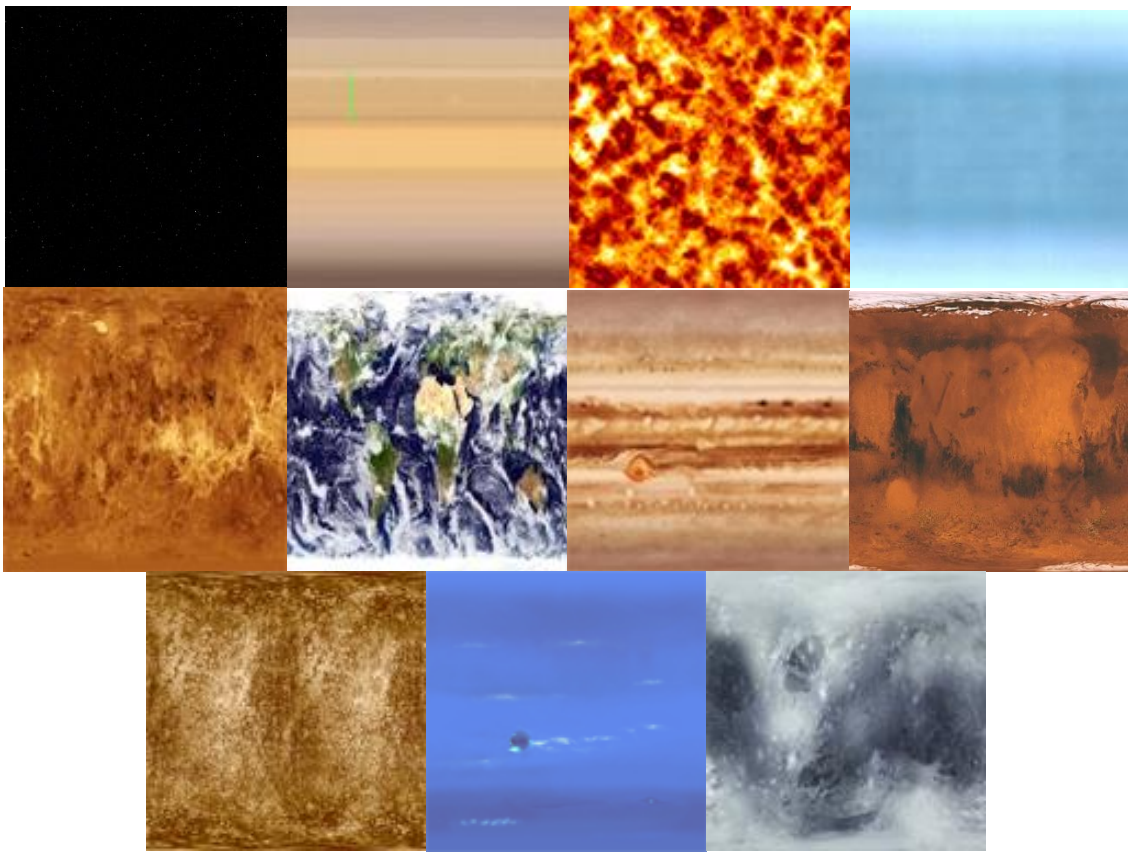


Figure 2.4 Planets textures

2.5 Modeling of the observer (camera), a view transformation

Let's say have constructed a scene (stationary objects) and want to see it from different angles. Just for this purpose, the gluLookAt () command is created, which will be even more useful if want to slide around the scene (animation). As such, there is no camera in OpenGL. There are functions for creating an artificial observer and objects. The view transformation positions and points the camera at the place where its object will be drawn. The gluLookAt () function, which consists of 9 parameters and is part of the utility library, is responsible for the camera vector.

9 camera parameters:

GLdouble eyex,GLdouble eyey,GLdouble eyez,GLdouble centerx,
GLdouble centery,GLdouble centerz,GLdouble upx,GLdouble upy, GLdouble upz,

Where:

- the first three arguments determine the location of the camera,
- the second three: where it is directed
- and the last three: which direction is considered the top one.

The upper direction vector sets the orientation of the camera. If gluLookAt() had not been called, the camera would have had the position and the orientation by it was set in default. It is known that the camera is always directed along the negative direction of the Z axis, like it is shown here: gluLookAt(0.0,0.0,0.0,0.0,0.0,-100.0,0.0,1.0,0.0);

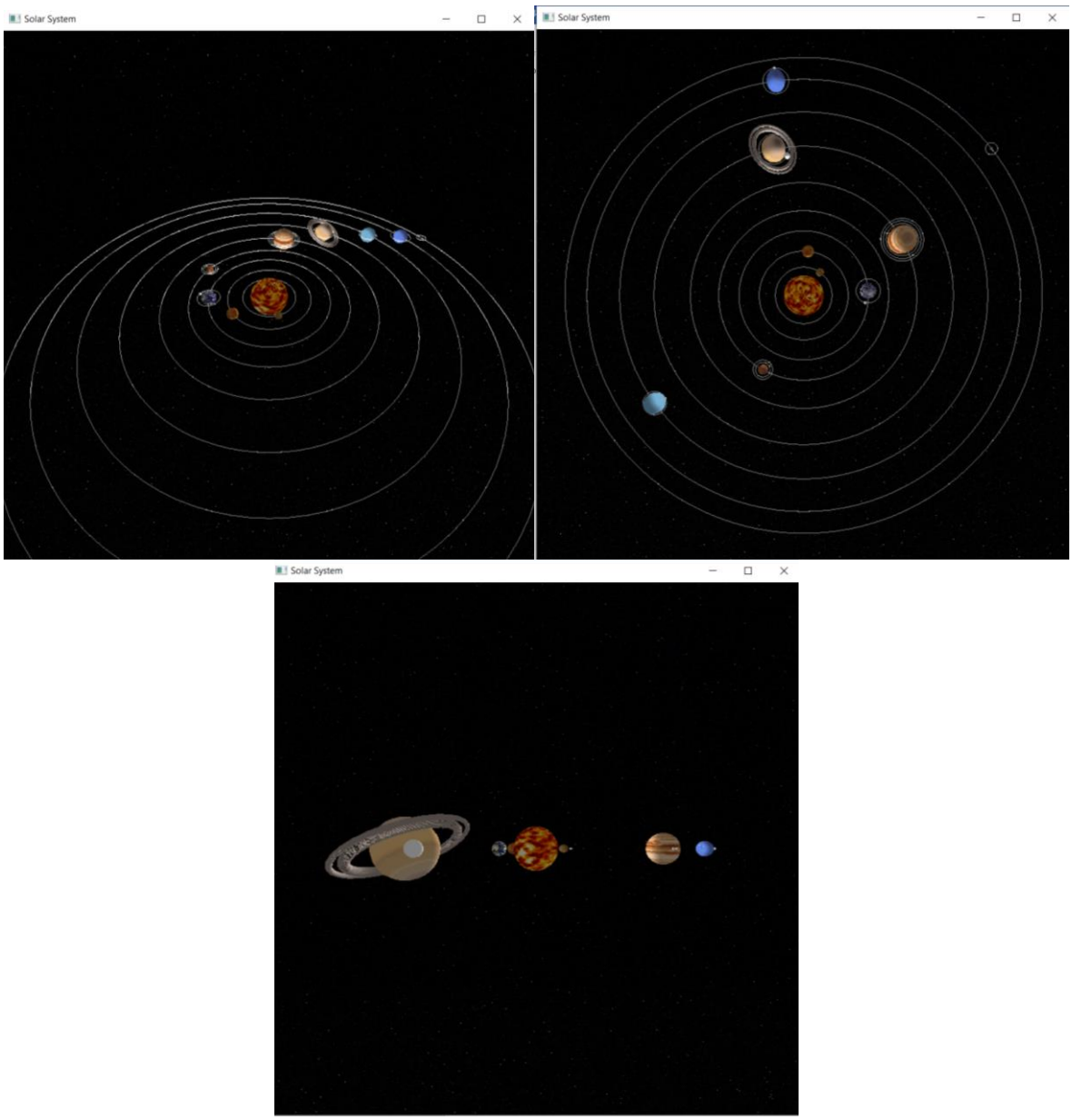
Note that the camera cannot be mounted on two tripods at the same time. When changing the camera position, it must call glLoadIdentity() to erase the effect of the current view transform.

The view transformation is enabled in the display() function. The smooth movement of the camera around the scene in a circle can be set by the following cycle:

```
 double i=0;
 gluLookAt(100*sin(i*3.14/180), 100*cos(i*3.14/180),1,….);
 glutSolidCube(12);// Drawing a scene
 i=i+1
if (changeCamera == 0)gluLookAt(0.0, zoom, 50.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
if (changeCamera == 1)gluLookAt(0.0, 0.0, zoom, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
if (changeCamera == 2)gluLookAt(0.0, zoom, 0.00001, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

void mouseWheel(int wheel, int direction, int x, int y)
{
        if (direction > 0 && zoom < 100) zoom++;
        if (direction < 0 && zoom > -75) zoom--;
        glutPostRedisplay();
}
```

The result of it in figure 2.5.



Figures 2.5 Three types of camera locations

Also, by using keyinput can be done several actions as written below:

```
void keyInput(unsigned char key, int x, int y){
	switch (key){
	case 27: exit(0); break;
	case ' ': if (isAnimate) isAnimate = 0; else{ isAnimate = 1; animate(1); } break;
	case 'o': if (smallOrbitActive) smallOrbitActive = 0; else smallOrbitActive = 1;
glutPostRedisplay(); break;
	case 'O': if (bigOrbitActive) bigOrbitActive = 0; else bigOrbitActive = 1;
glutPostRedisplay(); break;
	case 'm': if (moonsActive) moonsActive = 0; else moonsActive = 1;
glutPostRedisplay(); break;
```

```
        case 'M': if (moonsActive) moonsActive = 0; else moonsActive = 1;
glutPostRedisplay(); break;
        case 'l': if (labelsActive) labelsActive = 0; else labelsActive = 1;
glutPostRedisplay(); break;
        case 'L': if (labelsActive) labelsActive = 0; else labelsActive = 1;
glutPostRedisplay(); break;
        case '1': changeCamera = 0; glutPostRedisplay(); break;
        case '2': changeCamera = 1; glutPostRedisplay(); break;
        case '3': changeCamera = 2; glutPostRedisplay(); break;
        }
}
```

Instruction of the given actions are:

```
void intructions(void){
        cout << "SPACE to play/pause the simulation." << endl;
        cout << "ESC to exit the simulation." << endl;
        cout << "O to show/hide Big Orbital Trails." << endl;
        cout << "o to show/hide Small Orbital Trails." << endl;
        cout << "M/m to show/hide Moons." << endl;
        cout << "L/l to show/hide labels" << endl;
        cout << "1, 2 and 3 to change camera angles." << endl;
        cout << "Scroll to change camera movement" << endl;
}
```

Usage of it in figure 2.6.

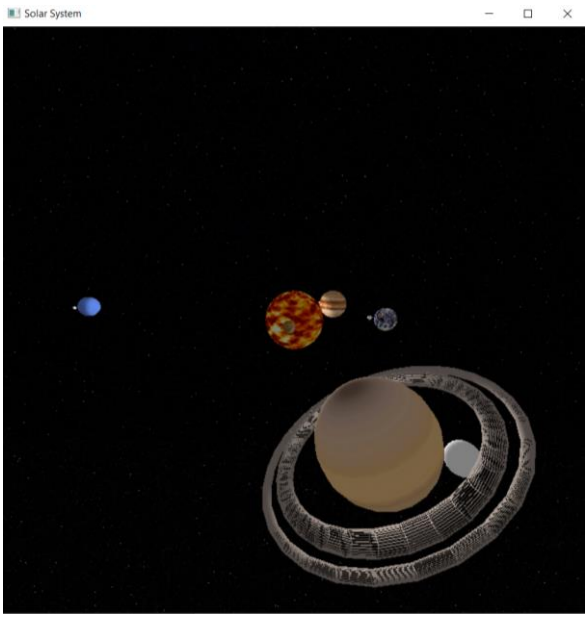


Figure 2.6 Using some keyinput commands

3 GAME MODELING

3.1 Snake game modeling

In this chapter, will be analyzed the game Snake. The game is considered very simple and it is not difficult to write it. Through this game, it intends to show the use of textures and other features of OpenGL.

Game Logic

The rules of the classic snake are simple:

- there is a field of cells where food randomly appears;
- there is a snake that moves all the time and that can be controled;
- the snake leaves its mark, that is, the trajectory.
- if the snake meets food on its way — the food disappears, appears in a new place, and the snake itself is lengthened by one cell;
- if the snake crashes into a wall or into itself, the game ends.
- the game starts again. [7]

To start writing code, let`s move by logic of the game. First, it is set the block size of the field. Next it is needed to define directions of the snake, that is why, write it like this:

```
static_cast<Direction>(rand() % 4))
```

Next, it is important to set block random appearance of fruit. To do so it is needed to send the x and y position of the fruit as well as setting x and y randomly. To move snake, first let's create it with fruit.

For creation of snake block need to know coordinates of given block, after sending those coordinates need to decide what form it will have.

