Балтийский государственный технический университет  
«ВОЕНМЕХ» им. Д. Ф. Устинова

Факультет «О» Естественнонаучный

Кафедра О7 «Информационные системы и программная инженерия»

**Практическая работа №5**по дисциплине «Программирование на ЯВУ»  
на тему «SDL»

Выполнил:  
Студент Костров Г. Ю. Группа О712Б  
Преподаватель: Васюков В.М.

Санкт-Петербург  
2022 г

**Постановка задачи:**

Построить график функции

y = exp(-2) / (x-5)

при x€[-5,3]

Изобразить на экране движение корабля по морю с помощью клавиш управления курсором:

при нажатии на клавишу «>» начинается движение вправо, при нажатии на клавишу «<» – вле-

во, при нажатии на клавишу «^» корабль останавливается.

**Текст программы:**

**// Задание 1**

#include "SDL.h"

#include "stdio.h"

#include "iostream"

#include "math.h"

#include "cmath"

#define SCREEN\_WIDTH 1280

#define SCREEN\_HEIGHT 720

float map(float minX, float maxX, float minPxX, float maxPxX, float x)

{

return (x - minX) / (maxX - minX) \* (maxPxX - minPxX) + minPxX;

}

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

{

SDL\_Init(SDL\_INIT\_EVERYTHING);

const auto Width = 1280;

const auto Heigth = 720;

SDL\_Window \*w = SDL\_CreateWindow("Graphs", 63, 126, Width, Heigth, SDL\_WINDOW\_BORDERLESS);

SDL\_Renderer \*r = SDL\_CreateRenderer(w, -1, SDL\_RENDERER\_ACCELERATED);

for (auto done = false; !done;)

{

SDL\_Event e;

while (SDL\_WaitEvent(&e))

{

switch (e.type)

{

case SDL\_QUIT:

done = true;

break;

}

SDL\_SetRenderDrawColor(r, 0xff, 0xff, 0xff, 0xff);

SDL\_RenderClear(r);

SDL\_SetRenderDrawColor(r, 0xee, 0xee, 0xee, 0xff);

// Линии, которые делят окно на квадратики)

{

for (float x = -10; x < 10; ++x) {

SDL\_RenderDrawLine(r,

map(-10, 10, 0, Width, x), map(-5, 5, Heigth, 0, -5),

map(-10, 10, 0, Width, x), map(-5, 5, Heigth, 0, +5));

}

for (float y = -5; y < 5; ++y) {

SDL\_RenderDrawLine(r,

map(-10, 10, 0, Width, -10), map(-5, 5, Heigth, 0, y),

map(-10, 10, 0, Width, 10), map(-5, 5, Heigth, 0, y));

}

}

// Ось абцисс и ординат + функция

{

SDL\_SetRenderDrawColor(r, 0xee, 0xc0, 0xc0, 0xff);

SDL\_RenderDrawLine(r,

map(-10, 10, 0, Width, -10), map(-5, 5, Heigth, 0, 0),

map(-10, 10, 0, Width, 10), map(-5, 5, Heigth, 0, 0));

SDL\_RenderDrawLine(r,

map(-10, 10, 0, Width, 0), map(-5, 5, Heigth, 0, -5),

map(-10, 10, 0, Width, 0), map(-5, 5, Heigth, 0, +5));

SDL\_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xff);

}

auto oldX = -1;

auto oldY = 0;

for (float x = -5; x <= 3; x += 0.01) // x от -5 до 3 по условию задачи

{

auto y = exp(-2) / (x - 5); // ТУТ МОЖНО ИЗМЕНЯТЬ ФУНКЦИЮ

auto newX = map(-10, 10, 0, Width, x);

auto newY = map(-5, 5, Heigth, 0, y);

if (oldX >= 0)

SDL\_RenderDrawLine(r, oldX, oldY, newX, newY);

oldX = newX;

oldY = newY;

}

SDL\_RenderPresent(r);

}

SDL\_Quit();

}

return 0;

}

**// Задание 2**  
  
//Using SDL, SDL\_image, SDL\_ttf, standard IO, strings, and string streams  
#include <SDL.h>  
#include <SDL\_image.h>  
#include <SDL\_ttf.h>  
#include <cstdio>  
#include <string>  
#include <sstream>  
#include <vector>  
  
