1. Обзор.

Необходимо создать игру «Гонки» на подобии тех, что были в семействе игр на платформе Тетрис.

Язык: C++.

IDE: Visual Studio 2016.

1. Модель программы.

2.1 После запуска программы, перед пользователем предстает полотно консоли, а затем и окно игры.

Окно игры представляет собой графическое окно OpenGL, размером 120х350 пикселов. На нем игроку предлагается выбрать один из трех вариантов сложности игры: EASY (легко), NORMAL (нормально) и HARD (Тяжело). Выбор осуществляется с помощью манипулятора «мышь» или путем нажатия клавиши, соответствующей первой букве сложности. Например для выбора сложности EASY, навести курсор на соответствующую надпись и нажать левой кнопкой мыши, либо нажать на клавиатуре ‘e’ или ‘E’ (регистронезависимость).

Выбранному уровню сложности соответствует количество жизней:

EASY – 5 жизней,

NORMAL – 3 жизни,

HARD – 1 жизнь.

2.2 После выбора подходящего уровня сложности, начнется игра.

На поле с двух сторон появляется движущаяся сверху вниз пунктирная линия, являющаяся аналогией обочины, и машина игрока, появляющаяся в нижней середине игрового поля. Машина представляет собой семь крестообразно расположенных квадратов белого цвета.

2.3 После этого сверху вниз начинают «падать» другие машины, за счет чего создается иллюзия обгона.

2.4 Для маневрирования юзер может пользоваться клавишами w, a, d, s, которые перемещают машину игрока вперед, влево, вправо и назад соответственно. Также для управления можно использовать стрелки и клавиши малой клавиатуры.

В случае столкновения, у игрока отнимается одна жизнь. Машина перемещается в исходное положение (нижний центр), остальные машины (если таковые имеются на игровом поле, исчезают, а после снова начинают появляться сверху.

По истечению жизней, игроку будет представлена информация об окончании игры (надпись GAME OVER) и заработанное им количество баллов. Далее приведена схема начисления баллов.

|  |  |
| --- | --- |
| Уровень сложности | Формула вычисления балов |
| EASY | x = y |
| NORMAL | x = y \* 2; |
| HARD | x = y \* 3; |

(y – количество машин, которые объехал юзер)

2.5 Если предыдущие игроки уже зарабатывали подобное количество баллов, то под указанием количества его баллов выйдет надпись «NEW GAME». Под надписью выйдет список чемпионов, который, при необходимости можно будет прокручивать с помощью колеса мыши.

2.6 Если предыдущие игроки не зарабатывали такого количества балов, то юзеру будет необходимо ввести свое имя. Ниже будет расположен список чемпионов, прокручивать который будет нельзя, дабы не отвлекать человека от основного занятия – ввода своего имени.

2.7 После набора имени, число символов которого ограничивается девятью, пользователь нажимает кнопку Enter и его имя занимает соответствующую нишу в списке чемпионов, который теперь можно прокручивать. А на месте ввода имени появится надпись «NEW GAME».

Нажав на кнопку «NEW GAME», пользователь начнет новую игру и перед ним снова выпадет меню выбора сложности…

1. Архитектура программы.

Самым базовым для программы является класс Car.



Этот класс содержит защищенные поля x и y для хранения координат положения машины.

Методы класса:

Car - перегруженный конструктор для создания машины с определенными координатами.

Move – метод, перемещающая машин на заданное количество пикселов. Если функция приняла 0 и 1, то она передвинет машину на 0 пикселов по иксу и 1 пиксел по игреку. Если функция приняла в качестве параметров -1 и 1, то к координатам x и y будет прибавлено -1 и 1 соответственно.

GetLocation – метод, возвращающий место положение машины по определенной оси. В качестве параметра функция принимает переменную типа Char, указывающую на необходимую к возврату ось. Если в качестве параметра функция приняла ‘y’, то функция вернет значение переменной y.

Draw – функция прорисовывающая машину в его координатах.

Методы класса Car наследует класс User. Он необходим для хранения специфических методов пользователя.



Этот класс наследует методы класса Car, находящиеся в общей видимости.

Он содержит защищенное поле life, для хранения количества жизней игрока.

User – перегруженный конструктор для создания машины юзера в определенном месте и с определенным количеством жизней.

ChangeLife – метод, отнимающую одну жизнь у объекта юзера и возвращающая 1, если еще есть жизни, либо 0, если жизней больше нет.

GetLife – метод, возвращающий текущее количество жизней пользователя.

Методы GetLocation, Move и Draw вызывают соответствующие им методы класса Car.