First let's try simple Quads:

```
glBegin(GL_QUADS);
glVertex2f(x1, y1);
glVertex2f(x2, y1);
glVertex2f(x2, y2);
glVertex2f(x1, y2);
glEnd();
glColor3f(0.3, 0.1, 0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x1, y1);
glVertex2f(x2, y1);
glVertex2f(x2, y2);
glVertex2f(x1, y2);
glEnd();
```

The result of it was as in this figure 3.1:

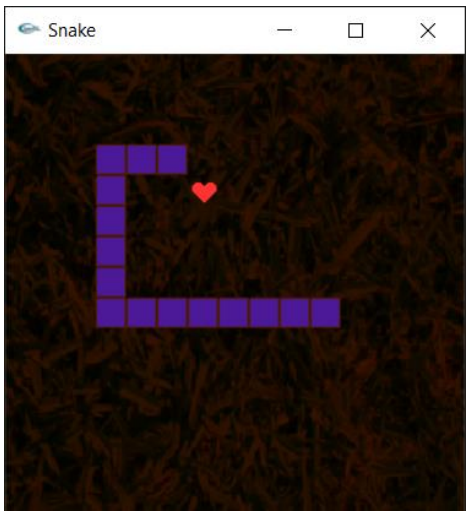
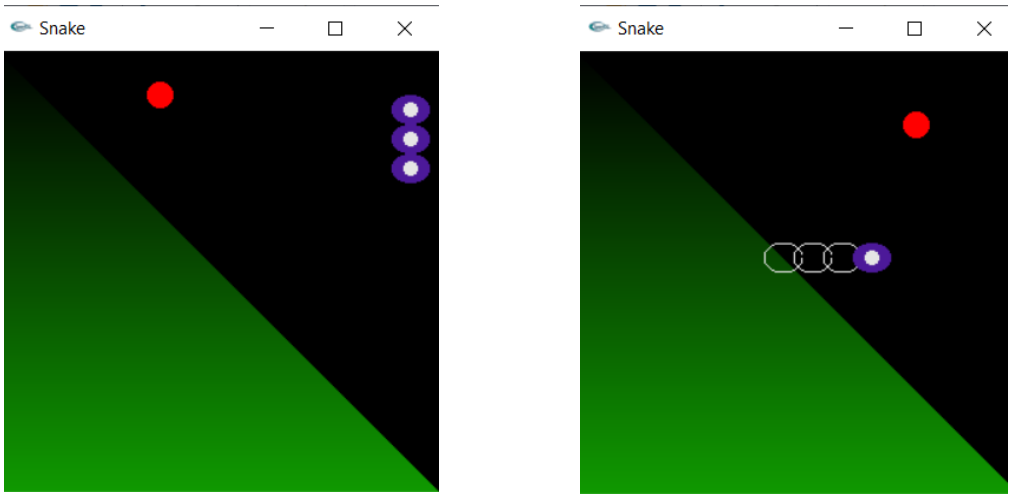


Figure 3.1 Creating a snake

Then try to make it as an ellipse by using GL_POLYGON:

```
glBegin(GL_POLYGON);
    for (int ii = 0; ii < 40; ii++)//head
    {
        //apply radius and offset
        glVertex2f(x * 13 + x1, y * 10 + y1);//output vertex
        //apply the rotation matrix
        t = x;
        x = c * x - s * y;
        y = s * t + c * y;
    }
    glEnd();
```

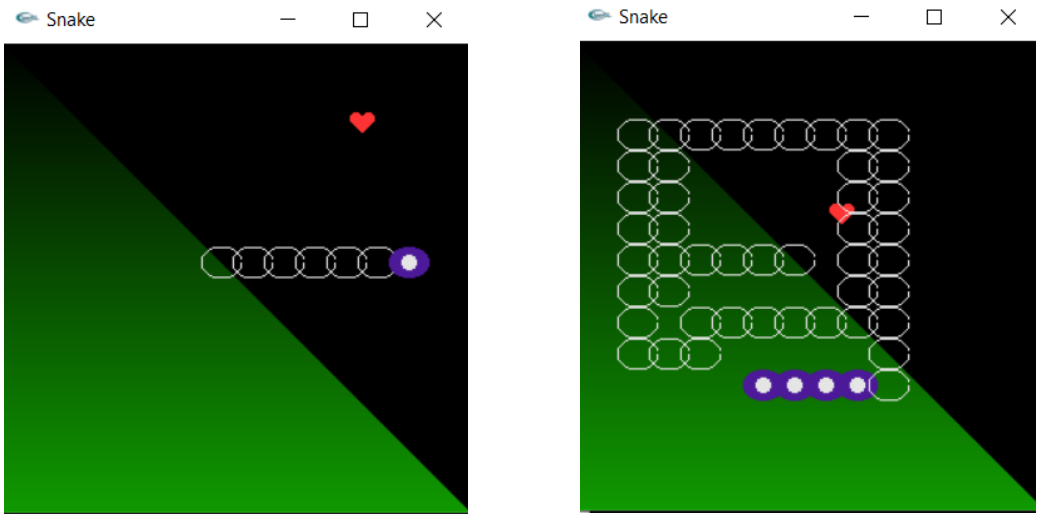
To show the trajectory of the snake use LINE_STRIP and connect the last position of the snake to the new position as in figures 3.2-3.3.



Figures 3.2 -3.3 The shape of the snake and its footprint

Also, to make the fruit different from the snake form, it was decided to take it as a heart, and the heart symbol equations was taken from wolfram Mathworld, the result is in figures 3.4-3.5.

```
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glBegin(GL_POLYGON);
glColor3f(1.0f, 0.2f, 0.2f);
for (int j = 0; j < 40; j++) {
        float const theta = 2.0f * 3.1415926f * (float)j / (float)40;
        float const x_ = scale * 16.0f * sinf(theta) * sinf(theta) * sinf(theta);
        float const y_ = -1 * scale * (13.0f * cosf(theta) - 5.0f * cosf(2.0f *
theta) - 2 * cosf(3.0f * theta) - cosf(4.0f * theta));
        glVertex2f(x + x_, y + y_);
}
glEnd();
```



Figures 3.4 -3.5 Fruit in the form of a heart

The motion of the snake is defined by it is direction, therefore it cannot go left, if it was going to right, as well as going to up while it was going to down.

```
case LEFT:
        if (lastMove_ == RIGHT)
                return;
        break;
case UP:
        if (lastMove_ == DOWN)
                return;
        break;
case RIGHT:
        if (lastMove_ == LEFT)
                return;
```

```
            break;
        case DOWN:
            if (lastMove_ == UP)
                return;
            break;
```

So, now it is clear how the snake moves. Next thing is needed to be considered is the block adding after meeting fruit by snake.

It is implemented by function, which calls block of snake, the pervious and new position of the snake.

```
setBlock(Type type, int x, int y)
```

By type is taken the snake, as an object, in this case as an ellipse. The trajectory was also added in the same way by using blocks.pop_back() to delete the last positions of the snake and leave only trajectory of the snake. The condition of adding new blocks is saying if snake does not meet fruit, then clear the last position of the snake and leave only the trajectory in the form of ellipse in STRIP_LINE and appearance of new fruit is written as if the condition is not working then add new block as well as new fruit:

```
        if (field.block(p.first, p.second) !=
                Field::FRUIT)
        {
                field.setBlock(Field::SNAKE_BLOCK, p.first, p.second);
                std::pair<int, int> p = blocks_.back();
                field.setBlock(Field::SNAKE_BLOCK_BACK, p.first, p.second);
                blocks_.pop_back();//to delete last positions of the snake
        }
        else
        {
                field.setBlock(Field::SNAKE_BLOCK, p.first, p.second);
                field.newFruit();
        }
```

It is important to write the condition when the snake cannot eat itself. To do so:

```
if (field.block(p.first, p.second) ==
                Field::SNAKE_BLOCK)
                return false;
```

The explanation of this condition is if the snake meets own block, then return false, which means the restart of the game.

The same condition is written for the frame of the window. Because it is considered to set frame as the obstacles:

```
if (p.first < 0 ||
                p.first >= Field::WIDTH ||
                p.second < 0 ||
                p.second >= Field::HEIGHT)
                return false;
```

Next interesting opportunity of OpenGL is the usage of texture. Let's use it on this game. To use the picture, need to load it to the project. Then define it as a char variable:

```
const char * grassFilename = "./sea_seamless.bmp";
```

Next stage is to load, for this, is used function GLuint loadBMP_custom(const char * imagepath) {}. This function helps to check the quality of selected picture and the correctness of the loading process. After recall this function in display() function, as it is shown here:

```
void display()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glPushAttrib(GL_ENABLE_BIT);
	glEnable(GL_TEXTURE_2D);
	glEnable(GL_BLEND);
	glEnable(GL_DEPTH_TEST);

	glBindTexture(GL_TEXTURE_2D, grassTexture);

	glPushMatrix();
	glBegin(GL_QUADS);

	glTexCoord2f(0, 0);
	glVertex2f(-310, 310);
	glTexCoord2f(1, 0);
	glVertex2f(310, 310);
	glTexCoord2f(1, 1);
	glVertex2f(310, -310);
	glTexCoord2f(0, 1);
	glVertex2f(-310, -310);

	glEnd();

	glPopMatrix();
	glPopAttrib();
	Painter p;
	game.draw(p);
	glutSwapBuffers();
}
```

And do not forget to write in int main(int argc, char **argv){} grassTexture to end the loading process. To control the snake is used the keyEvent as it is seen below:

```
void keyEvent(int key, int, int)
{
```

```
        switch (key)
        {
        case GLUT_KEY_LEFT:
                game.keyEvent(Snake::LEFT);
                break;
        case GLUT_KEY_UP:
                game.keyEvent(Snake::UP);
                break;
        case GLUT_KEY_RIGHT:
                game.keyEvent(Snake::RIGHT);
                break;
        case GLUT_KEY_DOWN:
                game.keyEvent(Snake::DOWN);
                break;
        }
}
```

To run all the written project is needed to put the time by using glutTimerFunc(300, timer, 0).

```
void timer(int = 0)
{
        game.tick();
        display();
        glutTimerFunc(300, timer, 0);
}
```

Here 300 is the milliseconds, timer is function recalling, and 0 is the initial value. So, it helps to run the process. Also in timer function was called game.tick() which defines the snake motion by directions and other conditions which were considered above.

```
bool Snake::tick(Field &field)
{
        lastMove_ = snake_direction;
        std::pair<int, int> p = blocks_.front();
        switch (snake_direction)
        {
        case LEFT:
                p.first--;
                break;
        case UP:
                p.second--;
                break;
        case RIGHT:
                p.first++;
```

```
            break;
        case DOWN:
            p.second++;
            break;
        }
        if (p.first < 0 ||
            p.first >= Field::WIDTH ||
            p.second < 0 ||
            p.second >= Field::HEIGHT)
            return false;
        if (field.block(p.first, p.second) ==
            Field::SNAKE_BLOCK)
            return false;

        blocks_.push_front(p);
        if (field.block(p.first, p.second) !=
            Field::FRUIT)
        {
            field.setBlock(Field::SNAKE_BLOCK, p.first, p.second);
            std::pair<int, int> p = blocks_.back();
            field.setBlock(Field::SNAKE_BLOCK_BACK, p.first, p.second);
            blocks_.pop_back();//to delete last positions of the snake
        }
        else
        {
            field.setBlock(Field::SNAKE_BLOCK, p.first, p.second);
            field.newFruit();
        }
        if (blocks_.size() >= Field::WIDTH * Field::HEIGHT - 1)
            return false;
        return true;
}
```

After added to the scale of the heart and expanded as a line, see figure 3.6:

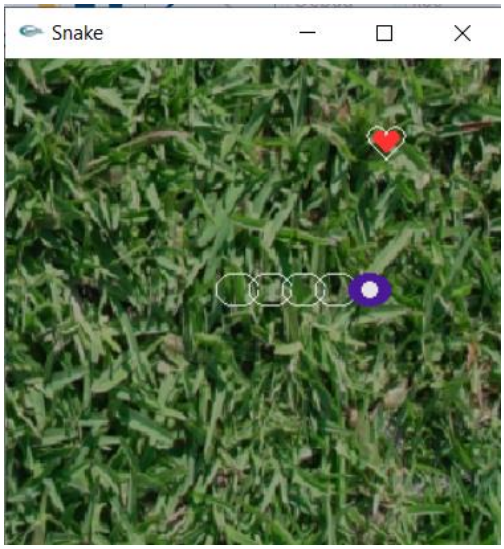


Figure 3.6 Magnification and circling of the heart

The result of made work in Appendix B is in figures 3.7-3.8:

Figures 3.7 -3.8 The final result of the Snake

3.2 Tetris game modeling

The most known game from 80th is Tetris game, who was born in 80th -90th know this game and surely played this game. So, the history of this game starts in 1985 and it was invented by Russian inventor Alexey Pazhitnov. Of course, the game was simultaneously in use and got popularity, by in this wave of popularity the game was upgraded many times. But the main logic and aim of the game was kept.

So, let's describe the aim of the game, this program displays a random drop of various shapes on the screen. The input data is the input of options for the speed of

movement of the figures from top to bottom and the control of falling figures. Dedication of the game is to improve skills like logic, concentration, and coordination.

There are many ways to implement this program. They can be divided by functionality:

1) mathematical description of the movements of the figures (left, right, drop, rotation);

2) graphical illustration of the movements of the figures on the screen (for this will be used OpenGL);

Let's start the game implementation with mathematical description, here will be considered elementary physic laws of movement of figures, as it was mentioned going left, right, drop, and rotation. It is believed the main part of the program is laid on the movement of figures and correctly program running is also depended on it. There are can be used a lot of ways to implement the game by certain algorithm. Let's consider the description of the movement of the figure with two linear functions and one of them will be for figure which is position is horizontally and the next linear function will be for the figure which is position is vertically.

```
void Game::MoveX(int delta) {
if (!blockFalling) return; //don't move if there's no piece falling
//if moving right, need to check block after last block
int rowDisplacement = delta>0 ? curBlockWidth : 0;
delta += posX;
for (int i = 0; i < BLOCK_SIDE_MAX; i++) {
	for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
		//...check if it can move to it's adjacent neighbor
		//don't check empty blocks
		if (block[i][j] == Empty_Block) continue;

		if (((delta + i < 0)) || (delta + i >= COLS) //if the X position cannot move further left / right
				|| ((grid[delta + rowDisplacement][posY + j] != Empty_Block))) { //OR if the position this brick segment is moving to isn't free
			return; //don't move the block
		}
	}
}
//update block[][]'s position within grid[][]
for (int i = 0; i < BLOCK_SIDE_MAX; i++) { //from bottom left (posX, posY)...
	for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
		if (block[i][j] == blockTypeCurrent) {
			grid[delta + i][posY + j] = block[i][j]; //move block piece to new location
```

```
                        grid[posX + i][posY + j] = Empty_Block; //replace old
location with an empty block
                    }
                }
            }
            posX = delta;
}

/**
* Move the current falling block downwards by 1 grid space
**/
void Game::MoveY() {
        //move block down by 1 grid space
        //by starting from bottom of 4x4 and checking up each column, stopping at first
taken grid space
        //able to check if next place for collision
        //check if the block can fall by 1
        bool foundBottom = false;
        for (int i = 0; i < BLOCK_SIDE_MAX; i++) { //from bottom left (posX,
posY)...
                for (int j = 0; j < BLOCK_SIDE_MAX; j++) {

                        //don't care about empty spaces in block
                        if (block[i][j] == Empty_Block || foundBottom) continue;

                        //must be at location of first solid space in
block[currentColumn][j]
                        //so check if this can move down one space
                        //also check for out of bounds
                        if (grid[posX + i][posY + j - 1] != Empty_Block || (posY + j) <=
0) {
                                blockFalling = false;
                                if (posY > ROWS - BLOCK_SIDE_MAX -
curBlockHeight) gameOver = true; //if it stops above the line, it's game over
                                return;
                        }
                        else {
                                foundBottom = true;
                        }
                }
                foundBottom = false;
        }
```

```
        //update block[][]'s position within grid[][]
        for (int i = 0; i < BLOCK_SIDE_MAX; i++) { //from bottom left (posX,
posY)...
                for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
                        if (block[i][j] == blockTypeCurrent) {
                                grid[posX + i][posY + j - 1] = block[i][j];
                                grid[posX + i][posY + j] = Empty_Block;
                        }
                }
        }
        posY -= BLOCK_VERT_SPEED;
    }
```

