МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВОЙ РАБОТЫ

по дисциплине: Объектно-ориентированное программирование
по профилю: Информатизация организаций
направления профессиональной подготовки: Прикладная информатика, бакалавриат

Тема: «Разработка компьютерной игры «Морской бой - 2»

Студент: Дмитриев Алексей Романович
Группа: ИНБО-04-18
Работа представлена к защите _____(дата)_____/_____/

Руководитель:_____

Работа допущена к защите_____(дата)_____/_____/

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /

М. МИРЭА. 2019г.

УДК 004.432.4

Дмитриев А.Р. Проектирование и создание игры «Морской бой» / Курсовая работа по дисциплине «Объектно-ориентированное программирование» профиля «Информатизация предприятия» направления профессиональной подготовки бакалавриата 09.03.03 «Прикладная информатика» (2-ой семестр) / руководитель асс. Хлебникова В.Л. / кафедра ИиППО Института ИТ РТУ МИРЭА – с. 70, табл. 1, илл. 9, ист. 5, (в т.ч. 1 на англ. яз.).

Целью данной работы является написание компьютерной игры «Морской бой» с использованием знаний в области объектно-ориентированного программирования, полученных в рамках лекционных и семинарских занятий по соответствующей дисциплине.

В рамках данной работы проведено моделирование нескольких классов, а также была создана программа на языке C++, представляющая собой реализацию компьютерной игры «Морской бой»

Dmitriev A.R. Computer game “Sea Battle” design and creating / Course work in the discipline “Object-oriented Programming” of the profile “Computerization of Enterprises” of the direction of undergraduate education 09.03.03 “Applied Informatics” (2nd semester) / Head Ass. Khlebnicova V.L. / Department of IIPPO Institute IT RTU MIREA – p. 70, tab. 1, ill 9, ist. 5, (including 1 in English Lang.).

The aim of the work is to create computer game “Sea Battle” using object-oriented programming knowledge and skills that were taken during lectures and seminars on the relevant discipline.

There were class modeling done and computer game “Sea Battle” creation completed. For this C++ programming language was used.

М. МИРЭА, Ин-т ИТ, каф. ИиППО. 2019. @ Дмитриев А.Р.

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: warerbit@mail.ru / ID: 6721090
Проверяющий: (warerbit@mail.ru / ID: 6721090)

Отчет предоставлен сервисом «Антиплагиат»- http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 1
Начало загрузки: 21.05.2019 14:55:35
Длительность загрузки: 00:00:02
Имя исходного файла: Отчет_CurseACH - копия
Размер текста: 787 кБ
Символов в тексте: 71498
Слов в тексте: 7786
Число предложений: 745

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 21.05.2019 14:55:37
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска: Модуль поиска Интернет

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
9,48%	0%	90,52%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Ссылка	Актуален на	Модуль поиска
[01]	4,16%	Перечень нормативно-технических документов, методических материалов, использованных при р...	http://megalektsii.ru	08 Июл 2017	Модуль поиска Интернет
[02]	0,61%	Техническое задание на разработку интернет-сайта структура документа	http://userdocs.ru	раньше 2011	Модуль поиска Интернет
[03]	1,37%	Министерство образования и науки Российской Федерации УДК 62-83::621.313.3 ГРНТИ 45.41.31 50....	http://netess.ru	04 Дек 2016	Модуль поиска Интернет

Еще источников: 5
Еще заимствований: 3,33%

ОТЗЫВ РУКОВОДИТЕЛЯ КУРСОВОЙ РАБОТЫ
ПО КАФЕДРЕ ИиППО ИНСТИТУТА ИТ МИРЭА

Настоящий отзыв составлен руководителем КР студента Дмитриева Алексея Романовича ИНБО-04-18 18И0361 по итогам выполнения им на кафедре ИиППО института ИТ Курсовой Работы по дисциплине Объектно-ориентированное программирование во 2 семестре 1 курса обучения в бакалавриаете на основании ФГОС ВО и действующих в РТУ МИРЭА нормативных актов: «Инструкция по организации и проведению курсового проектирования (СМКО МИРЭА 7.5.1/04.И.05)»; «Временное положение о текущем контроле успеваемости и промежуточной аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры (Положение. СМКО МИРЭА 7.5.1/03.П.08)»; «Рекомендации по оформлению письменных работ обучающимися (СМКО МИРЭА 7.5.1/03.П.69)»; а также Рабочей программы поименованной дисциплины.