Класс Enemy также наследует методы класса Car, и существует для описания специфических методов машин-«противников».



Этот класс также наследует методы класса Car, находящиеся в общей видимости.

Он содержит одно поле next – переменная типа указатель на объект типа Enemy, для образования вектора «врагов».

Enemy – перегруженный конструктор для создания машины врага в определенном месте. Так как полос движения в дороге только три, в качестве одного из параметров значение 1, -1 или 0 далее эта цифра умножается на -37 и посылается в качестве параметра х в конструктор класса Car (второй аргумент отправляется без изменений). Теперь машина-«противник» будет создана на определенной высоте и дороже.

Draw – вызывает соответствующий метод класса Car.

Move – метод для передвижения машины. Эта функция принимает в качестве параметров количество пикселей, на которое необходимо передвинуть машину по оси y и указатель на объект класса User.

Данный указатель необходим для опроса текущего положения машины игрока. Если передвижение возможно, функция возвращает 2; если нет, возвращается значение, возвращаемое методом класса User ChangeLife.

Класс Enemies наследует класс Enemy и необходим для создания непрерывного потока машин-«противников»



Для прямого доступа к полям класса Enemy, класс Enemies имеет тип наследования protected.

Класс содержит 4 поля:

count – для хранения информации о количестве пройденных машин;

и три указателя типа Enemy, для создания очереди из объектов типа Enemy. Указатель first указывает на начало очереди, last – на конец очереди и it указывает на текущий элемент.

Enemies – перегруженный конструктор для создания очереди с заданным в качестве параметра количеством «врагов».

Fill – метод для наполнения очереди заданным в качестве параметра количеством «врагов».

Timer – метод для поочередного перемещения элементов очереди. В качестве параметров он принимает указатель типа User. Благодаря указателям, организовывается цикл по всей очереди, у каждого из элементов которого вызывается функция Move, в качестве параметров которой передается необходимое для перемещения количество пикселов и, собственно, указатель на объект типа User, принятый ранее в качестве аргумента. Если метод какого-либо элемента вернет значение меньше двух, это значение возвращается и методом Move класса Enemies. Если все элементы совершили передвижение успешно, возвращается 2.

Draw – в качестве параметров эта функция принимает цифру, равную количеству пикселов экрана по высоте. Далее функция в цикле вызывает метод Draw каждого элемента очереди. Затем, если координата ‘y’ первого в очереди элемента равны или превышают высоте экрана, деленную на два, взятое с отрицательным знаком и отнятому от полученного значения еще двадцати, то это значит, что первый элемент вышел из игрового поля. Поле count наращивается единицей, первый элемент очереди удаляется, а в конец очереди добавляется новый элемент.

GetScore – возвращает значение поля count.

SetScore – устанавливает в поле count значение, принятое в качестве параметра.

Clean – очищает список, удаляя все его элементы.

Для прорисовки обочины применяется класс Road.



Данный класс имеет два поля целочисленного типа:

wid – для хранения данных о ширине экрана,

position – для хранения информации о позиции элемента бордюра, относительно оси y.

Road – перегруженный конструктор, присваивающий полям, значения принятые в качестве параметров.

GetLocation – метод возвращающий текущее положение элемента бордюра относительно оси y (значение поля position).

Move – метод, изменяющий позицию элемента бордюра путем изменения поля position. Так как движение дороги происходит по направлению сверху вниз, нет необходимости уточнять в какую сторону и по какой оси должен передвинуться бордюр.

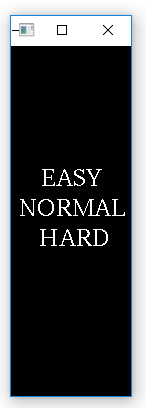
Draw – прорисовывает элемент бордюра в его координатах. Так как бордюр всегда по краям дороги, рисуются два элемента бордюра с каждой стороны. Благодаря полю wid известны координаты х, благодаря полю position известны координаты y элемента бордюра.

1. Форматы и протоколы.

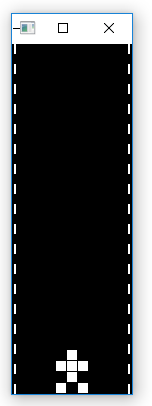
Для работы программы будет необходимо два текстовых файла для хранения информации о чемпионах – Top и Top2 для хранения очков и именах юзеров соответственно.

1. Тестирование.