After defining functions, the changing the given values of variables in certain time period, the figure will be changed, for example rotated. The example of rotation is given below, figure 3.9:

```
/**
* Rotate the current shape (in-place matrix rotation of a square, triangle swap trick)
**/
void Game::Rotate(int theta) {
        if (blockTypeCurrent == Square_Block) return; //don't bother rotating squares
don't flip if too close to bounds
        if ((posX + curBlockHeight > COLS) || (posY - curBlockWidth < 0)) return;

        int tempArr[BLOCK_SIDE_MAX][BLOCK_SIDE_MAX];

        //clear and update block within grid to remove residuals
        for (int i = 0; i < BLOCK_SIDE_MAX; i++) { //from bottom left (posX,
posY)...
                for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
                        if (block[i][j] != Empty_Block) {
                                grid[posX + i][posY + j] = Empty_Block;
                        }
                }
        }

        //rotate matrix in place
        for (int i = 0; i < BLOCK_SIDE_MAX; i++) {
                for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
                        tempArr[j][BLOCK_SIDE_MAX - i - 1] = block[i][j];
                }
        }
```

```
//if tempArr rotated to a bad spot, throw it away
for (int i = 0; i < BLOCK_SIDE_MAX; i++) {
        for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
                if (tempArr[i][j] != Empty_Block) {
                        if (grid[posX + i][posY + j - 1] != Empty_Block) {
                                //no need to dealoc, tempArr on stack
                                return;
                        }
                }
        }
}

//swap the current block's width and height
int temp = curBlockWidth;
curBlockWidth = curBlockHeight;
curBlockHeight = temp;

//copy over to block & put into grid
//no need to dealoc, tempArr on stack
for (int i = 0; i < BLOCK_SIDE_MAX; i++) {
        for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
                block[i][j] = tempArr[i][j];
                grid[posX + i][posY + j] = block[i][j];
        }
}}
```

Figure 3.9 Rotation of the shape

Let’s define next function by logic of the game, which will be responsible for clearing filled rows. In process of game implementation was the difficulty of this function is to declare the large number of variables which are be responsible for already fallen figures description, figure 3.10.

```
/**
* Clear out block[][]
**/
void Game::ClearBlockGrid() {
	for (int i = 0; i < BLOCK_SIDE_MAX; i++) {
		for (int j = 0; j < BLOCK_SIDE_MAX; j++) {
			block[i][j] = Empty_Block;
		}
	}
}

/**
* Clear out the grid of all blocks as well as the current falling block
**/
void Game::ClearGrid() {
	for (int c = 0; c < COLS; c++) {
		for (int r = 0; r < ROWS; r++) {
			grid[c][r] = Empty_Block;
		}
	}
	ClearBlockGrid();
}
```

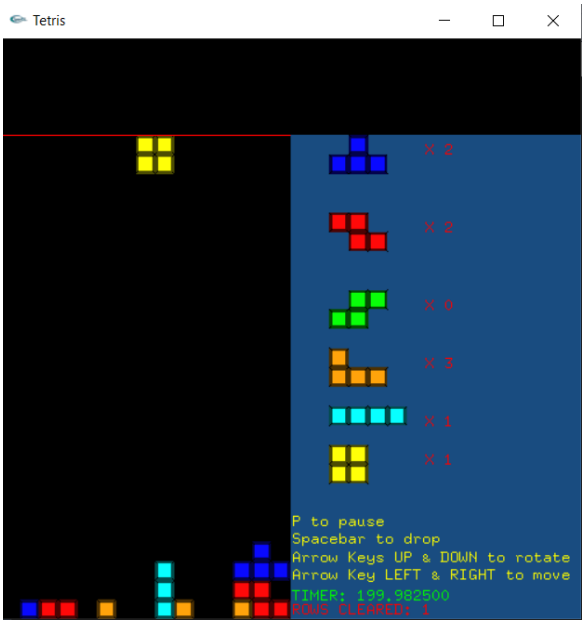


Figure 3.10 After clearing filled row

Another way to mathematically describe the movement of shapes and fill in the Tetris field is to create a two-dimensional matrix, where n will be the row and k will be the column. After a certain period of time, the values corresponding to the position

of the figures on the plane and the already fallen figures will change. To be sure, the nth row is corresponded to horizontal figures, as well as the kth column is corresponded for vertical figures.

Here the above-mentioned methods just one of the implementations of the game, it means that they are just popular ones that are widely used now.

Next, let's consider the graphical illustration of the game, and there are bunch of methods to do it. It is used to display figures and fields of the game. Let's give an example for more clearness, in Tetris game can be used ready shaped figures on the plane, but the drawback it is to collect all the shapes in the memory and it will influence to the running of the program as well. Also, another drawback is the rotation of the figure and to display the already fallen figures. In order to do it, it is needed to create every time a new figure which is field drawing, so as it was said above it will overload the computer memory as well.

So, in this case considered to find another way to implement the shapes, and here another way to graphically display Tetris shapes. Let's consider that all field is celled and the only way to shape figures is to color the cells by the movement of the figure, it sounds a little bit complicated, but it is better than loading every time from memory randomly shapes. So, there is one drawback of this graphical illustration is to work with huge number of elements in order to declare the movement of the shape and of course its disappearance when it is filled by row fully.

Next, interesting graphical illustration method is to use built-in graphical shapes as it is declared in programming language in this case in C++.

So, the next method of illustration is to create the square in order to draw certain shape of Tetris game, and for it just need to define the the upper-left corner, as well as the value of the width of the square, see figure 3.11.

```
case(Rectangle_Block):
            curBlockWidth = 1;
            curBlockHeight = 4;
            block[0][0] = Rectangle_Block;  //bottom left
            block[0][1] = Rectangle_Block;
            block[0][2] = Rectangle_Block;
            block[0][3] = Rectangle_Block;  //top
            break;
      case(L_Block):
            // X
            // XXX
            curBlockWidth = 3;
            curBlockHeight = 2;
            block[0][0] = L_Block; //lower left
            block[0][1] = L_Block; //top of L
            block[1][0] = L_Block; //mid section of long part
            block[2][0] = L_Block; //right most section of long part
            break;
```

```
case(Squigly_Block):
        //  XX
        // XX
        curBlockWidth = 3;
        curBlockHeight = 2;
        block[0][0] = Squigly_Block; //lower left
        block[1][0] = Squigly_Block; //lower right
        block[1][1] = Squigly_Block; //upper left
        block[2][1] = Squigly_Block; //upper right
        break;
case(RevSquigly_Block):
        // XX
        //  XX
        curBlockWidth = 3;
        curBlockHeight = 2;
        block[0][1] = RevSquigly_Block; //upper left
        block[1][1] = RevSquigly_Block; //upper right
        block[1][0] = RevSquigly_Block; //lower left
        block[2][0] = RevSquigly_Block; //lower right
        break;
case(T_Block):
        //  X
        // XXX
        curBlockWidth = 3;
        curBlockHeight = 2;
        block[0][0] = T_Block; //lower left
        block[1][0] = T_Block; //lower mid
        block[1][1] = T_Block; //upper mid
        block[2][0] = T_Block; //lower right
        break;
default: //AKA Square
        curBlockWidth = 2;
        curBlockHeight = 2;
        block[0][0] = Square_Block;  //lower left
        block[0][1] = Square_Block;  //top left
        block[1][0] = Square_Block;  //lower right
        block[1][1] = Square_Block;  //top right
        break;};
```

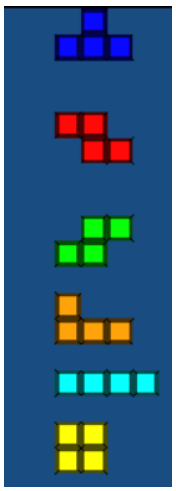


Figure 3.11 Shapes

So, let's move to the game application and how to run it. First, we have considered the mathematical illustration and after the graphical illustration of it, now it is time to apply it on program and run it. To do so it is important to set the time, to be sure that the program will run by time and it influences on the figure movements as well. So next thing is needed to be done is to declare two-dimensional array with empty values, because it has to be empty in order to fill it by time with elements.

```
/**
* Constructor; reset all vars and set timer to intial value
**/
Game::Game()
{
	Restart();
	timer = INIT_TIMER;
	onlyRectangleCheat = false;
	for (int i = 0; i < NUM_SHAPES; i++) {
		blockCounter[i] = 0;
	}
}
glutTimerFunc(game.timer, timer, 0);
```

Next, the shape is selected using a random number generator.

```
void seedRand() {
	srand(time(0));
}
```

The most important part of this game implementation and not only this game and for others is the controlling part. It is needed to be able to control from the beginning of the timer and needed to define control forms. As it was said, it involves to the creation of a user interface, in this case they are output and input data. Here the input data will be given by user and output data will be taken from the program to the user. Let's come closer and consider closer, here in Tetris game the input data is presenting

the actions that are needed for the user, as to move the shapes by pressing the keys Left, Right, Up, Space;

In accordance with the selected shape, the first two rows of the array are filled in. Then there is an entry into the fall cycle of the figure. The body of the loop is a shift of the shape one line below and a check for pressing the buttons "Right", "Left", "Rotate" (when these buttons are pressed, the shape is shifted to the right, left and rotated around its axis, respectively). Additionally, pressing the Pause button was added, see figure 3.12.

```
// Handes standard keyboard input (characters, space bar)
void Keyboard(unsigned char key, int x, int y) {
	if (!game.gameOver) {
		if (key == 'p') {
			game.paused = !game.paused;
		}
		else if ((key == ' ') && !game.paused) {
			while (game.blockFalling) {
				game.MoveY();
			}
			CheckRowClear();}}}
```

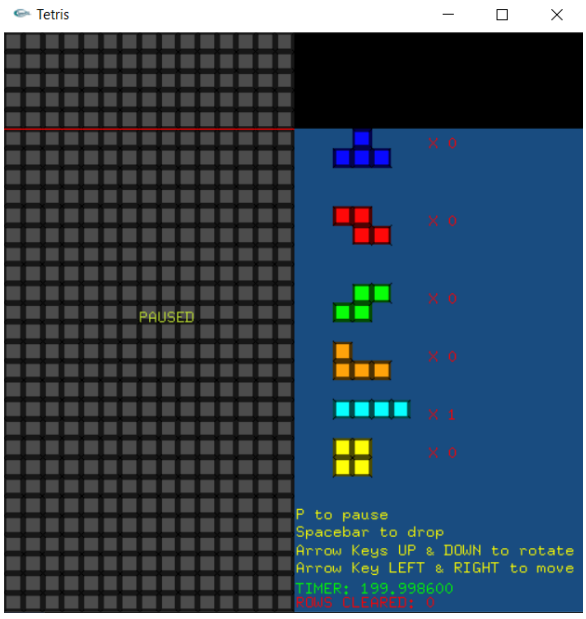


Figure 3.12 Game pausing

```
/**
* Handles special key input (arrow keys)
**/
void Special(int key, int x, int y) {
	//don't accept movement input while paused, gameOver or clearing row
	if (!game.paused && !game.gameOver && !isRowDestroying) {
		switch (key) {
```

```
            case GLUT_KEY_LEFT:
                game.MoveX(BLOCK_HORIZ_SPEED * -1);
                break;
            case GLUT_KEY_RIGHT:
                game.MoveX(BLOCK_HORIZ_SPEED);
                break;
            case GLUT_KEY_DOWN:
                game.Rotate(ROTATE_SHAPE_RIGHT);
                game.Rotate(ROTATE_SHAPE_RIGHT);
                game.Rotate(ROTATE_SHAPE_RIGHT);
                break;
            case GLUT_KEY_UP:
                game.Rotate(ROTATE_SHAPE_LEFT);
                break;
            default:
                break;
        }}}
```

The condition for the end of the cycle is that there is no free space under the shape.

```
if (posY > ROWS - BLOCK_SIDE_MAX - curBlockHeight) gameOver = true;
//game over case
if (posY > ROWS - 5) gameOver = true;
```

After exiting the loop, the check of fully filled rows begins. If there are any, then they are cleared and all the rows above are shifted.

```
void Game::CollapseRow(int row) {
    for (int i = row; i < ROWS - BLOCK_SIDE_MAX - 2; i++) {
        for (int j = 0; j < COLS; j++) {
            //from the bottom, for every block, nothing below it?
            //make it fall
            if (grid[j][i] == Empty_Block) {
                int temp = grid[j][i + 1];
                grid[j][i] = temp;
                grid[j][i + 1] = Empty_Block;
        }}}}
```

Then there is a new selection of the shape and the algorithm repeats, the source code in Appendix C, see figure 3.13.[8]

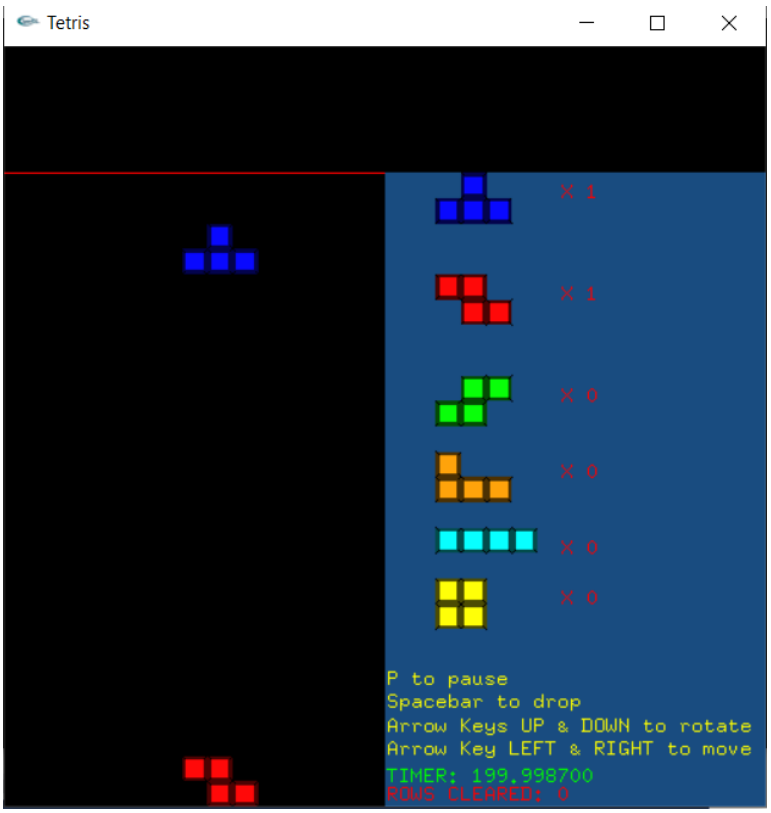


Figure 3.13 Result of the Tetris game coding

3.3 Breakout game modeling

Breakout is a successful arcade game released in 1976 by Atari. The goal of the game is to break several rows of bricks at the top of the screen using a ball and a small racket at the bottom of the screen. This game will use a structure of two floating-point numbers to represent the position on the screen.

