МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Математическое обеспечение и применение ЭВМ»

Курсовой проект

по дисциплине «Программирование»

на тему: «Разработка объектно-ориентированного приложения. Геометрические фигуры, состоящие из ломанных линий»

ПГУ 09.03.04 – 02КП201.06 ПЗ

Направление подготовки – 09.03.04 «Программная инженерия»

Выполнил студент: Дьячков Даниил Александрович

Группа: 20ВП1

Руководитель:

к.т.н., доцент _____ Гурьянов Л.В.

Проект защищен с оценкой _____

Преподаватели _____

_____

_____

Дата защиты _____

Пенза 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»

заведующий кафедрой

_____П.П. Макарычев

«___»_____ 2021г.

ЗАДАНИЕ

на курсовой проект

по дисциплине «Программирование»

на тему: «Разработка объектно-ориентированного приложения. Геометрические фигуры, состоящие из ломанных линий»

1.	**Студент гр.** *20ВП1* **факультета** ВТ **направления** 09.03.04
	Дьячков Даниил Александрович
2.	**Руководитель работы** *Гурьянов Лев Вячеславович*
3.	**Время проектирования** с «15» февраля 2021 по «30» мая 2021
4.	**Тема проекта**: Разработка объектно-ориентированного приложения. Геометрические фигуры, состоящие из ломанных линий
5.	**Техническое задание на курсовую работу (назначение, технические требования)**
	Назначение: *создание, визуализация и перемещение геометрических фигур,*
	состоящих из ломанных линий
	Основные функции приложения*:*
	а) создать геометрическую фигуру;
	б) отобразить фигуру;
	в) *переместить фигуру (перемещение и поворот);*
	г) *Совместить несколько геометрических фигур в одну;*
	Структура данных: *динамический список умных указателей*
	Технология разработки*: ООП*
	Язык программирования*: C++*

***Среда исполнения**: Windows 10 (x64)*

6. Содержание работы

6.1. **Пояснительная записка (перечень вопросов, подлежащих разработке, расчетов, обоснований, описаний)**

1) *Анализ предметной области*
2) *Анализ функциональных требований*
3) *Проектирование*
4) *Реализация*
5) *Тестирование*
6) *Оформление пояснительной записки*

7. Календарный график по выполнению работы

Наименование этапов работы	Объем работы (%)	Срок выполнения	Подпись руководителя
1) *Анализ предметной области и требований*	10	25.02.2021	
2) *Проектирование*	20	17.03.2021	
3) *Реализация*	30	20.04.2021	
4) *Тестирование*	20	10.05.2021	
5) *Оформление пояснительной записки*	20	5.06.2021	

Дата выдачи задания «15»февраля 2021г.

Руководитель курсового проекта _____ Гурьянов Л.В.

Задание к исполнению принял «15» февраля 2021г.

Студент Дьячков Даниил Александрович

Содержание

ВВЕДЕНИЕ

В курсовом проекте требуется разработать программные средства (ПС) создания, визуализации и перемещения геометрических фигур, состоящих из ломанных линий.

Для моделирования программных средств используется язык UML. Разработка осуществляется на языке C++.

Процесс создания ПС включает следующие этапы: анализ предметной области и требований к программным средствам, проектирование, реализация и тестирование.

Графическая часть проекта включает диаграмму классов и диаграмму компонентов, выполненных в нотации UML.

1. Разработка объектно-ориентированного приложения. Геометрические фигуры, состоящие из ломанных линий

1.1. Анализ предметной области и функциональных требований.

1.1.1. Анализ предметной области

В варианте тезисного задания (далее - ТЗ) на курсовой проект (далее - КП) представлена предметная область разработки, которая предписывает создание, отображение и перемещение геометрических фигур (далее - ГФ). При этом, в требованиях к КП указано на необходимость использования динамического списка умных указателей. Так как точная форма фигур не указана, необходимо реализовать такой алгоритм, который будет способен отобразить любую фигуру, состоящую из ломанных линий. Стоит учитывать, что ГФ, состоящая из ломанных линий – по сути, представляет из себя замкнутую ломанную линию с произвольным количеством изломов. Таким образом, приходим к выводу, что ГФ должна в себе содержать список (выбор списка, а не массива может быть обусловлен произвольным количеством точек) из точек или же из прямых.

Модель предметной области приведена на рисунке 1. В ней присутствуют класс точки, объекты которого содержатся к классе фигуры. Класс ConsoleRenderer содержит в себе множество объектов класса фигуры. Все объекты, которые содержатся в объекте данного класса будут отображены в консоли.