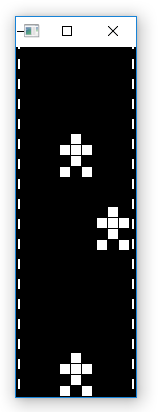
2.1



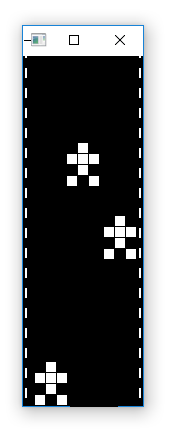
2.2



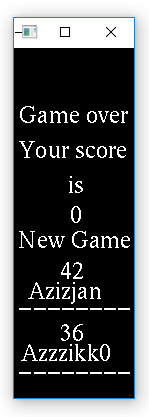
2.3



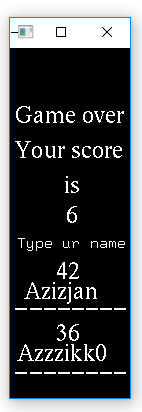
2.4



2.5



2.6



2.7



1. План работы. твыполнения Так как времени на выполнение курсового проекта выделяется много, составлять план просто бессмысленно. Само собой разумеется, что классы должны быть реализованы в порядке их наследования, то есть от базовых классов к классам-наследникам.

Листинг

Main

#include<GL\glut.h>

#include<ctime>

#include<vector>

#include<iostream>

#include <fstream>

#include"Road.h"

#include"Enemies.h"

#include"User.h"

#include <algorithm>

using namespace std;

vector <Road> road;

float sensitivity = 5.1f, speed, complexity = 0;

int WinWid = 120, WinHei = 350, WinScroll = 0;

User\* user;

Enemies enemies(30);

bool game = true, list = false;

string name;

bool isChamp = true; // if in the champions file is no same scores user well put his name in the list

struct Champions

{

string name;

int score;

Champions(int k, const std::string& s) : score(k), name(s) {}

};

struct less\_than\_key // to sort vector.

{

inline bool operator() (const Champions struct1, const Champions struct2)

{

return (struct1.score > struct2.score);

}

};

vector <Champions> Champs;

void Keyboard(unsigned char key, int x, int y);

void SKeyboard(int key, int x, int y);

void Initialize();

void Draw();

void Timer(int);

void Timer2(int);

void MousePressed(int, int, int, int);

void ComplexityTimer(int);

void ComplexityDraw();

void ComplexityMouse(int, int, int, int);

void ComplexityKeyboard(unsigned char key, int x, int y);

void GameOverDraw();

void GameOverTyping(unsigned char key, int x, int y);

void GameOverTimer(int);

int main(int argc, char\*\* argv)

{

srand(time(NULL));

for (int i = WinHei / 2; i >= -WinHei / 2; i -= 20)

{

Road curb(i, WinWid);

road.push\_back(curb);

}

//Initialization

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(WinWid, WinHei);

glutInitWindowPosition(450, 130);

glutCreateWindow("Race");

//Registration

glutDisplayFunc(ComplexityDraw);

glutTimerFunc(10, ComplexityTimer, 10);

Initialize();

glutMainLoop();

system("pause");

return 0;

}

void ComplexityTimer(int value)

{

if (complexity)

{

enemies.Clean();

enemies.Fill(20);

if (complexity == 1)

user = new User(WinHei, 5);

else if (complexity > 1 && complexity < 2)

user = new User(WinHei, 3);

else

user = new User(WinHei, 1);

glutKeyboardFunc(Keyboard);

glutSpecialFunc(SKeyboard);

glutDisplayFunc(Draw);//Drawing

glutPostRedisplay();

glutTimerFunc(speed = 20 / complexity, Timer, 1);

}

else

{

glutKeyboardFunc(ComplexityKeyboard);

glutDisplayFunc(ComplexityDraw);

glutPostRedisplay();

glutTimerFunc(10, ComplexityTimer, 10);

}

}

void ComplexityDraw()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glRasterPos2f(-30.0f, 35.0f);

for (char \*c = "EASY"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*c);

}

glRasterPos2f(-51.0f, 5.0f);

for (char \*c = "NORMAL"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*c);

}

glRasterPos2f(-31.0f, -25.0f);

for (char \*c = "HARD"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*c);

}

glutMouseFunc(ComplexityMouse);

glutSwapBuffers();

}

void ComplexityMouse(int button, int state, int x, int y)

{

y = -(y - (WinHei / 2));

x = x - (WinWid / 2);

if (x > -30 && x < 30 && y > 36 && y < 52 && state == GLUT\_DOWN)

{

complexity = 1;

}

if (x > -51 && x < 53 && y > 5 && y < 22 && state == GLUT\_DOWN)

{

complexity = 1.9;

}

if (x > -31 && x < 33 && y > -25 && y < -8 && state == GLUT\_DOWN)