The main object in the game will be the ball. The ball, figure 3.14, will have a position and speed represented by a two-dimensional structure.

```
void Breakout::newBall(float x = -10, float y = -10) {
        Ball b1;
        if (x < 0 || y < 0) {
                b1.xpos = WINWIDTH / 2.0;
                b1.ypos = WINHEIGHT - 30.0f;
        }
        else {
                b1.xpos = x;
                b1.ypos = y;
        }

        if (level == 1) {
```

```
        if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 2.5f;
        else
                b1.xvel = -2.5f;
b1.yvel = -5.0f;
}
else if (level == 2) {
        if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 3.5f;
        else
                b1.xvel = -3.5f;
        b1.yvel = -5.5f;
}
else if (level == 3) {
        if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 4.0f;
        else
                b1.xvel = -4.0f;
        b1.yvel = -6.0f;
}
else if (level == 4) {
        if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 4.0f;
        else
                b1.xvel = -4.0f;
        b1.yvel = -6.5f;
}
else if (level == 5) {
        if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 4.0f;
        else
                b1.xvel = -4.0f;
        b1.yvel = -7.0f;
}
else if (level == 6) {
        if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 4.0f;
        else
                b1.xvel = -4.0f;
        b1.yvel = -7.5f;
}
else if (level == 7) {
        if ((float)rand() / (RAND_MAX) < 0.5)
```

```
                b1.xvel = 4.0f;
            else
                b1.xvel = -4.0f;
            b1.yvel = -8.0f;
        }
        else if (level == 8) {
            if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 4.0f;
            else
                b1.xvel = -4.0f;
            b1.yvel = -8.5f;
        }
        else if (level == 9) {
            if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 4.0f;
            else
                b1.xvel = -4.0f;
            b1.yvel = -9.5f;
        }
        else if (level == 10) {
            if ((float)rand() / (RAND_MAX) < 0.5)
                b1.xvel = 5.0f;
            else
                b1.xvel = -5.0f;
            b1.yvel = -10.0f;
        }

        b1.radius = BALL_RADIUS;
        b1.r = 0.45f + (float)rand() / (RAND_MAX);
        b1.g = 0.25f + (float)rand() / (RAND_MAX);
        b1.b = 0.4f + (float)rand() / (RAND_MAX);
        balls.push_back(b1);
    }
```

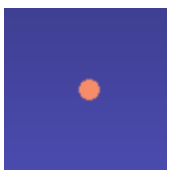


Figure 3.14 Ball, it can be in different color

The ball will bounce off the top, left, and right sides of the screen, but not the bottom side.

On the bottom side there will be a racket, figure 3.15, that the user can move horizontally with the mouse and keystroke.

```
void Breakout::mouseMove(int x, int y) {}

void Breakout::specialKeyPos(int key, int x, int y) {
	switch (key)
	{
	case GLUT_KEY_LEFT:
		if (paddle.xpos > 0) {
			paddle.xpos -= 5.0f;
			paddle.xpos -= 5.0f;
			glutPostRedisplay();
			paddle.xpos -= 5.0f;
			paddle.xpos -= 5.0f;
			glutPostRedisplay();
		}
		break;
	case GLUT_KEY_RIGHT:
		if (paddle.xpos + paddle.width < WINWIDTH) {
			paddle.xpos += 5.0f;
			paddle.xpos += 5.0f;
			glutPostRedisplay();
			paddle.xpos += 5.0f;
			paddle.xpos += 5.0f;
			glutPostRedisplay();
		}
		break;
	default:
		break;
	}
}
```

Figure 3.15 Paddle

The ball will be able to bounce off the racket and hit the bricks and bounce. The ball will be able to hit the bricks and bounce back, but it gets a bit more complicated because the brick can be hit from any direction, so will need to a bit of actual collision detection here.

Practical all games consist of 3 parts:

1) Collecting and processing user input

2) Game Logic

3) Drawing

For the first part, it is needed to capture the position of the mouse on the screen on the X-axis. The position of the racket on the Y-axis will be fixed. Another input to capture is a left-click to create a new ball after it goes off-screen.

```
void Breakout::mouseClick(int button, int state, int x, int y) {
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && lifesCount>0 && level<11) {
		newBall(x, y);
	}
	glutPostRedisplay();
}
```

For the second part, need to analyze the movement of the ball. To do this, take the current 2D position and add the 2D speed. Since the speed can also be negative, it should work in either direction. In order to detect when the ball hits the top of the screen, check the Y part of the ball's position. If Y is lower than the radius of the ball, make sure the Y part of the speed becomes positive. To check if the ball hits the left or right side of the screen, check the X part of the ball and compare it with the screen size, minus the ball's radius.

```
// Set new position
		it->xpos += it->xvel;
		it->ypos += it->yvel;

		// Collision with left/right/top window sides
		if ((it->xpos <= (2 * it->radius)) || (it->xpos >= (WINWIDTH - 2 * it->radius))) {
			it->xvel *= -1;
		}
		if ((it->ypos <= (2 * it->radius))) {
			it->yvel *= -1;
		}
		if (it->ypos >= (WINHEIGHT - 2 * it->radius)) {
			it = balls.erase(it);
			continue;
		}
```

To move the racket at the bottom of the screen, the mouse position on the X-axis is considered. And set the paddle's X position to that. Just need to make sure the paddle does not go outside the screen, see figure 3.16.

```
	y = WINHEIGHT - y;
	if (x - paddle.width / 2.0f >= 0 && x + paddle.width / 2.0f <= WINWIDTH) {
		paddle.xpos = x - paddle.width / 2.0f;
	}
```

```
else if (x - paddle.width / 2.0f <= 0) {
        paddle.xpos = 0;
}
else if (x + paddle.width / 2.0f >= WINWIDTH) {
        paddle.xpos = WINWIDTH - paddle.width;
}
```

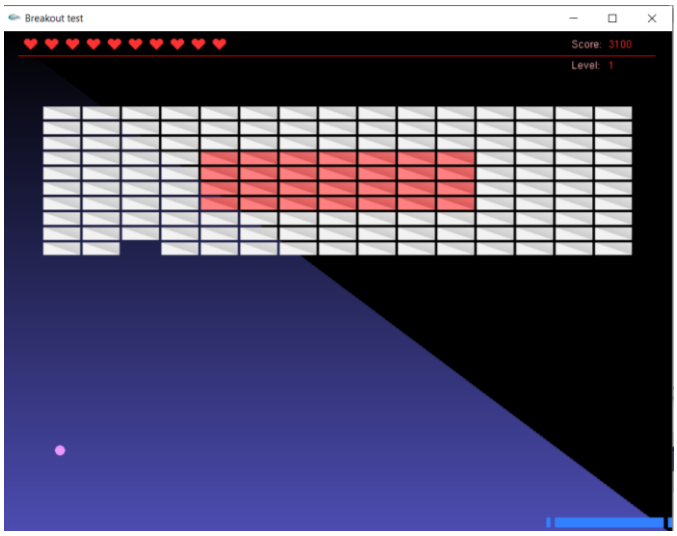


Figure 3.16 Paddle cannot go outside of the window

The next element to focus on is to check whether the ball bounces off the racket. This is similar to the ball hitting the walls, with 2 main differences. One is that only bounce it if the ball hits the paddle. So, it has to be checked if the X position of the ball is inside the paddle extremities. The other difference is that to make player to control the ball's direction after hitting the paddle, and to do that by changing the X part of the ball's speed, depending on where the ball hits the paddle. When the ball goes outside the screen on the Y axis, I will just block any logic in the engine, until the player clicks the mouse, which will reset the ball.

```
        // Check collision between paddle's top edge and bottom point on circle
        if (it->xpos >= paddle.xpos && it->xpos <= paddle.xpos +
paddle.width) {
                if ((it->ypos + it->radius - paddle.ypos) >= -10 && (it->ypos + it-
>radius - paddle.ypos) <= 0) {
                        it->yvel *= -1;
                        reward = 100;
                        score += reward;
                        continue;
                }
        }
```

Next, the collision of the ball with a brick is considered. To simplify collision detection, it will be assumed that the ball is square off-screen, not round. To detect a

collision, need to compare the sides of the ball and the brick. If the ball's top is above the brick's bottom and at the same time the ball's bottom is below the brick's top, have an overlap on the Y axis. Keep in mind the y axis goes from top 0 to bottom 600 on the graphical screen. If the ball's left side is to the left of the brick's right side and then at the same time the ball's right side is at the right of the brick's left side, have an overlap on the X axis. If both conditions are met, have collision. Now, to detect on which side the collision occurred, it is needed to find the smallest of the 4 segments, see the result in figure 3.17.

```
            // Collision with the bricks
            for (std::vector<Brick>::iterator br = bricks.begin(); br != bricks.end(); )
{
                  // Check collision between circle and vertical brick sides
                  if (it->ypos >= br->ypos && it->ypos <= br->ypos + br->height)
{
                        // brick right edge and left point on circle
                        if ((it->xpos - it->radius - br->xpos - br->width) <= 5 &&
(it->xpos - it->radius - br->xpos - br->width) >= 0) {
                              it->xvel *= -1;
                              br = hitBrick(br);
                              continue;
                        }

                        // brick left edge and right point on circle
                        if ((it->xpos + it->radius - br->xpos) >= -5 && (it->xpos +
it->radius - br->xpos) <= 0) {
                              it->xvel *= -1;
                              br = hitBrick(br);
                              continue;
                        }
                  }

                  // Check collision between circle and horizontal brick sides
                  if (it->xpos >= br->xpos && it->xpos <= br->xpos + br->width) {
                        // brick bottom edge and top point on circle
                        if ((it->ypos - it->radius - br->ypos - br->height) <= 10 &&
(it->ypos - it->radius - br->ypos - br->height) >= 0) {
                              it->yvel *= -1;
                              br = hitBrick(br);
                              continue;
                        }

                        // brick top edge and bottom point on circle
```

```
                        if ((it->ypos + it->radius - br->ypos) >= -10 && (it->ypos +
it->radius - br->ypos) <= 0) {
                                it->yvel *= -1;
                                br = hitBrick(br);
                                continue;
                        }
                }

                GLfloat d;
                // Check collision with top left corner
                d = pow((it->xpos - br->xpos), 2.0) + pow((it->ypos - br->ypos),
2.0);
                if (d < it->radius + 5.0) {
                        it->xvel *= -1;
                        it->yvel *= -1;
                        br = hitBrick(br);
                        continue;
                }

                // Check collision with top right corner
                d = pow((it->xpos - br->xpos - br->width), 2.0) + pow((it->ypos -
br->ypos), 2.0);
                if (d < it->radius + 5.0) {
                        it->xvel *= -1;
                        it->yvel *= -1;
                        br = hitBrick(br);
                        continue;
                }

                // Check collision with bottom left corner
                d = pow((it->xpos - br->xpos), 2.0) + pow((it->ypos - br->ypos -
br->height), 2.0);
                if (d < it->radius + 5.0) {
                        it->xvel *= -1;
                        it->yvel *= -1;
                        br = hitBrick(br);
                        continue;
                }

                // Check collision with bottom right corner
                d = pow((it->xpos - br->xpos - br->width), 2.0) + pow((it->ypos -
br->ypos - br->height), 2.0);
                if (d < it->radius + 5.0) {
```

```
                it->xvel *= -1;
                it->yvel *= -1;
                br = hitBrick(br);
                continue;
            }

            ++br; // next brick
        }
```

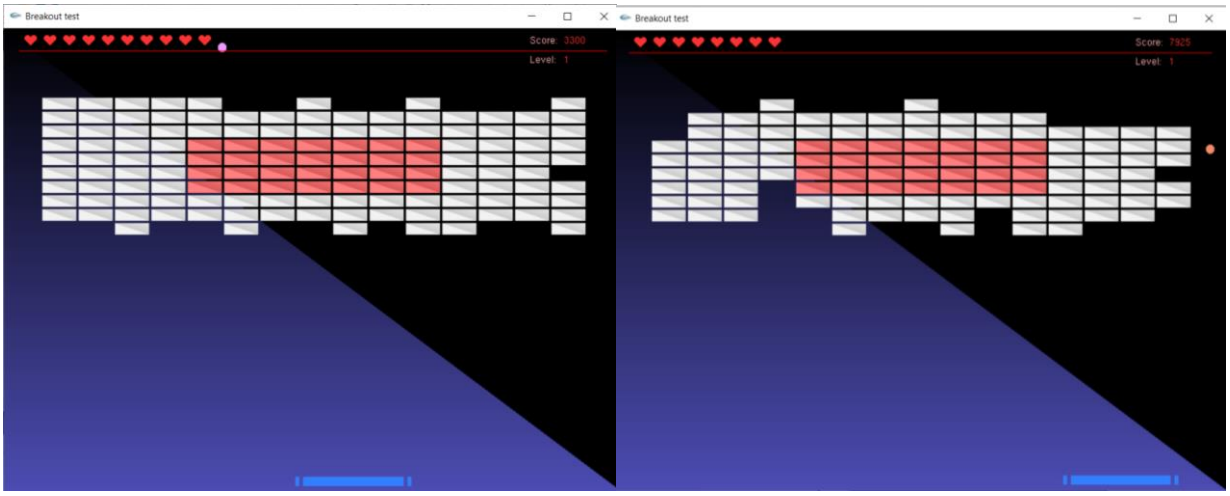


Figure 3.17 Collision with the bricks

The last part of the game is drawing: ball, racket, bricks.