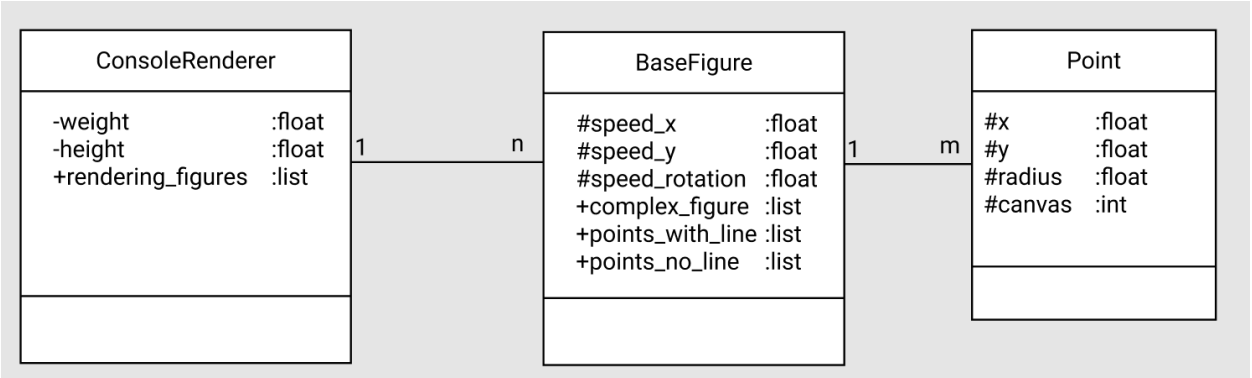

Рисунок 1 – Модель предметной области

1.1.2. Анализ функциональных требований

В техническом задании на курсовой проект определены следующие функциональные требования:

- Создать ГФ
- Отобразить ГФ
- Переместить ГФ
- Объединить (создать сложную) ГФ

Диаграмма вариантов использования, соответствующая этим требованиям, приведена на рисунке 2.

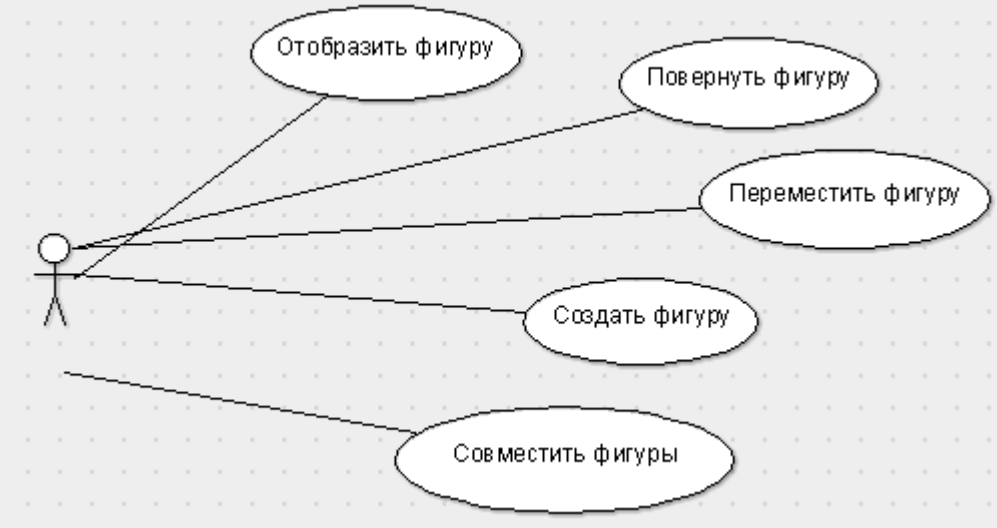

Рисунок 2 – диаграмма вариантов использования

Сценарии варианта использования «Переместить фигуру» и «Создать сложную фигуру» приведены в таблицах 1 – 2.

Таблица 1 – Сценарий варианта использования «Создать сложную фигуру»

Наименование: Совместить фигуры (Создать Сложную фигуру)
ID: 1
Краткое описание: система создает объект «Фигура» состоящий из двух и более фигур (не обязательно простых)
Действующие лица: пользователь
Предусловие: используемые при создании фигуры созданы
Основной поток: 1. Пользователь перетаскивает одну из фигур внутрь другой (так, чтобы линии фигур соприкасались) 2. Пользователь отпускает зажатую левую кнопку мыши в тот момент, когда фигуры буду соприкасаться

3. . Система создает объект «Фигура» из соприкасающихся фигур
Постусловие: сложная фигура создана

Таблица 2 – Сценарий варианта использования «переместить фигуру»

Наименование: Переместить фигуру мышкой
ID: 2
Краткое описание: система перемещает фигуру вслед за движением курсора мыши
Действующие лица: пользователь
Предусловие: Фигура существует, консоль поддерживает события мыши, фигура отображается не за границами консоли
Основной поток: 1. Пользователь помещает курсор мыши внутрь отображаемой геометрической фигуры 2. Пользователь зажимает левую клавишу курсора мыши 3. Пользователь перетаскивает фигуру (двигает мышью с зажатой левой клавишей) 4. Система перемещает фигуру вслед за движением мыши
Постусловие: левая кнопка мыши отпущена

1.2. Проектирование

1.2.1. Диаграмма классов

Диаграмма классов приведена на рисунке 3. На приведённой диаграмме класс BaseFigure может включать в себя множество объектов класса Point (Отношение агрегации, поскольку разрушение одного объекта не обязательно приводит к разрушению второго[1]). В тоже время, класс ConsoleRenderer может включать в себя множество объектов класса BaseFigure (Отношение агрегации, поскольку разрушение одного объекта не обязательно приводит к разрушению второго).

Рисунок 3 – диаграмма классов

1.2.2. Спецификация классов

Спецификация классов приведена в таблицах 3 – 5.

