МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Восточно-Сибирский государственный университет технологий и управления»

Электротехнический факультет
Кафедра систем информатики

Курсовой проект

по дисциплине «Программирование»

Тема:

«Реализация редактора космической ракеты»

Выполнил: студент гр. Б661
_____ Коковихин А.В.
Руководитель: стар.преп. каф. СИ
_____ Мердыгеев Б.Д.
Оценка: _____
Дата защиты: _____

Улан-Удэ
2022

ВОСТОЧНО-СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра систем информатики

З А Д А Н И Е

на курсовой проект

Дисциплина: Программирование

Тема: Реализация редактора космической ракеты

Исполнитель: Коковихин А.В.

Руководитель: Мердыгеев Б.Д.

Краткое содержание проекта: данная курсовая работа посвящена написанию редактора космической ракеты, прототипом для которой является редактор из игры: «Kerbal Space Programm» (KSP)

1. Теоретическая часть: словесная постановка задачи, описание оригинальной игры, Описание функций редактора, отличия от оригинала, специфика консольных приложений

2. Практическая часть: формальная постановка задачи, UML модель, алгоритм решения задачи, реализация управления интерфейсом «горячими» клавишами, разработка приложения и тестирование полученного программного продукта

Сроки выполнения работы по календарному плану*:

Этап 1. Теоретический раздел – 15% к 5 неделе.

Этап 2. Проектный раздел – 40% к 8 неделе.

Этап 3. Программный раздел – 70% к 12 неделе.

Этап 4. Экспериментальный раздел – 90% к 14 неделе.

Этап 5. Защита – 100 % к 16 неделе.

Требования к оформлению:

1. Отчет по курсовой работе должен быть представлен в электронной и твердой копиях.
2. Объем отчета должен быть не менее 20 машинописных страниц без учета приложений.
3. Отчет оформляется по ГОСТ 7.32-2001.

Руководитель работы _____

Исполнитель _____

Дата выдачи "__" _____ 2020 г.

АННОТАЦИЯ

Данный проект будет представлять консольное приложение для «рисования» космических ракет используя символы ASCII таблицы. В качестве «эталона» был выбран цех вертикальной сборки из игры KSP. Ключевое отличие от оригинала – графика будет 2-х мерной и в консоли.

Целью курсовой работы является закрепление теоретических знаний и получение практических знаний по основам программирования и объектно-ориентированному программированию. Для этого были рассмотрены теоретические вопросы реализации объектно-ориентированного программирования, на основании которых была разработана программная реализация редактора космических ракет

Основные результаты работы: созданы ASCII спрайты частей ракеты, разработана система строительства «по частям», созданы алгоритмы сохранения и загрузки конструкций, скрипт копирование изображения с экрана в буфер обмена и программа, реализующая данный редактор, написанный на языке программирования C++.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Недавно по сети прокатился тренд на создание ASCII графики. И 3-х мерный рендеринг вращающегося бублика, Doom в консоли и т.д. Вдохновившись этой идей было принято решение создать полноценный редактор космической ракеты используя ASCII графику.

Результатом работы является игра, работающая на компьютере с ОС Windows. Данная реализация является наиболее приближенной к оригиналу, но при этом обладает собственной уникальной графикой.

Целью курсовой работы является закрепление теоретических знаний и получение практических знаний по объектно-ориентированному программированию. Для достижения поставленной цели были решены следующие задачи:

1) анализ предметной области;
2) разработка объектной модели;
3) создание спрайтов деталей
4) разработка интерфейса и управления;
5) разработка программы;
6) тестирование работоспособности программы.

Расчетно-пояснительная записка состоит из:

- теоретической части, содержащей словесную постановку задачи, описание оригинальной игры, описание функций редактора, отличия реализации от оригинала, специфика консольных приложений;
- практической части, содержащей формальную постановку задачи, UML модель и алгоритм работы программы.

1 Теоретический раздел

1.1 Словесная постановка задачи

Необходимо написать программу, повторяющую функционал «Цеха вертикальной сборки» из игры (Kerbal Space Program) KSP, используя ASCII графику. Редактор должен быть приспособлен к деталям различных форм и размеров, так же в нём должны быть реализованы механики присоединения, удаления и добавления деталей. Задачу необходимо решить, используя объектно-ориентированное программирование и язык C++ без использования сторонних фреймворков и движков. Дополнительно требуется создать дружественный пользователю интерфейс используя для отрисовки только консоль.

1.2 Описание оригинальной игры (KSP)

Kerbal Space Program – Игра песочница, симулятор космической компании: «Почувствуйте себя Илоном Маском». Игроку предстоит:

- проектировать ракеты, самолёты, спутники, роверы и прочие «машины» для того, чтобы проводить исследования в Солнечной системе и выполнять контракты;
- пилотировать собственные миссии;
- исследовать новые технологии и закупать новое оборудование/улучшать здания.

В цехе вертикальной сборки корабли центрируются по вертикальной оси, в нём игрок может размещать детали на проектируемый корабль перетаскивая их из списка деталей на ту позицию, где он её хочет закрепить. У деталей есть свои «точки крепления» для ровной установки одной детали на другую, чаще всего их 2: сверху и снизу. Помимо этого, игроку доступна функция симметричного копирования детали вокруг оси, для этого он выбирает число копий и устанавливает первую деталь, а уже редактор ставит остальные копии на свои места. Следующее «удобство» проецирование детали и её копий: как она будет расположена если подтвердить её размещение сейчас.

Правила редактора:

- первая деталь – главная, все остальные крепятся к ней:
- если отсоединить деталь - то отсоединятся все прикреплённые к ней;
- детали не прикреплённые к главной будут «неактивными», но их можно перемещать, и они не будут учувствовать при запуске;
- детали не должны пересекаться.

1.3 Описание функций редактора (продукта)

Ключевая функция – это добавление на ракету новых деталей. Пользователь будет выбирать деталь из списка (как и в оригинале) и размещать её на ракете. Так же при размещении будет учитываться пересечение с другими деталями (как в оригинале).

Удаление детали – ещё одна необходимая функция для редактирования ракеты. Потребуется дополнительно продумать поведение при удалении детали, имеющей несколько соединений.

Сохранение структуры ракеты в файл – в оригинале так же предусмотрено сохранить чертёж аппарата (вероятно, тоже в отдельный файл) оно и понятно, удобно хранить все свои наработки в памяти ПК, а не у себя в голове регулярно воспроизводя его.

Загрузка структуры ракеты из файла – неотъемлемый компаньон сохранения, зачем сохранять не имея возможность потом загрузить? В оригинале игрок может выбрать загрузить чертёж заместо текущего или добавить загружаемый чертёж как «неактивные» детали. Надо будет продумать в дальнейшем добавление шаблона к текущему или полное замещение.

Сохранение ASCII «картинки в» буфер обмена – то ради чего всё делается: получить изображение ракеты с возможностью его вставки в комментарии под видео, мессенджер и т.д.

1.4 Отличия от оригинала

В целом редактор будет максимально возможно соответствовать оригиналу. Но увы всё скопировать не получится. Так, например, симметрия не имеет смысла в 2х мерной графике, а также весьма проблематично будет реализовать «неактивные» детали. В эту же категорию уходит управление мышью, свободное перемещение деталей, поворот деталей (тоже есть в оригинале.

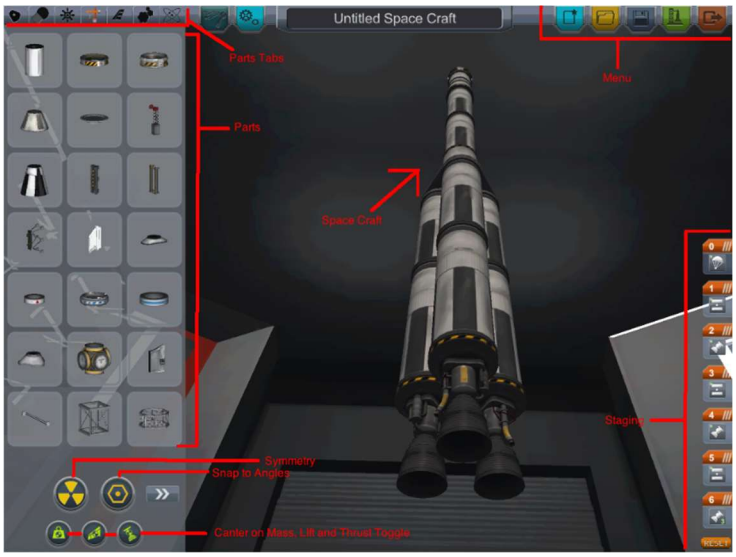

Рисунок 1 – Интерфейс редактора

Рисунок 2 – Прототип интерфейса

Правила редактора:

- первая деталь главная, всё дальнейшее строительство толкается от неё;
- нельзя сохранить пустой корабль;
- детали не должны пересекаться;
- для добавления детали игроку придётся переводить «курсор» в необходимую позицию используя горячие клавиши, например WASD.

1.5 Специфика консольных приложений

Консольный интерфейс накладывает ряд сложностей, таких как: отсутствие уже привычного всем курсора, выводимая графика ограничена набором символов ASCII (что конечно в нашем проекте не сильно и мешается). Отсюда следует трудность реализации привычных глазу кнопок, слайдеров и прочих удобных виджетов. Выходит, что управление будет осуществляется горячими клавишами и/или консольными командами, а вся графика будет символьной.

Ограничение номер один – интерфейс не получится сделать бесшовным. Ограничение номер два – «перетащить» деталь из списка на корабль не получится никак. Это означает то, что придётся полностью переосмыслить управление (оно будет в корне отличатся от оригинала).

Исходя из этого будет проще убрать «неактивные» детали т. к. они будут путать пользователя наслаиваясь на другие детали. Можно кончено попробовать возможность переключение (например клавишей TAB) между активным кораблём и неактивными деталями, но скорее всего это принесёт много хлопот.

Так же под нож попадает и вставка шаблона в текущий проект. Причина тут в сложности присоединить готовый шаблон, состоящий из множества деталей в текущий, да можно найти крайние точки присоединения, и отталкиваться от них, но данный алгоритм будет тяжёл в отладке.

Поворот деталей – к сожалению в таблице ASCII UTF-8 отсутствуют зеркальные символы, опять же да они есть в других таблицах, но синхронизировать это всё тоже задачка не из простых, а ведь ещё при выводе символа из другой таблице надо так же будет переключить текущую таблицу в консоли, и следить за тем, чтобы они совпадали. Что, конечно, неприятно, но что есть то есть, а что касается поворота на 180 градусов, то тут можно создать зеркальный спрайт, к счастью, некоторые спец символы имеют свои отражённые копии: «({[/ \]})».

Остаётся последний вопрос – консольные команды или горячие клавиши. Если учесть тот факт что управление в оригинальной игре не использует контекстное меню, а большая часть кнопок отвечает за функционал необходимый в KSP но не в текущем проекте то разумнее будет применять горячие клавиши: навигация – «WASD», удаление детали – X или Delete, отмена/возврат – Esc, Backspace, сохранение, загрузка, копия в буфер соответственно – Ctrl + S, Ctrl + O, Ctrl + C, справка/помощь – F2.

2 Проектный раздел

2.1 Формальная постановка задачи

Входные данные: горячие клавиши, файлы сохранённых проектов:

- ←/a/ф/4 (Влево), ↑w/ц/8 (Вверх), →/d/в/6 (Вправо), ↓/s/ы/2 (Вниз) – для навигации по интерфейсу;
- 0, Space, Enter (OK) – для подтверждения действия;
- Esc (Cancel) – для закрытия списка деталей/перехода к главной детали;
- Ctrl + C (Copy), Ctrl + S (Save), Ctrl + O (Open) – для копирования в буфер, сохранения, открытия файла;
- F2, Ctrl + H, Ctrl + I (Help) – выводит сообщение со справочной информацией;
- Ctrl + Q (Quit) – закрывает программу;
- Файлы содержащие необходимую информацию для загрузки проекта (*.asdat).

Выходные данные: графическое отображение интерфейса, а также информационные сообщения (диалоговые окна).

Метод решения: для решения задачи должны быть созданы классы для следующих объектов: Window_Args – хранящий информацию необходимую для отрисовки интерфейса (движок), Window_Parts_Panel_Args – подкласс Window_Args отвечающий за отрисовку списка деталей, Part – класс детали ракеты, PartConnector – точка для соединения деталей между собой, Point2 –двухмерный вектор (используется как позиция или размер). А также перечислений: Color – 8-ми битный цвет и Part_Type – тип детали. Основными методами классов должны быть вывод графического изображения объекта и изменение его данных. В свою очередь «спрайты» деталей будут «вшиты» в соответствующие Get методы, например Get_Cockpit() представлен на рисунке 3.

```
Part* Get_Cocpit()
{
    const Point2 center = Point2(6, 3);
    const Point2 size = Point2(12, 5);
    char** sprite = new char* [5]
    {
        new char[12] {R"(  /|||||\  )"},
        new char[12] {R"( /||/ \||\ )"},
        new char[12] {R"([|°|\ /|°|])"},
        new char[12] {R"(||||<=>||||)"},
        new char[12] {R"([|||<=>|||])"},
    };
    int con_count = 4;
    PartConnector* conectors = new PartConnector[4]
    {
        PartConnector(Point2(0, -2), Point2(0, -1)),
        PartConnector(Point2(5, 1), Point2(1, 0)),
        PartConnector(Point2(0, 2), Point2(0, 1)),
        PartConnector(Point2(-5, 1), Point2(-1, 0)),
    };
```

Рисунок 3 – Пример спрайта

Здесь центр спрайта – точка относительно которой расположено всё остальное, размер спрайта – размер массива символов (включая «\0»), спрайт – представленный в виде массива символов, массив точек соединения – где каждая точка обладает своей позицией (относительно центральной точки) и направлением.