//Screen dimension constants  
const int SCREEN\_WIDTH = 1280;  
const int SCREEN\_HEIGHT = 960;  
  
//Texture wrapper class  
class LTexture {  
public:  
 //Initializes variables  
 LTexture();  
  
 //Deallocates memory  
 ~LTexture();  
  
 //Loads image at specified path  
 bool loadFromFile(std::string path);  
  
#if defined(SDL\_TTF\_MAJOR\_VERSION)  
  
 //Creates image from font string  
 bool loadFromRenderedText(std::string textureText, SDL\_Color textColor);  
  
#endif  
  
 //Deallocates texture  
 void free();  
  
 //Set color modulation  
 void setColor(Uint8 red, Uint8 green, Uint8 blue);  
  
 //Set blending  
 void setBlendMode(SDL\_BlendMode blending);  
  
 //Set alpha modulation  
 void setAlpha(Uint8 alpha);  
  
 //Renders texture at given point  
 void render(int x, int y, SDL\_Rect \*clip = NULL, double angle = 0.0, SDL\_Point \*center = NULL,  
 SDL\_RendererFlip flip = SDL\_FLIP\_NONE);  
  
 //Gets image dimensions  
 int getWidth();  
  
 int getHeight();  
  
private:  
 //The actual hardware texture  
 SDL\_Texture \*mTexture;  
  
 //Image dimensions  
 int mWidth;  
 int mHeight;  
};  
  
//Starts up SDL and creates window  
bool init();  
  
//Loads media  
bool loadMedia();  
  
//Frees media and shuts down SDL  
void close();  
  
//The window we'll be rendering to  
SDL\_Window \*gWindow = NULL;  
  
//The window renderer  
SDL\_Renderer \*gRenderer = NULL;  
  
//Globally used font  
TTF\_Font \*gFont = NULL;  
  
//Scene textures  
LTexture gPromptTextTexture;  
LTexture gInputTextTexture;  
  
std::string nameOfFilesForClock[4] = {"clock.bmp", "hour.bmp", "minute.bmp", "second.bmp"};  
  
std::vector<LTexture> clockTextures(4);  
  
LTexture::LTexture() {  
 //Initialize  
 mTexture = NULL;  
 mWidth = 0;  
 mHeight = 0;  
}  
  
LTexture::~LTexture() {  
 //Deallocate  
 free();  
}  
  
bool LTexture::loadFromFile(std::string path) {  
 //Get rid of preexisting texture  
 free();  
  
 //The final texture  
 SDL\_Texture \*newTexture = NULL;  
  
 //Load image at specified path  
 SDL\_Surface \*loadedSurface = IMG\_Load(path.c\_str());  
 if (loadedSurface == NULL) {  
 printf("Unable to load image %s! SDL\_image Error: %s\n", path.c\_str(), IMG\_GetError());  
 } else {  
 //Color key image  
 SDL\_SetColorKey(loadedSurface, SDL\_TRUE, SDL\_MapRGB(loadedSurface->format, 0, 0xFF, 0xFF));  
  
 //Create texture from surface pixels  
 newTexture = SDL\_CreateTextureFromSurface(gRenderer, loadedSurface);  
 if (newTexture == NULL) {  
 printf("Unable to create texture from %s! SDL Error: %s\n", path.c\_str(), SDL\_GetError());  
 } else {  
 //Get image dimensions  
 mWidth = loadedSurface->w;  
 mHeight = loadedSurface->h;  
 }  
  
 //Get rid of old loaded surface  
 SDL\_FreeSurface(loadedSurface);  
 }  
  
 //Return success  
 mTexture = newTexture;  
 return mTexture != NULL;  
}  
  
#if defined(SDL\_TTF\_MAJOR\_VERSION)  
  
bool LTexture::loadFromRenderedText(std::string textureText, SDL\_Color textColor) {  
 //Get rid of preexisting texture  
 free();  
  
