МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Балтийский федеральный университет имени Иммануила Канта»

ОНК «Институт высоких технологий»

ОТЧЁТ О ПРОХОЖДЕНИИ

УЧЕБНОЙ ТЕХНОЛОГИЧЕСКОЙ (ПРОЕКТНО-ТЕХНОЛОГИЧЕСКОЙ) ПРАКТИКИ

на базе Высшей школы компьютерных наук и искусственного интеллекта

Выполнил Атмакин Дмитрий Михайлович

студент очной формы обучения 1 курса

направления подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

профиль обучения «Разработка баз данных и интернет-приложений»

Руководитель практики

Ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Полковский О.А.

г. Калининград 2024 г.

Оглавление

[ВВЕДЕНИЕ 3](#_Toc171374305)

[ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc171374306)

[Базовые типы данных 4](#_Toc171374307)

[Функции и методы 5](#_Toc171374308)

[Передача параметров по ссылке 6](#_Toc171374309)

[Классы 7](#_Toc171374310)

[Конструкторы класса 9](#_Toc171374311)

[CMake 9](#_Toc171374312)

[ГЛАВА 2. ЗАДАНИЕ НА ПРАКТИКУ 11](#_Toc171374313)

[Задача №1 «Камень, ножницы, бумага» 11](#_Toc171374314)

[Глава 3. Выполнение заданий на практику 12](#_Toc171374315)

[Решение задачи №1 12](#_Toc171374316)

[ЗАКЛЮЧЕНИЕ 18](#_Toc171374317)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc171374318)

[ПРИЛОЖЕНИЕ 1 20](#_Toc171374319)

**ВВЕДЕНИЕ**

Вид практики – Учебная технологическая (проектно-технологическая) практика (далее Учебная практика).

Цель учебной практики: получение первичных профессиональных умений навыков.

Задачи учебной практики:

1. Закрепление и углубление теоретических знаний в области информационных технологий;
2. Приобретение и развитие первичных профессиональных навыков и умений в области прикладной математики и информатики.

**ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**Базовые типы данных**

Тип переменной – характеристика, определяющая формат представления данных в памяти компьютера, множество допустимых значений этих данных и совокупность операций над ними. Зная тип переменной, компилятор выделит для нее необходимое количество ячеек памяти. Основные типы переменных приведены в таблице 1.

*Таблица 1*

**Типы данных в С++**

|  |  |  |
| --- | --- | --- |
| **Тип** | **Размер**  **(в байтах)** | **Диапазон** |
| unsigned char | 1 | 0 – 255 |
| char | 1 | -128 – 127 |
| unsigned short | 2 | 0 – 65535 |
| short | 2 | -32 768 – 32 767 |
| unsigned int (16 разрядов) | 2 | 0 – 65 535 |
| unsigned int (32 разряда) | 4 | 0 – 4 294 967 295 |
| int (16 разрядов) | 2 | -32 768 – 32 767 |
| int (32 разряда) | 4 | -2 147 483 648 – 2 147 483 647 |
| unsigned long | 4 | 0 – 4294967295 |
| long | 4 | -2 147 483 648 – 2 147 483 647 |
| unsigned long long (С99) | 8 | 0 – 18 446 744 073 709 551 615 |
| long long (С99) | 8 | -9 223 372 036 854 775 808 –  9 223 372 036 854 775 807 |
| float | 4 | 1.2e-38 – 3.4e+38 |
| double | 8 | 2.2e-308 – 1.7e+308 |
| long double | 10 | 1.7e-4932 – 1.7e+4932 |

**Функции и методы**

Функция определяет действия, которые выполняет программа. Они позволяют выделить набор инструкций и назначить ему имя. В будущем можно многократно по присвоенному имени вызывать в различных частях программы. Иначе говоря, функция – это именованный блок кода.

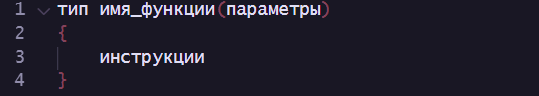


Рисунок 1. Формальное определение функции.

В начале указывается возвращаемый тип функции. Если возвращаемого значения нет, то используется тип void.

Затем идет имя функции, которое имеет произвольное обозначение. К именованию функций применяются те же правила, что и к переменным (имена начинаются с буквы английского алфавита или символа подчеркивания, дальше могут идти цифры, буквы, символ подчеркивания).

После имени идет перечисление параметров. Они могут отсутствовать, в таком случае указываются пустые скобки.