Таблица 3 – спецификация класса Point

Название класса	Point
Назначение класса	Класс сущности точки
Члены класса:	x: float – координата точки по оси x y: float – координаьа точки по оси y radius: float – радиус от центра фигуры canvas: int – id консоли
Функции класса:	get_canvas() – получить id консоли get_raw_x() – получить координату по x get_raw_y() - получить координату но y reboot_radius() – сбросить радиус get_x() - получить координату по x get_y() - получить координату но y border_control() – контроль выхода за границы консоли get_polar_coord() – получить полярные координаты точки get_change_polar_coord() – получить новые полярные координаты точки set_polar_coord() – установить полярные координаты move_point() – переместить точку set_coord() – установить координаты точки Point() - конструктор get_extreme_points() – получить крайние точки фигуры

	draw() - нарисовать точку draw_line() – нарисовать линию

Таблица 4 – спецификация класса BaseFigure

Название класса	BaseFigure
Назначение класса	Класс сущности точки
Члены класса:	canvas: int – id консоли norm_speed_x: float – скорость перемещения в секунду по оси x norm_speed_y: float - скорость перемещения в секунду по оси y norm_speed_rotation: float скорость поворота в секунду speed_x: float - скорость перемещения по оси x за один кадр speed_y: float - скорость перемещения по оси y за один кадр speed_rotation: float – скорость поворота фигуры за 1 кадр complex_figure: list<shared_ptr<BaseFigure>> - список простых фигур внутри сложной points_with_line: list<shared_ptr<Point>> - список точек, соеденённых линией points_no_line: list<shared_ptr<Point>> - список точек, не соеденённых линией left_button_pressed: bool – перемещает ли пользователь эту фигуру is_active: bool – может ли фигура свободно перемещаться user_move_me_status: int статус перемещения фигуры пользователем
Функции класса:	get_random_speed() – получить случайную скорость get_random_rotation_speed() – полуить случайную скорость поворота is_complex_figure() - является ли фигура сложной set_speed() установить скорость get_canvas() получить id консоли get_raw_center_point() получить центр фигуры (float) get_center_point() получить центр фигуры BaseFigure() конструктор draw() нарисовать фигуру get_all_points_with_line() получить все точки, учавствующие в рисовании линии

	get_all_points() получить все точки intersection_of_figures пересекаются ли фигуры move_figure() переместить фигуру rotation_figure() повернуть фигуру get_extreme_points(); получить крайние точки border_regulate() контроль столкновения с границами консоли is_2_lines() пересекаются ли 2 отрезка is_point_into_figure() находится ли точка внутри фигуры renderer() обработка всего перемещения и отрисовки фигуры reboot_radius() сбросить радиусы в точках фигуры add_figure() добавить еще одну фигуру к этой sum_figures() сложить две фигуры

Таблица 5 – Спецификация класса ConsoleRenderer

Название класса	ConsoleRenderer
Назначение класса	Класс отрисовки от консоли
Члены класса:	mouse_pos_x: float – текущая позиция мышки по оси x mouse_pos_y: float – текущая позиция мышки по оис y last_mouse_pos_x: float – прошлая позиция мыши по оси x last_mouse_pos_y: float – прошлая позиция мыши по оис y delay: int – планируемое время между кадрами отрисовки weight: float – ширина консоли height: float – высота консоли current_time: uint64_t – текущее время с начала эпохи update_time: uint64_t – время следующего планируемого обновления с начала эпохи last_time_update: uint64_t прошлое время обновления с начала эпохи difference_between_times: int разница между временем прошлого обновления и текущего обновлением rendering_figures: list<shared_ptr<BaseFigure>>
Функции класса:	renderer() – отрисовка всех фигур на экране

	add_figure() – добавить фигуру в отрисовку на данную консоль event_processing() – обработка событий мыши ConsoleRenderer() - конструктор