 //Render text surface  
 SDL\_Surface \*textSurface = TTF\_RenderText\_Solid(gFont, textureText.c\_str(), textColor);  
 if (textSurface != NULL) {  
 //Create texture from surface pixels  
 mTexture = SDL\_CreateTextureFromSurface(gRenderer, textSurface);  
 if (mTexture == NULL) {  
 printf("Unable to create texture from rendered text! SDL Error: %s\n", SDL\_GetError());  
 } else {  
 //Get image dimensions  
 mWidth = textSurface->w;  
 mHeight = textSurface->h;  
 }  
  
 //Get rid of old surface  
 SDL\_FreeSurface(textSurface);  
 } else {  
 printf("Unable to render text surface! SDL\_ttf Error: %s\n", TTF\_GetError());  
 }  
  
  
 //Return success  
 return mTexture != NULL;  
}  
  
#endif  
  
void LTexture::free() {  
 //Free texture if it exists  
 if (mTexture != NULL) {  
 SDL\_DestroyTexture(mTexture);  
 mTexture = NULL;  
 mWidth = 0;  
 mHeight = 0;  
 }  
}  
  
void LTexture::setColor(Uint8 red, Uint8 green, Uint8 blue) {  
 //Modulate texture rgb  
 SDL\_SetTextureColorMod(mTexture, red, green, blue);  
}  
  
void LTexture::setBlendMode(SDL\_BlendMode blending) {  
 //Set blending function  
 SDL\_SetTextureBlendMode(mTexture, blending);  
}  
  
void LTexture::setAlpha(Uint8 alpha) {  
 //Modulate texture alpha  
 SDL\_SetTextureAlphaMod(mTexture, alpha);  
}  
  
void LTexture::render(int x, int y, SDL\_Rect \*clip, double angle, SDL\_Point \*center, SDL\_RendererFlip flip) {  
 //Set rendering space and render to screen  
 SDL\_Rect renderQuad = {x, y, mWidth, mHeight};  
  
 //Set clip rendering dimensions  
 if (clip != NULL) {  
 renderQuad.w = clip->w;  
 renderQuad.h = clip->h;  
 }  
  
 //Render to screen  
 SDL\_RenderCopyEx(gRenderer, mTexture, clip, &renderQuad, angle, center, flip);  
}  
  
int LTexture::getWidth() {  
 return mWidth;  
}  
  
int LTexture::getHeight() {  
 return mHeight;  
}  
  
bool init() {  
 //Initialization flag  
 bool success = true;  
  
 //Initialize SDL  
 if (SDL\_Init(SDL\_INIT\_VIDEO) < 0) {  
 printf("SDL could not initialize! SDL Error: %s\n", SDL\_GetError());  
 success = false;  
 } else {  
 //Set texture filtering to linear  
 if (!SDL\_SetHint(SDL\_HINT\_RENDER\_SCALE\_QUALITY, "1")) {  
 printf("Warning: Linear texture filtering not enabled!");  
 }  
  
 //Create window  
 gWindow = SDL\_CreateWindow("Time", SDL\_WINDOWPOS\_UNDEFINED, SDL\_WINDOWPOS\_UNDEFINED, SCREEN\_WIDTH,  
 SCREEN\_HEIGHT, SDL\_WINDOW\_SHOWN);  
 if (gWindow == NULL) {  
 printf("Window could not be created! SDL Error: %s\n", SDL\_GetError());  
 success = false;  
 } else {  
 //Create vsynced renderer for window  
 gRenderer = SDL\_CreateRenderer(gWindow, -1, SDL\_RENDERER\_ACCELERATED | SDL\_RENDERER\_PRESENTVSYNC);  
 if (gRenderer == NULL) {  
 printf("Renderer could not be created! SDL Error: %s\n", SDL\_GetError());  
 success = false;  
 } else {  
 //Initialize renderer color  
 SDL\_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);  
  
 //Initialize PNG loading  
 int imgFlags = IMG\_INIT\_PNG;  
 if (!(IMG\_Init(imgFlags) & imgFlags)) {  
 printf("SDL\_image could not initialize! SDL\_image Error: %s\n", IMG\_GetError());  
 success = false;  
 }  
  