Далее следуют фигурные скобки, в которых содержится тело функции, имеющее выполняемые инструкции.

Для возвращения результата работы функции используется оператор return. Если указан любой тип функции, кроме void, то return обязательно должен возвращать какое-либо значение.

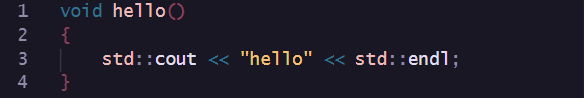


Рисунок 2. Пример функции, выводящей в консоль “hello”.

Для использования написанной функции нужно написать имя функции и аргументы в круглых скобках, если необходимо.



Рисунок 3. Вызов функции.



Рисунок 4. Пример вызова функции.

Метод – функция, которая вызывается по имени, связанному с объектом. Во многих отношениях он идентичен функции, за исключением двух отличий: методу неявно передаются данные для работы с объектом и метод способен оперировать данными, содержащимися внутри класса.

**Передача параметров по ссылке**

Аргументы, обозначающие переменные, могут передаваться в функцию по значению и по ссылке.

При передаче по значению функция получает копию значения переменной и работает с ней. Переменная, указанная в аргументах, не будет изменена.

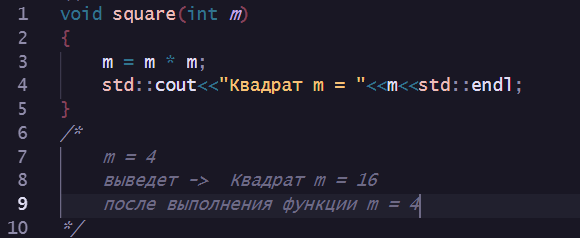


Рисунок 5. Пример передачи переменной по значению.

Для манипуляции с объектом с помощью функций используют передачу аргументов по ссылке. Ссылочный параметр связывается непосредственно с объектом, поэтому через ссылку можно менять сам объект.

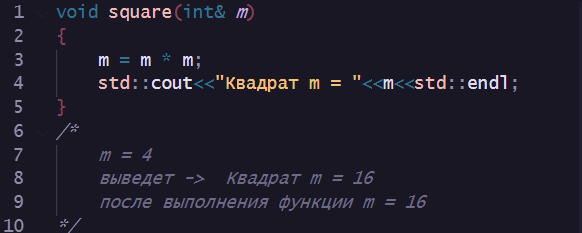


Рисунок 6. Пример передачи аргумента по ссылке.

**Классы**

Кроме использования встроенных типов, таких как int, double и так далее, мы можем определять собственные типы или классы. Класс представляет составной тип, который может использовать другие типы.

Класс предназначен для описания некоторого типа объектов. Другими словами, класс является планом объекта, а объект представляет конкретное воплощение класса, его реализацию.

Для объявления класса используют ключевое слово class, после которого идет имя класса.

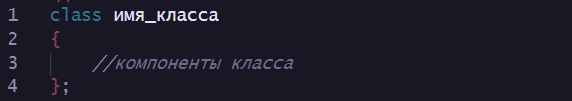


Рисунок 7. Объявление класса.

Переменные класса называют полями класса.

Существуют модификаторы доступа, которые определяют доступность компонентов класса. Их всего три: public(общий), private(закрытый), protected(защищенный). Public делает свойства и методы доступными без ограничений, private делает компоненты доступными только внутри класса, а protected дает доступ только наследникам класса.

Существуют структуры, которые мало чем отличаются от классов. Структуры обычно служат для хранения общедоступных данных в виде публичных переменных.

Для определения структуры применяется ключевое слово struct.

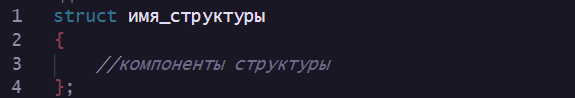


Рисунок 8. Объявление структуры.

Ключевое слово this представляет указатель на текущий объект данного класса. С его помощью можно обращаться внутри класса к любым его членам.

**Конструкторы класса**

Конструкторы – специальная функция, которая имеет то же имя, что и класс, не возвращает никакого значения, но позволяет инициализировать объект класса с полями, имеющими определенные значения. При каждом создании нового объекта класса вызывается конструктор класса.

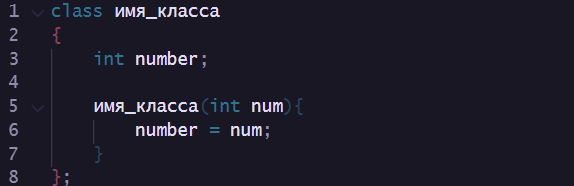


Рисунок 9. Пример конструктора класса.