1.3. Реализация

Структура проекта представлена на рисунке 4

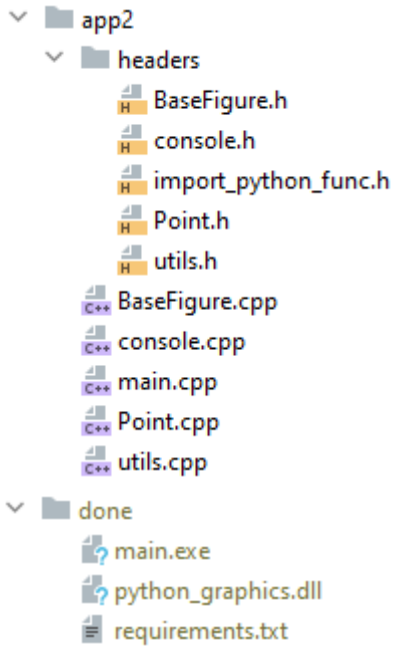

Рисунок 4 – структура проекта

Диаграмма компонентов приведена на рисунке 5

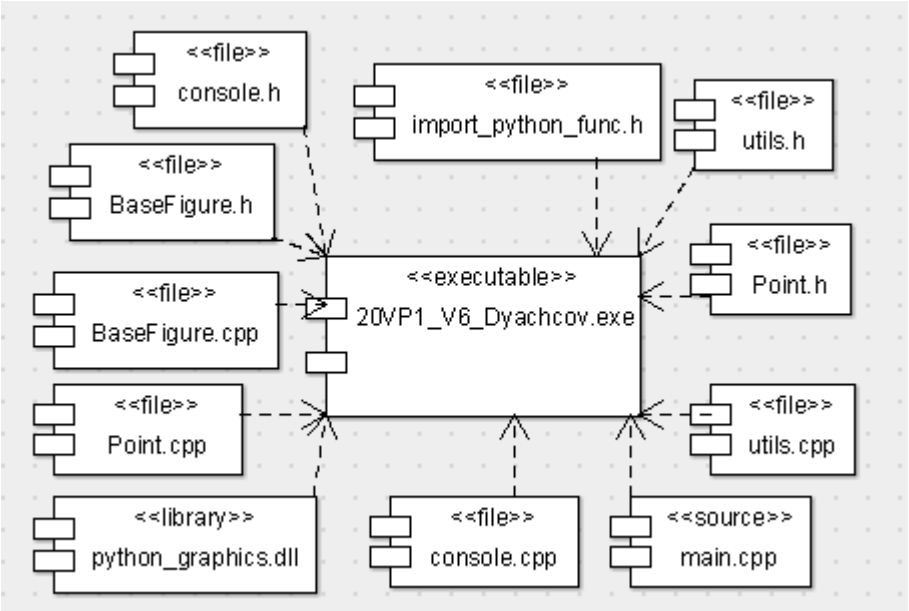

Рисунок 5 – диаграмма компонентов

Описание компонентов приведено в таблице 6.

Таблица 6 – Описание компонентов

Компонент	Назначение
20VP1_V6_Dyachcov.exe	Исполняемый файл приложения
Point.h	Заголовочный файл класса Point
BaseFigure.h	Заголовочный файл класса BaseFigure
Import_python_func.h	Заголовочный файл библиотеки графики
Utils.h	Заголовочный файл утилит
console.h	Заголовочный файл класса ConsoleRenderer
Point.cpp	Файл реализации класса Point
main.cpp	Главный файл приложения (с созданием фигур и запуском консоли)
Utils.cpp	Файл реализации утилит
Console.cpp	Файл реализации класса ConsoleRebderer
BaseFigure.cpp	Файл реализации класса BaseFigure
python_graphics.dll	Библиотека для реализации отрисовки графики в консоли

Приведем некоторые соображения, касающиеся реализации. Начнем с анализа того, что требуется хранить в динамическом списке: линии или точки. Так как прямая, сама по себе, будет в любом случае содержать в себе 2-е точки (начальная и конечная), необходимость в подобной структуре данных отпадает (под структурой данных понимается непосредственно способ хранения данных в программе, а не structure в языке программирования C++). Приняв во внимание вышеизложенный факт, выбор структуры данных остановим на динамическом массиве из точек. Следовательно, каждая фигура должна хранить список точек. В виду того, что хранить список точек не экономично с точки зрения памяти (т.к. точки могут быть объявлены, а уже потом занесены в список) будем хранить в списке указатели на точки. Принимая во внимание новые возможности языка C++, будем реализовывать не просто список указателей на точки, а список умных указателей на точки (их особенность заключается в том, что при уничтожении последнего указателя на объект, уничтожается и сам объект автоматически. Как будет видно в дальнейшем, список указателей – единственная структура в классе, которая могла бы потребовать специально прописанного удаления при уничтожении

объекта. Из этого делаем вывод, что переопределение системного деструктора (который неявно присутствует в каждом классе) будет избыточной). Т.к. в ходе реализации КП потребуется передача точек из массива в сторонние функции, будем использовать указатель std::shared_ptr вместо std::unique_ptr, более экономичного по памяти.

Так как в ТЗ сказано, что необходимо реализовать сложение фигур, следовательно количество фигур будет непостоянным. Значит необходимо будет также иметь какой-либо контейнер, в котором будут храниться фигуры. В виду вышеизложенных соображений выбираем динамически создаваемый список умных указателей на фигуру.

В виду того же условия ТЗ, встает вопрос: «Как реализовывать сложение?»

Автор КП видит два пути: создание отдельной структуры данных для «сложной фигуры» (т.е. фигуры, состоящей из нескольких фигур) и реализация «сложной фигуры» внутри структуры данных для простой фигуры (один класс, который может выступать в роли как простой фигуры, так и сложной) (Примечание: под простой фигурой понимается фигура, заданная набором точек. Под сложной фигурой понимается фигура, включающая в себя другие фигуры). Рассмотрим первый случай: возникает вопрос о сложении двух «сложных фигур», или же вопрос о сложении сложной фигуры с простой. Если это реализовывать еще одной структурой данных (условно сложно-сложная фигура, или сложная фигура второго порядка), то приходим к немаштабируемому решению: порядок сложности будет жестко ограничен программным кодом. В виду предубежденности автора КП о том, что программный код должен быть максимально «гибким», данное решение считается неподходящим. Реализация сложной фигуры 1-ого и последующих порядков в одном контейнере, по своей сути, не будет отличаться от такой же реализации «внутри» простой фигуры. В этом случае автору КП видится избыточным иметь две структуры для одних и тех же целей, когда можно иметь одну.

В целях единообразия и реализации концепций полиморфизма (одного из «3-х китов ООП»), к тому же принимая во внимание вышеизложенные соображения, автор КП считает необходимым реализовывать данный функционал (создание «сложных» фигур) в виде динамического списка умных указателей на «простые фигуры» (Примечание: при подобной реализации нет ограничения на содержание только простых фигур в списке, однако подобная возможность не будет использоваться в данном КП в силу своей ненадобности). В виду решения задач, которые, возможно, потребуется решить в будущем, необходимо закладывать основу для реализации всех возможных задач, даже тех, которые не прописаны в ТЗ. А именно, возможность создания фигур, не связанных ломанной линией (фигура из отдельных точек, график какой-либо функции, кривая, окружность и т.д.). Для этого необходимо иметь динамический список умных указателей на точки, через которые не будет проведена линия. По своей сути, данные точки будут отличаться только методом отрисовки, поэтому реализовать данный extend (если говорить языком диаграммы UML) не составит больших временных затрат.

