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 <TODO Fix>**

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, оригинальная игра, |
| отличия от оригинала |
| **2. Практическая часть:** Формальная постановка задачи, UML модель, создание |
| спрайтов деталей, алгоритм решения задачи реализация и тестирование редактора |
| **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 <TODO Fix>**

Данный проект будет представлять консольное приложение для «рисования» космических ракет используя символы ASCII таблицы. В качестве эталона был выбран редактор кораблей в игре KSP. Правда в отличии от оригинала графика будет 2-х мерной.

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», и программа, реализующая данный редактор, написанный на языке программирования С++.

Содержание

[Введение 5](#_Toc100065971)

[1 Теоретический раздел 6](#_Toc100065972)

[1.1 Словесная постановка задачи 6](#_Toc100065973)

[1.2 Описание функций редактора 6](#_Toc100065974)

[1.3 Отличия от оригинала 7](#_Toc100065975)

[2 Проектный раздел 12](#_Toc100065976)

[2.1 Формальная постановка задачи 12](#_Toc100065977)

[2.2 UML модель 13](#_Toc100065978)

[2.3 Алгоритм решения задачи 17](#_Toc100065979)

[2.4 Реализация автомата для управления поведением танка 18](#_Toc100065980)

[3 Программный раздел 20](#_Toc100065981)

[3.1 Описание программы 20](#_Toc100065982)

[3.2 Описание структуры данных 20](#_Toc100065983)

[3.3 Описание основных функций 22](#_Toc100065984)

[4 Экспериментальный раздел 24](#_Toc100065985)

[4.1 Тестирование в нормальных условиях 24](#_Toc100065986)

[4.2 Тестирование в исключительных условиях 26](#_Toc100065987)

[4.3 Тестирование в экстремальных условиях 28](#_Toc100065988)

[4.4 Итоги тестирования 28](#_Toc100065989)

[Заключение 29](#_Toc100065990)

[Список использованных источников 30](#_Toc100065991)

[Приложение А 30](#_Toc100065992)

# ВВЕДЕНИЕ <TODO Fix>

Недавно по сети прокатился тренд на создание ASCII графики. 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 программы;
5. тестирование работоспособности программы;
6. анализ вычислительной сложности.

Расчетно-пояснительная записка состоит из:

* теоретической части, содержащей словесную постановку задачи, отличия реализации от оригинала.
* практической части, содержащей формальную постановку задачи, создание спрайтов деталей, UML модель и алгоритм работы программы.

1. **Теоретический раздел**

## 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 С++ без использования сторонних фреймворков и движков. Дополнительно требуется создать дружественный пользователю интерфейс используя для отрисовки только консоль.

## Описание оригинальной игры (KSP)

Kerbal Space Program – Игра песочница, симулятор космической компании. 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 игрок 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, их можно перемещать но они не будут учувствовать при запуске;
* детали не должны пересекаться.

## 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.

## 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.

## 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.

1. **Проектный раздел**

## 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 (ОК) – 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*: длярешениязадачи 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.

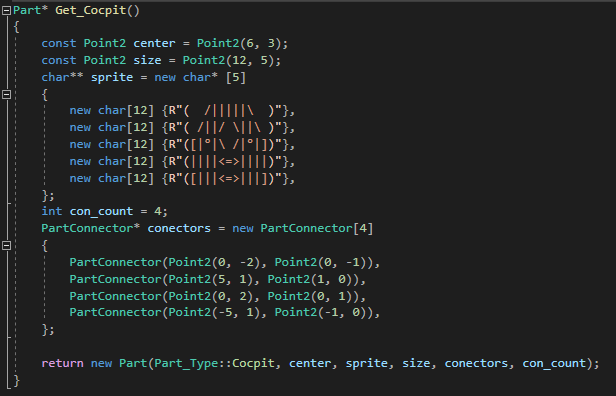


Рисунок 3 – Пример спрайта

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.