Руководителем КР отмечается, что задание на курсовую работу выдано в первые две недели семестра. В это же время со студентом проведено собеседование по вопросам организации, учебно-трудовой дисциплины, графика работ и взаимодействия с руководителем, поставлены, обеспечены информационными ресурсами и обсуждены цель, основная задача, содержание, технологии, информационно-методическое обеспечение, формы и сущности отчётности и её защиты под дифференцированную аттестацию. Проведён инструктаж по соблюдению требований охраны труда, техники безопасности (ТБ), а также правилам внутреннего трудового (учебного) распорядка (ПВР).

За студентом закреплен тьютор-стажёр из числа актива подразделения СНК при кафедре ИиППО, выход в кафедральный тематический мини-портал Кабинета дипломного проектирования и магистерской подготовки при кафедре ИиППО (далее: «Кабинет»); в портале создана соответствующая директория курсовой работы, в распоряжение проектанта предоставлены стартовые информационные консисты тематических библиотек портала, выданы методические указания и типовые шаблоны документального оформления пояснительной записки (ПЗ), относящиеся к конкретной дисциплине – в совокупности позволяющие эффективно использовать средства Онтонет управления образовательным макромедиа контентом, а также всемирных технологий WorkBooks.

Обозначено и реализовано обретение обучающимся профессиональных компетенций, а именно ОПК-2, ПК-8.

Отчётные материалы студента в целом отвечают заданной теме, поставленной задаче, требованиям по оформлению. Предложенные решения обладают полнотой и качественной глубиной раскрытия

изучаемых вопросов, демонстрируя признаки новизны, креативности и меры самостоятельности, а также проявление проектантом дисциплинированности, элементов профессиональной этики, умения работать в коллективе.

Качество, полнота и технико-эстетический уровень оформления ПЗ удовлетворяют типовым требованиям. Проверенная руководителем на работоспособность, соответствие и незашумлённость электронная версия ПЗ в форматах Word и Pdf удовлетворительна.

По итогам курсовой работы, представления ПЗ руководитель согласно перечисленным выше нормативам подводит следующие итоги:

1. Качественная сторона выполненных работ - _____*(отл., хор., удовл., неудовл.)*.
2. Количественная сторона (характеризующая охват, объём, детализацию и т.п.) выполненных работ _____ *(отл., хор., удовл., неудовл.)*.
3. Содержание планово-отчётной документации проектанта_____ *(отл., хор., удовл., неудовл.)*.
4. Качество оформления документации на бумажном носителе и в электронной форме (в том числе, соблюдение требований стандартов и упомянутых выше нормативов) _____*(отл., хор., удовл., неудовл.)*.
5. Учебно-производственная дисциплина, включая соблюдение установленных сроков выполнения, отчётности и аттестаций, а также норм учебно-профессиональной этики _____*(отл., хор., удовл., неудовл.)*.

Итоговая оценка, рекомендованная руководителем:_____ *(отл., хор., удовл., неудовл.).*

Таким образом, курсовая работа успешно и в срок завершено. Курсовая работа рекомендована к защите.

Руководитель:_____(должность, ф.и.о., дата)

Прим.: «Кабинет» каф. ИиППО м-лы КР на хранение принял»
Подпись отв. лица._____ Должность. Дата.

Оглавление

1. ВВЕДЕНИЕ

Целями данной курсовой работы являются:

- приобретение практических навыков по проектированию, поэтапной разработке распределённых объектно-ориентированных программ, а также их тестированию и документированию;
- закрепление навыков разработки собственных классов, создания многофайловых проектов.

Для достижения поставленных целей необходимо решить данные задачи:

- разработать техническое задание на программный продукт;
- изучить и научиться применять принципы поэтапной разработки и отладки программ средней сложности;
- проанализировать предметную область и разработать собственные классы и алгоритмы;
- разработать интерфейс программы;
- разработать тесты для проверки работоспособности программного продукта;
- разработать требуемую документацию на программный продукт.