У класса может быть несколько конструкторов.

**CMake**

CMake – кроссплатформенная утилита для автоматической сборки программы из исходного. Для того, чтобы собрать проект средствами CMake, необходимо в корне дерева исходников разместить файл CMakeLists.txt, хранящий правила и цели сборки.

**ГЛАВА 2. ЗАДАНИЕ НА ПРАКТИКУ**

**Задача №1 «Камень, ножницы, бумага»**

Перед началом нужно собрать информацию о текущей задачи: что и как нужно сделать? Главным вектором движения будет создание игры, в которой существа показывают разное поведение в зависимости от ситуации. К примеру, можно воспользоваться правилом игры “Камень, ножницы, бумага”, где камень бьет ножницы, ножницы бьют бумагу, бумага – камень. Соответственно, камень должен скрываться от бумаги, чтобы не быть уничтоженным или съеденным. Для большей красочности будем создавать игру, способную обработать поведение не пары сущностей, а несколько десятков.

Концептом определились, но как это визуализировать? Решением стал С++, точнее библиотека SFML, которая позволяет визуализировать все, что угодно душе.

В движении наших существ добавим капельку случайности, чтобы они могли догонять друг друга. Или убегать – зависит от удачи существ.

Но во время поиска информации возник вопрос: как лучше всего оптимизировать поведение существ, чтобы они могли видеть только тех, кто находится в радиусе их зрения? В нашем проекте планируется иметь десятки существ, что крайне замедляет работу программы – обработка сотней циклов в сотне других циклов тратит безумное количество ресурсов компьютера.

**Глава 3. Выполнение заданий на практику**

**Решение задачи №1**

Первым шагом стала разработка поведения существ – передвижение. В нашей программе будет три типа передвижения: преследование, побег, отдых. Во время преследования существо должно гнаться за своей жертвой, во время побега убегать от хищника, а во время отдыха спокойно двигаться в разные стороны.

Для своего удобства создадим класс body, в котором будем хранить методы и поля, связанные с передвижением.

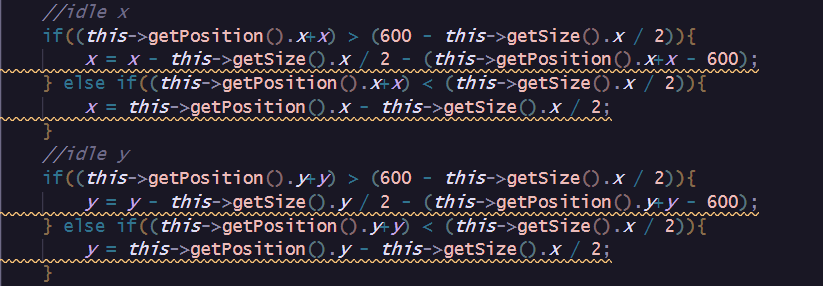


Рисунок 10. Реализация отдыха.

Благодаря SFML можно получать координаты существа относительно окна, в котором происходят действия. Так же были наложены ограничения на передвижение существ: координаты существа не должны уходить за границы программы. В нашем случае границы равны 600 (шесть сотен) для каждой стороны.

Над реализацией поведения хищника и жертвы пришлось немного подумать: куда должна убегать жертва и как должен преследовать поедатель?

С помощью относительной системы координат был найден ответ: передвижение зависит от нахождения в определенном квадранте относительно данной сущности.

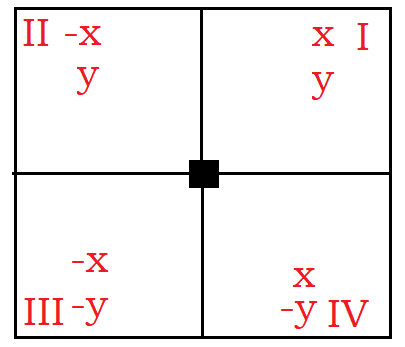


Рисунок 11. Визуализация решения

Если жертва находится в первой четверти, относительно черного квадрата, то мы сближаемся по х и у, если жертва во второй четверти, то сближаемся по у, отдаляемся по х. Инвертируем знаки для поведения жертвы.