2.2 UML модель

Иерархия классов данной модели представляет собой служебный класс Point2 и перечисления Color, Part_Type. Которые используются основными классами. Главным классом можно назвать класс Window_Args, т.к. функция отрисовки (расположенная вне классов) многократно обращается к полям и методам объекта этого класса. Внутри этого класса расположены несколько объектов Point2: Size, cursor_pos, единственный экземпляр Window_Parts_Panel_Args, и массив объектов Part parts и указатели Part* selected_part, PartConnector* selected_connector.

Краткое описание классов:

- *Window_Args* представляет собой основной класс, объединяющий другие, а также содержащий необходимые для корректной отрисовки параметры;
- *Window_Parts_Panel_Args* является классом для работы с окном, содержащем список деталей, когда данное окно открыто управление переходит на список;
- *Part* деталь корабля, её содержимое было описано под рисунком 3;
- *PartConnector* содержит поля: необходимые для того чтобы можно было вычислить совместимость двух коннекторов (Connected_Position – направление), получить указатель на присоединённую деталь, вычислить новую позицию детали если её присоединить к этому коннектору;
- класс *Point2* является реализацией двухмерного вектора, обладает арифметическими операторами и двумя координатами: X, Y.

Маленькая оговорочка, для удобства были использованы структуры, при использовании ключевого слова class возникали некоторые трудности, а замена его на struct сразу отбросила их все.

Диаграмма классов (рис. 4) была сгенерирована с использованием компонента «Конструктор классов» встроенной в саму Visual Studio.

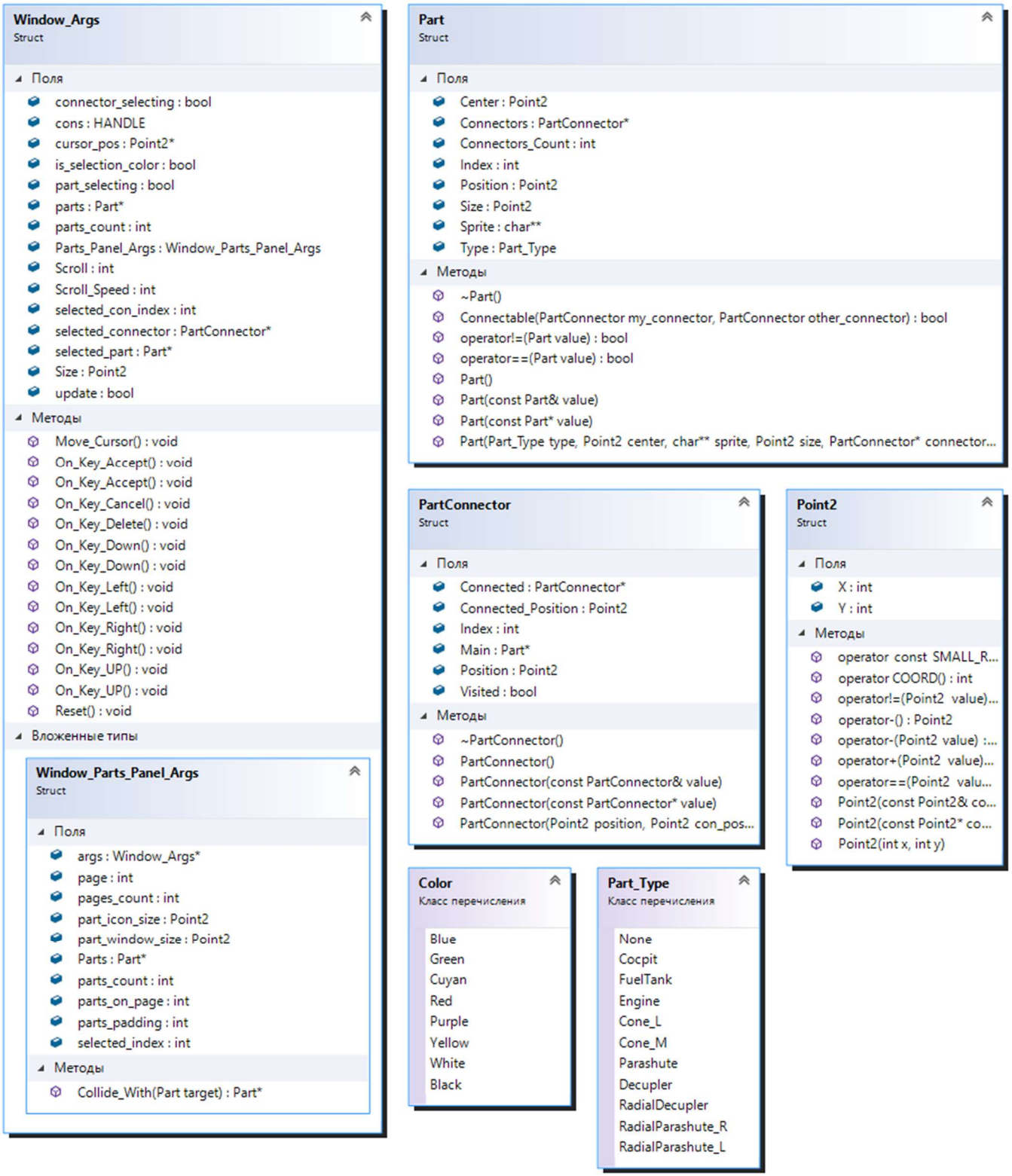

Рисунок 4 – Диаграмма классов

2.3 Алгоритм решения задачи

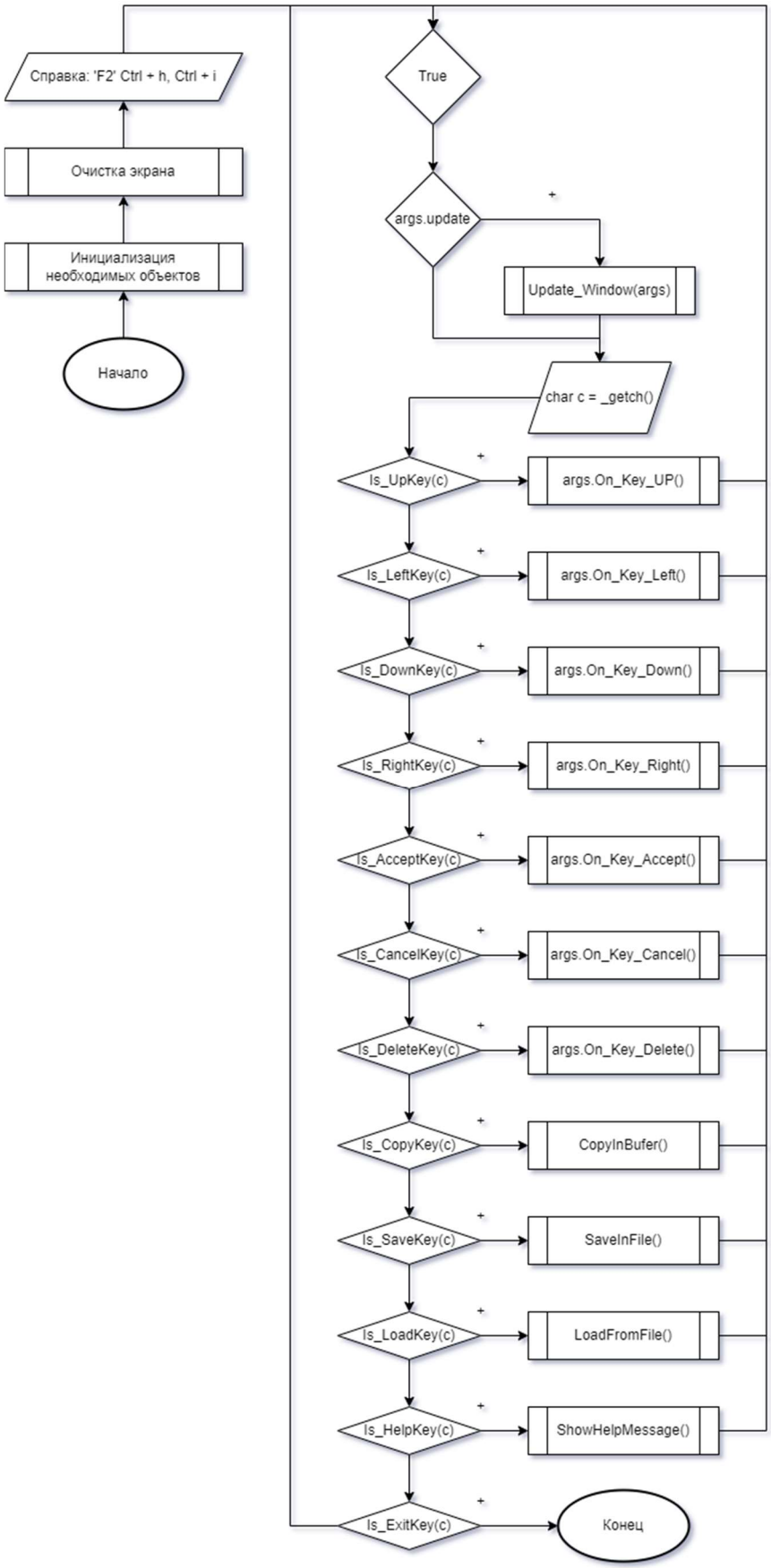

Рисунок 5 – Основной цикл программы

Данный цикл выполняется да тех пор, пока пользователь не выйдет из программы, а на нажатие каждой из горячих клавиш происходит соответствующие действие.

Так же помимо копирования, сохранения, загрузки и отображения справки, код которых прописан сразу по месту, все команды переадресовываются в экземпляр класса (структуры) Window_Args, а команда Exit завершает работу программу. Все коды символов были получены экспериментально.

2.4 Реализация управления интерфейсом «горячими» клавишами

Во-первых, интерфейс разделён на два режима: режим выбора детали и режим выбора коннектора. А во-вторых, была произведена оптимизация по сокращению обновлений экрана.

Для начала общие клавиши:

- (Copy) – копирует «изображение» текущего корабля в буфер обмена (если может);
- (Save) – вызывает диалог выбора файла и сохраняет в выбранный файл данные текущего корабля;
- (Load) – вызывает диалог выбора файла и загружает данные из выбранного файла в текущий редактор;
- (Help) – выводит справочное сообщение со списком горячих клавиш;
- (Quit) – закрывает программу.

В-третьих, результаты работы каждого режима сохраняются, и поэтому вставка детали осуществляется по месту, выбранному в предыдущем режиме. Ну и куда же без оптимизации, весь экран обновляется если:

- ✓ Добавлена новая деталь;
- ✓ Удалена деталь;
- ✓ Загружены данные из файла.

В своё время при переходах между коннекторами обновлений экрана не происходит, просто курсор меняет своё положение.

В-четвёртых, окно выбора деталей обновляется отдельно от всего экрана, опять же в угоду оптимизации, а именно если:

- ✓ изменился индекс выбранной детали (в том числе смена страницы);
- ✓ если пользователь загрузил данные из файла (обновляется весь экран);
- ✓ при открытии/закрытии окна выбора деталей (переход между режимами).

Горячие клавиши в режиме выбора детали (который встречает пользователя первым):

- (Вверх) – предыдущая деталь (если первая не обновляет экран) (если первая деталь на странице – предыдущая страница);

— (Вниз) – следующая деталь (если последняя не обновляет экран) (если последняя деталь на странице – следующая страница);
— (Влево) – предыдущая страница (если первая не обновляет страницу);
— (Вправо) – следующая страница (если последняя не обновляет страницу)
— (ОК) – размещает деталь на месте курсора (или в центре если первая);
— (Cancel) – переводит в режим выбора коннектора (если хотя бы одна деталь установлена на корабль);
— (Delete) – ничего не делает.

Горячие клавиши в режиме выбора коннектора:

— (Вверх)/(Вниз) – прокрутка корабля вверх/вниз;
— (Влево)/(Вправо) – смена коннектора против часовой/по часовой;
— (ОК) – если есть соединение – переходит «в деталь» к которой присоединён коннектор, если соединения нет, но коннекторы в «нужном» положении – соединяет их, иначе – открывает режим выбора детали и по завершению которого присоединит выбранную деталь к выбранному, заранее, коннектору или, если места не хватит или у детали нет коннектора с подходящим «направлением», выдаст сообщение с ошибкой.
— (Cancel) – переводит «курсор» в положение первого коннектора, «главной» детали;
— (Delete) – удаляет деталь по ту сторону коннектора или, если детали по ту сторону нет или же деталь имеет другие соединения – выводит сообщение об ошибке.