Также необходимо предусмотреть в фигуре поля для хранения скорости движения по осям x и y, а также скорость поворота фигуры. Более того, так как при движении фигура должна проходить за единицу времени фиксированное расстояние (и поворачиваться на фиксированный угол) необходимо также предусмотреть поля для фиксированных (привязанных ко времени) скоростей.

Для реализации перемещения фигур пользователем необходимо реализовать переменную, в которой хранится состояние мыши (нажата или отжата) и состояние мыши относительно данной фигуры (мышь отжата, нажата, но не наведена на фигуру, нажата и наведена на конкретную фигуру).

Возможность сложения фигур (т.е. изменения числа отрисовываемых фигур) будет реализована при помощи хранения всех фигур в одном контейнере – класса ConsoleRenderer, который будет отвечать за равномерное

движение фигур, сложение фигур друг с другом, обработку событий консоли, если таковые будут иметься.

Для реализации объединения фигур необходимо реализовать функцию контроля пересечения фигур, функцию захвата фигуры пользователем (поместить курсор внутрь отрисовываемой фигуры и зажать левую кнопку мыши и, не отжимая кнопки, передвинуть мышь в место, куда необходимо передвинуть фигуру) и функцию перемещения фигуры пользователем.

1.4. Тестирование

Результаты функционального тестирования представлены в таблице 7

Таблица 7 – Результаты тестирования

Вариант использования	**Тест**	**Результат**
Создать фигуру	**Тест1** Заданы точки одной фигуры с координатами (10, 10); (30, 10); (30, 30)	Тест пройден, Фигура создана
	Тест2 Заданы точки одной фигуры с координатами (-10, -10) (-30, -60)	Тест пройден, Фигура создана не смотря на частичный выход за границы консоли
	Тест3 Заданы точки одной фигуры с координатами (-10, -10); (-30, -60); (50, 70); (10, 10)	Тест пройден, Фигура создана, не смотря на полный выход за границы консоли
Отобразить фигуру	**Тест1** Заданы точки одной фигуры с координатами (10, 10); (30, 10); (30, 30)	Тест пройден, Фигура нарисована (рисунок 6.а)
	Тест2	Тест пройден, Фигура нарисована за границей консоли и поэтому не видна (на

	Заданы точки одной фигуры с координатами (-10, -10) (-30, -60)	самом деле, используемая библиотека не станет рисовать эту фигуру)
	Тест3 Заданы точки одной фигуры с координатами (-10, -10); (-30, -60); (50, 70); (10, 10)	Тест пройден, Фигура нарисована частично, другая часть нарисована за границами консоли (на самом деле, используемая библиотека не станет рисовать все, что находится за пределами отображаемой консоли)
Переместить фигуру (автоматически со случайной скоростью)	**Тест1** Заданы точки одной фигуры с координатами (10, 10); (30, 10); (30, 30)	Тест пройден, Фигура перемещается со случайной скоростью (рисунок 6.б)
	Тест2 Заданы точки одной фигуры с координатами (-10, -10) (30, 60)	Тест пройден, Фигура перемещается со случайной скоростью таким образом, чтобы с течением времени оказаться полностью в окне консоли
	Тест3 Заданы точки одной фигуры с координатами (-10, -10); (-30, -60);(-50, -70); (-1, -1)	Тест пройден, Фигура перемещается со случайной скоростью, таким образом, чтобы с течением времени оказаться полностью в окне консоли
Поворачивать фигуру (автоматически со случайной скоростью	**Тест1** Заданы точки одной фигуры с координатами (10, 10); (30, 10); (30, 30)	Тест пройден, Фигура поворачивается со случайной скоростью (рисунок 6.б)
	Тест2 Заданы точки одной фигуры с координатами (-10, -10) (30, 60)	Тест пройден, Фигура поворачивается со случайной скоростью не смотря на то, что половина фигуры находится за границами консоли
	Тест3 Заданы точки одной фигуры с координатами	Тест пройден, Фигура поворачивается со случайной скоростью, при этом центр вращения смещён в точку соприкосновения с границей

	(10, 0); (30, 60); (50, 70); (50, 90)	консоли ((10, 0)) с целью избежать выхода фигуры за границы консоли в результате вращения
Совместить фигуры (Создавать сложную фигуру)	**Тест1** Заданы точки одной фигуры с координатами (10, 10); (30, 10); (30, 30) и другой с координатами (50, 50); (80, 50); (80, 80)	Тест пройден, При перемещении одной фигуры на другую фигуры объединяются (рисунок 6.в)
	Тест2 Заданы точки одной фигуры с координатами (10, 10) (30, 60) (5, 7) и другой с координатами (110, 110); (130, 110); (130, 130)	Тест пройден, При перемещении одной фигуры на другую фигуры объединяются
	Тест3 Заданы точки одной фигуры с координатами (10, 10); (30, 60); (50, 70); (0, 0) и другой с координатами (110, 110); (130, 110); (130, 130)	Тест пройден, При перемещении одной фигуры на другую фигуры объединяются