 //Initialize SDL\_ttf  
 if (TTF\_Init() == -1) {  
 printf("SDL\_ttf could not initialize! SDL\_ttf Error: %s\n", TTF\_GetError());  
 success = false;  
 }  
 }  
 }  
 }  
  
 return success;  
}  
  
bool loadMedia() {  
 //Loading success flag  
 bool success = true;  
  
 //Open the font  
 gFont = TTF\_OpenFont("courier.ttf", 28);  
 if (gFont == NULL) {  
 printf("Failed to load font! SDL\_ttf Error: %s\n", TTF\_GetError());  
 success = false;  
 } else {  
 //Render the prompt  
 SDL\_Color textColor = {0, 0, 0, 0xFF};  
 if (!gPromptTextTexture.loadFromRenderedText("Enter time in format \*\*:\*\* :", textColor)) {  
 printf("Failed to render prompt text!\n");  
 success = false;  
 }  
 }  
 for (int i = 0; i < 4; i++) {  
 if (!clockTextures.at(i).loadFromFile(nameOfFilesForClock[i])) {  
 printf("Failed to load texture!\n");  
 success = false;  
 }  
 }  
 return success;  
}  
  
void close() {  
 //Free loaded images  
 gPromptTextTexture.free();  
 gInputTextTexture.free();  
  
 //Free global font  
 TTF\_CloseFont(gFont);  
 gFont = NULL;  
  
 //Destroy window  
 SDL\_DestroyRenderer(gRenderer);  
 SDL\_DestroyWindow(gWindow);  
 gWindow = NULL;  
 gRenderer = NULL;  
  
 //Quit SDL subsystems  
 TTF\_Quit();  
 IMG\_Quit();  
 SDL\_Quit();  
}  
  
int main(int argc, char \*args[]) {  
 //Start up SDL and create window  
 if (!init()) {  
 printf("Failed to initialize!\n");  
 } else {  
 //Load media  
 if (!loadMedia()) {  
 printf("Failed to load media!\n");  
 } else {  
 //Main loop flag  
 bool quit = false;  
  
 //Event handler  
 SDL\_Event e;  
  
 //Set text color as black  
 SDL\_Color textColor = {0, 0, 0, 0xFF};  
  
 //The current input text.  
 std::string inputText = "text";  
 gInputTextTexture.loadFromRenderedText(inputText.c\_str(), textColor);  
  
 //Enable text input  
 SDL\_StartTextInput();  
  
 int hour, minute, second;  
 int angleOfSeconds = 0;  
 bool inputTime = false;  
 //While application is running  
 while (!quit) {  
 //The rerender text flag  
 bool renderText = false;  
  