3 Программный раздел

3.1 Описание программы

Данная программа реализована на языке C/C++ и выполнена в среде разработки Visual Studio 2019. В ней используются библиотеки *Windows.h, cstdio, chrono, thread, stdlib.h, conio.h, stdio.h, iostream, shobjidl.h,* подключаемые в файле *Define.h*.

Так же подключаются в *header.h* перечислены все классы(структуры) и перечисления: *struct Point2*, *enum class Color*, *struct Part*, *struct PartConnector*, *struct Window_Args*, *enum class Part_Type*. Определения самих классов, находятся в соответствующих файлах с расширением «.h», а реализация некоторых методов в файлах с расширением «*.cpp»*.

Данная программа является много файловым проектом, функция *main* представлена в файле «main*.cpp»*.

3.2 Описание структуры данных

Данная программы была написана, используя объектно-ориентированный подход, с использованием классов, инкапсуляции и полиморфизма, механизмов управления памятью и сборкой мусора. А что касается «сохранения» данных то для это цели есть отдельные скрипты:

- В файл сохраняется только количество деталей; типы, позиции, с какой деталью и через какой коннектор соединён коннектор каждой детали (рис. 7);
- В буфер формируется картинка только спрайтов всех деталей, без лишних пробелов (рис. 8).

```
FILE* f = NULL;
fopen_s(&f, convertWstringToAstringWinapi(wstring(file_path)).c_str(), "w");
fwrite(&args.parts_count, sizeof(int), 1, f);
for (int i = 0; i < args.parts_count; i++)
{
    Part* p = &args.parts[i];
    fwrite(&p->Type, sizeof(Part_Type), 1, f);
    fwrite(&p->Position, sizeof(Point2), 1, f);
}
int NULL_key = -1;
for (int i = 0; i < args.parts_count; i++)
{
    Part* p = &args.parts[i];
    for (int ic = 0; ic < p->Connectors_Count; ic++)
    {
        PartConnector* c = &p->Connectors[ic];
        if (c->Connected == NULL)
            fwrite(&NULL_key, sizeof(int), 1, f);
        else
        {
            fwrite(&c->Connected->Main->Index, sizeof(int), 1, f);
            fwrite(&c->Connected->Index, sizeof(int), 1, f);
        }
    }
}
```

Рисунок 7 – Скрипт записи в файл.

```
Point2 size = Point2(
    (right_part->Position.X - right_part->Center.X + right_part->Size.X-1) - (left_part->Position.X - left_part->Center.X),
    (bottom_part->Position.Y - bottom_part->Center.Y + bottom_part->Size.Y) - (top_part->Position.Y - top_part->Center.Y)
);
int len = (size.X + 1) * size.Y;
HGLOBAL hGlob = GlobalAlloc(GMEM_FIXED, len);

char* data = new char[len];
for (int y = 0; y < size.Y; y++)
{
    for (int x = 0; x < size.X; x++)
    {
        int index = y * (size.X+1) + x;
        data[index] = ' ';
    }
    data[(y + 1) * (size.X+1)-1] = '\n';
}
data[len - 1] = '\0';

Point2 start = Point2(left_part->Position.X - left_part->Center.X, top_part->Position.Y - top_part->Center.Y);
for (int i = 0; i < args.parts_count; i++)
{
    Part* p = &args.parts[i];
    Point2 d_pos = p->Position - start - p->Center;
    for (int y = 0; y < p->Size.Y; y++)
    {
        for (int x = 0; x < p->Size.X-1; x++)
        {
            int index = (d_pos.X + x) + (size.X + 1) * (d_pos.Y + y);
            data[index] = p->Sprite[y][x];
        }
    }
}
strcpy_s((char*)hGlob, len, data);
```

Заполнение массива пробелами и \n

"Отрисовка" спрайтов всех деталей в строке

Рисунок 8 – Скрипт «отрисовки» в буфер обмена

Структура каждого из классов была показана на рисунке 4. А исходники всех их методов будут в разделе Приложение.

3.3 Описание основных функций

Ранее уже были приведены отрывки кода функций, но ниже уже даны основные функции и их описание, демонстрирующее логику и функционирование программы.

Отдельно хочется упомянуть макросы вида: «IS_<Key Title>Key(key)» (их код показан на рисунке 6). Данная группа макросов хранит условия, в которых код введённого символа сравниваются с кодами соответствующими корячим клавишам.

```
#define Is_UpKey(key_code) key_code == 72 || key_code == 119 || key_code == 56 || key_code == -26
#define Is_LeftKey(key_code) key_code == 75 || key_code == 97 || key_code == 52 || key_code == -28
#define Is_DownKey(key_code) key_code == 80 || key_code == 115 || key_code == 50 || key_code == -21
#define Is_RightKey(key_code) key_code == 77 || key_code == 100 || key_code == 54 || key_code == -94
#define Is_AcceptKey(key_code) key_code == 13 || key_code == 32 || key_code == 53
#define Is_CancelKey(key_code) key_code == 27
#define Is_DeleteKey(key_code) key_code == 83 ||  key_code == 120 ||  key_code == -25
#define Is_CopyKey(key_code) key_code == 3
#define Is_SaveKey(key_code) key_code == 19
#define Is_LoadKey(key_code) key_code == 15
#define Is_ExitKey(key_code) key_code == 17
#define Is_HelpKey(key_code) key_code == 60 || key_code == 8 ||  key_code == 9
```

Рисунок 6 – Макросы

Функция обновления экрана, вскользь уже говорилось, что данная функция лишний раз не перерисовывает весь экран. Код данной функции делится на 2 части: первая – чистит весь

экран и рисует все детали вызывая функцию отрисовки деталей, вторая – чистит и рисует только окно выбора деталей.

Функция отрисовки деталей в свою очередь в цикле рисует спрайты деталей в их позициях. Но т.к. отрисовка окна происходит поверх деталей то при открытом окне часть деталей затирается этим окном.

Так же отдельного внимания заслуживает метод, проверяющий наложение деталей. Его алгоритм сопоставляет каждый символ испытуемой детали с каждым символом других деталей на поле, и, если его не пустой символ накладывается на не пустой символ другого спрайта – данная функция возвращает деталь, с которой столкнулась деталь. (ранее это использовалось) …

```
void Update_Window(Window_Args& args);
void Draw_Parts(Window_Args args);
Part* Window_Args::Window_Parts_Panel_Args::Collide_With(Part target);
```

Остальные функции представлены в приложении листинга программы.

4 Экспериментальный раздел

4.1 Тестирование в нормальных условиях

Было проведено тестирование игры на наличие критичных ошибок и багов. И все команды в обычных условиях выполняются корректно.

Так на рисунке 7 видно, что сверху выведен текст с горячими клавишами для вызова справки, открылся режим выбора детали, и «курсор» – выделение зелёным цветом – расположен на первой детали первой страницы.

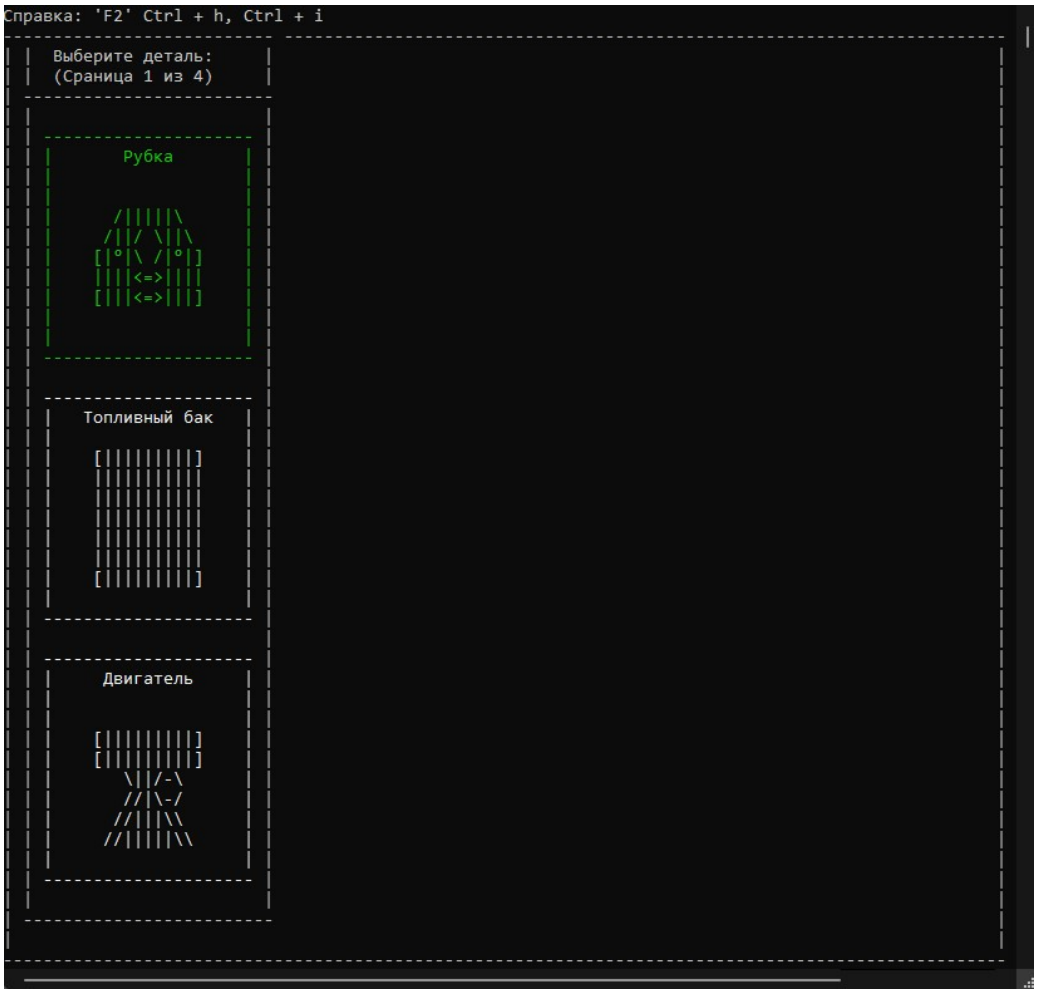

Рисунок 7 – Запуск

На рисунках 8, 9 продемонстрирован отклик интерфейса на команду (Вниз) и последующую команду (Вверх), а на рисунках 10, 11 – отклик на (Вправо), (Влево).

Рисунок 8 – (Вниз)

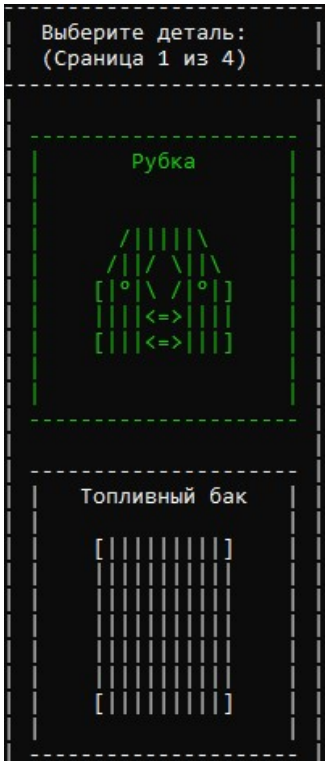

Рисунок 9 – (Вверх)

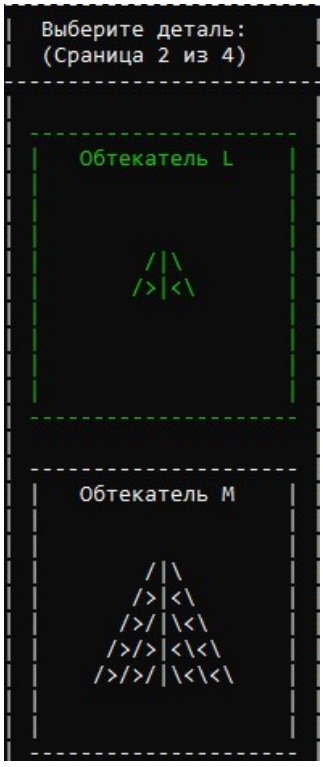

Рисунок 10 – (Вправо)

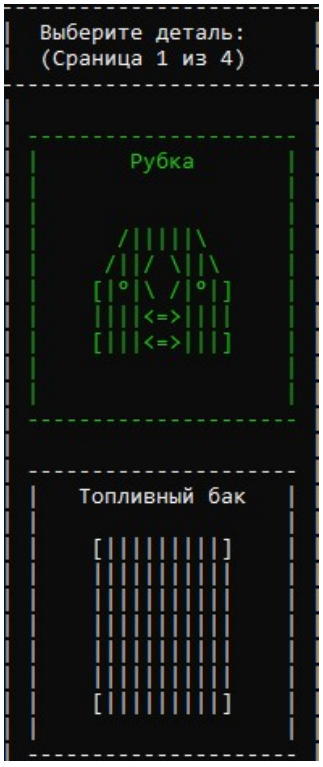

Рисунок 11 – (Влево)

После команды (ОК) программа корректно осуществила переход во второй режим, предварительно разместив первую деталь на поле и переместив курсор в позицию первого коннектора (рисунок 12). Помимо этого, скрылся и текст «Справка…». Повторным нажатием (ОК) вновь включается режим выбора детали, а уже установленная продолжает покоиться на своём месте (рисунок 13).