## 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.

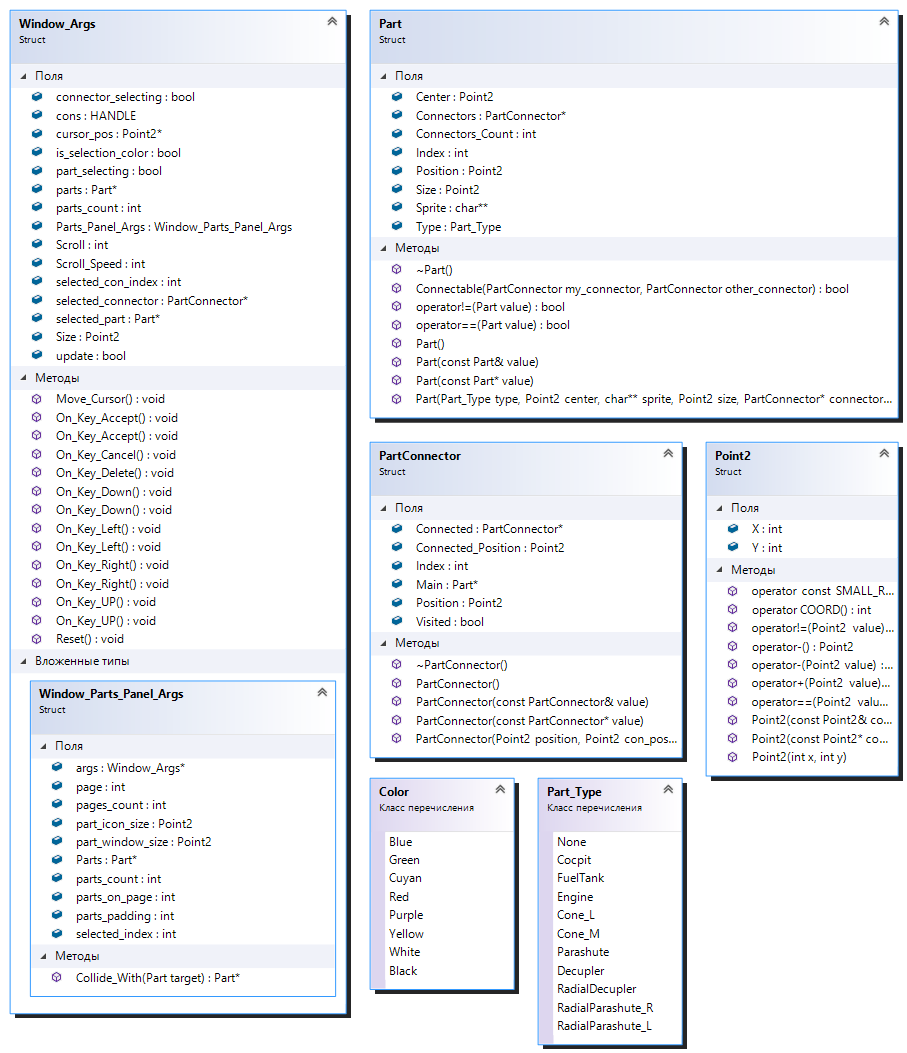


Рисунок 4 – Диаграмма классов

## Алгоритм решения задачи

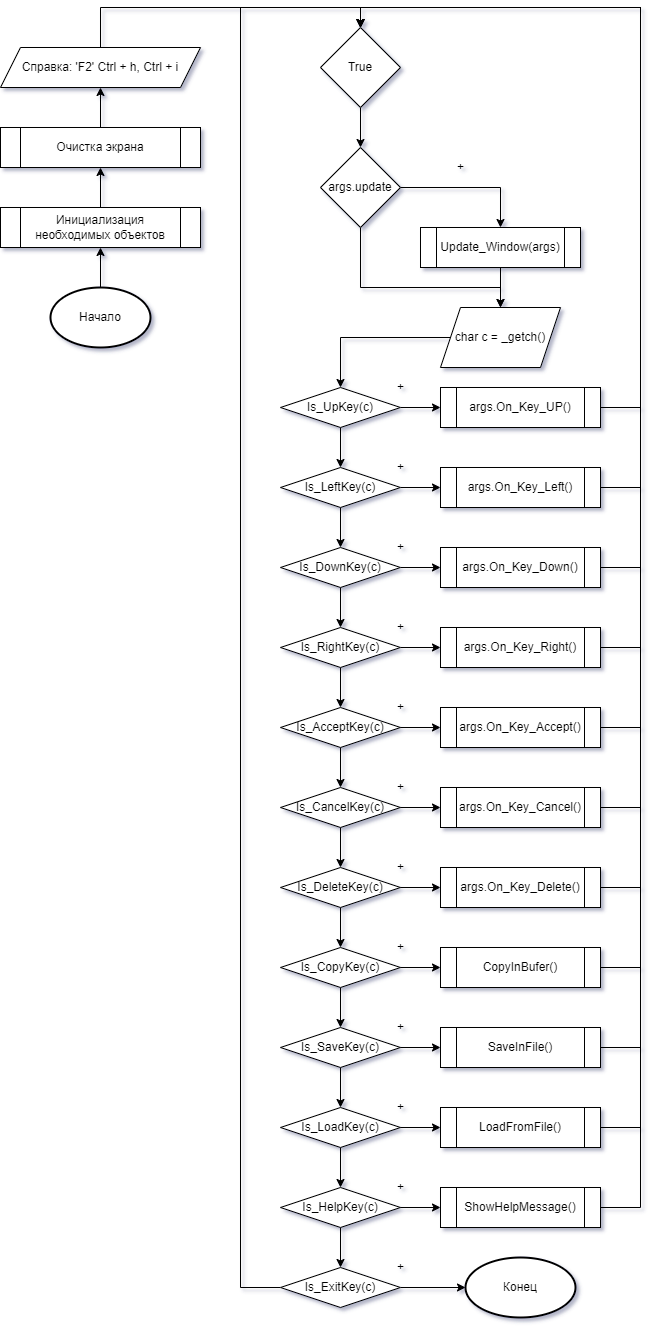


Рисунок 5 – Основной цикл программы

Данный цикл выполняется да тех пор, пока пользователь не выйдет из программы. В данном алгоритме используются макросы IS\_XKey(key) их код показан на рисунке 6.

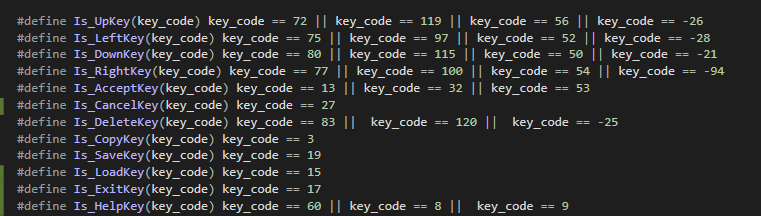


Рисунок 6 – Макросы

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.

## 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.

1. **Программный раздел**