 //Handle events on queue  
 while (SDL\_PollEvent(&e) != 0) {  
 //User requests quit  
 if (e.type == SDL\_QUIT) {  
 quit = true;  
 }  
 //Special key input  
 else if (e.type == SDL\_KEYDOWN) {  
 //Handle backspace  
 if (e.key.keysym.sym == SDLK\_BACKSPACE && inputText.length() > 0) {  
 //lop off character  
 inputText.pop\_back();  
 renderText = true;  
 }  
 //Handle copy  
 else if (e.key.keysym.sym == SDLK\_c && SDL\_GetModState() & KMOD\_CTRL) {  
 SDL\_SetClipboardText(inputText.c\_str());  
 }  
 //Handle paste  
 else if (e.key.keysym.sym == SDLK\_v && SDL\_GetModState() & KMOD\_CTRL) {  
 inputText = SDL\_GetClipboardText();  
 renderText = true;  
 }  
 }  
 //Special text input event  
 else if (e.type == SDL\_TEXTINPUT) {  
 //Not copy or pasting  
 if (!(SDL\_GetModState() & KMOD\_CTRL &&  
 (e.text.text[0] == 'c' || e.text.text[0] == 'C' || e.text.text[0] == 'v' ||  
 e.text.text[0] == 'V'))) {  
 //Append character  
 inputText += e.text.text;  
 renderText = true;  
 }  
 }  
 if (e.type == SDL\_KEYUP && e.key.repeat == 0 && e.key.keysym.sym == SDLK\_RETURN &&  
 !inputText.empty() && (inputText.length() == 5)) {  
 if (std::isdigit(inputText[0]) && std::isdigit(inputText[1]) && std::isdigit(inputText[3]) && std::isdigit(inputText[4]) && inputText[2] == ':') {  
 if ((std::stoi(inputText.substr(0, 2)) < 24) && (std::stoi(inputText.substr(3, 2)) < 60)) {  
 hour = std::stoi(inputText.substr(0, 2));  
 minute = std::stoi(inputText.substr(3, 2));  
 if (hour >= 12)  
 hour -= 12;  
 inputTime = true;  
 SDL\_StopTextInput();  
 }  
 }  
 }  
 }  
 //Clear screen  
 SDL\_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);  
 SDL\_RenderClear(gRenderer);  
 if (!inputTime) {  
 //Rerender text if needed  
 if (renderText) {  
 //Text is not empty  
 if (!inputText.empty()) {  
 //Render new text  
 gInputTextTexture.loadFromRenderedText(inputText, textColor);  
 }  
 //Text is empty  
 else {  
 //Render space texture  
 gInputTextTexture.loadFromRenderedText(" ", textColor);  
 }  
 }  
 //Render text textures  
 gPromptTextTexture.render((SCREEN\_WIDTH - gPromptTextTexture.getWidth()) / 2, 0);  
 gInputTextTexture.render((SCREEN\_WIDTH - gInputTextTexture.getWidth()) / 2,  
 gPromptTextTexture.getHeight());  
 } else {  
 clockTextures.at(0).render((SCREEN\_WIDTH - clockTextures.at(0).getWidth()) / 2, (SCREEN\_HEIGHT - clockTextures.at(0).getHeight()) / 2);  
 clockTextures.at(3).render((SCREEN\_WIDTH - clockTextures.at(0).getWidth()) / 2 + 195, (SCREEN\_HEIGHT - clockTextures.at(0).getHeight()) / 2 + 20,  
 nullptr, angleOfSeconds++);  
 if (angleOfSeconds % 360 == 0)  
 minute += 1;  
 clockTextures.at(1).render((SCREEN\_WIDTH - clockTextures.at(0).getWidth()) / 2 + 200, (SCREEN\_HEIGHT - clockTextures.at(0).getHeight()) / 2 + 120, nullptr, hour \* 30);  
 clockTextures.at(2).render((SCREEN\_WIDTH - clockTextures.at(0).getWidth()) / 2 + 200, (SCREEN\_HEIGHT - clockTextures.at(0).getHeight()) / 2 + 80, nullptr, minute \* 6);  
  
 }  
  
 //Update screen  
 SDL\_RenderPresent(gRenderer);  
 }  
  
 //Disable text input  
 SDL\_StopTextInput();  
 }  
 }  
  
 //Free resources and close SDL  
 close();  
  
 return 0;  
}

**Результат работы программы:**

Рисунок 1:Задание 1.

Рисунок 2: Задание 2.