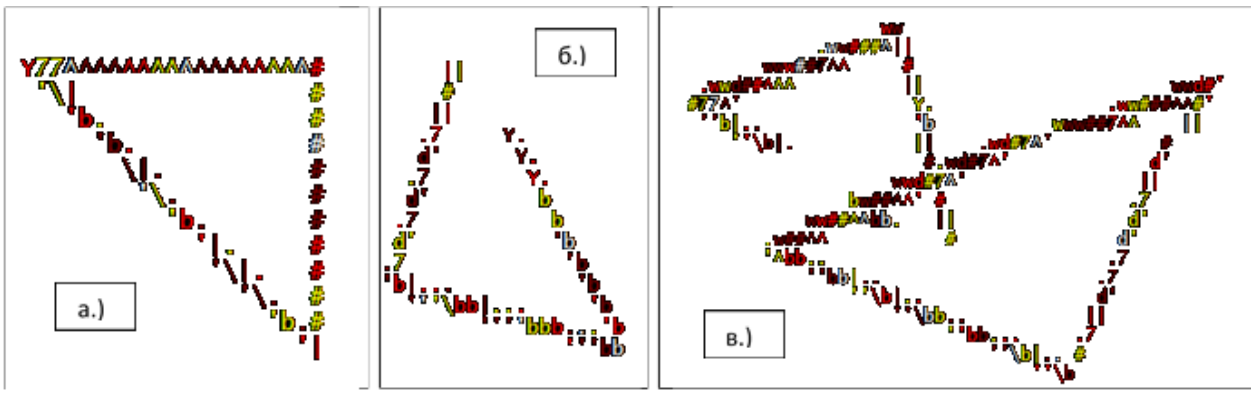

Рисунок 6.а – Тестирование. Создание треугольника, Рисунок 6.б – Тестирование, Автоматическое перемещение фигуры со случайной

скоростью, автоматический поворот фигуры со случайной скоростью.
Рисунок 6.в – Тестирование. Совмещение фигур

ЗАКЛЮЧЕНИЕ

В процессе разработки курсового проекта произведён анализ предметной области (составлена модель предметной области) и функциональных требований к проекту (разработана диаграмма вариантов использования, описаны сценарии варианта использования «Создать сложную фигуру» и «Переместить фигуру мышкой»). В процессе проектирования построена диаграмма классов, спроектированы и реализованы классы: BaseFigure, Point, ConsoleRenderer; описаны их спецификации. Результатом разработки стало приложение «20VP1_V6_Dyachcov.exe», для работы с произвольными геометрическими фигурами, состоящими из ломанных линий. При разработке приложения (написании программного кода) использовались практические руководстве по программированию на C++ [2], [3]. Структура приложения отражена на диаграмме компонентов. Заключительным этапом разработки приложения стало его тестирование, которое было пройдено успешно. Фрагмент программного кода (реализация класса Point) из файла Point.cpp представлена в приложении 1. Весь код проекта представлен на репозитории gitHub: https://github.com/DaniinXorchenabo/coursework1

Список использованных источников

1. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование/Джим Арлоу, Айла Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624с

2. Л.В.Гурьянов. Введение в программирование на языке С++/Л.В.Гурьянов, Л.С. Гурьянова, Е.А.Дзюба, Д.В.Такташкин. – Лабораторный практикум: Издательство ПГУ, 2010. – 91с

3. Т. А. Павловская. С/С++. Программирование на языке высокого уровня. – СПб.: Питер, 2003.

Приложение А. Код приложения

Заголовочный файл класса Point Point.h

```
#ifndef POINT_H
#define POINT_H

#include <list>
#include <iterator>
#include <memory>
#include <iostream>
#include <windows.h>
#include <conio.h>
#include <chrono>
#include <cmath>
#include <random>
#include "import_python_func.h"
#include "utils.h"

using namespace std;

class Point {

protected:
    float x, y, radius = -1;
    int canvas;

public:
    int get_x() const;
    int get_y() const;
    int get_canvas() const;
    float get_raw_x() const;
    float get_raw_y() const;
    bool border_control(int max_x, int max_y);
    pair<float, float> get_polar_coord(float c_x, float c_y);
    pair<float, float> get_change_polar_coord(float r, float angle, float
c_x, float c_y);

    void reboot_radius();
    void set_coord(float new_x, float new_y);
    void set_polar_coord(float r, float angle, float c_x, float c_y);

    Point();
    Point(int my_canvas, int s_x, int s_y);
    Point(const Point &copy);

    tuple <shared_ptr<Point>, shared_ptr<Point>, shared_ptr<Point>,
shared_ptr<Point>>
    get_extreme_points(tuple <shared_ptr<Point>, shared_ptr<Point>,
            shared_ptr<Point>, shared_ptr<Point>> data);

    void draw();
    void draw_line(Point &point);
    void move_point(float add_x, float add_y);
};

#endif
```