Программная часть данной курсовой работы составлялась на объектно-ориентированном языке C++ в среде разработки Visual Studio 2017.

Основная часть данного отчёта разбита на три части:

- Теоретическое введение
- Проектирование системы
- Тестирование системы

2. ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

2.1. Основные понятия объектно-ориентированного программирования

Объектно-ориентированное программирование – это подход к проектированию и разработке программных продуктов, в основе которого лежат такие понятия как объекты и классы. Класс является сложным типом данных, который описывает состояние и поведение некоторой сущности или сущностей. Каждый класс состоит из набора данных (полей класса, переменных) и функций (методов класса), предназначенных для работы с этими полями. Объект же является конкретным представителем данного класса, имеющим определённое поведение и состояние в каждый момент времени, которые в точности определены классом, к которому они принадлежат.

2.2. Основные принципы объектно-ориентированного программирования

В объектно-ориентированном программировании имеется набор определённых принципов, которые должны использоваться при проектировании и разработке программного продукта.

Основным принципом ООП является абстрагирование. Абстрагированием является исключение из проектируемой модели всех характеристик, которые не являются значимыми, и добавление в нее всех необходимых. В соответствии с этим принципом разработчик программного продукта имеет право выбирать, какие свойства и действия описываемой сущности ему требуется переносить в разрабатываемый класс. От того, насколько хорошо будет совершён переход к абстракциям, зависит качество всей спроектированной модели, а также качество написанного вследствие кода программного решения. При хорошем переходе можно будет выделить общие части различных классов и сильно упростить написание кода ещё на этапе проектирования классов. В случае

неудачного перехода к абстракциям у разработчика программного решения может получиться весьма сложный и громоздкий код, сложный для понимания и, как следствие, для дальнейшей поддержки не только сторонними разработчиками, но и тем, кто написал данный код.

Следующим базовым принципом ООП является наследование. Этот принцип сильно упрощает написание кода разработчику программного продукта, позволяя объединить совпадающие поля и методы нескольких различных классов в отдельный класс, от которого в дальнейшем и будут наследоваться эти классы. При реализации наследования в программном продукте класс-наследник получает абсолютно все поля и методы класса-родителя. Некоторые языки программирования позволяют классу-наследнику унаследовать поля и методы сразу нескольких классов-родителей.

Одним из основополагающих принципов ООП является инкапсуляция. Инкапсуляция позволяет скрывать от пользователя методы и поля проектируемого класса. Таким образом, в рамках этого принципа в классе будут иметься поля и методы доступные для изменения или выполнения соответственно только внутри класса, в то время как другие будут доступны извне класса.

В объектно-ориентированном программировании присутствует такое явление как интерфейс класса. На основании общепринятых стандартов программирования все поля класса должны быть закрытыми от прямого изменения пользователем во избежание каких-либо ошибок в работе программного продукта. Именно здесь и появляется интерфейс класса. Интерфейс класса - это определённый набор функций, посредствам которых пользователь может влиять на изменение полей класса, или по-другому все методы открытые для внешнего доступа.

Последним принципом объектно-ориентированного программирования является полиморфизм. Полиморфизм идёт рядом с наследованием. Это свойство позволяет переопределять методы (то есть

создать в классе-наследнике метод с той же сигнатурой, что и в родительском классе, но, выполняющий другие действия). На принципе полиморфизма основываются такие механизмы как интерфейс и абстрактный класс. Абстрактный класс - это класс, в котором не определён хотя бы один метод. Интерфейс же показывает, как должен выглядеть класс-наследник. В интерфейсе все методы не являются определёнными, и должны быть переопределены при имплементировании интерфейса.

2.3. Объектно-ориентированное программирование в языке C++

Одним из главных нововведений в языке C++ в отличие от C стала поддержка объектно-ориентированного программирования. В C++ соблюдаются все принципы ООП, но при этом есть определённые особенности их использования.

Для создания класса в языке программирования C++ используется ключевое слово class, вслед за которым идёт название класса. После имени класса следуют фигурные скобки, в которых уже прописываются поля и методы класса. При этом в C++ принято разделять класс на два файла. В первый файл с расширением .h принято помещать описание полей класса, подключаемые библиотеки и прототипы методов класса, а во второй с расширением .cpp - реализации методов.