{

complexity = 2.6;

}

}

void ComplexityKeyboard(unsigned char key, int x, int y)

{

switch ((char)key)

{

case 'e': complexity = 1; break;

case 'E': complexity = 1; break;

case 'n': complexity = 1.9; break;

case 'N': complexity = 1.9; break;

case 'h': complexity = 2.6; break;

case 'H': complexity = 2.6; break;

}

}

void MousePressed(int button, int state, int x, int y)

{

if ((button == 3) || (button == 4)) // It's a wheel event

{

// Each wheel event reports like a button click, GLUT\_DOWN then GLUT\_UP

if (state == GLUT\_UP) return; // Disregard redundant GLUT\_UP events

WinScroll += (button == 3) ? (WinScroll > 0) ? -1 : 0 : 1;

}

y = -(y - (WinHei / 2));

x = x - (WinWid / 2);

if (!isChamp && x > -55 && x < 56 && y > -25 && y < -13 && state == GLUT\_DOWN)

{

game = true;

complexity = 0;

enemies.SetScore(0);

isChamp = true;

WinScroll = 0;

}

}

void Timer(int value)

{

for (vector<Road>::iterator i = road.begin(); i != road.end(); i++)

i->Move();

short cond;

switch (enemies.Timer(user))

{

case 1:

enemies.Clean();

enemies.Fill(20);

cond = user->GetLife();

user->~User();

user = new User(WinHei, cond);

glutPostRedisplay();

glutTimerFunc(speed = 20/complexity, Timer, speed);

break;

case 2:

glutPostRedisplay();

glutTimerFunc(speed -= speed \* 0.001 / 100, Timer, speed);

break;

case 0:

user->~User();

game = false;

ifstream scores;

scores.open("Top.txt");

int num;

int sc = round(complexity) \* enemies.GetScore();

while (scores >> num)

{

if (num == sc)

isChamp = false;

}

//complexity = 0;

scores.close();

glutDisplayFunc(GameOverDraw);

glutKeyboardFunc(GameOverTyping);

glutPostRedisplay();

glutTimerFunc(10, GameOverTimer, 1);

break;

}

}

void GameOverDraw()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glRasterPos2f(-54.0f, 100.0f);

for (char \*c = "Game over"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*c);

}

glRasterPos2f(-54.0f, 65.0f);

for (char \*c = "Your score"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*c);

}

glRasterPos2f(-6.0f, 30.0f);

for (char \*c = "is"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*c);

}

int sc = round(complexity) \* enemies.GetScore();

std::string s = std::to\_string(sc);

char const \*pchar = s.c\_str();

float num = s.length() \* 7.5 / 2;

glRasterPos2f(-num, 0.0f);

while (\*pchar != '\0')

{

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*pchar);

pchar++;

}

if (name.length() && isChamp) {

glRasterPos2f(-50, -25);

char const \*pcharr = name.c\_str();

while (\*pcharr != '\0')

{

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*pcharr);

pcharr++;

}

}

else

{

if (isChamp) {

glRasterPos2f(-52, -25);

for (char \*c = "Type ur name"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_9\_BY\_15, \*c);

}

}

else

{

glRasterPos2f(-55, -25);

for (char \*c = "New Game"; \*c != '\0'; c++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*c);

}

glutMouseFunc(MousePressed);

}

}

ifstream names, scores;

names.open("Top2.txt", ios::in);

scores.open("Top.txt", ios::in);

int i = -55, scr;

while (scores >> scr) {

string ss;

getline(names, ss);

std::string sss = std::to\_string(scr);

float n = sss.length() \* 14 / 2;

if (i + WinScroll < -45) {

glRasterPos2f(-n, i + WinScroll);

char const \*ppchar = sss.c\_str();

while (\*ppchar != '\0') {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*ppchar);

ppchar++;

}

}

i -= 20;

if (i + WinScroll < -45) {

float numm = ss.length() \* 13 / 2;

glRasterPos2f(-numm, i + WinScroll);

char const \*har = ss.c\_str();

while (\*har != '\0') {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*har);

har++;

}

}

i -= 18;

if (i + WinScroll < -45) {

glRasterPos2f(-WinWid / 2 + 5, i + WinScroll);

for (const char \*ar = "---------------"; \*ar != '\0'; \*ar++) {

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*ar);

ar++;

}

}

i -= 25;

}

names.close(); scores.close();

glutSwapBuffers();

}

void GameOverTimer(int value)

{

if (game && !complexity)