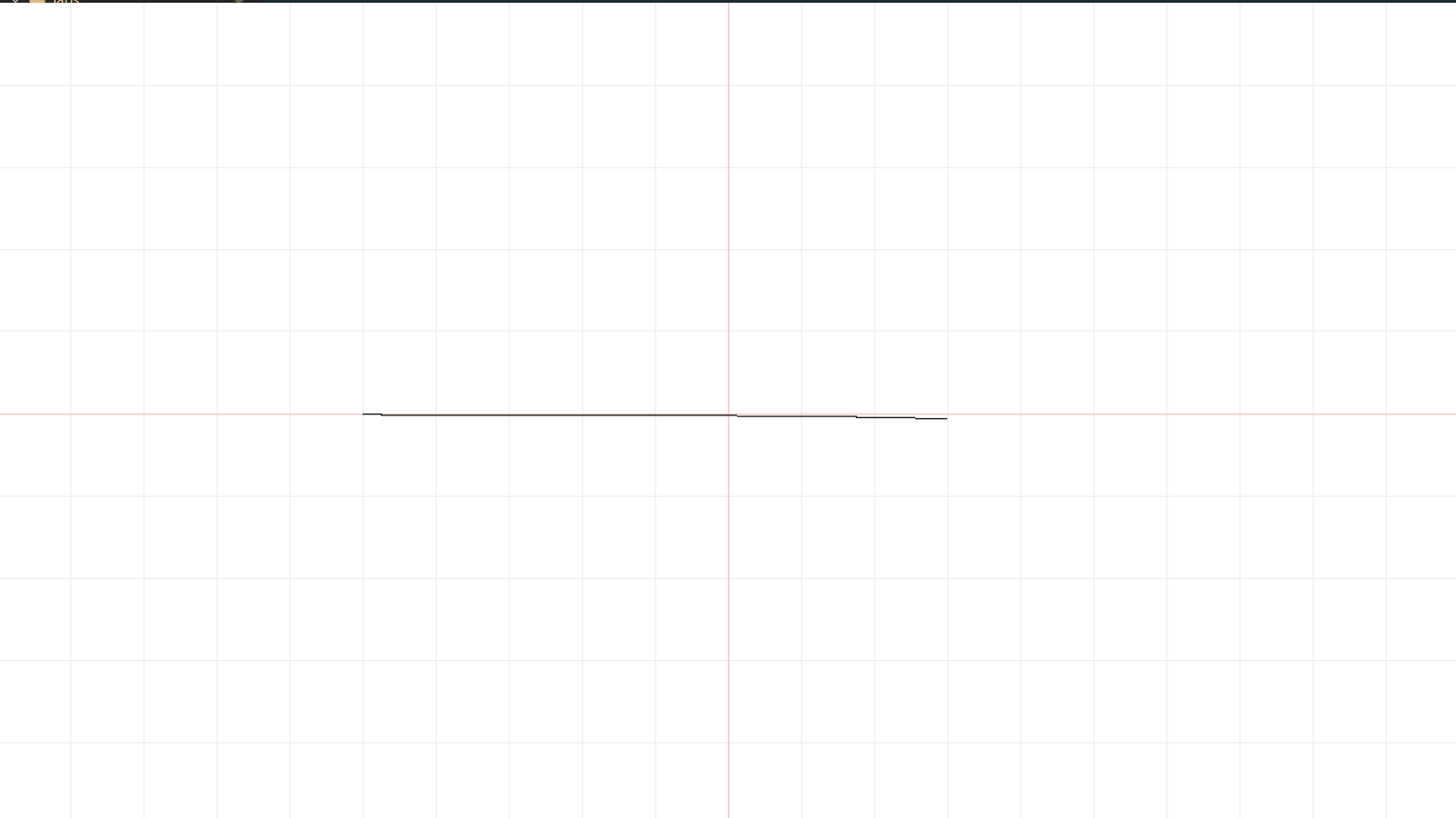


Рисунок 1

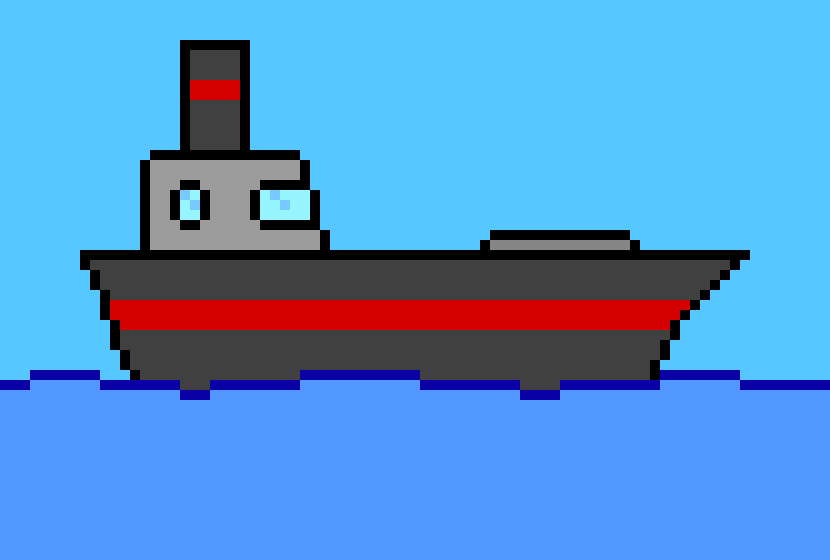


Рисунок 2