Файл реализации класса Point Point.cpp

```
#include <list>
#include <iterator>
#include <memory>
#include <iostream>
#include <windows.h>
#include <conio.h>
```

```cpp
#include <chrono>
#include <cmath>
#include <random>
#include "headers/import_python_func.h"
#include "headers/Point.h"

using namespace std;


int Point::get_canvas() const {
    return canvas;
}

float Point::get_raw_x() const {
    return x;
}

float Point::get_raw_y() const {
    return y;
}

void Point::reboot_radius() {
    radius = -1;
}

int Point::get_x() const {
    return (int) x;
}

int Point::get_y() const {
    return (int) y;
}

bool Point::border_control(int max_x, int max_y) {
    // Вернет true, если точка находится в пределах экрана
    return x >= 0 && y >= 0 && x <= max_x && y <= max_y;
}

pair<float, float> Point::get_polar_coord(float c_x, float c_y) {
    float r_y = y - c_y;
    float r_x = x - c_x;
    if (radius == -1) {
        radius = sqrt(r_x * r_x + r_y * r_y);
    }
    return {radius, atan2(r_y, r_x)};
}

pair<float, float> Point::get_change_polar_coord(float r, float angle, float
c_x, float c_y) {
    return {r * cos(angle) + c_x - x, r * sin(angle) + c_y - y};
}

void Point::set_polar_coord(float r, float angle, float c_x, float c_y) {
    x = r * cos(angle) + c_x;
    y = r * sin(angle) + c_y;
}

void Point::move_point(float add_x, float add_y) {
    x += add_x;
    y += add_y;
}

void Point::set_coord(float new_x, float new_y) {
```

```
    x = new_x;
    y = new_y;
}

Point::Point() {
    x = 0;
    y = 0;
}

Point::Point(int my_canvas, int s_x, int s_y) {
    canvas = my_canvas;
    x = s x;
    y = s_y;
}

Point::Point(const Point &copy) {
    canvas = copy.get_canvas();
    x = copy.get_raw_x();
    y = copy.get_raw_y();
}

tuple <shared_ptr<Point>, shared_ptr<Point>, shared_ptr<Point>,
shared_ptr<Point>>
Point::get_extreme_points(tuple <shared_ptr<Point>, shared_ptr<Point>,
shared_ptr<Point>, shared_ptr<Point>> data) {
    auto[max_x, max_y, min_x, min_y] = data;
    if (max_x->get_raw_x() <= x) max_x = make_shared<Point>(*this);
    if (max_y->get_raw_y() <= y) max_y = make_shared<Point>(*this);
    if (min_x->get_raw_x() >= x) min_x = make_shared<Point>(*this);
    if (min_y->get_raw_y() >= y) min_y = make_shared<Point>(*this);
    return {max_x, max_y, min_x, min_y};
}

void Point::draw() {
    cout << "Draw point";
    cout << get_x();
    cout << ", ";
    cout << get_y() << endl;
//            draw_point_python(canvas, x, y);
}

void Point::draw_line(Point &point) {
    new_draw_line_python(canvas, x, y, point.get_raw_x(), point.get_raw_y());
//            draw_line_python(canvas, x, y, point.get_raw_x(),
point.get_raw_y());
}
```