To make the game more complicated were added bricks with different levels of health, as it is shown here:

```
Iterator Breakout::hitBrick(Iterator brick) {
    score += reward;
    reward += 25;

    // Decrease brick health
    if (brick->health == 2) {
        brick->r = 0.95f;
        brick->g = 0.95f;
        brick->b = 0.95f;
        brick->health -= 1;
        return ++brick;
    }else  if (brick->health == 3) {
        brick->r = 1.0f;
        brick->g = 0.9f;
        brick->b = 0.1f;
        brick->health -= 1;
        return ++brick;
```

```
        }
        else    if (brick->health == 4) {
                brick->r = 1.0f;
                brick->g = 0.0f;
                brick->b = 0.1f;
                brick->health -= 1;
                return ++brick;
        }
        else {
                return bricks.erase(brick);
        }
}
```

 -1 health

 -2 health

 -3 health

 -4 health

Also, were added 10 levels by using different bricks. For 1st level bricks positions and bricks health are defined by condition, figure 3.18:

```
void Breakout::bricksLevel1(void) {
        Brick newBrick;
        newBrick.r = 0.95f;
        newBrick.g = 0.95f;
        newBrick.b = 0.95f;
        newBrick.health = 1;
        newBrick.width = (WALLWIDTH - (WALLCOLS - 2) * WALLSPACE) / WALLCOLS;
        newBrick.height = (WALLHEIGHT - (WALLROWS - 2) * WALLSPACE) / WALLROWS;

        for (int i = 0; i < WALLROWS; ++i) {
                for (int j = 0; j < WALLCOLS; ++j) {
                        // Set stronger bricks
                        if (i + 1 > ceil(WALLROWS / 2.0) - 2 && i < ceil(WALLROWS / 2.0) + 2 && j + 2 > ceil(WALLCOLS / 2.0) - 3 && j < ceil(WALLCOLS / 2.0) + 3) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.5f;
                                newBrick.b = 0.5f;
```

```
                newBrick.health = 2;
            }
            else {
                newBrick.r = 0.95f;
                newBrick.g = 0.95f;
                newBrick.b = 0.95f;
                newBrick.health = 1;
            }

            newBrick.xpos = WALLX + j * newBrick.width + j *
WALLSPACE;
            newBrick.ypos = WALLY + i * newBrick.height + i *
WALLSPACE;
            bricks.push_back(newBrick);
        }
    }
}
```

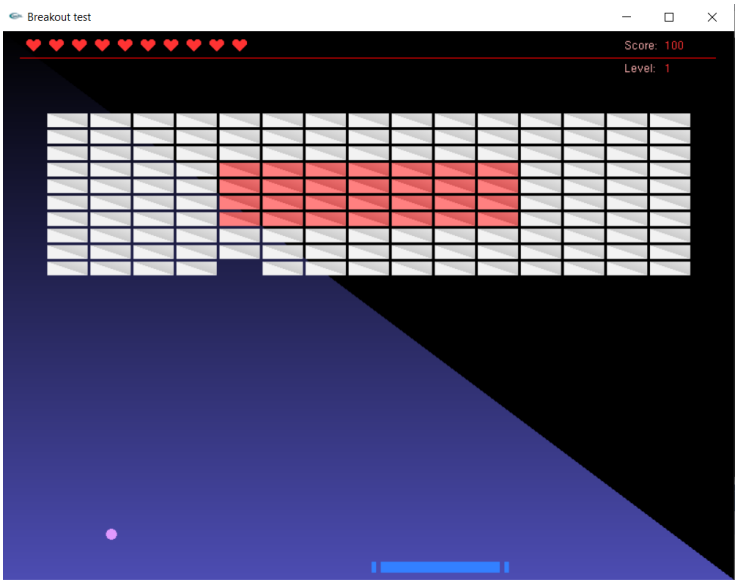


Figure 3.18 1st level

For 2nd level, figure 3.19:

```
for (int i = 0; i < WALLROWS; i++) {
        for (int j = 0; j < WALLCOLS; j++) {
            // Set stronger bricks
            if (i == 1 || i == WALLROWS - 2 || j == 1 || j == WALLCOLS -
2) {
                newBrick.r = 1.0f;
                newBrick.g = 0.5f;
                newBrick.b = 0.5f;
```

```
                newBrick.health = 2;
            }
            else {
                newBrick.r = 0.95f;
                newBrick.g = 0.95f;
                newBrick.b = 0.95f;
                newBrick.health = 1;}
```

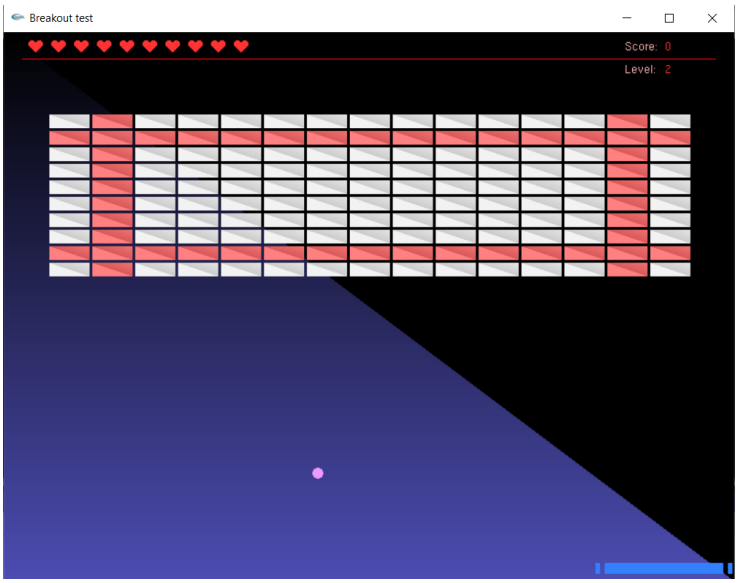


Figure 3.19 2nd level

For 3rd level, figure 3.20:

```
for (int i = 0; i < WALLROWS; i++) {
            for (int j = 0; j < WALLCOLS; j++) {
                // Set stronger bricks
                if (i == WALLROWS - 3) {
                    newBrick.r = 1.0f;
                    newBrick.g = 0.0f;
                    newBrick.b = 0.1f;
                    newBrick.health = 3;
                }
                else if (i == WALLROWS - 1 || i == WALLROWS - 2 || i ==
WALLROWS - 5 || i == WALLROWS - 6) {
                    newBrick.r = 1.0f;
                    newBrick.g = 0.9f;
                    newBrick.b = 0.1f;
                    newBrick.health = 2;
                }
                else {
                    newBrick.r = 0.95f;
```

```
                newBrick.g = 0.95f;
                newBrick.b = 0.95f;
                newBrick.health = 1;}
```

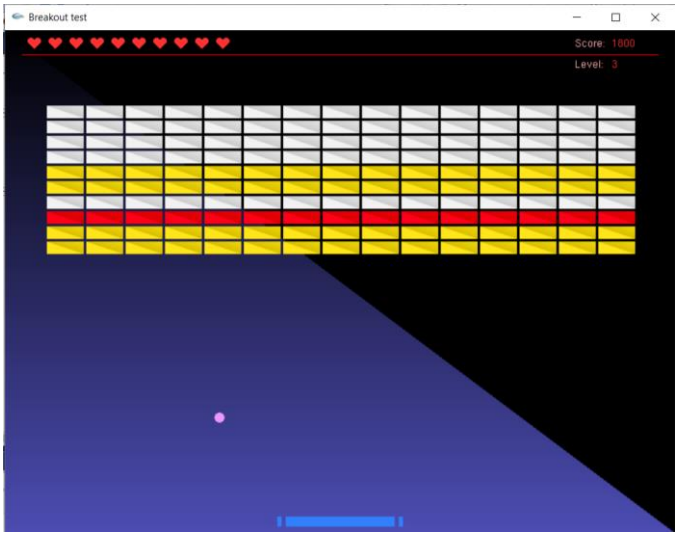


Figure 3.20 3rd level

For 4th level, figure 3.21:

```
for (int i = 0; i < WALLROWS; i++) {
            for (int j = 0; j < WALLCOLS; j++) {
                // Set stronger bricks
                if (i == j || i==WALLCOLS-1-j||j==WALLCOLS-8) {
                    newBrick.r = 1.0f;
                    newBrick.g = 0.0f;
                    newBrick.b = 0.1f;
                    newBrick.health = 3;
                }
                else if (i == WALLROWS - 1 || i == WALLROWS - 2 || i ==
WALLROWS - 5 || i == WALLROWS - 6) {
                    newBrick.r = 1.0f;
                    newBrick.g = 0.9f;
                    newBrick.b = 0.1f;
                    newBrick.health = 2;
                }
                else {
                    newBrick.r = 0.95f;
                    newBrick.g = 0.95f;
                    newBrick.b = 0.95f;
                    newBrick.health = 1;
                }
```

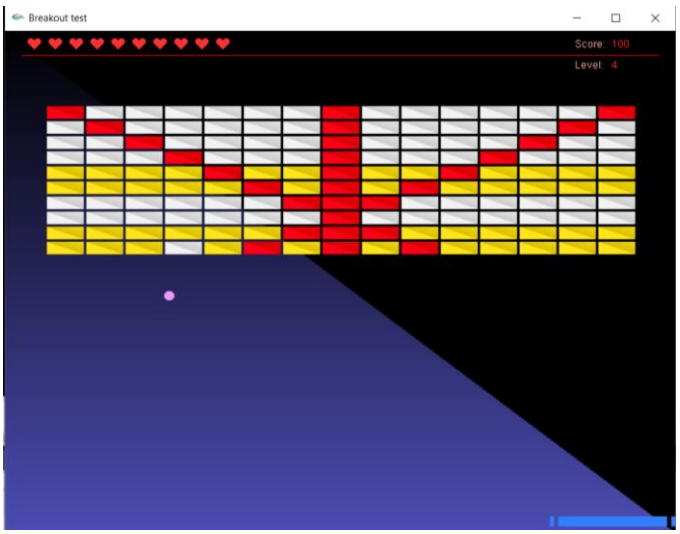


Figure 3.21 4th level

For 5th level, figure 3.22:

```
for (int i = 0; i < WALLROWS; i++) {
                for (int j = 0; j < WALLCOLS; j++) {
                        // Set stronger bricks
                        if (i + 1 > ceil(WALLROWS / 2.0) - 2 && i < ceil(WALLROWS
/ 2.0) + 2 && j + 2 > ceil(WALLCOLS / 2.0) - 3 && j < ceil(WALLCOLS / 2.0) +
3) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 0.1f;
                                newBrick.health = 3;
                        }
                        else if (i == WALLROWS - 1||i==WALLROWS-
10||j==WALLCOLS-1||j==0) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.9f;
                                newBrick.b = 0.1f;
                                newBrick.health = 2;
                        }
                        else {
                                newBrick.r = 0.95f;
                                newBrick.g = 0.95f;
                                newBrick.b = 0.95f;
                                newBrick.health = 1;}
```

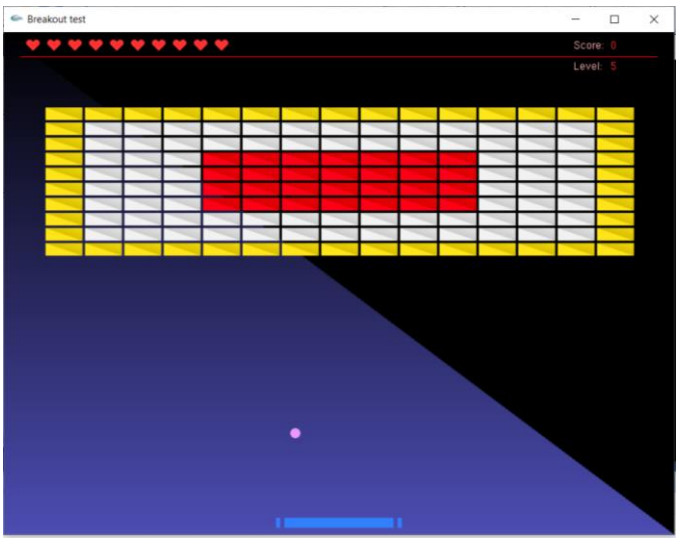


Figure 3.22 5th level

For 6th level, figure 3.23:

```
for (int i = 0; i < WALLROWS; i++) {
                for (int j = 0; j < WALLCOLS; j++) {
                        // Set stronger bricks
                        if (i == j || i == WALLCOLS - 1 - j || j == WALLCOLS - 8) {
                                newBrick.r = 0.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 1.0f;
                                newBrick.health = 4;
                        }
                        else if (i == j || i == WALLCOLS - 1 - j || j == WALLCOLS - 8) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 0.1f;
                                newBrick.health = 3;
                        }
                        else if (i == WALLROWS - 1 || i == WALLROWS - 2 || i ==
WALLROWS - 5 || i == WALLROWS - 6) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.9f;
                                newBrick.b = 0.1f;
                                newBrick.health = 2;
                        }
                        else {
                                newBrick.r = 0.95f;
                                newBrick.g = 0.95f;
                                newBrick.b = 0.95f;
                                newBrick.health = 1; }
```