На данном этапе нам нужно определять видимость существ для друг друга. Решение стало создать подобие коллизии: запихиваем в существо квадрат, который следит за пересечением с телом другого существа. В нашем случае создаем прямоугольник area c размерами 450, 450. И пишем метод проверки видимости.

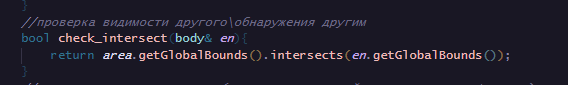


Рисунок 12. Метод проверки видимости другого существа.

Для собственного удобства создадим метод check\_type для определения типа поведения данного существа относительно другого. Все, передвижение готово.

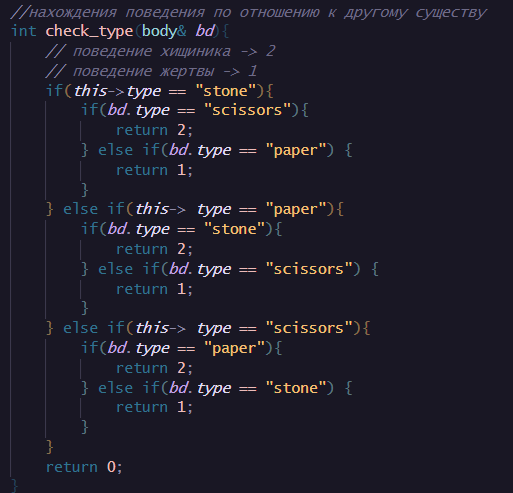


Рисунок 13. Определение типа поведения

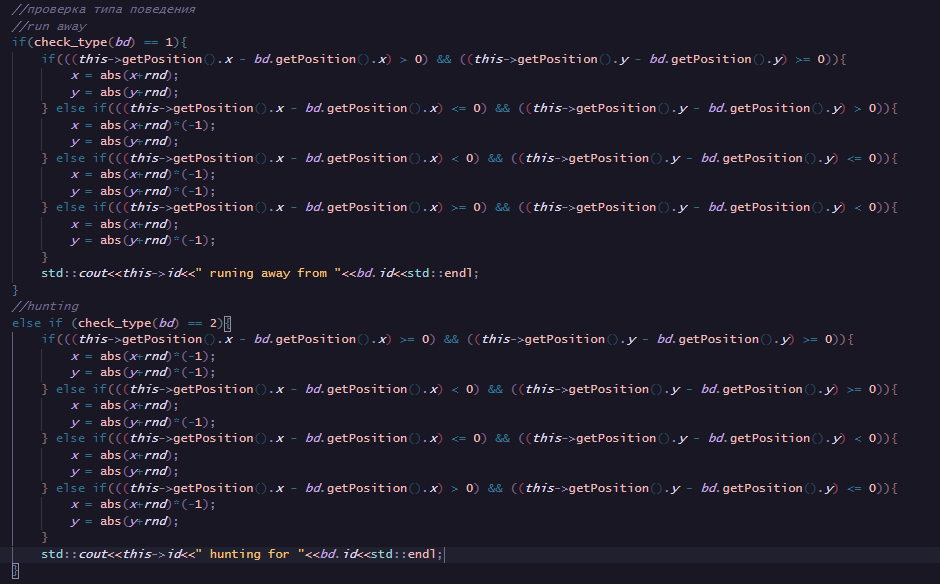


Рисунок 14. Реализация поведений типа хищник и жертва

Передвижение готово, но наша программа не готова к работе с десятками существ: одно существо может видеть только одно. Использовать проходку по всем существам с помощью цикла - не вариант. Будем решать один из главных вопросов разработки игр – оптимизация программы методом подразделения пространства. Мною был выбран способ quadtree (подразделение пространства четверным древом).

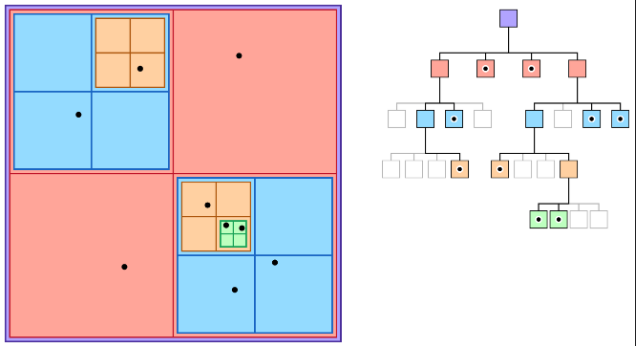


Рисунок 15. Метод quadtree

После старательного мозгового штурма были созданы rect – узел, quadtree – само древо. Во время тестирования древа возникли некоторые проблемы.