Принцип инкапсуляции в языке программирования C++ обеспечивается с помощью ключевых слов public, protected, private. Они являются модификаторами доступа к элементам класса. По умолчанию, если не указан какой-либо модификатор, поле или метод, обозначенные в классе, имеют модификатор private. Это означает, что к данному элементу класса нельзя обратиться извне класса и что он может использоваться и изменяться только внутри данного класса. Модификатор public позволяет вызывать методы класса и изменять поля из любого места программы. Модификатор доступа protected при создании объекта данного класса ничем не отличается от private. Все его особенности проявляются при наследовании от данного класса.

В языке программирования C++ так же присутствует такой механизм, как абстрактные классы. Для того, чтобы создать такой класс, разработчик должен добавить в данный класс хотя бы один полностью виртуальный метод класса. Такой метод создается с помощью ключевого слова virtual перед прототипом метода и =0 после прототипа. В языке C++, в отличие от некоторых других, нельзя создавать объект абстрактного класса, но при этом можно создавать указатели на объекты абстрактных классов. Абстрактные классы чаще всего используются как родительские классы. Для того, чтобы можно было создать объект класса-наследника от абстрактного класса, требуется определить все полностью виртуальные функции, которые есть в классе родителе.

В языке программирования C++ существует несколько способов наследовать классы. Они отличаются тем, какие поля и методы будут наследоваться и какие модификаторы доступа будут они иметь после наследования. Более подробная информация об этом приведена в таблице (табл. 1).

Таблица 1 - Модификаторы доступа при различных видах наследования

		Исходный модификатор доступа		
		public	private	protected
Модификатор доступа наследования	public	public	private	protected
	private	private	private	private
	protected	protected	private	protected

Принцип полиморфизма реализован в языке программирования C++ без особых нюансов. При наследовании, если в классе-родителе существует поле, которое инициализируется при объявлении, в классе-наследнике можно проинициализировать его по-другому, написав его определение в полях класса-наследника и проинициализировав, если это требуется для решения поставленной задачи. Все методы, унаследованные от родительского класса, в классе-наследнике можно то же переопределить. Для этого требуется переписать прототип метода в заголовочный файл данного класса и написать новый алгоритм в файле реализаций. В языке программирования C++ при наследовании можно также расширять методы родительского класса. Это значит, что программа будет выполнять сначала код, написаный в реализации класса-родителя, а потом - класса-наследника.

3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1. Уточнение задания

Основная задача, которую надо выполнить в рамках данной курсовой работы - написать игру «Морской бой», в которой пользователь будет играть против компьютера, который будет анализировать свои предыдущие ходы.

Правила игры:

- У каждого игрока есть 10 кораблей: 4 однопалубных, 3 двухпалубных, 2 трехпалубных, 1 четырехпалубный, а также свое поле 10x10 клеток (каждая палуба – 1 клетка), на котором он должен расставить свои корабли.
- Для того, чтобы начать игру, все корабли должны быть расставлены на полях так, чтобы никакие из них не соприкасались гранями или углами.
- После того, как все корабли расставлены, начинается игра. Цель игры – убить все корабли соперника. Ходы делаются поочередно, первый ход делает пользователь.
- Ход заключается в том, что игрок должен «выстрелить» в одну из 100 клеток на поле соперника. Если в клетке, в которую «выстрелил» игрок, нет палубы какого-либо корабля, то игроку выводится сообщение о том, что он «не попал», а право хода переходит к сопернику. Если же в клетке оказывается палуба какого-либо корабля, то игроку выводится сообщение о том, что он «попал» или «убил», и ход этого игрока продолжается. Корабль считается убитым, если игрок попал во все его палубы.
- Игра заканчивается, когда все корабли одного из игроков убиты.

Таким образом пользователь должен иметь возможность расставить свои корабли на поле 10x10, а программа должна контролировать правильность расстановки кораблей. После того, как пользователь

расставит все свои корабли, он должен будет нажать на кнопку «Start». Когда начнется игра, появится поле компьютера, на котором корабли расставлены случайно(соблюдая правила расстановки), а так же появятся поля, на которых после каждого хода игрока, будут выводится информационные сообщения.