## Описание программы

Данная программа реализована на языке С/С++ 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»*.

## 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).



Рисунок 7 – Скрипт записи в файл.

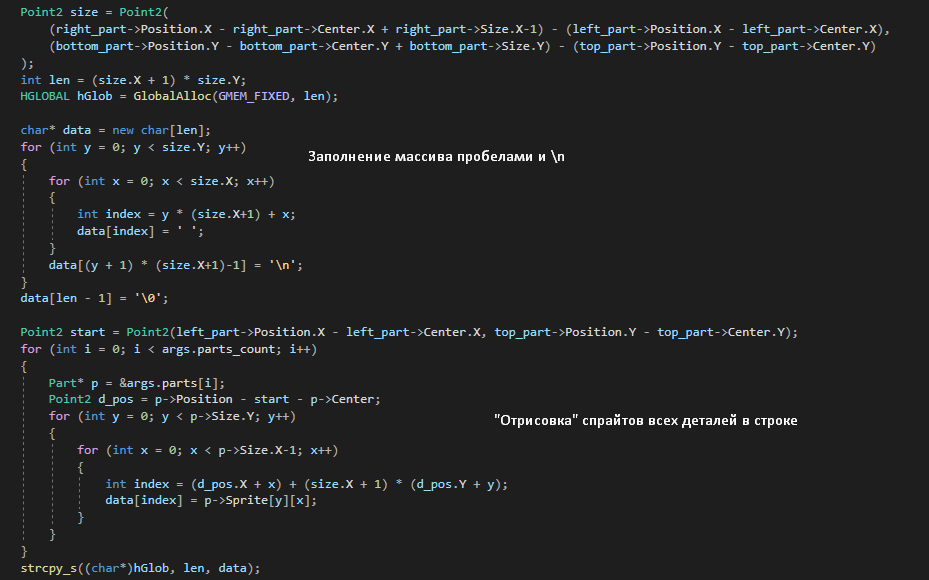


Рисунок 8 – Скрипт «отрисовки» в буфер обмена

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

## Описание основных функций

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

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.

void Update\_Window(Window\_Args& args);

void Draw\_Parts(Window\_Args args);

Остальные функции представлены в приложении листинга программы.