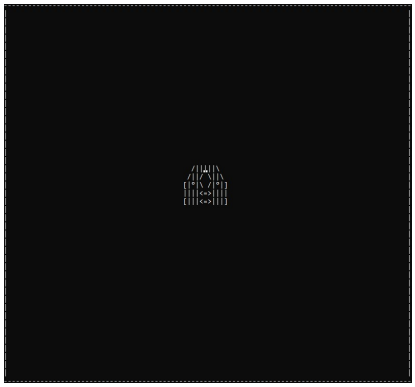

Рисунок 12 – Команда (ОК) x1

Рисунок 13 Команда (ОК) x2

На рисунке 14 «курсор» перемещён на носовой парашют, а на рисунке 15 он уже размещён на ракете.

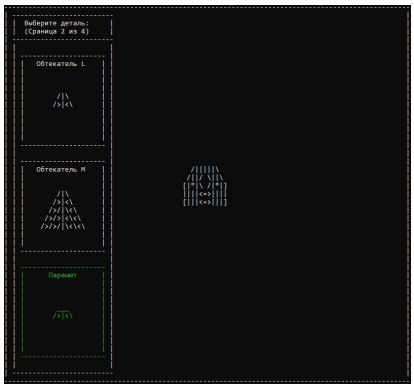

Рисунок 14 – Выбор детали

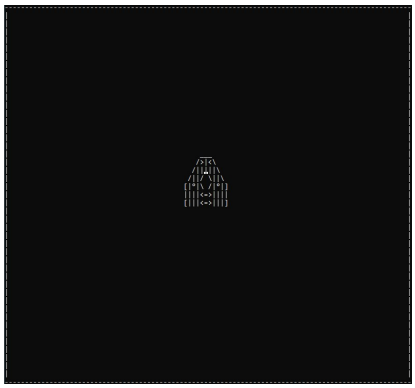

Рисунок 15 – Установка детали

Команда (Copy) сохраняет «изображение ракеты в буфер (рисунок 16), а команда (Save) вызывает диалоговое окно выбора файла (рисунок 17) и сохраняет в него данные о ракете (рисунок 18).

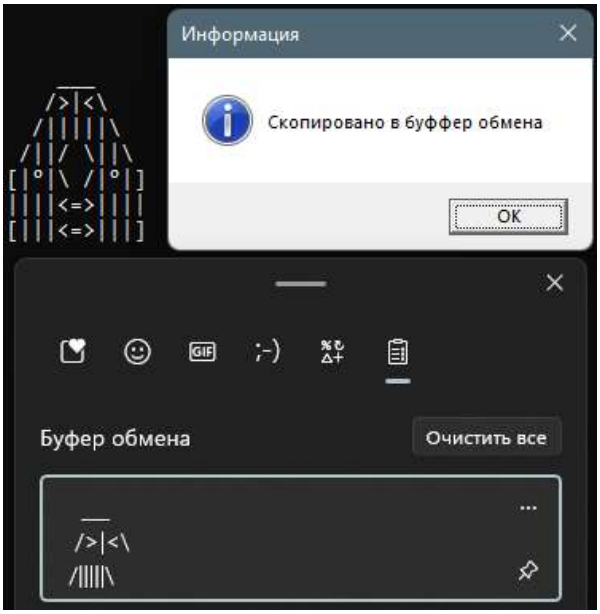

Рисунок 16 – Сохранение в буфер обмена

В качестве цели был выбран файл «test.asdat», данные которого были перезаписаны.

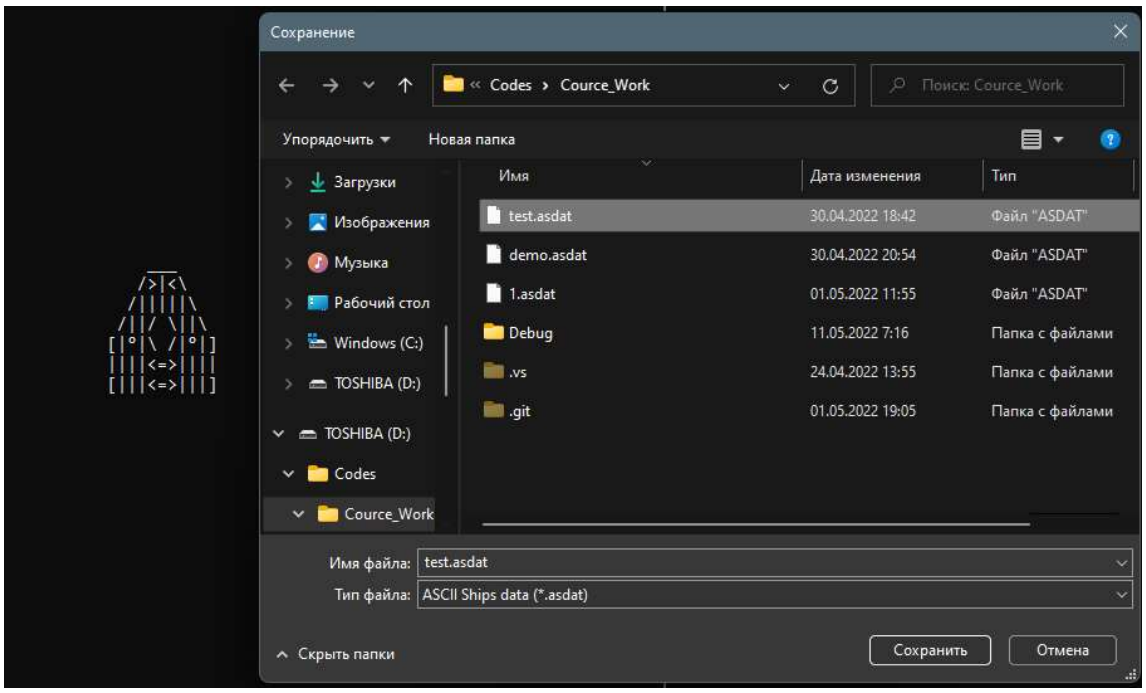

Рисунок 17 – Диалоговое окно сохранения

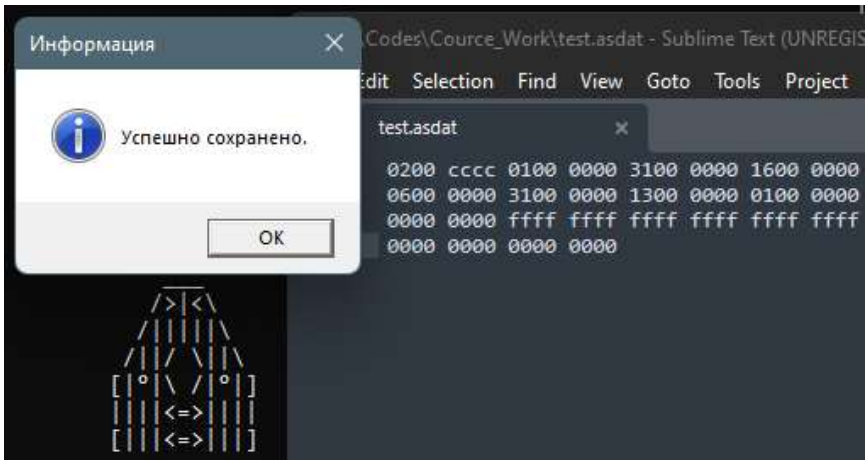

Рисунок 18 – Результат сохранения

Последняя тестируемая функция – (Delete), программа вывела сообщение о операции (рисунок 19) и после обновления экрана парашют исчез.

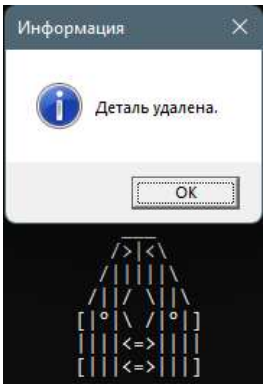

Рисунок 19 – Удаление детали

4.2 Тестирование в исключительных условиях

Предусмотрена проверка на пересечение новой детали с имеющимися. Для этого была воссоздана ситуация, когда место справа и слева от кокпита ограничено (рисунок 20). В таком случае расположить туда крупную деталь не получится (рисунок 21) в отличие от маленького парашюта (рисунок 22).

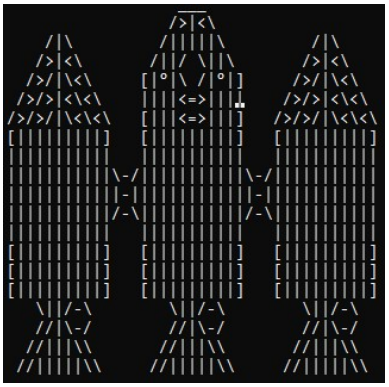

Рисунок 20 – Плотная расстановка

Так же для отладки был создан отдельный скрипт для визуализации процесса проверки коллизии: зелёным отмечены проверенные «пиксели», красным – «пиксель», который пересекается с другой деталью. При расчёте коллизии не принимают участия «пиксели» символа «пробел». Так на рисунке 21 пересечение было обнаружено при проверке коллизии с правым топливным баком.

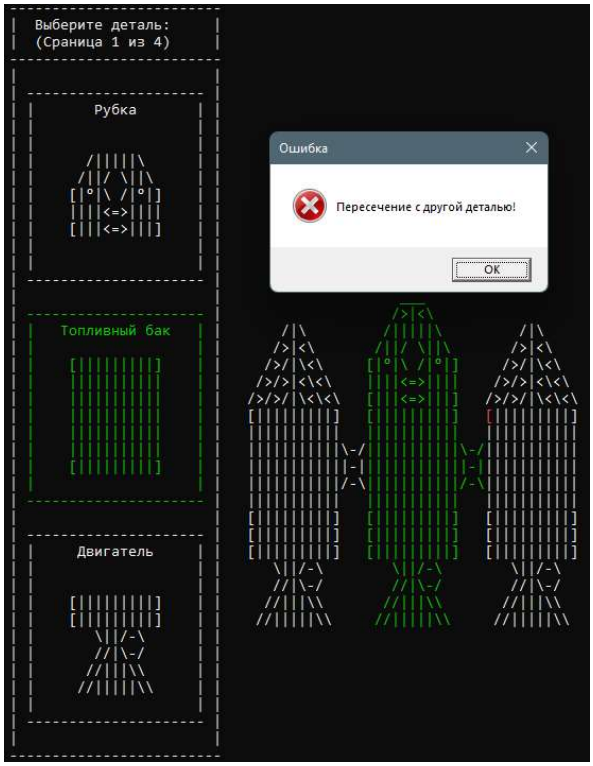

Рисунок 21 – Пересечение со спрайтом другой детали

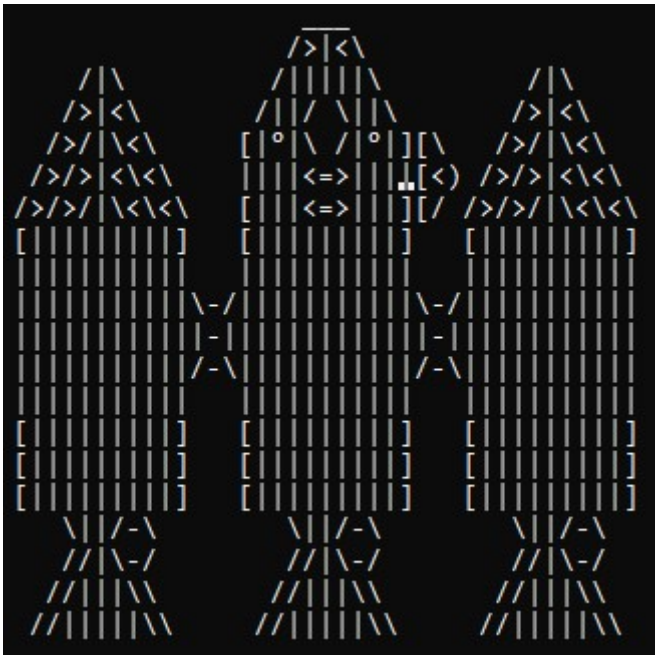

Рисунок 22 – Отсутствие коллизии с маленькой деталью

Следующее ограничение: нельзя удалить деталь, которая обладает другими соединениями (рисунок 23)

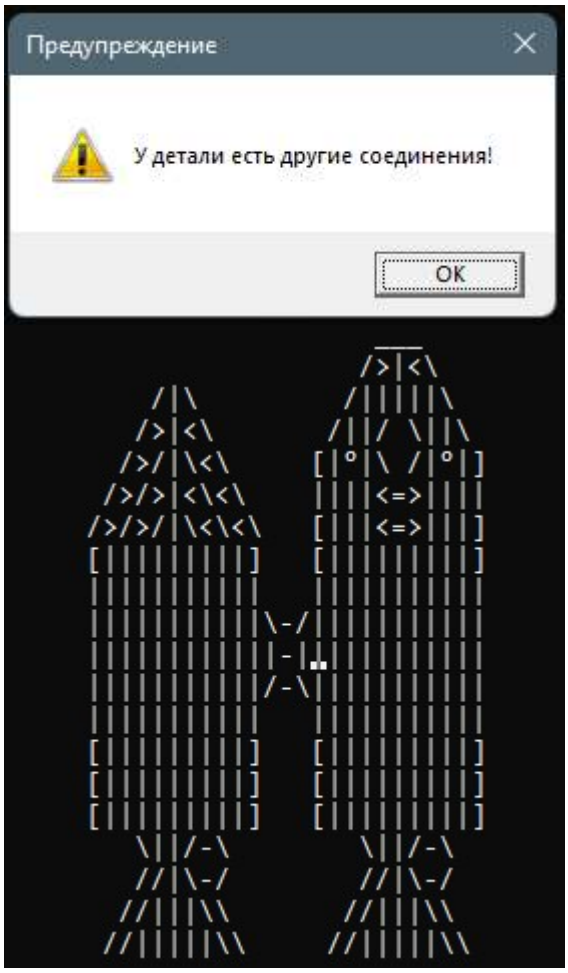

Рисунок 23 – Запрет на удаление

Так же нельзя добавить деталь, которая не обладает подходящим коннектором (рисунок 24).