Приложение Б. Графическая часть

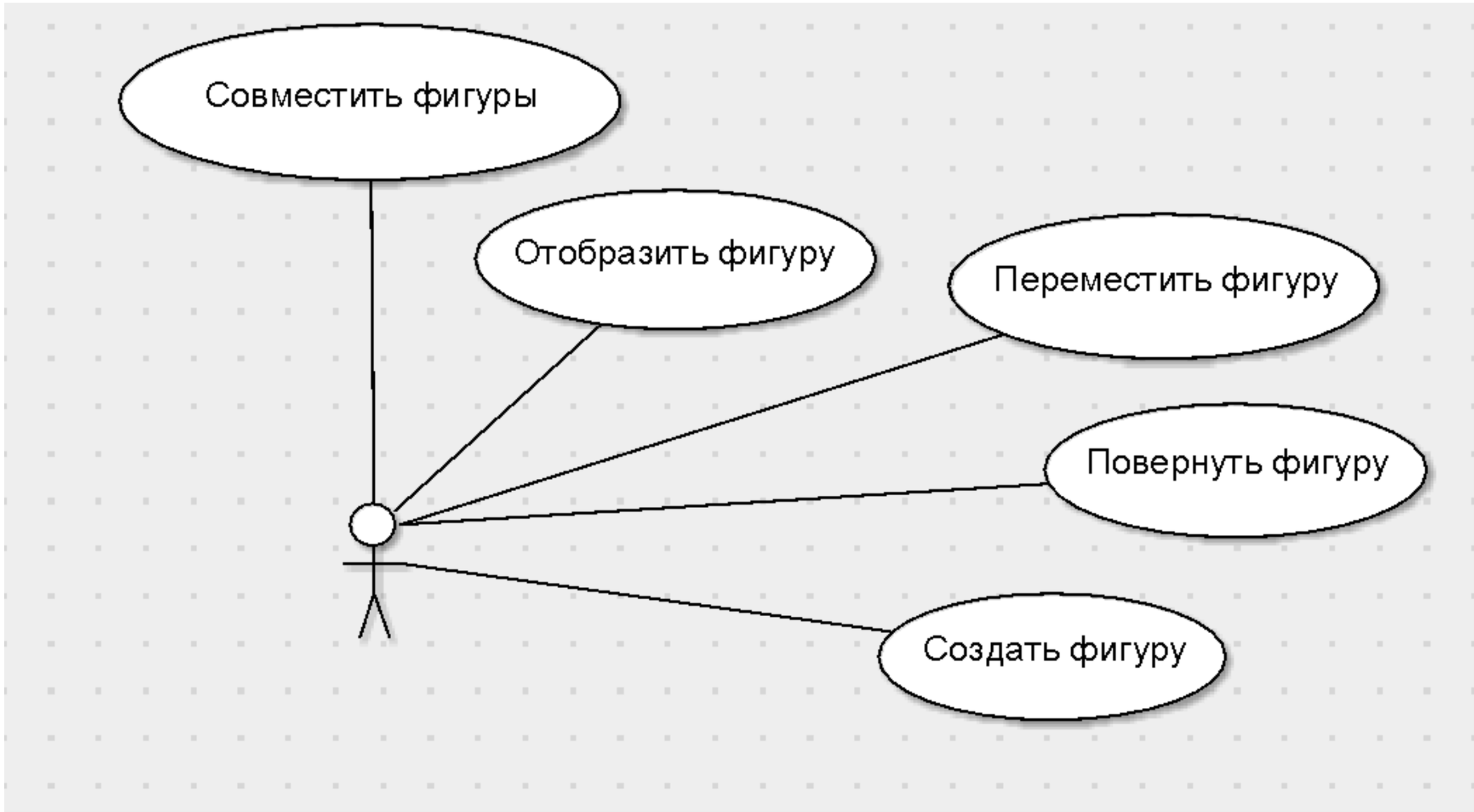

Рисунок Б1 – Диаграмма классов

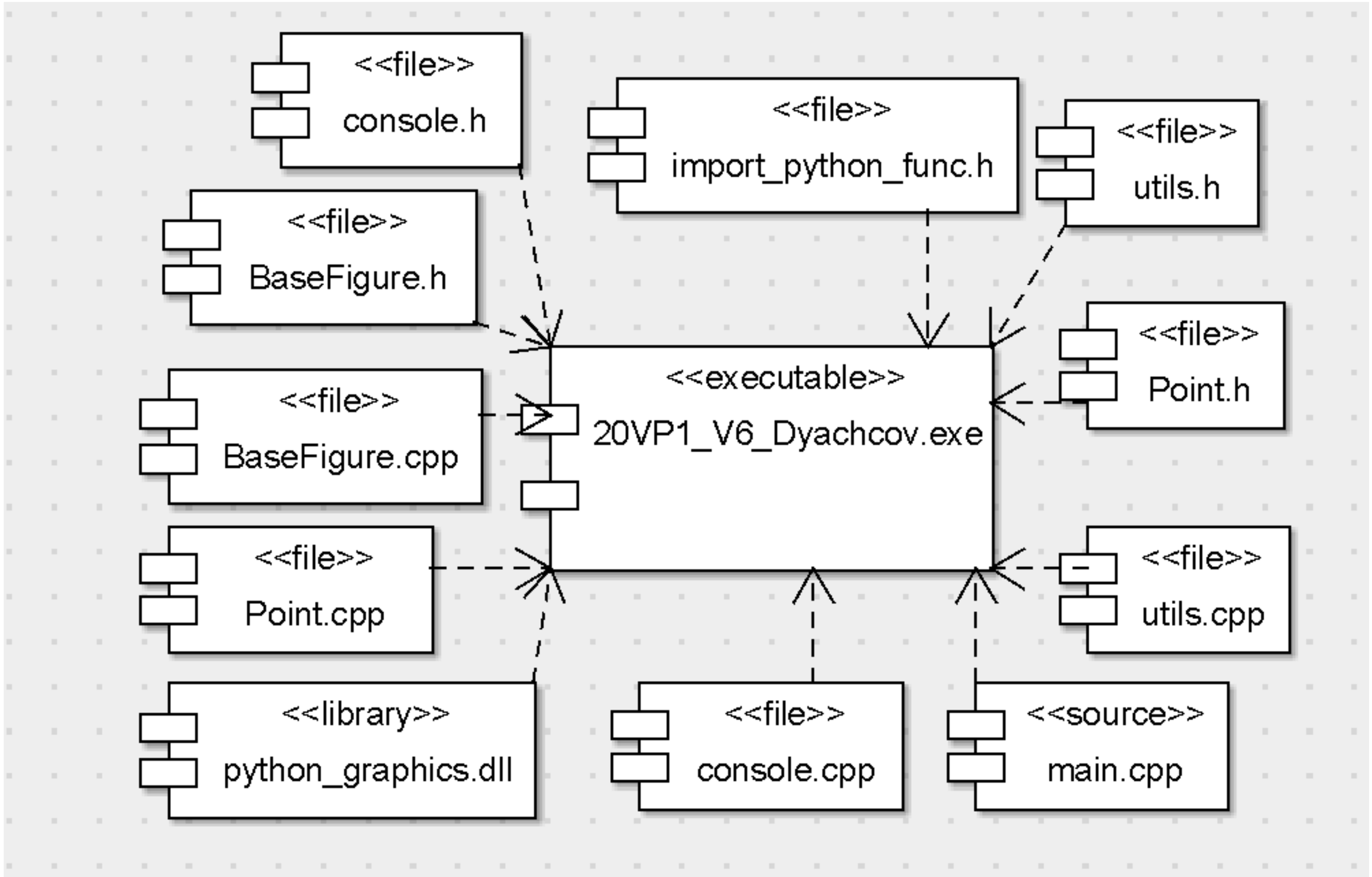

Рисунок Б2 – Диаграмма компонентом