1. **Экспериментальный раздел**

## Тестирование в нормальных условиях

Было проведено тестирование игры на наличие критичных ошибок и багов. Танк нормально реагировал на нажатия клавиш для управления его передвижением (рисунки 13–15), стрельбы.

Для этого проведения теста был использован первый игровой уровень. Ниже приведены скриншоты работы игры.

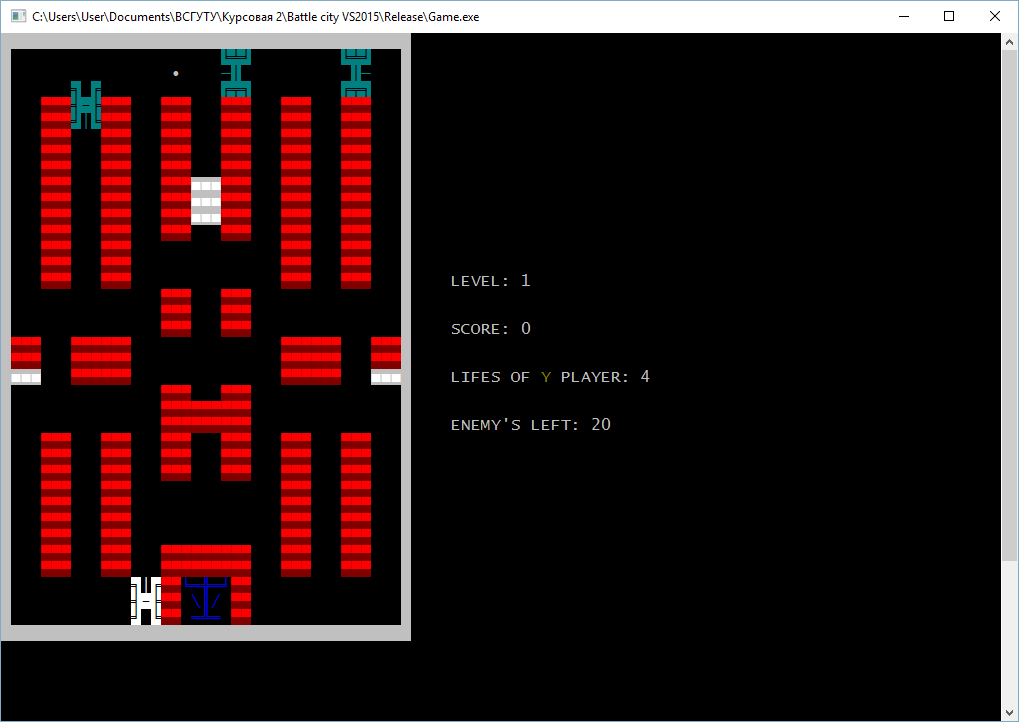


Рисунок 13 – Начальное положение танка



Рисунок 14 – Движение вверх

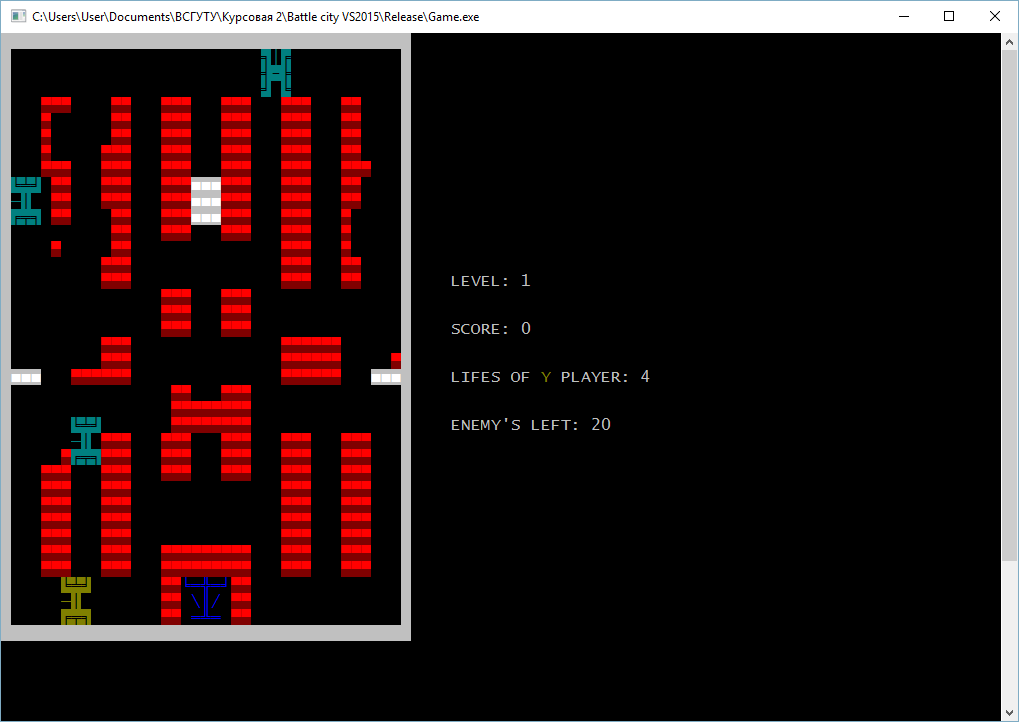


Рисунок 15 – Движение влево

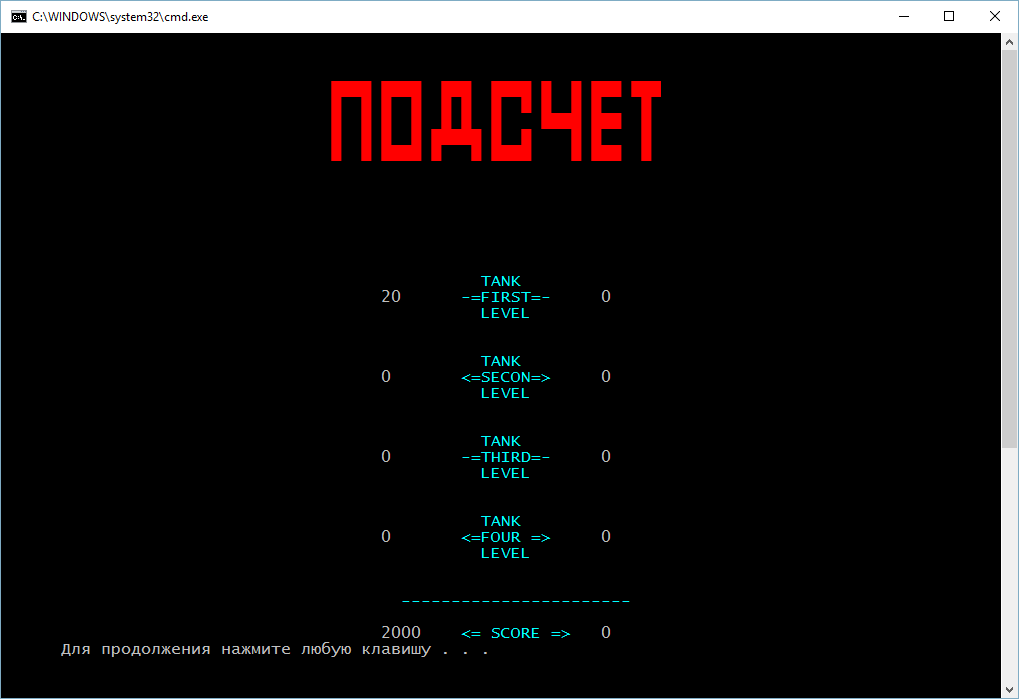


Рисунок 16 – Подсчет очков первого уровня

Как видно из рисунка 16 по результатам тестирования игры можно судить о нормальной реакции игры по завершению уровня и перехода на следующий.

## Тестирование в исключительных условиях

Была произведена проверка столкновения танков и снарядов с объектами. Снаряды уничтожали только разрешаемые объекты (танки, бетонные блоки на рисунке 17) и пролетали сквозь воду, кусты и т.п. на рисунке 19. Танки же не могли проехать сквозь непроходимые объекты. Так же объекты не выходили за границы карты на рисунке 18.