{

glutDisplayFunc(ComplexityDraw);

glutPostRedisplay();

glutMouseFunc(ComplexityMouse);

glutTimerFunc(10, ComplexityTimer, 1);

}

else

{

glutDisplayFunc(GameOverDraw);

if (isChamp)

glutKeyboardFunc(GameOverTyping);

glutPostRedisplay();

glutTimerFunc(10, GameOverTimer, 1);

}

}

void GameOverTyping(unsigned char key, int x, int y)

{

if (key == 13) // Enter key

{

fstream names, scores;

names.open("Top2.txt", ios::in | ios::out);

scores.open("Top.txt", ios::in | ios::out);

int num;

while (scores >> num)

{

string n; getline(names, n);

Champs.push\_back(Champions(num, n));

}

Champs.push\_back(Champions(enemies.GetScore() \* round(complexity), name)); // to add current user

name = ""; // to empty the name field so it will not appear in new game

names.close(); scores.close();

std::ofstream ofs, ofs2; // to empty file

ofs.open("Top.txt", std::ofstream::out | std::ofstream::trunc); ofs2.open("Top2.txt", std::ofstream::out | std::ofstream::trunc);

ofs.close(); ofs2.close();

names.open("Top2.txt", ios::in | ios::out);

scores.open("Top.txt", ios::in | ios::out);

std::sort(Champs.begin(), Champs.end(), less\_than\_key());

for (vector<Champions>::iterator i = Champs.begin(); i != Champs.end(); i++)

{

names << i->name << "\n";

scores << i->score << "\n";

}

Champs.clear();

names.close(); scores.close();

glutKeyboardFunc(Keyboard);

isChamp = false;

}

else if (key == 8) // backspace

{

if (name.length())

name.erase(name.length() - 1, 1);

}

else // regular text

{

if (name.length() < 9) {

string l;

l = (char)key;

name.append(l);

}

}

}

void Timer2(int value)

{

if (game)

{

glutKeyboardFunc(ComplexityKeyboard);

glutDisplayFunc(ComplexityDraw);

glutPostRedisplay();

glutTimerFunc(1, ComplexityTimer, 1);

}

else

{

ifstream scores;

scores.open("Top.txt");

int num; float flag = true; // if we need to insert the user to the champions list

while (scores >> num)

{

if (num == enemies.GetScore())

flag = false; //exit if there is the same result

}

scores.close();

if (flag)

{

glutKeyboardFunc(GameOverTyping);

glutDisplayFunc(GameOverDraw);

glutPostRedisplay();

glutTimerFunc(10, GameOverTimer, 1);

}

}

}

void Draw()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

vector <Road>::iterator i = road.begin();

while (i != road.end())

{

i->Draw();

if (i->GetPosition() <= -WinHei / 2 - 5) {

Road r(WinHei / 2 + 5, WinWid);

road.push\_back(r);

i = road.erase(i);

}

else

++i;

}

enemies.Draw(WinHei);

user->Draw();

glutSwapBuffers(); // for GLUT\_DOUBLE

}

void SKeyboard(int key, int x, int y)

{

switch (key)

{

case GLUT\_KEY\_LEFT:

if (user->GetLocation("x") >= -WinWid / 2 + 19.5 + sensitivity)

user->Move(-sensitivity, 0);

break;

case GLUT\_KEY\_RIGHT:

if (user->GetLocation("x") <= WinWid / 2 - 19.5 - sensitivity)

user->Move(sensitivity, 0);

break;

case GLUT\_KEY\_UP:

if (user->GetLocation("y") <= WinHei / 2 - 17 - sensitivity)

user->Move(0, sensitivity);

break;

case GLUT\_KEY\_DOWN:

if (user->GetLocation("y") >= -WinHei / 2 + 30)

user->Move(0, -sensitivity);

break;

}

}

void Keyboard(unsigned char key, int x, int y)

{

switch (key)

{

case 'a':

if (user->GetLocation("x") >= -WinWid / 2 + 19.5 + sensitivity)

user->Move(-sensitivity, 0);

break;

case 'd':

if (user->GetLocation("x") <= WinWid / 2 - 19.5 - sensitivity)

user->Move(sensitivity, 0);

break;

case 'w':

if (user->GetLocation("y") <= WinHei / 2 - 17 - sensitivity)

user->Move(0, sensitivity);

break;

case 's':

if (user->GetLocation("y") >= -WinHei / 2 + 30)

user->Move(0, -sensitivity + 1);

break;

}

}

void Initialize()

{

glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-WinWid / 2, WinWid / 2, -WinHei / 2, WinHei / 2, -200.0, 200);

glMatrixMode(GL\_MODELVIEW);

}