Так же компьютер будет анализировать ситуацию на поле: если он попадет по кораблю, у которого больше 1 палубы, то следующими ходами он будет стараться убить этот корабль.

После победы или проигрыша пользователя будет открываться окно с соответствующей информацией, в этом окне будет присутствовать кнопка «Restart», которая позволит пользователю сыграть заново, без выключения и повторного запуска игры.

Для удобного пользовательского ввода данных было решено использовать графическую библиотеку SFML для создания графического интерфейса.

3.2. Проектирование классов

Для данной игры было разработано четыре класса.

3.2.1. Класс Pos

Данный класс представляет собой координату.

В нём имеются поля, представляющие собой:

1) Координата x (int x).
2) Координата y (int y).

Этот класс содержит в себе следующие методы:

Pos() - базовый конструктор, создающий координату (0,0).

Pos(int,int) – конструктор, создающий координату (x,y).

setPos(int,int) – меняет значение координаты на (x,y).

3.2.2. Класс Ship

Этот класс представляет собой корабль.

В нём имеются поля, представляющие собой:

1) Количество палуб, в которые еще не попали (int heads).
2) Количество палуб (int maxHeads).
3) Состояние корабля «подбит» (bool isDamaged).
4) Состояние корабля «жив» (bool isAlive).
5) Координаты палуб корабля (vector<Pos> posShip).
6) Координаты клеток, которые находятся рядом с кораблем (vector<Pos> shipSurroundings).

Этот класс содержит в себе следующие методы:

Ship() - конструктор, создающий корабль без палуб, в состоянии «не подбит» и «жив».

setHeads(int) – задает количество палуб.

setPos(Pos,bool) – задает координаты корабля и окружающих его клеток, в зависимости от того, как стоит корабль(вертикально или горизонтально).

getHeads() – получить количество палуб, которые еще не подбили.

getMaxHeads() – получить изначальное количество палуб.

Damage() – подбить или убить корабль.

3.2.3. Класс Player

Класс игрока, который используется во время самой игры.

В нём имеются поля, представляющие собой:

1) Количество живых кораблей(int numberOfShips).
2) Корабли игрока (vector<Ship> pships).
3) Ход игрока (bool turn).

Этот класс содержит в себе следующие методы:

Player() – конструктор, который создает игрока с 10 не расставленными кораблями.

setShip(Pos,int,bool) – вызывает методы класса Ship setHeads и setPos в зависимости от количества палуб.

killShip() – убивает 1 корабль.

getNumberOfShips() – получить количество кораблей, которые еще живы.

3.2.4. Класс TheGame

Данный класс представляет собой основное управление игрой. В нём принимаются решения о создании или закрытии того или иного окна.

В нём имеются поля, представляющие собой:

1) Игрок-пользователь (Player player).
2) Игрок-компьютер (Player playerCPU).
3) Корабли компьютера (vector<FloatRect> cpuShips).
4) Корабли пользователя (vector<Sprite> ships).
5) Поле игрока (Sprite plain1).
6) Поле компьютера (Sprite plain2).
7) Текстура «красного крестика» (Texture markText).
8) Текстура корабля (Texture shipText).
9) Главная текстура (Texture Text).
10) Главная текстура информации (Texture TextInformation).
11) Изображение «красного крестика» (Image markImg).
12) Поворот корабля (vector<bool> isRotated).
13) Можно ли повернуть корабль (vector<bool> canRotate).
14) Поле, отвечающее за продолжение игры (bool goOn).
15) Победа или поражение (bool win).
16) Сыграть снова (bool restart).

Этот класс содержит в себе следующие методы:

TheGame() – задает основным полям начальные значения.

Texturising() – создает основные спрайты.

StageOne() – Первый этап: расстановка кораблей пользователем.

StageTwo() – Второй этап: запись координат кораблей в соответствии с тем, как расставил корабли пользователь. Расстановка кораблей компьютера и запись их координат.

StageThree – Третий этап: Сама игра.