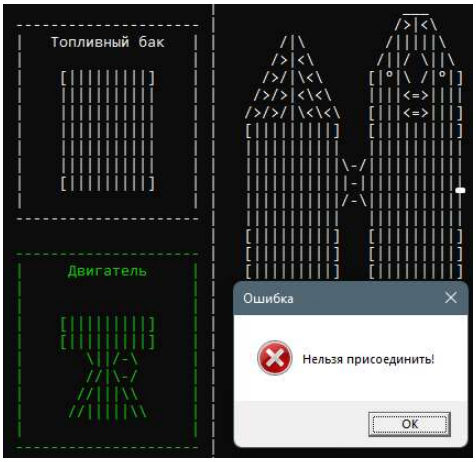

Рисунок 24 – Нет нужного коннектора

Так, например, нельзя соединить двигатель и разъединитель, а также носовые обтекатели и парашюты с боковыми деталями: левый и правый парашюты, боковой разъединитель. Ограничение реализовывается через установку направления для коннектора. Так у двигателя два коннектора (рисунок 25), а у носовых обтекателей всего один (рисунок 26).

```
char** sprite = new char* [6]
{
    new char[12] {R"([||||▯||||])"},
    new char[12] {R"([||||||||||])"},
    new char[12] {R"(   \||/-\  )"},
    new char[12] {R"(   //|\-/  )"},
    new char[12] {R"(  //|||\\  )"},
    new char[12] {R"( //||▯||\\ )"},
};
int con_count = 2;
PartConnector* conectors = new PartConnector[2]
{
    ▯PartConnector(Point2(0, -2), Point2(0, -1)),
    ▯PartConnector(Point2(0, 3), Point2(0, 1)),
```

Рисунок 25 – Расположение коннекторов на двигателе

```
const Point2 center = Point2(6, 3);
const Point2 size = Point2(12, 5);
char** sprite = new char* [5]
{
    new char[12] {R"(    /|\    )"},
    new char[12] {R"(   />|<\   )"},
    new char[12] {R"(  />/|\<\  )"},
    new char[12] {R"( />/>|<\<\ )"},
    new char[12] {R"(/>/>/▯\<\<\)"},
};
int con_count = 1;
PartConnector* conectors = new PartConnector[1]
{
    ▯PartConnector(Point2(0, 2), Point2(0, 1)),
```

Рисунок 26 – Расположение коннектора на носовом обтекателе

Как видно, программа работает корректно и сопровождает все необходимые действия соответствующими сообщениями.

4.3 Тестирование в экстремальных условиях

Первое что было проверено – ограничение отрисовки вне пределов экрана (рисунок 27).

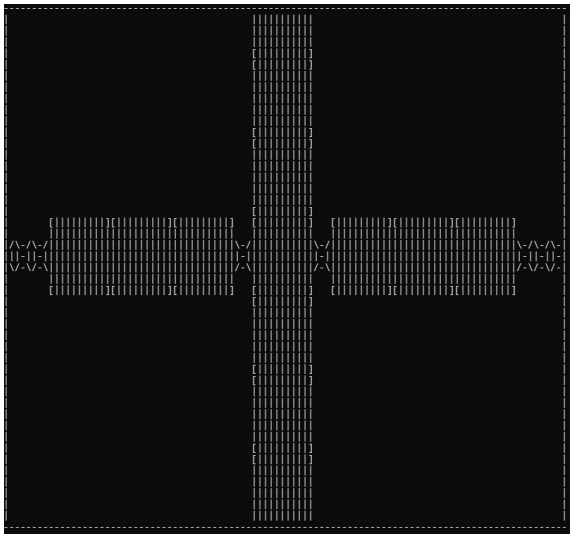

Рисунок 27 –Большая для экрана ракета обрезана по краям

Так же было протестировано создание большой ракеты (из 50 деталей). Работа программы проходила в обычном режиме без задержек и тормозов. А также данные этой ракеты успешно сохранились в файл (рисунок 28).

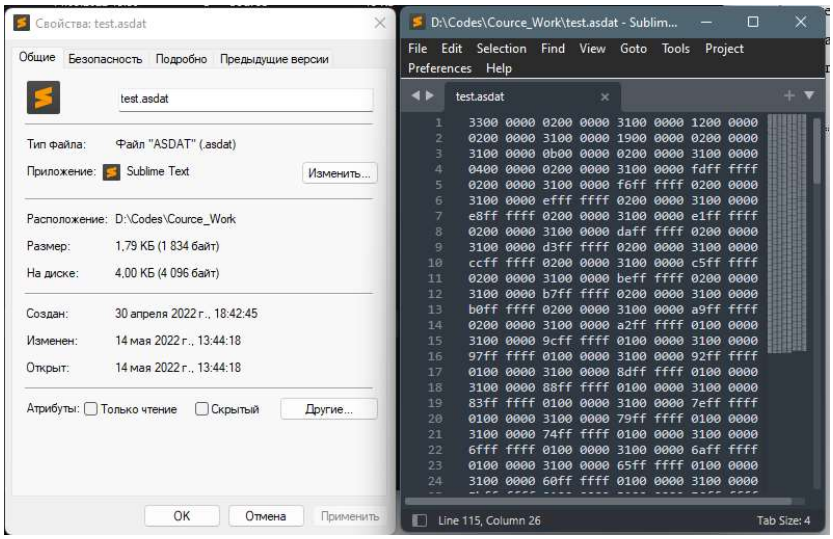

Рисунок 28 – Данные «большой» ракеты

4.4 Итоги тестирования

После проведенных тестов было выявления отсутствие критичных проблем и багов игры, о полной работоспособности всех элементов игры. Из чего можно сделать вывод об отсутствии багов и ошибок программы, то есть о полноценном рабочем редакторе ASCII ракет для семейства ОС Windows, выполненное в консольном приложении.

ЗАКЛЮЧЕНИЕ

В ходе разработки была создана удобная база и возможности для будущих версий программы. В неё с лёгкостью можно добавить другие типы деталей, стилизовать список деталей чтобы он выводил миниатюры, создать другие классы деталей, например контейнеры, добавить цветов, поработать над интерфейсом и т.д.

Так же в ходе разработки была идея сделать управление мышью, т.к. в консоли есть так называемое «выделение», можно попытаться её реализовать в будущем.

Вывод: проект доведён до финальной стадии: ключевые механики реализованы, все неисправности были устранены на этапе тестирования, что свидетельствует о том, что все поставленные цели достигнуты, а задачи выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Эккель, Б. Философия C++ [Текст]: учебник / Б. Эккель, Чак Эллисон. – М.: Питер, 2004. – 577 с.
2. Лафоре, Р. Объектно-ориентированное программирование в C++ [Текст]: учебник / Р. Лафоре. – М.: Питер , 2004. – 992 с.
3. Kerbal Space Program [Электронный ресурс]: – URL: https://www.kerbalspaceprogram.com/game/kerbal-space-program/
4. Чистый код. Создание, анализ и рефакторинг [Текст]: Книга / Мартин Роберт К: Питер 2010. – 464 с.
5. Кондитерская программиста [Электронный ресурс]: Habr.com – URL: https://habr.com/ru/post/650011/

GitHub: https://github.com/AndyTech-IT/Cource_Work

Приложение А

Define.h

```
#include <thread>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <iostream>
#include <shobjidl.h>
using namespace std;

#define DEBUG false

struct Point2;
enum class Color;
struct Part;
struct PartConnector;
struct Window_Args;
enum class Part_Type;
```

Point2.h

```
#pragma once
#include "Define.h"

struct Point2
{
	int X;
	int Y;
	Point2(int x = 0, int y = 0);
	Point2(const Point2& copy);
	Point2(const Point2* copy);
	operator COORD()
	{
		return COORD{ (short)X, (short)Y };
	}
	operator const SMALL_RECT*()
	{
		const SMALL_RECT* result = new SMALL_RECT{ 0, 0, (short)X, (short)Y };
		return result;
	}
	Point2 operator +(Point2 value);
	Point2 operator -(Point2 value);
	Point2 operator -();
	bool operator ==(Point2 value);
	bool operator !=(Point2 value);
};
```

Point2.cpp

```
#include "Point2.h"

Point2::Point2(int x, int y)
{
	X = x;
	Y = y;
}

Point2::Point2(const Point2& copy)
{
	X = copy.X;
	Y = copy.Y;
}

Point2::Point2(const Point2* copy)
{
	X = copy->X;
	Y = copy->Y;
}

Point2 Point2::operator+(Point2 value)
{
	return Point2(X + value.X, Y + value.Y);
}
```

```
Point2 Point2::operator-(Point2 value)
{
	return Point2(X - value.X, Y - value.Y);
}

Point2 Point2::operator-()
{
	return Point2(-X, -Y);
}

bool Point2::operator==(Point2 value)
{
	return value.X == X && value.Y == Y;
}

bool Point2::operator!=(Point2 value)
{
	return value.X != X && value.Y != Y;
}
```