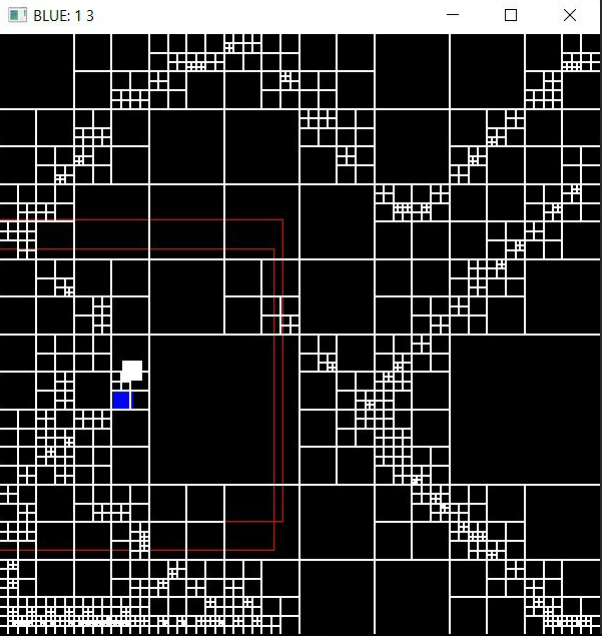


Рисунок 16. Проблема с бесконечным делением

Решение данной проблемы относительно быстро было найдено: нужно отслеживать вход и выход существа из данного узла. Если было недостаточно существ для деления узла, то он возвращался в состояние до деления. Иначе говоря, подузлы объединялись в один узел.

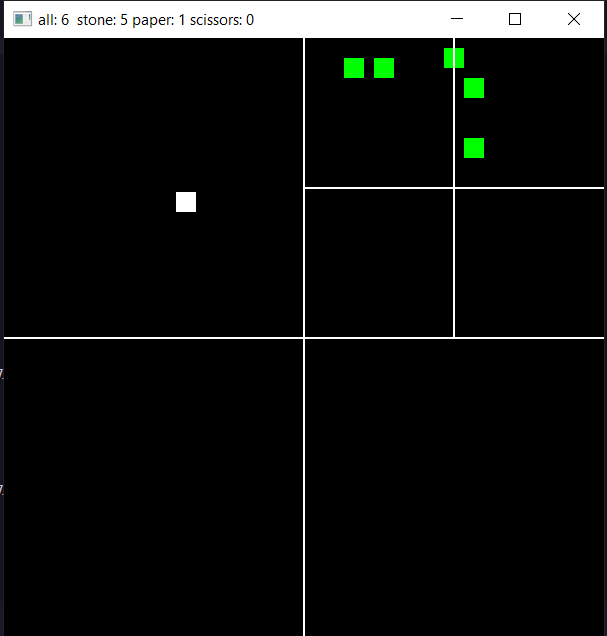


Рисунок 17. Корректная работа древа

Проблема с древом решена, проблема с движением существ решено. Осталось только добавить корректное зрение в определенных границах. Способ с коллизией лишь частично решает данную проблему.

Решением стало написание метода query(поиск) в quadtree. Query находит всех существ, находящихся в передвигаемом узле range. Теперь мы можем получать массив с видимыми существами. Правда, это частично решает нашу проблему: в этот массив попадают и существа такой же группы, что и то, относительно которого осуществляется поиск. В некоторых случаях нам нужно будет обработать 20 (двадцать) бессмысленных циклов лишь для одного существа.

Изменим условие для попадания существа в массив найденных: тип существ должен отличаться. Иначе говоря, камень не может увидеть себе подобных. Так же, если массив пуст, то будем добавлять в него само существо. Чтобы оно двигалось относительно самого себя.

Пришло время для финального тестирования. Для него мы создаем массив со всеми существами (для удобства), в который засовываем три типа существ: камень, бумага, ножницы. Во время тестирования был замечен баг, во время которого хищники, подобно падальщикам, временно крутятся вокруг своей жертвы. В итоге, баг стали воспринимать фичей, он добавляет интереса в наблюдении за действиями существ.

Так же для удобства были добавлены переменные, которыми можно управлять популяцией существ перед запуском симуляции.