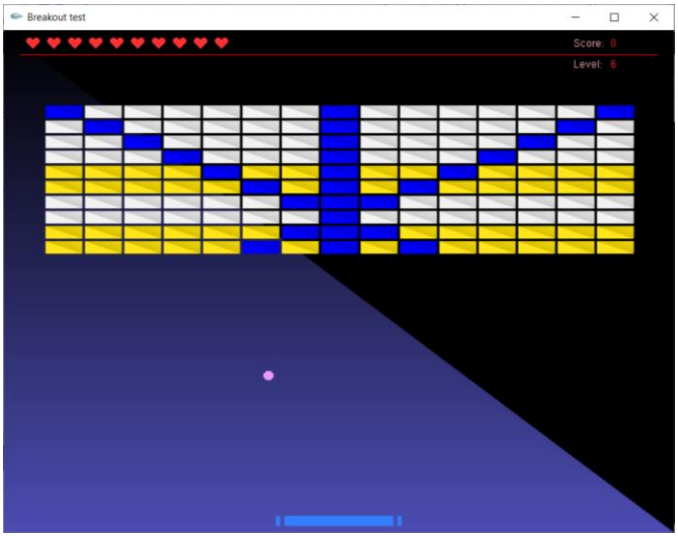


Figure 3.23 6th level

For 7th level, figure 3.24:

```
for (int i = 0; i < WALLROWS; i++) {
                for (int j = 0; j < WALLCOLS; j++) {
                        // Set stronger bricks
                        if (i + 1 > ceil(WALLROWS / 1.0) - 2 && i < ceil(WALLROWS
/ 1.0) + 2 && j + 2 > ceil(WALLCOLS / 2.0) - 3 && j < ceil(WALLCOLS / 2.0) +
3) {
                                newBrick.r = 0.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 1.0f;
                                newBrick.health = 4;
                        }
                        else if (i + 1 > ceil(WALLROWS / 2.0) - 2 && i <
ceil(WALLROWS / 2.0) + 2 && j + 2 > ceil(WALLCOLS / 2.0) - 3 && j <
ceil(WALLCOLS / 2.0) + 3) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 0.1f;
                                newBrick.health = 3;
                        }
                        else if (i == WALLROWS - 1 || i == WALLROWS - 10 || j ==
WALLCOLS - 1 || j == 0) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.9f;
                                newBrick.b = 0.1f;
                                newBrick.health = 2;
                        }
                        else {
```

```
                newBrick.r = 0.95f;
                newBrick.g = 0.95f;
                newBrick.b = 0.95f;
                newBrick.health = 1; }
```

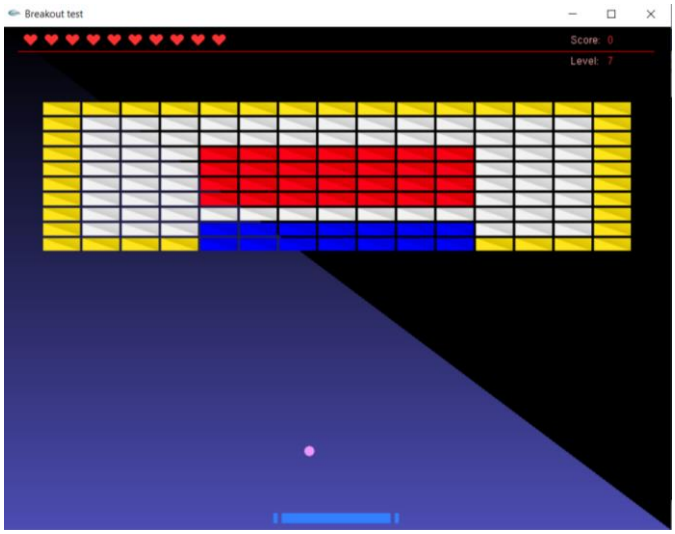


Figure 3.24 7th level

For 8th level, figure 3.25:

```
for (int i = 0; i < WALLROWS; i++) {
                for (int j = 0; j < WALLCOLS; j++) {
                        // Set stronger bricks
                        if (i + 1 > ceil(WALLROWS / 1.0) - 2 && i < ceil(WALLROWS
/ 1.0) + 2 && j + 2 > ceil(WALLCOLS / 2.0) - 6 && j < ceil(WALLCOLS / 2.0) +
6) {
                                newBrick.r = 0.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 1.0f;
                                newBrick.health = 4;
                        }else if ( j == WALLCOLS - 8||i==0) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 0.1f;
                                newBrick.health = 3;
                        }
                        else if (i == WALLROWS - 1) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.9f;
                                newBrick.b = 0.1f;
                                newBrick.health = 2;
                        }
```

```
else {
        newBrick.r = 0.95f;
        newBrick.g = 0.95f;
        newBrick.b = 0.95f;
        newBrick.health = 1;}
```

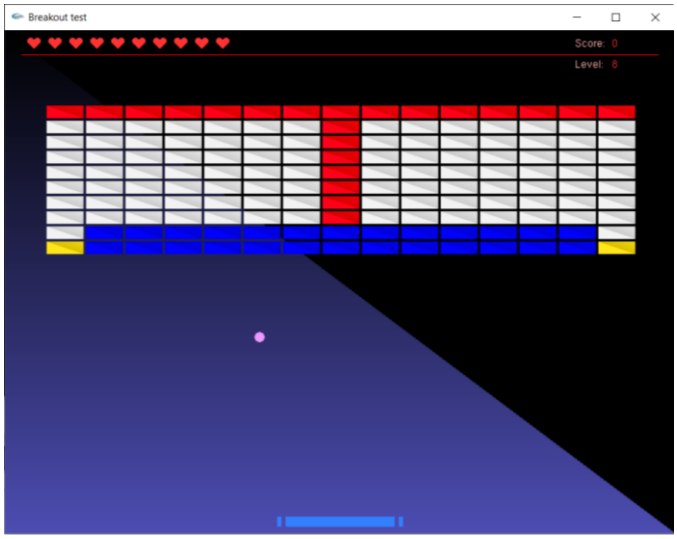


Figure 3.25 8th level

For 9th level, figure 3.26:

```
for (int i = 0; i < WALLROWS; i++) {
        for (int j = 0; j < WALLCOLS; j++) {
            // Set stronger bricks
            if (i == j || i == WALLCOLS - 1 - j) {
                newBrick.r = 0.0f;
                newBrick.g = 0.0f;
                newBrick.b = 1.0f;
                newBrick.health = 4;
            }
            else {
                newBrick.r = 0.95f;
                newBrick.g = 0.95f;
                newBrick.b = 0.95f;
                newBrick.health = 1; }
```

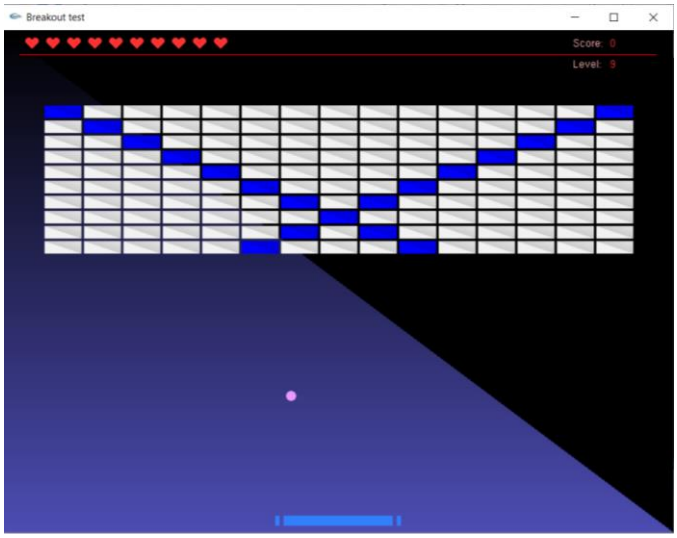


Figure 3.26 9th level

For 10th level, figure 3.27:

```
for (int i = 0; i < WALLROWS; i++) {
                for (int j = 0; j < WALLCOLS; j++) {
                        // Set stronger bricks
                        if (i + 1 > ceil(WALLROWS / 2.0) - 2 && i < ceil(WALLROWS
/ 2.0) + 2 && j + 2 > ceil(WALLCOLS / 2.0) - 3 && j < ceil(WALLCOLS / 2.0) +
3) {
                                newBrick.r = 0.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 1.0f;
                                newBrick.health = 4;
                        }
                        else if (i == WALLROWS - 1 || i == WALLROWS - 10 || j ==
WALLCOLS - 1 || j == 0) {
                                newBrick.r = 1.0f;
                                newBrick.g = 0.0f;
                                newBrick.b = 0.1f;
                                newBrick.health = 3;
                        }
                        else {
                                newBrick.r = 0.95f;
                                newBrick.g = 0.95f;
                                newBrick.b = 0.95f;
                                newBrick.health = 1;}
```

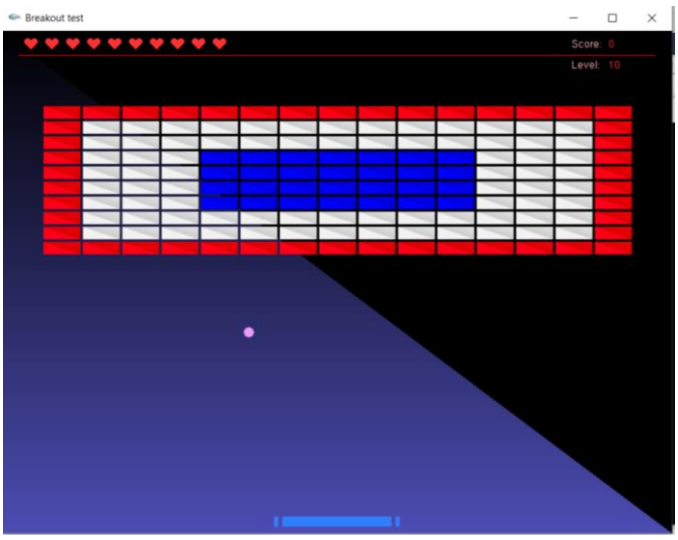


Figure 3.27 10th level

Also added for special effects, music of failure, in figure 3.29, and victory, in figure 3.30, by using PlaySound(). The PlaySound function helps to play the certain sound which is declared in the program, also important to declare the source of the file for correctness of the location of the sound or the music. [9]

```
if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
lifesCount==0) {
        PlaySound(TEXT("death.wav"), NULL, SND_APPLICATION);
    }
```

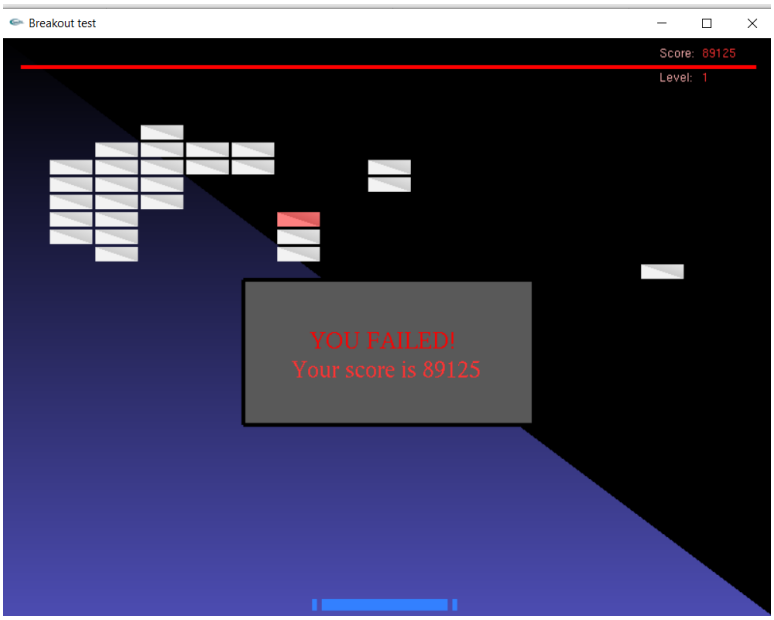


Figure 3.29 Failure

```
PlaySound(TEXT("happy.wav"), NULL, SND_ASYNC);
```

```
void Breakout::newBallWON(float x = -10, float y = -10) {
	Ball b1;
	if (x < 0 || y < 0) {
		b1.xpos = WINWIDTH / 2.0;
		b1.ypos = WINHEIGHT - 20.0f;
	}

		if ((float)rand() / (RAND_MAX) < 0.5)
			b1.xvel = 5.5f;
		else
			b1.xvel = -5.5f;
		b1.yvel = -5.0f;

	b1.radius = BALL_RADIUS;
	b1.r = 0.75f + (float)rand() / (RAND_MAX);
	b1.g = 0.25f + (float)rand() / (RAND_MAX);
	b1.b = 0.4f + (float)rand() / (RAND_MAX);
	balls.push_back(b1);}
```

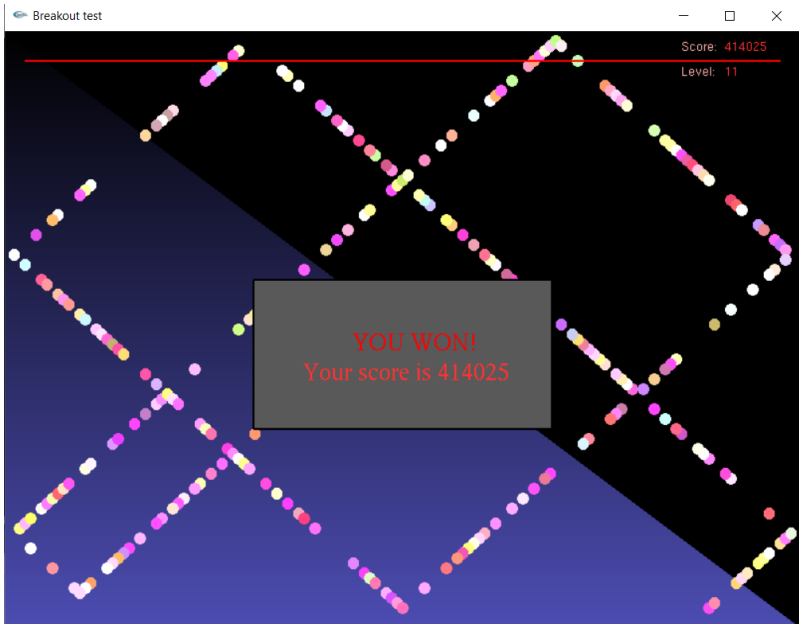


Figure 3.30 Victory

The main source code is in Appendix A.