Main.cpp

```
#include "Define.h"
#include "Point2.h"
#include "Part.h"
#include "Color.h"
#include "Window_Args.h"

HANDLE cons = GetStdHandle(STD_OUTPUT_HANDLE);
void Draw_Parts(Window_Args args)
{
	for (Part* p = &args.parts[0]; p < &args.parts[args.parts_count]; p++)
	{
		Point2 draw_pos = p->Position - p->Center + Point2(1, args.Scroll + 1);
		for (int y = 0; y < p->Size.Y; y++)
		{
			if (draw_pos.Y < 0 + 1 || draw_pos.Y > args.Size.Y - 2)
			{
				draw_pos.Y++;
				continue;
			}
			for (int x = 0; x < p->Size.X - 1; x++)
			{
				if (draw_pos.X + x < 1 || draw_pos.X + x > args.Size.X - 2)
				{
					continue;
				}
				SetConsoleCursorPosition(cons, draw_pos + Point2(x));
				cout << p->Sprite[y][x];
			}
			draw_pos.Y++;
		}
	}
}

void Update_Window(Window_Args& args)
{
	SetConsoleCursorPosition(cons, Point2());
	Point2 draw_pos;
	bool is_selection_color = false;
	if (args.connector_selecting)
	{
		if (args.parts_count == 0)
			args.part_selecting = true;
		else
		{
			for (int y = 0; y < args.Size.Y; y++)
			{
				for (int x = 0; x < args.Size.X; x++)
				{
					if (y == 0 || y == args.Size.Y - 1)
						cout << '-';
					else if (x == 0 || x == args.Size.X - 1)
						cout << '|';
					else
						cout << ' ';
				}
```

```
                        cout << "\n";
                }
                Draw_Parts(args);
        }
        args.Move_Cursor();
        return;
}

if (args.part_selecting)
{
        Window_Args::Window_Parts_Panel_Args panel_args = args.Parts_Panel_Args;
        draw_pos = Point2(panel_args.parts_padding+1, panel_args.parts_padding);

        // Clear part`s window place
        SetConsoleCursorPosition(cons, draw_pos);
        for (int y = 0; y < panel_args.part_window_size.Y; y++)
        {
                for (int x = 0; x < panel_args.part_window_size.X+1; x++)
                {
                        if (x == panel_args.part_window_size.X)
                        {
                                cout << ' ';
                                continue;
                        }
                        char c = ' ';
                        if (y == 0 || y == panel_args.part_window_size.Y - 1)
                                c = '-';
                        else if (x == 0 || x == panel_args.part_window_size.X - 1)
                                c = '|';
                        cout << c;
                }
                SetConsoleCursorPosition(cons, Point2(draw_pos.X, draw_pos.Y+y+1));
        }
        draw_pos = draw_pos + Point2(2, 1);

        // Draw panel Head
        SetConsoleCursorPosition(cons, draw_pos);
        cout << " Выберите деталь:";
        draw_pos.Y++;
        SetConsoleCursorPosition(cons, draw_pos);
        cout << " (Сраница " << panel_args.page + 1 << " из " << panel_args.pages_count << ')';
        draw_pos.Y++;
        SetConsoleCursorPosition(cons, draw_pos-Point2(2));
        for (int x = 0; x < panel_args.part_window_size.X; x++)
                cout << '-';

        // Add padding
        draw_pos.Y += 1 + panel_args.parts_padding;

        for (int i = panel_args.page * panel_args.parts_on_page; i < (panel_args.page + 1) * pan-
el_args.parts_on_page; i++)
        {
                Point2 padding = Point2(panel_args.part_icon_size.X / 2 - panel_args.Parts[i].Center.X + 1,
panel_args.part_icon_size.Y / 2 - panel_args.Parts[i].Center.Y);

                // Set selection color
                SetConsoleCursorPosition(cons, draw_pos);
                if (panel_args.selected_index == i && is_selection_color == false)
                {
                        SetConsoleTextAttribute(cons, (WORD)Color::Green);
                        is_selection_color = true;
                }

                // Draw top border
                for (int x = 0; x < panel_args.part_icon_size.X; x++)
                        cout << '-';
                draw_pos.Y++;
                string title = Get_PartTitle(panel_args.Parts[i].Type);
                SetConsoleCursorPosition(cons, draw_pos + Point2((panel_args.part_icon_size.X - ti-
tle.length())/2));
                cout << title;

                for (int y = 0; y < panel_args.part_icon_size.Y; y++, draw_pos.Y++)
                {
                        // Left border
                        SetConsoleCursorPosition(cons, draw_pos);
                        cout << '|';

                        // Right border
```

```
                                SetConsoleCursorPosition(cons, Point2(draw_pos.X + panel_args.part_icon_size.X-1,
draw_pos.Y));
                                cout << '|';

                                // Try draw part`s row
                                int row = y - padding.Y-1;
                                if (row >= 0 && row < panel_args.Parts[i].Size.Y && i < panel_args.parts_count)
                                {
                                        SetConsoleCursorPosition(cons, Point2(draw_pos.X + padding.X + pan-
el_args.parts_padding-1, draw_pos.Y));
                                        cout << panel_args.Parts[i].Sprite[row];
                                }
                        }

                        // Draw bottom border
                        SetConsoleCursorPosition(cons, draw_pos);
                        for (int x = 0; x < panel_args.part_icon_size.X; x++)
                                cout << '-';

                        // Add padding
                        draw_pos.Y += 1 + panel_args.parts_padding;

                        // UnSet selection color
                        SetConsoleCursorPosition(cons, draw_pos);
                        if (is_selection_color == true)
                        {
                                SetConsoleTextAttribute(cons, (WORD)Color::White);
                                is_selection_color = false;
                        }
                }
        }
}
string convertWstringToAstringWinapi(const  std::wstring& src)   throw (std::runtime_error)
{
        if (src.length() < 1024)
        {
                char temp[1048];
                temp[0] = '\0';

                int winapires = WideCharToMultiByte(CP_ACP, 0, src.c_str(), -1, temp, 1048, NULL, NULL);
                if (winapires == 0)
                {
                        throw std::runtime_error("Error call WideCharToMultiByte");
                }


                return std::string(temp);
        }
        return "";
}
PWSTR SaveFile_Dialoge()
{
        PWSTR pszFilePath = PWSTR();
        bool success = false;
        COMDLG_FILTERSPEC FileTypes[] = {
                { L"ASCII Ships data", L"*.asdat" },
                { L"All files", L"*.*" }
        };
        HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED |
                COINIT_DISABLE_OLE1DDE);
        if (SUCCEEDED(hr))
        {
                IFileSaveDialog* pFileOpen;
                hr = CoCreateInstance(CLSID_FileSaveDialog, NULL, CLSCTX_ALL,
                        IID_IFileSaveDialog, reinterpret_cast<void**>(&pFileOpen));

                if (SUCCEEDED(hr))
                {
                        pFileOpen->SetFileTypes(_countof(FileTypes), FileTypes);
                        hr = pFileOpen->Show(NULL);
                        if (SUCCEEDED(hr))
                        {
                                IShellItem* pItem;
                                hr = pFileOpen->GetResult(&pItem);
                                if (SUCCEEDED(hr))
                                {
                                        hr = pItem->GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                                        if (SUCCEEDED(hr))
                                        {
```

```
                                        success = true;
                                        CoTaskMemFree(pszFilePath);
                                }
                                pItem->Release();
                        }
                }
                pFileOpen->Release();
        }
        CoUninitialize();
    }
    return success ? pszFilePath : NULL;
}
PWSTR OpenFile_Dialoge()
{
    PWSTR pszFilePath = PWSTR();
    bool success = false;
    COMDLG_FILTERSPEC FileTypes[] = {
        { L"ASCII Ships data", L"*.asdat" },
        { L"All files", L"*.*" }
    };
    HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED | COINIT_DISABLE_OLE1DDE);
    if (SUCCEEDED(hr))
    {
        IFileOpenDialog* pFileOpen;
        hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_ALL,
            IID_IFileOpenDialog, reinterpret_cast<void**>(&pFileOpen));
        if (SUCCEEDED(hr))
        {
            pFileOpen->SetFileTypes(_countof(FileTypes), FileTypes);
            hr = pFileOpen->Show(NULL);
            if (SUCCEEDED(hr))
            {
                IShellItem* pItem;
                hr = pFileOpen->GetResult(&pItem);
                if (SUCCEEDED(hr))
                {
                    hr = pItem->GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                    if (SUCCEEDED(hr))
                    {
                        success = true;
                        CoTaskMemFree(pszFilePath);
                    }
                    pItem->Release();
                }
            }
            pFileOpen->Release();
        }
        CoUninitialize();
    }
    return success ? pszFilePath : NULL;
}

#define Is_UpKey(key_code) key_code == 72 || key_code == 119 || key_code == 56 || key_code == -26
#define Is_LeftKey(key_code) key_code == 75 || key_code == 97 || key_code == 52 || key_code == -28
#define Is_DownKey(key_code) key_code == 80 || key_code == 115 || key_code == 50 || key_code == -21
#define Is_RightKey(key_code) key_code == 77 || key_code == 100 || key_code == 54 || key_code == -94
#define Is_AcceptKey(key_code) key_code == 13 || key_code == 32 || key_code == 53
#define Is_CancelKey(key_code) key_code == 27
#define Is_DeleteKey(key_code) key_code == 83 ||  key_code == 120 ||  key_code == -25
#define Is_CopyKey(key_code) key_code == 3
#define Is_SaveKey(key_code) key_code == 19
#define Is_LoadKey(key_code) key_code == 15
#define Is_ExitKey(key_code) key_code == 17
#define Is_HelpKey(key_code) key_code == 60 || key_code == 8 ||  key_code == 9

int main()
{
    setlocale(LC_ALL, "rus");
    Window_Args args{ cons };
    Point2 debug_point = Point2(0, args.Size.Y + 1);
    PWSTR file_path = PWSTR();

    cout << "Справка: 'F2' Ctrl + h, Ctrl + i" << endl;
    for (int y = 0; y < args.Size.Y; y++)
    {
        for (int x = 0; x < args.Size.X; x++)
        {
            if (y == 0 || y == args.Size.Y - 1)
                cout << '-';
```

```
                        else if (x == 0 || x == args.Size.X - 1)
                                cout << '|';
                        else
                                cout << ' ';
                }
                cout << "\n";
        }

        while (true)
        {
                if (args.update)
                {
                        Update_Window(args);
                        SetConsoleWindowInfo(cons, TRUE, args.Size);
                }
                char c = _getch();
                if (c == -32)
                        c = _getch();

                if (Is_UpKey(c))
                {
                        args.On_Key_UP();
                }
                else if (Is_LeftKey(c))
                {
                        args.On_Key_Left();
                }
                else if (Is_DownKey(c))
                {
                        args.On_Key_Down();
                }
                else if (Is_RightKey(c))
                {
                        args.On_Key_Right();
                }
                else if (Is_AcceptKey(c))
                {
                        args.On_Key_Accept();
                }
                else if (Is_CancelKey(c))
                {
                        args.On_Key_Cancel();
                }
                else if (Is_DeleteKey(c))
                {
                        args.On_Key_Delete();
                }
                else if (Is_CopyKey(c))
                {
                        if (args.parts_count == 0)
                                MessageBoxW(NULL, L"Нечего сохранять", L"Предупреждение", MB_ICONEXCLAMATION);
                        else if (OpenClipboard(NULL))
                        {
                                // Remove the current Clipboard contents
                                if (EmptyClipboard())
                                {
                                        // Get the currently selected data
                                        Part* top_part = NULL;
                                        Part* left_part = NULL;
                                        Part* right_part = NULL;
                                        Part* bottom_part = NULL;
                                        for (int i = 0; i < args.parts_count; i++)
                                        {
                                                Part* p = &args.parts[i];
                                                if (top_part == NULL || top_part->Position.Y + top_part->Center.Y >
p->Position.Y + p->Center.Y)
                                                        top_part = p;
                                                if (bottom_part == NULL || bottom_part->Position.Y + bottom_part-
>Center.Y - bottom_part->Size.Y < p->Position.Y + p->Center.Y - p->Size.Y)
                                                        bottom_part = p;

                                                if (left_part == NULL || left_part->Position.X - left_part->Center.X
> p->Position.X - p->Center.X)
                                                        left_part = p;
                                                if (right_part == NULL || right_part->Position.X + right_part-
>Center.X < p->Position.X + p->Center.X)
                                                        right_part = p;
                                        }
```

```
                                        Point2 size = Point2(
                                                (right_part->Position.X - right_part->Center.X + right_part->Size.X-
1) - (left_part->Position.X - left_part->Center.X),
                                                (bottom_part->Position.Y - bottom_part->Center.Y + bottom_part-
>Size.Y) - (top_part->Position.Y - top_part->Center.Y)
                                        );
                                        int len = (size.X + 1) * size.Y;
                                        HGLOBAL hGlob = GlobalAlloc(GMEM_FIXED, len);

                                        char* data = new char[len];
                                        for (int y = 0; y < size.Y; y++)
                                        {
                                                for (int x = 0; x < size.X; x++)
                                                {
                                                        int index = y * (size.X+1) + x;
                                                        data[index] = ' ';
                                                }
                                                data[(y + 1) * (size.X+1)-1] = '\n';
                                        }
                                        data[len - 1] = '\0';

                                        Point2 start = Point2(left_part->Position.X - left_part->Center.X, top_part-
>Position.Y - top_part->Center.Y);
                                        for (int i = 0; i < args.parts_count; i++)
                                        {
                                                Part* p = &args.parts[i];
                                                Point2 d_pos = p->Position - start - p->Center;
                                                for (int y = 0; y < p->Size.Y; y++)
                                                {
                                                        for (int x = 0; x < p->Size.X-1; x++)
                                                        {
                                                                int index = (d_pos.X + x) + (size.X + 1) * (d_pos.Y
+ y);

                                                                data[index] = p->Sprite[y][x];
                                                        }
                                                }
                                        }
                                        strcpy_s((char*)hGlob, len, data);
                                        // For the appropriate data formats...
                                        if (SetClipboardData(CF_TEXT, hGlob) == NULL)
                                        {
                                                CloseClipboard();
                                                GlobalFree(hGlob);
                                        }
                                        else
                                        {
                                                CloseClipboard();
                                                MessageBoxW(NULL, L"Скопировано в буффер обмена", L"Информация",
MB_ICONINFORMATION);
                                        }
                                }
                        }

                }
                else if (Is_SaveKey(c))
                {
                        if (args.parts_count == 0)
                        {
                                MessageBoxW(NULL, L"Сохронять нечего!", L"Предупреждение", MB_ICONEXCLAMATION);
                        }
                        else
                        {
                                file_path = SaveFile_Dialoge();
                                if (file_path == NULL)
                                        MessageBoxW(NULL, L"Вы не выбрали файл!", L"Предупреждение",
MB_ICONEXCLAMATION);
                                else
                                {
                                        FILE* f = NULL;
                                        fopen_s(&f, convertWstringToAstringWinapi(wstring(file_path)).c_str(), "w");
                                        fwrite(&args.parts_count, sizeof(int), 1, f);
                                        for (int i = 0; i < args.parts_count; i++)
                                        {
                                                Part* p = &args.parts[i];
                                                fwrite(&p->Type, sizeof(Part_Type), 1, f);
                                                fwrite(&p->Position, sizeof(Point2), 1, f);
                                        }
                                        int NULL_key = -1;
                                        for (int i = 0; i < args.parts_count; i++)
```

```
                                        {
                                                Part* p = &args.parts[i];
                                                for (int ic = 0; ic < p->Connectors_Count; ic++)
                                                {
                                                        PartConnector* c = &p->Connectors[ic];
                                                        if (c->Connected == NULL)
                                                                fwrite(&NULL_key, sizeof(int), 1, f);
                                                        else
                                                        {
                                                                fwrite(&c->Connected->Main->Index, sizeof(int), 1,
f);
                                                                fwrite(&c->Connected->Index, sizeof(int), 1, f);
                                                        }
                                                }
                                        }
                                        fclose(f);
                                        MessageBoxW(NULL, L"Успешно сохранено.", L"Информация", MB_ICONINFORMATION);
                                }
                        }
                }
                else if (Is_LoadKey(c))
                {
                        file_path = OpenFile_Dialoge();
                        if (file_path == NULL)
                                MessageBoxW(NULL, L"Вы не выбрали файл!", L"Предупреждение", MB_ICONEXCLAMATION);
                        else
                        {
                                FILE* f = NULL;
                                fopen_s(&f, convertWstringToAstringWinapi(wstring(file_path)).c_str(), "r");
                                fread(&args.parts_count, sizeof(int), 1, f);
                                args.parts = new Part[args.parts_count];
                                for (int i = 0; i < args.parts_count; i++)
                                {
                                        Part_Type t;
                                        fread(&t, sizeof(Part_Type), 1, f);
                                        Part* p = Get_Part(t);
                                        fread(&p->Position, sizeof(Point2), 1, f);
                                        p->Index = i;
                                        args.parts[i] = *p;
                                }
                                for (int i = 0; i < args.parts_count; i++)
                                {
                                        for (int ic = 0; ic < args.parts[i].Connectors_Count; ic++)
                                        {
                                                PartConnector* c = &args.parts[i].Connectors[ic];
                                                int index;
                                                fread(&index, sizeof(int), 1, f);
                                                if (index != -1)
                                                {
                                                        int pci;
                                                        fread(&pci, sizeof(int), 1, f);
                                                        c->Connected = &args.parts[index].Connectors[pci];
                                                }
                                        }
                                }
                                args.connector_selecting = true;
                                args.part_selecting = false;
                                args.Reset();
                                fclose(f);
                                MessageBoxW(NULL, L"Загрузка данных прошла успешно.", L"Информация",
MB_ICONINFORMATION);
                        }
                }
                else if (Is_HelpKey(c))
                        MessageBoxW(NULL, L"Добро пожаловать в редактор.\nУправление:\n WASD, Numpad, стрелочки -
навигация\n Space, Enter, Num0 - Выполнить действие\n Delete - Удалить деталь с корабля\n Esc - отменить/на главную
деталь\n Ctrl + c - Скопировать картинку в буфер обмена\n Ctrl + s - Сохранить данные в файл\n Ctrl + o - Загрузить
данные из файла", L"Справочная Информация", MB_ICONQUESTION);
                else if (Is_ExitKey(c))
                        return 0;
        }
}
```