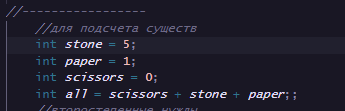


Рисунок 18. Переменные для удобства

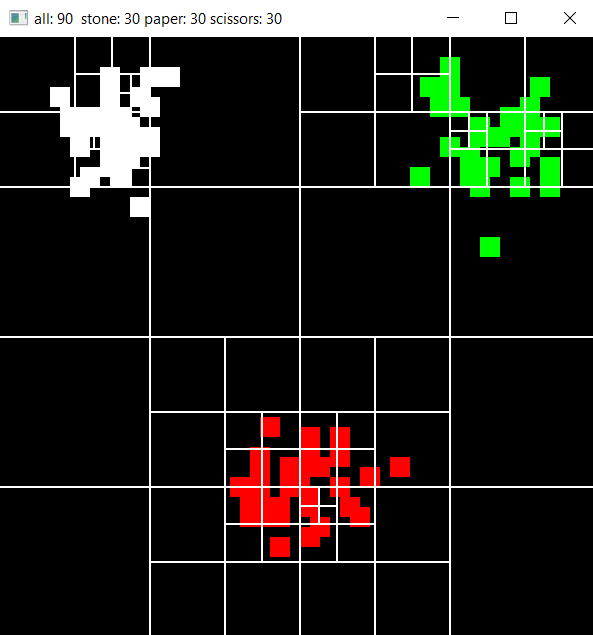


Рисунок 19. Финальное тестирование с большим количеством существ

**ЗАКЛЮЧЕНИЕ**

В ходе практики были изучены основы языка программирования С++. Задачи были направлены на закрепление теоретического материала по типам данных, арифметическим, логическим операторам. Были изучены условия и циклы. Особое внимание уделялось работе с текстом, в частности массиву char.

Предлагалось решить задачи, связанные со спортивным программированием, длинной арифметикой, битовыми операциями.

В результате практики были усовершенствованы мои компетенции, закреплены теоретические навыки.

В ходе учебной практики я научилась: выполнять поставленную задачу, следуя условию, изменять готовое решения по мере нахождения ошибок, искать альтернативные пути решения задач.

Также я освоила новую для меня среду программирования Visual Studio и работу с GitHub (создание репозиториев для выгрузки решений задач). Я научилась использовать функции и циклы, обрабатывать строки, изучила библиотеки fstream (для чтения данных из файла и ввода в него полученных результатов), iostrem (для ввода и вывода данных), vector (для работы с динамическим массивом), string (для работы со строками), выполнила работу с разными видами массивов и приемами работы с ними.

По мере прохождения учебно-технологической практики я выполнила пять задач, в которых использовала полученные знания и закрепила навыки работы с C++.

В течение практики задачи были выполнены, а цели достигнуты.

# СПИСОК ЛИТЕРАТУРЫ

**Перечень учебной литературы ресурсов сети «Интернет», необходимой для проведения практики**

1. Варфоломеева, Т. Н. Структуры данных и основные алгоритмы их обработки : учебное пособие / Т. Н. Варфоломеева. - Москва : ФЛИНТА, 2017. - 159 с. - ISBN 978-5-9765-3691-3. - Текст : электронный. - URL: https://znanium.com/catalog/product/1860018 (дата обращения: 19.01.2023). – Режим доступа: по подписке.
2. Затонский, А. В. Программирование и основы алгоритмизации. Теоретические основы и примеры реализации численных методов: учебное пособие / А.В. Затонский, Н.В. Бильфельд. — 2-е изд. — Москва: РИОР : ИНФРА-М, 2022. — 167 с. — (Высшее образование). — DOI: https: //www.dx.doi.org/10.12737/20468. - ISBN 978-5-369-01195-9. - Текст: электронный. - URL: https://znanium.com/catalog/product/1860435 (дата обращения: 16.02.2023). – Режим доступа: по подписке.

**Перечень ресурсов информационно-телекоммуникационной сети «Интернет»**

1. Основы программирования. – режим доступа: http://www.intuit.ru/studies/courses/648/504/info
2. Сайт «Структуры и алгоритмы». – режим доступа: <http://www.structur.h1.ru/>
3. Сайт с обучением С++ - режим доступа: <https://metanit.com/cpp/tutorial>
4. Форум по SFML – режим доступа: <https://en.sfml-dev.org/forums>
5. Пример готового quadtree – режим доступа: https://github.com/RamezzE/QuadTree-SFML

## ПРИЛОЖЕНИЕ 1

Код слишком большой, поэтому его можно увидеть в гитхабе.

<https://github.com/Dumaqkumaq/praktika>