CONCLUSION

The basics of creating a realistic picture of the invention are studied:

- lighting;
- texture overlay;

Next stage:

1) In order to develop games, have studied: - the basics of realistic scenes: scene modeling, lighting, texture overlay;

2) Synthesized observers and species transformations are studied on the example of games;

3) The basics of animation of objects are studied: through the control of information input, mouse, keyboard;

4) As an example of motion animation, Solar Systems are considered;

5) Games: Game models with conditions and restrictions have been developed;

6) Game Results;

7) The results of the study were presented in the form of abstracts and heard at the International Scientific Conference of Students and Young Scientists "Farabi Alemi" (Almaty, April 6-9, 2021), see Appendix D.

REFERENCES

1. Васильев В.Е., Морозов А. В. Компьютерная графика: учебное пособие: - СПб.: СЗТУ, 2005.-101 с.
2. 24hostel.msk.ru. (n.d.). Компьютерная графика в современном мире. [online] Available at: http://24hostel.msk.ru/2012/06/19/komputernaja-grafika-v-sovremennom-mire/ [Accessed 14 Jun. 2021].
3. FB.ru. (n.d.). Компьютерная графика что такое? Виды компьютерной графики. [online] Available at: https://fb.ru/article/190005/kompyuternaya-grafika-chto-takoe-vidyi-kompyuternoy-grafiki [Accessed 14 Jun. 2021].
4. Openglbook.com. (2009). Preface: What is OpenGL? | OpenGLBook.com. [online] Available at: http://openglbook.com/chapter-0-preface-what-is-opengl.html.
5. bourabai.kz. (n.d.). *Технология OpenGL*. [online] Available at: http://bourabai.kz/graphics/OpenGL/index.htm [Accessed 14 Jun. 2021].
6. 3DNews - Daily Digital Digest. (n.d.). *Что такое OpenGL?* [online] Available at: https://3dnews.ru/169184 [Accessed 14 Jun. 2021].
7. Журнал “Код”: программирование без снобизма. (2020). *Своя игра: создаём собственную “Змейку.”* [online] Available at: https://thecode.media/snake-js/ [Accessed 14 Jun. 2021].
8. www.bestreferat.ru. (n.d.). *Курсовая работа: Разработка программы для игры Тетрис - BestReferat.ru*. [online] Available at: https://www.bestreferat.ru/referat-281883.html#_Toc283235714 [Accessed 14 Jun. 2021].
9. docs.microsoft.com. (n.d.). *PlaySound function (Windows)*. [online] Available at: https://docs.microsoft.com/en-us/previous-versions/dd743680(v=vs.85).
10. Д. Шрайнер — OpenGL Redbook
11. Rice.edu. (2019). *Why is computer graphics important?* [online] Available at: https://www.cs.rice.edu/~jwarren/360/outline/subsection3_1_2.html.
12. www.cprogramming.com. (n.d.). *C++ OpenGL Programming Tutorial - Cprogramming.com*. [online] Available at: https://www.cprogramming.com/tutorial/opengl_introduction.html [Accessed 14 Jun. 2021].
13. www.glprogramming.com. (n.d.). *Chapter 1. Introduction to OpenGL*. [online] Available at: https://www.glprogramming.com/blue/ch01.html [Accessed 14 Jun. 2021].
14. www.songho.ca. (n.d.). *OpenGL Camera*. [online] Available at: http://www.songho.ca/opengl/gl_camera.html [Accessed 14 Jun. 2021].
15. open.gl. (n.d.). *OpenGL - Transformations*. [online] Available at: https://open.gl/transformations.
16. www.khronos.org. (n.d.). *How lighting works - OpenGL Wiki*. [online] Available at: https://www.khronos.org/opengl/wiki/How_lighting_works [Accessed 14 Jun. 2021].
17. Дональд Херн, М. Паулин Бейкер. Компьютерная графика и стандарт OpenGL , 2005.— 1168с.

18. Бубнов А.Е. Компьютерный дизайн. Основы, Мн: Знание, 2008 г.

APPENDIX A

Breakout game Source.cpp

```
#include <Windows.h>
#include <iostream>
#include <assert.h>
#include <glut.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "breakout.h"
using namespace std;
Breakout game;
void myDisplay()
{
	game.display();
}
// Define the reshape function
void myReshape(int w, int h)
{
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		glOrtho(-10.5, 10.5, -10.5*(GLfloat)h / (GLfloat)w, 10.5*(GLfloat)h /
(GLfloat)w, -100.0, 100.0);
	else
		glOrtho(-10.5*(GLfloat)w / (GLfloat)h, 10.5*(GLfloat)w / (GLfloat)h, -
10.5, 1.5, -100.0, 100.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	game.reshape(w, h);
}
// Define the mouse click events
void myMouseClick(int button, int state, int x, int y)
{
	game.mouseClick(button, state, x, y);
}
// Define the mouse drag events
void myMouseMove(int x, int y)
{
	game.mouseMove(x, y);
}
// Define keystroke events
```

```
void myKeyStroke(unsigned char key, int x, int y)
{
	game.keyStroke(key, x, y);
}
void mySpecialKeyStroke(int key, int x, int y)
{
	game.specialKeyPos(key, x, y);
}
void main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |
GLUT_MULTISAMPLE | GLUT_DEPTH);
	glutInitWindowSize(WINWIDTH, WINHEIGHT);
	glutInitWindowPosition(100, 100);
	glutCreateWindow("Breakout test");
	game.init();
	glutDisplayFunc(myDisplay);
	// Handle reshape
	glutReshapeFunc(myReshape);
	// Handle mouse clicks
	glutMouseFunc(myMouseClick);
	// Handle mouse motion
	glutPassiveMotionFunc(myMouseMove);
	// Handle keyboard strokes
	glutKeyboardFunc(myKeyStroke);
	// specify keyboard special key input events
	glutSpecialFunc(mySpecialKeyStroke);
	// Enter the opengl event processing loop
	glutMainLoop();

}
```

APPENDIX B

Snake game Source.cpp

```
#define _CRT_SECURE_NO_DEPRECATE
#include "painter.h"
#include "game.h"
#include "glut.h"
#include <iostream>
Game game;
const char * grassFilename = "./sea_seamless.bmp";
GLuint grassTexture;
GLuint loadBMP_custom(const char * imagepath) {
	// Data read from the header of the BMP file
	unsigned char header[54]; // Each BMP file begins by a 54-bytes header
	unsigned int dataPos;     // Position in the file where the actual data begins
	unsigned int width, height;
	unsigned int imageSize;   // = width*height*3
	                                        // Actual RGB data
	unsigned char * data;

	FILE * file = fopen(imagepath, "rb");
	if (!file) {
		printf("Image could not be opened\n");
		return 0;
	}
	// If not 54 bytes read : problem
	if (fread(header, 1, 54, file) != 54) {
		printf("Not a correct BMP file\n");
		return false;
	}
	if (header[0] != 'B' || header[1] != 'M') {
		printf("Not a correct BMP file\n");
		return 0;
	}
	// Read ints from the byte array
	dataPos = *(int*)&(header[0x0A]);
	imageSize = *(int*)&(header[0x22]);
	width = *(int*)&(header[0x12]);
	height = *(int*)&(header[0x16]);
	// Some BMP files are misformatted, guess missing information
	if (imageSize == 0)    imageSize = width*height * 3; // 3 : one byte for each
Red, Green and Blue component
	if (dataPos == 0)      dataPos = 54; // The BMP header is done that way
// Create a buffer
```

```
	data = new unsigned char[imageSize];
	// Read the actual data from the file into the buffer
	fread(data, 1, imageSize, file);
	//Everything is in memory now, the file can be closed
	fclose(file);
	if (false) {
		for (int i = 0; i < imageSize; i += 3) {
			int tmp = data[i];
			data[i] = data[i + 2];
			data[i + 2] = tmp;
			std::cout << i / 3 << " -- R:" << (int)data[i] << " G:" << (int)data[i
+ 1] << " B:" << (int)data[i + 2] << std::endl << std::endl;
		}
	}
	// Create one OpenGL texture
	GLuint textureID;
	glGenTextures(1, &textureID);
	// "Bind" the newly created texture : all future texture functions will modify
this texture
	glBindTexture(GL_TEXTURE_2D, textureID);
	// Give the image to OpenGL
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_BGR_EXT, GL_UNSIGNED_BYTE, data);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
	return textureID;
}
void display()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glPushAttrib(GL_ENABLE_BIT);
	glEnable(GL_TEXTURE_2D);
	glEnable(GL_BLEND);
	glEnable(GL_DEPTH_TEST);
	glBindTexture(GL_TEXTURE_2D, grassTexture);
	glPushMatrix();
	glBegin(GL_QUADS);
	glTexCoord2f(0, 0);
	glVertex2f(-310, 310);
	glTexCoord2f(1, 0);
	glVertex2f(310, 310);
```

```
        glTexCoord2f(1, 1);
        glVertex2f(310, -310);
        glTexCoord2f(0, 1);
        glVertex2f(-310, -310);
        glEnd();
        glPopMatrix();
        glPopAttrib();
        Painter p;
        game.draw(p);
        glutSwapBuffers();
}
void timer(int = 0)
{       game.tick();
        display();
        glutTimerFunc(300, timer, 0);
}
void keyEvent(int key, int, int)
{
        switch (key)
        {
        case GLUT_KEY_LEFT:
                game.keyEvent(Snake::LEFT);
                break;
        case GLUT_KEY_UP:
                game.keyEvent(Snake::UP);
                break;
        case GLUT_KEY_RIGHT:
                game.keyEvent(Snake::RIGHT);
                break;
        case GLUT_KEY_DOWN:
                game.keyEvent(Snake::DOWN);
                break;
        }
}
int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowSize(Field::WIDTH * Field::BLOCK_WIDTH,
Field::HEIGHT * Field::BLOCK_HEIGHT);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Snake");
        glClearColor(0, 0, 0, 1.0);
```

```
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        //WINHEIGHT
        glOrtho(0, Field::WIDTH * Field::BLOCK_WIDTH, Field::HEIGHT *
Field::BLOCK_HEIGHT, 0, -1.0, 1.0);
        glutDisplayFunc(display);
        glutSpecialFunc(keyEvent);
        timer();
        grassTexture = loadBMP_custom(grassFilename);
        glutMainLoop();
}
```