Приложение В

Part.h

```
#pragma once
#include "Define.h"

#include "Point2.h"
#include "PartConnector.h"

enum class Part_Type
{
	None,
	Cocpit,
	FuelTank,
	Engine,
	Cone_L,
	Cone_M,
	Parashute,
	Decupler,
	RadialDecupler,
	RadialParashute_R,
	RadialParashute_L
};
struct Part
{
	Part_Type Type = Part_Type::None;
	Point2 Position = Point2();
	Point2 Size = Point2();
	Point2 Center = Point2();
	int Index = 0;

	PartConnector* Connectors = NULL;
	int Connectors_Count = 0;

	char** Sprite = NULL;

	Part();
	Part(const Part& value);
	Part(const Part* value);
	Part(Part_Type type, Point2 center, char** sprite, Point2 size, PartConnector* connectors = NULL, int con-
nectors_count = 0);
	bool Connectable(PartConnector my_connector, PartConnector other_connector);
	bool operator ==(Part value)
	{
		return value.Position == this->Position && value.Size == this->Size;
	}
	bool operator !=(Part value)
	{
		return (*this == value) == false;
	}
	~Part();
};

Part* Get_Cocpit();
Part* Get_FuelTank();
Part* Get_Engine();
Part* Get_Cone_L();
Part* Get_Cone_M();
Part* Get_Parashute();
Part* Get_Decupler();
Part* Get_RadialDecupler();
Part* Get_RadialParashute_R();
Part* Get_RadialParashute_L();

Part* Get_Part(Part_Type type);
Part* Get_Parts(int count);
string Get_PartTitle(Part_Type type);
```

PartConnector.h

```
#pragma once
#include "Define.h"

#include "Point2.h"
#include "Part.h"
```

```
struct PartConnector
{
	Part* Main = NULL;
	bool Visited = false;
	PartConnector* Connected = NULL;
	Point2 Position = Point2();
	Point2 Connected_Position = Point2();
	int Index = 0;

	PartConnector();
	PartConnector(const PartConnector& value);
	PartConnector(const PartConnector* value);
	PartConnector(Point2 position, Point2 con_position);
	~PartConnector();
};
```

Part.cpp

```
#include "Part.h"

Part* Get_Cocpit()
{
	const Point2 center = Point2(6, 3);
	const Point2 size = Point2(12, 5);
	char** sprite = new char* [5]
	{
		new char[12] {R"(  /|||||\  )"},
		new char[12] {R"( /||/ \||\ )"},
		new char[12] {R"([|°|\ /|°|])"},
		new char[12] {R"(||||<=>||||)"},
		new char[12] {R"([|||<=>|||])"},
	};
	int con_count = 4;
	PartConnector* conectors = new PartConnector[4]
	{
		PartConnector(Point2(0, -2), Point2(0, -1)),
		PartConnector(Point2(5, 1), Point2(1, 0)),
		PartConnector(Point2(0, 2), Point2(0, 1)),
		PartConnector(Point2(-5, 1), Point2(-1, 0)),
	};

	return new Part(Part_Type::Cocpit, center, sprite, size, conectors, con_count);
}

Part* Get_FuelTank()
{
	const Point2 center = Point2(6, 4);
	const Point2 size = Point2(12, 7);
	char** sprite = new char* [7]
	{
		new char[12] {R"([||||||||||])"},
		new char[12] {R"(||||||||||)"},
		new char[12] {R"(||||||||||)"},
		new char[12] {R"(||||||||||)"},
		new char[12] {R"(||||||||||)"},
		new char[12] {R"(||||||||||)"},
		new char[12] {R"([||||||||||])"},
	};
	int con_count = 4;
	PartConnector* conectors = new PartConnector[4]
	{
		PartConnector(Point2(0, -3), Point2(0, -1)),
		PartConnector(Point2(5, 0), Point2(1, 0)),
		PartConnector(Point2(0, 3), Point2(0, 1)),
		PartConnector(Point2(-5, 0), Point2(-1, 0)),
	};

	return new Part(Part_Type::FuelTank, center, sprite, size, conectors, con_count);
}

Part* Get_Engine()
{
	const Point2 center = Point2(6, 3);
	const Point2 size = Point2(12, 6);
	char** sprite = new char* [6]
	{
		new char[12] {R"([||||||||||])"},
```

```
                new char[12] {R"([||||||||||])"},
                new char[12] {R"(   \||/-\  )"},
                new char[12] {R"(   //|\-/  )"},
                new char[12] {R"(  //|||\\  )"},
                new char[12] {R"( //||||||\\ )"},
        };
        int con_count = 2;
        PartConnector* conectors = new PartConnector[2]
        {
                PartConnector(Point2(0, -2), Point2(0, -1)),
                PartConnector(Point2(0, 3), Point2(0, 1)),
        };

        return new Part(Part_Type::Engine, center, sprite, size, conectors, con_count);
}

Part* Get_Decupler()
{
        const Point2 center = Point2(6, 2);
        const Point2 size = Point2(12, 2);
        char** sprite = new char* [2]
        {
                new char[12] {R"(/<<<<=>>>>\)"},
                new char[12] {R"(\<<<<=>>>>/)"},
        };
        int con_count = 2;
        PartConnector* conectors = new PartConnector[2]
        {
                PartConnector(Point2(0, -1), Point2(0, -1)),
                PartConnector(Point2(0, 0), Point2(0, 1)),
        };

        return new Part(Part_Type::Decupler, center, sprite, size, conectors, con_count);
}

Part* Get_Parashute()
{
        const Point2 center = Point2(3, 2);
        const Point2 size = Point2(6, 2);
        char** sprite = new char* [2]
        {
                new char[6] {R"( ___ )"},
                new char[6] {R"(/>|<\)"},
        };
        int con_count = 1;
        PartConnector* conectors = new PartConnector[1]
        {
                PartConnector(Point2(0, 0), Point2(0, 1)),
        };

        return new Part(Part_Type::Parashute, center, sprite, size, conectors, con_count);
}
Part* Get_Cone_L()
{
        const Point2 center = Point2(3, 2);
        const Point2 size = Point2(6, 2);
        char** sprite = new char* [2]
        {
                new char[6] {R"( /|\ )"},
                new char[6] {R"(/>|<\)"},
        };
        int con_count = 1;
        PartConnector* conectors = new PartConnector[1]
        {
                PartConnector(Point2(0, 0), Point2(0, 1)),
        };

        return new Part(Part_Type::Cone_L, center, sprite, size, conectors, con_count);
}

Part* Get_Cone_M()
{
        const Point2 center = Point2(6, 3);
        const Point2 size = Point2(12, 5);
        char** sprite = new char* [5]
        {
                new char[12] {R"(    /|\    )"},
                new char[12] {R"(   />|<\   )"},
                new char[12] {R"(  />/|\<\  )"},
```

```
                new char[12] {R"( />/>|<\<\ )"},
                new char[12] {R"(/>/>/|\<\<\)"},
        };
        int con_count = 1;
        PartConnector* conectors = new PartConnector[1]
        {
                PartConnector(Point2(0, 2), Point2(0, 1)),
        };

        return new Part(Part_Type::Cone_M, center, sprite, size, conectors, con_count);
}

Part* Get_RadialDecupler()
{
        const Point2 center = Point2(2, 2);
        const Point2 size = Point2(4, 3);
        char** sprite = new char* [3]
        {
                new char[4] {R"(\-/)"},
                new char[4] {R"(|-|)"},
                new char[4] {R"(/-\)"},
        };
        int con_count = 2;
        PartConnector* conectors = new PartConnector[2]
        {
                PartConnector(Point2(-1, 0), Point2(-1, 0)),
                PartConnector(Point2(1, 0), Point2(1, 0)),
        };

        return new Part(Part_Type::RadialDecupler, center, sprite, size, conectors, con_count);
}

Part* Get_RadialParashute_R()
{
        const Point2 center = Point2(2, 2);
        const Point2 size = Point2(4, 3);
        char** sprite = new char* [3]
        {
                new char[4] {R"([\ )"},
                new char[4] {R"([<))"},
                new char[4] {R"([/ )"},
        };
        int con_count = 1;
        PartConnector* conectors = new PartConnector[1]
        {
                PartConnector(Point2(-1, 0), Point2(-1, 0)),
        };

        return new Part(Part_Type::RadialParashute_R, center, sprite, size, conectors, con_count);
}

Part* Get_RadialParashute_L()
{
        const Point2 center = Point2(2, 2);
        const Point2 size = Point2(4, 3);
        char** sprite = new char* [3]
        {
                new char[4] {R"( /])"},
                new char[4] {R"((>])"},
                new char[4] {R"( \])"},
        };
        int con_count = 1;
        PartConnector* conectors = new PartConnector[1]
        {
                PartConnector(Point2(1, 0), Point2(1, 0)),
        };

        return new Part(Part_Type::RadialParashute_L, center, sprite, size, conectors, con_count);
}

Part* Get_Part(Part_Type type)
{
        switch (type)
        {
        case Part_Type::Cocpit:
                return Get_Cocpit();
        case Part_Type::FuelTank:
                return Get_FuelTank();
        case Part_Type::Engine:
```

```
                return Get_Engine();
        case Part_Type::Cone_L:
                return Get_Cone_L();
        case Part_Type::Cone_M:
                return Get_Cone_M();
        case Part_Type::Parashute:
                return Get_Parashute();
        case Part_Type::Decupler:
                return Get_Decupler();
        case Part_Type::RadialDecupler:
                return Get_RadialDecupler();
        case Part_Type::RadialParashute_R:
                return Get_RadialParashute_R();
        case Part_Type::RadialParashute_L:
                return Get_RadialParashute_L();
        default:
                return NULL;
        }
}

string Get_PartTitle(Part_Type type)
{
        switch (type)
        {
        case Part_Type::Cocpit:
                return "Рубка";
        case Part_Type::FuelTank:
                return "Топливный бак";
        case Part_Type::Engine:
                return "Двигатель";
        case Part_Type::Cone_L:
                return "Обтекатель L";
        case Part_Type::Cone_M:
                return "Обтекатель M";
        case Part_Type::Parashute:
                return "Парашют";
        case Part_Type::Decupler:
                return "Разъединитель";
        case Part_Type::RadialDecupler:
                return "Разъединитель";
        case Part_Type::RadialParashute_R:
                return "Парашют R";
        case Part_Type::RadialParashute_L:
                return "Парашют L";
        default:
                return "<blank>";
        }
}

Part* Get_Parts(int count)
{
        Part* parts = new Part[count];
        for (int i = 0; i < count; i++)
        {
                parts[i] = *Get_Part((Part_Type)(i+1));
        }
        return parts;
}


Part::Part()
{
}

Part::Part(const Part& value)
{
        this->Center = value.Center;
        this->Connectors_Count = value.Connectors_Count;
        Connectors = new PartConnector[Connectors_Count];
        for (int i = 0; i < Connectors_Count; i++)
        {
                Connectors[i] = value.Connectors[i];
                Connectors[i].Main = this;
                Connectors[i].Index = i;
        }
        this->Connectors_Count = value.Connectors_Count;
        this->Type = value.Type;
        this->Position = value.Position;
        this->Size = value.Size;
```

```
	this->Sprite = new char* [Size.Y];
	for (int y = 0; y < Size.Y; y++)
	{
		Sprite[y] = new char[Size.X];
		for (int x = 0; x < Size.X; x++)
			Sprite[y][x] = value.Sprite[y][x];
	}
	this->Index = value.Index;
}

Part::Part(const Part* value)
{
	this->Center = value->Center;
	this->Connectors_Count = value->Connectors_Count;
	Connectors = new PartConnector[Connectors_Count];
	for (int i = 0; i < Connectors_Count; i++)
	{
		Connectors[i] = value->Connectors[i];
		Connectors[i].Main = this;
		Connectors[i].Index = i;
	}
	this->Position = value->Position;
	this->Size = value->Size;
	this->Sprite = new char* [Size.Y];
	for (int y = 0; y < Size.Y; y++)
	{
		Sprite[y] = new char[Size.X];
		for (int x = 0; x < Size.X; x++)
			Sprite[y][x] = value->Sprite[y][x];
	}
	this->Type = value->Type;
	this->Index = value->Index;
}

Part::Part(Part_Type type, Point2 center, char** sprite, Point2 size, PartConnector* connectors, int connect-
ors_count)
{
	Type = type;
	Size = size;
	this->Sprite = new char* [Size.Y];
	for (int y = 0; y < Size.Y; y++)
	{
		Sprite[y] = new char[Size.X];
		for (int x = 0; x < Size.X; x++)
			Sprite[y][x] = sprite[y][x];
	}
	Center = center;
	Connectors = new PartConnector[connectors_count];
	for (int i = 0; i < connectors_count; i++)
	{
		Connectors[i] = connectors[i];
		Connectors[i].Main = this;
		Connectors[i].Index = i;
	}
	Connectors_Count = connectors_count;
}

bool Part::Connectable(PartConnector my_connector, PartConnector other_connector)
{
	return my_connector.Connected_Position == -other_connector.Connected_Position;
}



Part::~Part()
{
	for (int i = 0; i < Size.Y; i++)
	{
		delete[] Sprite[i];
	}
	delete[] Sprite;

	delete[] Connectors;
}
```