Рисунок 17 – Тестирование коллизии объектов



Рисунок 18 – Тестирование игры

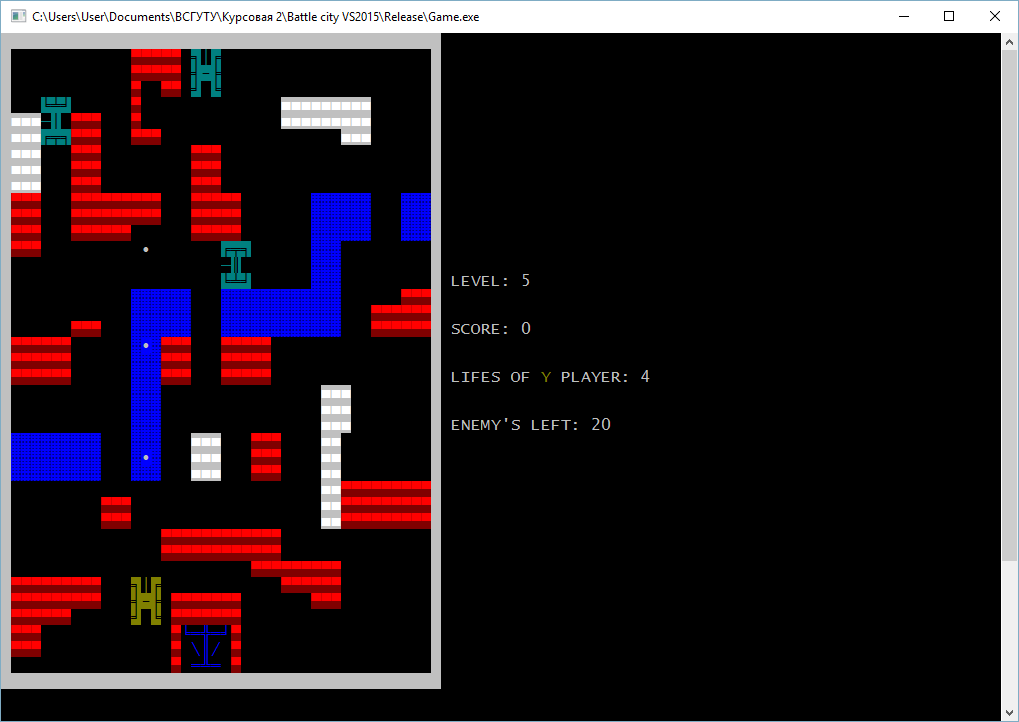


Рисунок 19 – Тестирование снарядов с водой



Рисунок 20 – Тестирование маскировочных кустов

Как видно из рисунков 17, 18, 19 и 20 по результатам тестирования игры можно судить о полной работоспособности программы, то есть ее играбельности. Из данных скриншотов видно о работоспособности всех основных игровых элементов, помогающих в создании игровой атмосферы. Можно судить о правильности работы одиночного игрового режима.

## Тестирование в экстремальных условиях

Тестирование мульти плеера игры для двоих игроков, показало себя стабильно. Оба танка передвигаются и взаимодействуют друг с другом. Ниже на рисунке 18 приведен скриншот мультиплеера.



Рисунок 21 – Проверка мультиплеера

Из рисунка 21 можно судить о работоспособности мультиплеера и режима игры на двоих.

## 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 клоне игры «Battle City» для семейства ОС Windows, выполненное в консольном приложении.

**Заключение**

Разработка механизма имитации искусственного интеллекта и программного обеспечения для создания игры «Battle City» были выполнены в полном объеме.

В ходе выполнения данной работы был рассмотрен теоретический материал, посвященный принципам объектно-ориентированному программированию и искусственному интеллекту в играх. Затем была разработана объектная модель игровых классов. После чего программа была реализована на языке высокого уровня C/C++ и выполнено тестирование ее работоспособности.

Можно сделать вывод об оптимальной работе игры, так как не было выявлено ошибок программы при ее тестировании.

**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 С++ [Текст]: учебник / Р. Лафоре. – М.: Питер , 2004. – 992 с.
3. Battle City [Электронный ресурс]: [опубл. 15.10.2015] / Wikimedia Foundation, Inc.. – URL: [https://ru.wikipedia.org/wiki/Battle\_City](https://ru.wikipedia.org/wiki/Матрица_(математика)).
4. Создание искусственного интеллекта для игр [Электронный ресурс]: [опубл. 1.09.2015] / Хабрахабр. – URL: [https://habrahabr.ru/company/intel/blog/265679/](https://ru.wikipedia.org/wiki/Матрица_(математика)).

**Приложение А**

**Листинг программы**

…