APPENDIX C

Tetris game main.cpp

```
#include "Game.h"
#include "main.h"
using namespace std;
Game game; //game instance
                //function headers
                /**
                * Game callback timer; updates current falling block
                **/
void timer(int id) {
        if (game.gameOver) {      //mode for when blocks cross red line
                        game.gameOver =true;
                        game.Update();
                        game.ClearGrid();
        }
        else if (isRowDestroying) {
                //use bIsRowDestroying to anim falling bricks
                destroy_Row_Alpha -= ROW_DESTROYED_ALPHA_DELTA;
                glutTimerFunc(destroyTimer, timer, DESTROY_TIMER_ID);
                glutPostRedisplay();
                return;
        }
        else if (!game.paused) {
                game.Update();
                if (!game.blockFalling) {
                        CheckRowClear();
                }
                else if (game.gameOver) {
                        game.ClearGrid();
                        glutTimerFunc(10, timer, 1);}}
        if (!isRowDestroying) {
                glutPostRedisplay();
                glutTimerFunc(game.timer, timer, 0);}}
/**
* GLUT Callback for displaying image on screen
**/
void display(void) {
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glBegin(GL_QUADS);
        glColor3f(0.1f, 0.3f, 0.5f);
```

```
glVertex2f(1500, 2500);
glVertex2f(-1500, 2500);
glVertex2f(-1000, -1000);
glVertex2f(2500,-1000);
glEnd();
//Draw play space (area w/ blocks)
glViewport(0, 0, WINDOW_WIDTH / 2, WINDOW_HEIGHT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, COLS * 100, 0, ROWS * 100);
if (!game.paused) {
	if (game.blockFalling) {
		for (int r = 0; r < ROWS; r++) {
			for (int c = 0; c < COLS; c++) {
				switch (game.grid[c][r]) {
				case(Empty_Block):
					glColor4f(0.0, 0.0, 0.0, 1.0); //black
					break;
				case(Square_Block):
					glColor4f(1.0, 1.0, 0.03, 1.0); //yellow
					break;
				case(Rectangle_Block):
					glColor4f(0.03, 1.0, 1.0, 1.0); //cyan
					break;
				case(L_Block):
					glColor4f(1.0, 0.64, 0.043, 1.0); //orange
					break;
				case(Squigly_Block):
					glColor4f(0.035, 1.0, 0.035, 1.0); //green
					break;
				case(RevSquigly_Block):
					glColor4f(1.0, 0.035, 0.035, 1.0); //red
					break;
				case(T_Block):
					glColor4f(0.035, 0.035, 1.0, 1.0); //blue
					break;
				default:
					glColor4f(1.0, 1.0, 1.0, 1.0); //white for debug
					break;
				};
				//if the current row is within destroyRows[]
				for (int k = 0; k < BLOCK_SIDE_MAX; k++) {
					if (destroyRows[k] == r) {
```

```
                                        isRowDestroying = true;
                                        //fade block pieces via alpha
                                        setAlpha(destroy_Row_Alpha /
MAX_DESTROY_ROW_ALPHA);
                                    }
                                }

                                //if the row has been fully displaced, move it
                                if (isRowDestroying && ((destroy_Row_Alpha /
MAX_DESTROY_ROW_ALPHA) <= 0.0)) {
                                    //and remove the row
                                    for (int h = 0; h < BLOCK_SIDE_MAX; h++)
{
                                        if (destroyRows[h] !=
DESTROY_ROW_INIT) {
                                            rowsCleared++;
                                            for (int j = 0; j < COLS; j++) {

        game.grid[j][destroyRows[h]] = Empty_Block;
                                            }

        game.CollapseRow(destroyRows[h]);
                                            destroyRows[h] =
DESTROY_ROW_INIT;
                                            isRowDestroying = false;
                                        }
                                    }
                                    //ResetDestroyRows();
                                }
                                //display square
                                DrawSquare(c, r);
                            }
                        }
                    }
        }
        else if(game.paused) {
                //if paused, cover screen
                for (int r = 0; r < ROWS; r++) {
                    for (int c = 0; c < COLS; c++) {
                        glColor4f(0.3, 0.3, 0.3, 1.0); //grey to cover
                        DrawSquare(c, r);
                    }
                }
```

```
                glColor4f(random(), random(), random(), 1.0);
                char msg[100];
                if (game.paused) {
                        sprintf_s(msg, "PAUSED");
                        BitmapText(msg, (ROWS / 4) * BLOCK_PIXEL_SIZE, (COLS)*
BLOCK_PIXEL_SIZE);
                }}
        /*else if(game.gameOver){
                glColor4f(random(), random(), random(), 1.0);
                char msg[100];
                if (game.gameOver) {
                        sprintf_s(msg, "GAME OVER");
                        BitmapText(msg, (ROWS / 4) * BLOCK_PIXEL_SIZE, (COLS)*
BLOCK_PIXEL_SIZE);
                }
        }*/

        //draw game-over line
        glColor4f(1.0, 0.0, 0.0, 1.0);
        glBegin(GL_LINES);
        glVertex2f(0, 2500);
        glVertex2f(WINDOW_WIDTH * 4, 2500);
        glEnd();

        // Write messages on right hand side of screen
        glViewport(WINDOW_WIDTH / 2, 0, WINDOW_WIDTH,
WINDOW_HEIGHT);


        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, COLS * 200, 0, ROWS * 100);

        //draw dividing line
        glColor4f(random(), random(), random(), 1.0);
        glBegin(GL_LINES);
        glVertex2f(0, 0);
        glVertex2f(0, 3000);
        glEnd();

        char msg[100];
        glColor3f(1, 0, 0);
        sprintf_s(msg, "ROWS CLEARED: %i", rowsCleared);
```

```
BitmapText(msg, 10, 30);
glColor3f(0, 1, 0);
sprintf_s(msg, "TIMER: %f", game.timer);
BitmapText(msg, 10, 100);
glColor3f(1, 1, 0);
sprintf_s(msg, "Arrow Key LEFT & RIGHT to move");
BitmapText(msg, 10, 210);
sprintf_s(msg, "Arrow Keys UP & DOWN to rotate");
BitmapText(msg, 10, 300);
sprintf_s(msg, "Spacebar to drop");
BitmapText(msg, 10, 390);
sprintf_s(msg, "P to pause");
BitmapText(msg, 10, 480);
/*sprintf_s(msg, "F1 to toggle next block 1x4");
BitmapText(msg, 10, 570);*/

//draw shape tracker
//Square_Block
glColor4f(1.0, 1.0, 0.03, 1.0); //yellow
DrawSquare(2, 7);
glColor4f(1.0, 1.0, 0.03, 1.0); //yellow
DrawSquare(2, 8);
glColor4f(1.0, 1.0, 0.03, 1.0); //yellow
DrawSquare(3, 7);
glColor4f(1.0, 1.0, 0.03, 1.0); //yellow
DrawSquare(3, 8);
//Rectangle_Block
for (int i = 2; i < 6; i++) {
	glColor4f(0.03, 1.0, 1.0, 1.0); //cyan
	DrawSquare(i, 10);
}
//L_Block
glColor4f(1.0, 0.64, 0.043, 1.0); //orange
DrawSquare(2, 12);
glColor4f(1.0, 0.64, 0.043, 1.0); //orange
DrawSquare(2, 13);
glColor4f(1.0, 0.64, 0.043, 1.0); //orange
DrawSquare(3, 12);
glColor4f(1.0, 0.64, 0.043, 1.0); //orange
DrawSquare(4, 12);
//Squigly_Block
glColor4f(0.035, 1.0, 0.035, 1.0); //green
DrawSquare(2, 15);
```

```
        glColor4f(0.035, 1.0, 0.035, 1.0); //green
        DrawSquare(3, 15);
        glColor4f(0.035, 1.0, 0.035, 1.0); //green
        DrawSquare(3, 16);
        glColor4f(0.035, 1.0, 0.035, 1.0); //green
        DrawSquare(4, 16);
        //RevSquigly_Block
        glColor4f(1.0, 0.035, 0.035, 1.0); //red
        DrawSquare(2, 20);
        glColor4f(1.0, 0.035, 0.035, 1.0); //red
        DrawSquare(3, 20);
        glColor4f(1.0, 0.035, 0.035, 1.0); //red
        DrawSquare(3, 19);
        glColor4f(1.0, 0.035, 0.035, 1.0); //red
        DrawSquare(4, 19);
        //T_Block
        glColor4f(0.035, 0.035, 1.0, 1.0); //blue
        DrawSquare(2, 23);
        glColor4f(0.035, 0.035, 1.0, 1.0); //blue
        DrawSquare(3, 24);
        glColor4f(0.035, 0.035, 1.0, 1.0); //blue
        DrawSquare(3, 23);
        glColor4f(0.035, 0.035, 1.0, 1.0); //blue
        DrawSquare(4, 23);

        //Text for block counter
        glColor3f(1, 0, 0);
        sprintf_s(msg, "X %i", game.blockCounter[Square_Block - 1]);
        BitmapText(msg, 700, 800);
        sprintf_s(msg, "X %i", game.blockCounter[Rectangle_Block - 1]);
        BitmapText(msg, 700, 1000);
        sprintf_s(msg, "X %i", game.blockCounter[L_Block - 1]);
        BitmapText(msg, 700, 1300);
        sprintf_s(msg, "X %i", game.blockCounter[Squigly_Block - 1]);
        BitmapText(msg, 700, 1600);
        sprintf_s(msg, "X %i", game.blockCounter[RevSquigly_Block - 1]);
        BitmapText(msg, 700, 2000);
        sprintf_s(msg, "X %i", game.blockCounter[T_Block - 1]);
        BitmapText(msg, 700, 2400);

        glutSwapBuffers();
}
```

```
/**
* Draw square by drawing dark square in background, smaller & brighter square on top
*   and 4 black lines from each corner of larger & darker square to smaller & brighter square on top
*   to give 3d effect
**/
void DrawSquare(int column, int row) {
        //back block that will form trapezoids
        darkenColor(TRAPEZOID_DARK_DELTA);
        glBegin(GL_POLYGON); //BLOCK_PIXEL_SIZExBLOCK_PIXEL_SIZE block
        glVertex2f(column * BLOCK_PIXEL_SIZE, (row + 1) * BLOCK_PIXEL_SIZE); //top left
        glVertex2f((column + 1) * BLOCK_PIXEL_SIZE, (row + 1) * BLOCK_PIXEL_SIZE); //top right
        glVertex2f((column + 1) * BLOCK_PIXEL_SIZE, row * BLOCK_PIXEL_SIZE); //bottom right
        glVertex2f(column * BLOCK_PIXEL_SIZE, row * BLOCK_PIXEL_SIZE); //bottom left
        glEnd();

        //brighten color and draw square on top
        darkenColor(1 / TRAPEZOID_DARK_DELTA);
        glBegin(GL_POLYGON);
        glVertex2f((column * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT, ((row + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT); //top left
        glVertex2f(((column + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT, ((row + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT); //top right
        glVertex2f(((column + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT, (row * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT); //bottom right
        glVertex2f((column * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT, (row * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT); //bottom left
        glEnd();

        //draw lines from darker square to lighter square
        glColor4f(0.0, 0.0, 0.0, 1.0);
        glBegin(GL_LINES);
        glVertex2f(column * BLOCK_PIXEL_SIZE, (row + 1) * BLOCK_PIXEL_SIZE);
        glVertex2f((column * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT, ((row + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT);
```

```
        glVertex2f((column + 1) * BLOCK_PIXEL_SIZE, (row + 1) *
BLOCK_PIXEL_SIZE);
        glVertex2f(((column + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT,
((row + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT);

        glVertex2f((column + 1) * BLOCK_PIXEL_SIZE, row *
BLOCK_PIXEL_SIZE);
        glVertex2f(((column + 1) * BLOCK_PIXEL_SIZE) - TRAPEZOID_HEIGHT,
(row * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT);

        glVertex2f(column * BLOCK_PIXEL_SIZE, row * BLOCK_PIXEL_SIZE);
        glVertex2f((column * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT,
(row * BLOCK_PIXEL_SIZE) + TRAPEZOID_HEIGHT);
        glEnd();
}

/**
* Check if a row has been cleared
**/
void CheckRowClear() {
        int blockCount = 0;
        for (int i = 0; i < ROWS; i++) {
                //trace through all columns
                for (int j = 0; j < COLS; j++) {
                        if (game.grid[j][i] != Empty_Block) {
                                blockCount++;
                        }
                }
                //found a column? destroy it
                if (blockCount == COLS)
                        DestroyRow(i);
                blockCount = 0;
        }
}

/**
* Break each block in a row into chuncks
**/
void DestroyRow(int row) {
        //set next available value in destroy[BLOCK_SIDE_MAX] (destroy[] can hold
up to 4 ROW values)
        for (int i = 0; i < BLOCK_SIDE_MAX; i++) {
                if (destroyRows[i] == DESTROY_ROW_INIT) {
```

```
                    destroyRows[i] = row;
                    break;
                }
            }
        }
        //set destroy row displacement = max_destroy displacement
        destroy_Row_Alpha = MAX_DESTROY_ROW_ALPHA;
}

/**
* Sets up GLUT and callbacks as well as timer and random seeds
**/
void main(int argc, char *argv[]) {
        seedRand();
        glutInit(&argc, argv);

        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

        glutInitWindowPosition(100, 100);
        glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);

        glutCreateWindow("Tetris");

        glutDisplayFunc(display);
        glutSpecialFunc(Special);
        glutKeyboardFunc(Keyboard);
        glutTimerFunc(game.timer, timer, 0);

        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        ResetDestroyRows();

        rowsCleared = 0;
        deadTime = 0;

        glutMainLoop();
}

/**
* Reset array used for tracking destroyed rows
**/
void ResetDestroyRows() {
        isRowDestroying = false;
```

```
	for (int i = 0; i < BLOCK_SIDE_MAX; i++) {
		destroyRows[i] = DESTROY_ROW_INIT;
	}
}

/**
* Displays a char* to the given coordinates
**/
void BitmapText(char *str, int wcx, int wcy) {
	glRasterPos2i(wcx, wcy);
	for (int i = 0; str[i] != '\0'; i++) {
		glutBitmapCharacter(GLUT_BITMAP_8_BY_13, str[i]);
	}
}

/**
* Handes standard keyboard input (characters, space bar)
**/
void Keyboard(unsigned char key, int x, int y) {
	if (!game.gameOver) {
		if (key == 'p') {
			game.paused = !game.paused;
		}
		else if ((key == ' ') && !game.paused) {
			while (game.blockFalling) {
				game.MoveY();
			}
			CheckRowClear();
		}
	}
	else { game.gameOver = true; }
}

/**
* Handles special key input (arrow keys)
**/
void Special(int key, int x, int y) {
	//don't accept movement input while paused, gameOver or clearing row
	if (!game.paused && !game.gameOver && !isRowDestroying) {
		switch (key) {
		case GLUT_KEY_LEFT:
			game.MoveX(BLOCK_HORIZ_SPEED * -1);
			break;
```

```
                case GLUT_KEY_RIGHT:
                        game.MoveX(BLOCK_HORIZ_SPEED);
                        break;
                case GLUT_KEY_DOWN:
                        game.Rotate(ROTATE_SHAPE_RIGHT);
                        game.Rotate(ROTATE_SHAPE_RIGHT);
                        game.Rotate(ROTATE_SHAPE_RIGHT);
                        break;
                case GLUT_KEY_UP:
                        game.Rotate(ROTATE_SHAPE_LEFT);
                        break;
                default:
                        break;
                }
        }
}
```

APPENDIX D

Интерактивная графика и моделирование игр в среде OpenGL

Какибай А. Қ.
Научный руководитель: проф., доктор физ.-мат. наук Л.А. Хаджиева.
Казахский Национальный Университет имени аль-Фараби
aru.kakibay@gmail.com

OpenGL (Open Graphics Library — открытая графическая библиотека, графическое API) — спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику. Включает более 250 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. [2]

Данная работа посвящена разработке программного кода по моделированию игр, а также его визуализации с использованием открытой графической библиотеки OpenGL. Выбор указанной графической среды обусловлен ее довольно простым процедурным интерфейсом, который дает возможность создавать программные комплексы в 2D и 3D пространствах. Для визуализации используется редактор Microsoft Visual Studio, т.к. является кроссплатформенным графическим интерфейсом. В качества языка выбран язык объектно-ориентированного программирования C++. Также на основу моделирования игры взят идея игры “Breakout”.

“Breakout” – это успешная аркадная игра, выпущенная в 1976 году компанией Atari. Цель игры – разбить несколько рядов кирпичей в верхней части экрана, используя мяч и небольшую ракетку внизу экрана. В данной игре будет использовано структура из двух чисел с плавающей запятой для представления позиции на экране. Основным предметом в игре будет мяч. Мяч будет иметь положение и скорость, представленные двухмерной структурой. Мяч будет отскакивать от верхней, левой и правой стороны экрана, но не от нижней стороны. На нижней стороне будет ракетка, которую пользователь может перемещать горизонтально с помощью мыши. Мяч сможет подпрыгивать от ракетки и бить по кирпичам и отскакивать.

Практический все игры состоят из 3 частей:

1) Сбор и обработка пользовательского ввода
2) Логика игры
3) Рисовка

Для первой части нужно захватить положение мыши на экране по оси X. Положение ракетки по оси Y будет фиксированным. Другой ввод, который надо захватить, — это щелчок левой кнопкой мыши, чтобы создавать новый мяч после того, как он выйдет за пределы экрана.

Для второй части, надо разобрать движение мяча. Для этого нужно брать текущую 2D-позицию и добавляем 2D-скорость. Поскольку скорость также может быть отрицательной, она должна работать в любом направлении. Чтобы переместить ракетку в нижней части экрана, рассматривается позиция мыши по оси X. Следующий элемент, на котором нужно сосредоточиться, — это проверить, отскакивает ли мяч от ракетки. Далее рассматривается столкновение мяча с кирпичом. Чтобы упростить обнаружение столкновений, будет считаться, что мяч квадратный за кадром, а не круглый. Чтобы обнаружить столкновение, нужно сравнить стороны шара и кирпича.

Последняя часть игры — это рисовка: мяч, ракетка, кирпичи.

СПИСОК ЛИТЕРАТУРЫ

1. http://netlib.narod.ru/library/book0052/ch03_07.htm
2. http://bourabai.kz/graphics/OpenGL/index.htm