PartConnector.cpp

```
#include "PartConnector.h"
```

```
PartConnector::PartConnector()
{
}

PartConnector::PartConnector(const PartConnector& value)
{
	this->Connected = value.Connected;
	this->Connected_Position = value.Connected_Position;
	this->Main = value.Main;
	this->Position = value.Position;
	this->Index = value.Index;
}

PartConnector::PartConnector(const PartConnector* value)
{
	this->Connected = value->Connected;
	this->Connected_Position = value->Connected_Position;
	this->Main = value->Main;
	this->Position = value->Position;
	this->Index = value->Index;
}

PartConnector::PartConnector(Point2 position, Point2 con_position)
{
	Position = position;
	Connected_Position = con_position;
}

PartConnector::~PartConnector()
{
	if (Connected)
	{
		Connected->Connected = NULL;
	}
}
```

Приложение С

Window_Args.h

```
#pragma once
#include "Define.h"
#include "Point2.h"
#include "Part.h"


struct Window_Args
{
	HANDLE& cons;
	struct Window_Parts_Panel_Args
	{
		Window_Args* args;
		Point2 part_window_size;
		int parts_count = 10;
		int parts_padding = 1;
		int parts_on_page = 3;
		int pages_count = parts_count / parts_on_page + (parts_count % parts_on_page != 0 ? 1 : 0);
		int page = 0;
		int selected_index = 0;
		Point2 part_icon_size = Point2(19 + 2, 10);
		Part* Parts = Get_Parts(10);

		Part* Collide_With(Part target)
		{
			for (Part* p = &args->parts[0]; p < &args->parts[args->parts_count]; p++)
			{
				Point2 d_pos = target.Position - target.Center - p->Position + p->Center;
				for (int y = 0; y < p->Size.Y; y++)
				{
					int y1 = y - d_pos.Y;
					for (int x = 0; x < p->Size.X-1; x++)
					{
						int x1 = x - d_pos.X;
						if (x1 >= 0 && x1 < target.Size.X - 1 && y1 >= 0 && y1 < tar-
get.Size.Y)
						{
#if DEBUG
							SetConsoleTextAttribute(args->cons, (WORD)Color::Red);
							SetConsoleCursorPosition(args->cons, p->Position + Point2(x
+ 1, y + 1) - p->Center);
							if (p->Sprite[y][x] != '\0')
								cout << p->Sprite[y][x];
							else
								cout << 0;
							SetConsoleTextAttribute(args->cons, (WORD)Color::White);
#endif
							return p;
						}
#if DEBUG
						else
						{
							SetConsoleTextAttribute(args->cons, (WORD)Color::Green);
							SetConsoleCursorPosition(args->cons, p->Position + Point2(x
+ 1, y + 1) - p->Center);
							if (p->Sprite[y][x] != '\0')
								cout << p->Sprite[y][x];
							else
								cout << 0;
							SetConsoleTextAttribute(args->cons, (WORD)Color::White);
						}
#endif
					}
				}
			}
			return NULL;
		}
		void On_Key_UP()
		{
			if (selected_index > 0)
			{
				selected_index--;
				page = selected_index / parts_on_page;
				args->update = true;
```

```
                        }
                        else
                        {
                                args->update = false;
                        }
                }
                void On_Key_Left()
                {
                        if (page > 0)
                        {
                                page--;
                                selected_index = page * parts_on_page;
                                args->update = true;
                        }
                        else
                        {
                                args->update = false;
                        }
                }
                void On_Key_Down()
                {
                        if (selected_index < parts_count - 1)
                        {
                                selected_index++;
                                page = selected_index / parts_on_page;
                                args->update = true;
                        }
                        else
                        {
                                args->update = false;
                        }
                }
                void On_Key_Right()
                {
                        if (page < pages_count - 1)
                        {
                                page++;
                                selected_index = page * parts_on_page;
                                args->update = true;
                        }
                        else
                        {
                                args->update = false;
                        }
                }
                void On_Key_Accept()
                {
                        int index = selected_index;
                        page = 0;
                        selected_index = 0;
                        args->part_selecting = false;
                        args->connector_selecting = true;

                        if (args->parts_count == 0)
                        {
                                // Add main part in arr
                                args->parts = new Part[1]{ Get_Part((Part_Type)(index + 1)) };
                                args->parts_count = 1;

                                // Add main part

                                args->parts[0].Position = Point2(args->Size.X / 2 - 1, args->Size.Y / 2 - 1 + args-
>Scroll);
                                args->selected_part = &args->parts[0];
                                args->selected_con_index = 0;
                                args->selected_connector = &args->selected_part->Connectors[args-
>selected_con_index];
                                args->cursor_pos = new Point2(args->selected_part->Position + args-
>selected_connector->Position);
                                args->update = true;
                                return;
                        }

                        // Create new part
                        Part* adding = new Part(Parts[index]);
                        PartConnector* con = NULL;

                        // Finding of connection to new part
                        for (int i = 0; i < adding->Connectors_Count; i++)
```

```
                        {
                                PartConnector* temp = &adding->Connectors[i];
                                if (args->selected_part->Connectable(*args->selected_connector, *temp))
                                {
                                        con = temp;
                                        break;
                                }
                        }
                        if (con == NULL)
                        {
                                MessageBoxW(NULL, L"Нельзя присоединить!", L"Ошибка", MB_ICONERROR);
                                args->update = true;
                                args->part_selecting = true;
                                args->connector_selecting = false;
                                return;
                        }

                        adding->Position = *args->cursor_pos + args->selected_connector->Connected_Position - con-
>Position;
                        if (Collide_With(adding) != NULL)
                        {
                                MessageBoxW(NULL, L"Пересечение с другой деталью!", L"Ошибка", MB_ICONERROR);
                                args->update = true;
                                args->part_selecting = true;
                                args->connector_selecting = false;
                                return;
                        }

                        // Connect new part to exist
                        con->Connected = args->selected_connector;
                        args->selected_connector->Connected = con;

                        // add new part to arr
                        Part* parts = new Part[args->parts_count + 1];
                        for (int i = 0; i < args->parts_count; i++)
                        {
                                parts[i] = args->parts[i];
                                parts[i].Index = i;
                        }
                        adding->Index = args->parts_count;
                        parts[args->parts_count++] = *adding;
                        args->parts = parts;
                }
        };
        int Scroll = 0;
        int Scroll_Speed = 3;
        Point2 Size = Point2(100, 47);
        bool is_selection_color = false;
        bool part_selecting = true;
        bool connector_selecting = false;
        bool update = true;
        Window_Parts_Panel_Args Parts_Panel_Args{ this, Point2(25, Size.Y - 2) };

        Part* parts = NULL;
        USHORT parts_count = 0;

        Part* selected_part = NULL;
        int selected_con_index = 0;
        PartConnector* selected_connector = NULL;
        Point2* cursor_pos = new Point2();
        void Move_Cursor()
        {
                Point2 con_point = *cursor_pos + Point2(0, Scroll);
                if (con_point.X > 0 && con_point.X < Size.X - 1 &&
                        con_point.Y > 0 && con_point.Y < Size.Y - 1)
                        SetConsoleCursorPosition(cons, *cursor_pos + Point2(0, Scroll));
        }
        void Reset()
        {
                if (parts_count != 0)
                {
                        selected_part = &parts[0];
                        selected_con_index = 0;
                        selected_connector = &selected_part->Connectors[0];
                        cursor_pos = new Point2(selected_part->Position + selected_connector->Position);
                }
        }
        void On_Key_Cancel()
        {
```

```
		if (part_selecting && parts_count != 0)
		{
			connector_selecting = true;
			part_selecting = false;
			update = true;
			Parts_Panel_Args.selected_index = 0;
			Parts_Panel_Args.page = 0;
			return;
		}
		if (connector_selecting)
		{
			update = false;
			Reset();
			Move_Cursor();
		}
	}
	void On_Key_Delete()
	{
		if (connector_selecting)
		{
			if (selected_connector->Connected == NULL)
			{
				MessageBoxW(NULL, L"Нечего удалять!", L"Ошибка", MB_ICONERROR);
				update = false;
				return;
			}

			Part* deleting = selected_connector->Connected->Main;
			for (int i = 0; i < deleting->Connectors_Count; i++)
			{
				// Mast be only one connection with curent
				if (deleting->Connectors[i].Connected != NULL && deleting->Connectors[i].Connected
!= selected_connector)
				{
					MessageBoxW(NULL, L"У детали есть другие соединения!", L"Предупреждение",
MB_ICONEXCLAMATION);
					update = false;
					return;
				}
			}

			bool founded = false;
			--parts_count;
			for (int i = 0; i < parts_count; i++)
			{
				if (parts[i] == deleting)
					founded = true;
				if (founded)
				{
					parts[i] = parts[i + 1];
					parts[i].Index = i;
				}
			}
			MessageBoxW(NULL, L"Деталь удалена.", L"Информация", MB_ICONINFORMATION);
			update = true;
			return;
		}

		update = false;
	}
	void On_Key_UP()
	{
		if (part_selecting)
		{
			Parts_Panel_Args.On_Key_UP();
			return;
		}

		Scroll += Scroll_Speed;
		update = true;
	}
	void On_Key_Down()
	{
		if (part_selecting)
		{
			Parts_Panel_Args.On_Key_Down();
			return;
		}
```

```
                Scroll -= Scroll_Speed;
                update = true;
        }
        void On_Key_Left()
        {
                if (part_selecting)
                {
                        Parts_Panel_Args.On_Key_Left();
                        return;
                }
                if (connector_selecting)
                {
                        selected_con_index = selected_con_index > 0 ? selected_con_index - 1 : selected_part-
>Connectors_Count - 1;
                        selected_connector = &selected_part->Connectors[selected_con_index];
                        cursor_pos = new Point2(selected_part->Position + selected_connector->Position);
                        Move_Cursor();
                        update = false;
                        return;
                }
        }
        void On_Key_Right()
        {
                if (part_selecting)
                {
                        Parts_Panel_Args.On_Key_Right();
                        return;
                }
                if (connector_selecting)
                {
                        selected_con_index = selected_con_index < selected_part->Connectors_Count - 1 ? select-
ed_con_index + 1 : 0;
                        selected_connector = &selected_part->Connectors[selected_con_index];
                        cursor_pos = new Point2(selected_part->Position + selected_connector->Position);
                        Move_Cursor();
                        update = false;
                        return;
                }
        }
        void On_Key_Accept()
        {
                if (part_selecting)
                {
                        Parts_Panel_Args.On_Key_Accept();
                        return;
                }

                if (connector_selecting)
                {
                        if (selected_connector->Connected)
                        {
                                selected_part = selected_connector->Connected->Main;
                                selected_connector = &selected_part->Connectors[0];
                                selected_con_index = 0;
                                cursor_pos = new Point2(selected_part->Position + selected_connector->Position);
                                Move_Cursor();
                                update = false;
                                return;
                        }

                        for (int i = 0; i < parts_count; i++)
                        {
                                Part* p = &parts[i];
                                for (int j = 0; j < p->Connectors_Count; j++)
                                {
                                        PartConnector* c = &p->Connectors[j];
                                        if ((c->Position + c->Connected_Position + p->Position) == (select-
ed_connector->Position + selected_part->Position))
                                        {
                                                c->Connected = selected_connector;
                                                selected_connector->Connected = c;
                                                MessageBoxW(NULL, L"Соединение установлено.", L"Информация",
MB_ICONINFORMATION);
                                                update = false;
                                                return;
                                        }
                                }
                        }
```

```
                        connector_selecting = false;
                        part_selecting = true;
                        update = true;
                        return;
                    }
                }
            }
        };
```