StageFinal – Последний этап: Окно с результатом игры(победа или поражение) и кнопка «Restart».

3.3. Диаграмма классов:

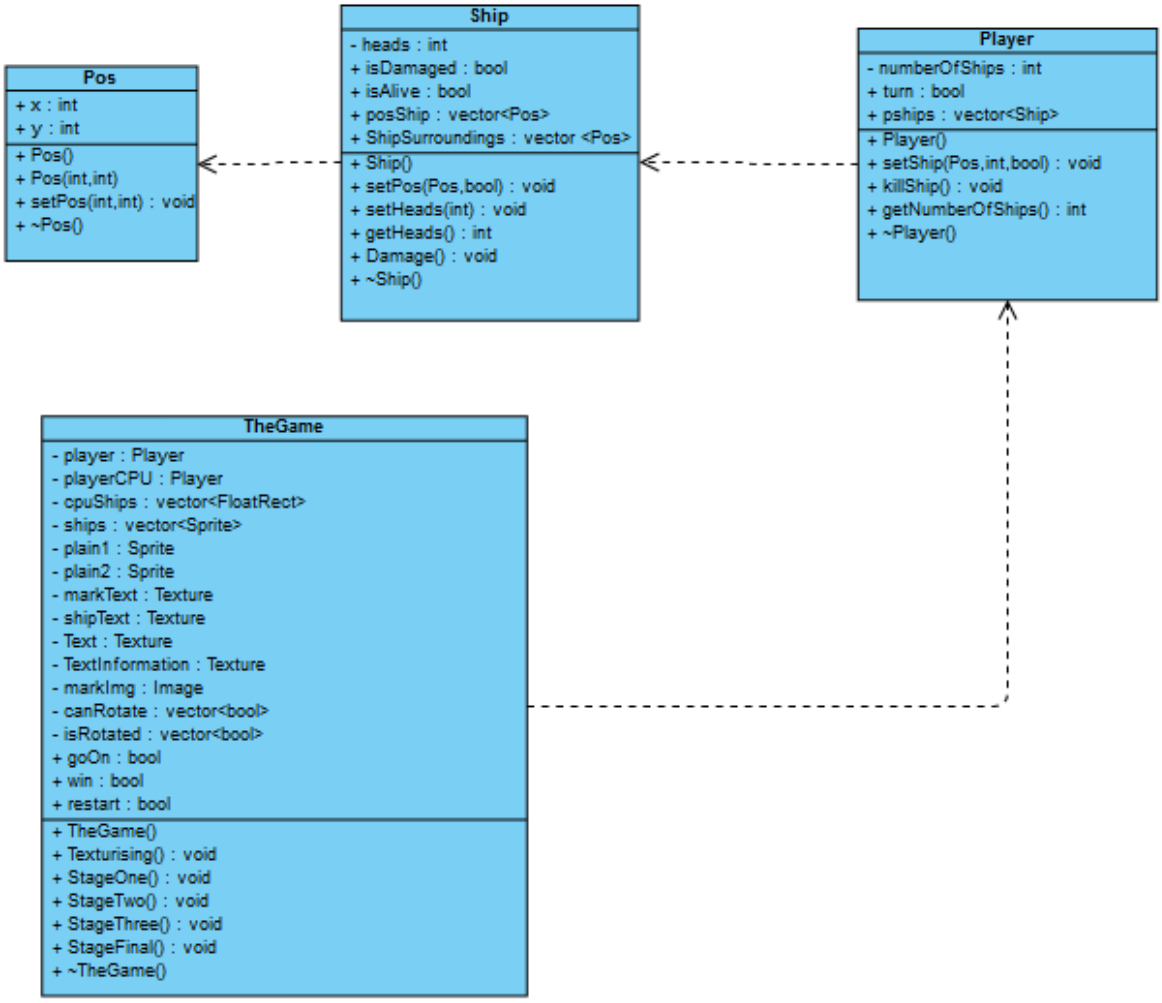

Рис. 3.1. UML диаграмма классов

4. ТЕСТИРОВАНИЕ СИСТЕМЫ

После написания программы был создан набор тестов для проверки правильности работы программы.

4.1. Тест №1

Проверка на правильность расстановки.

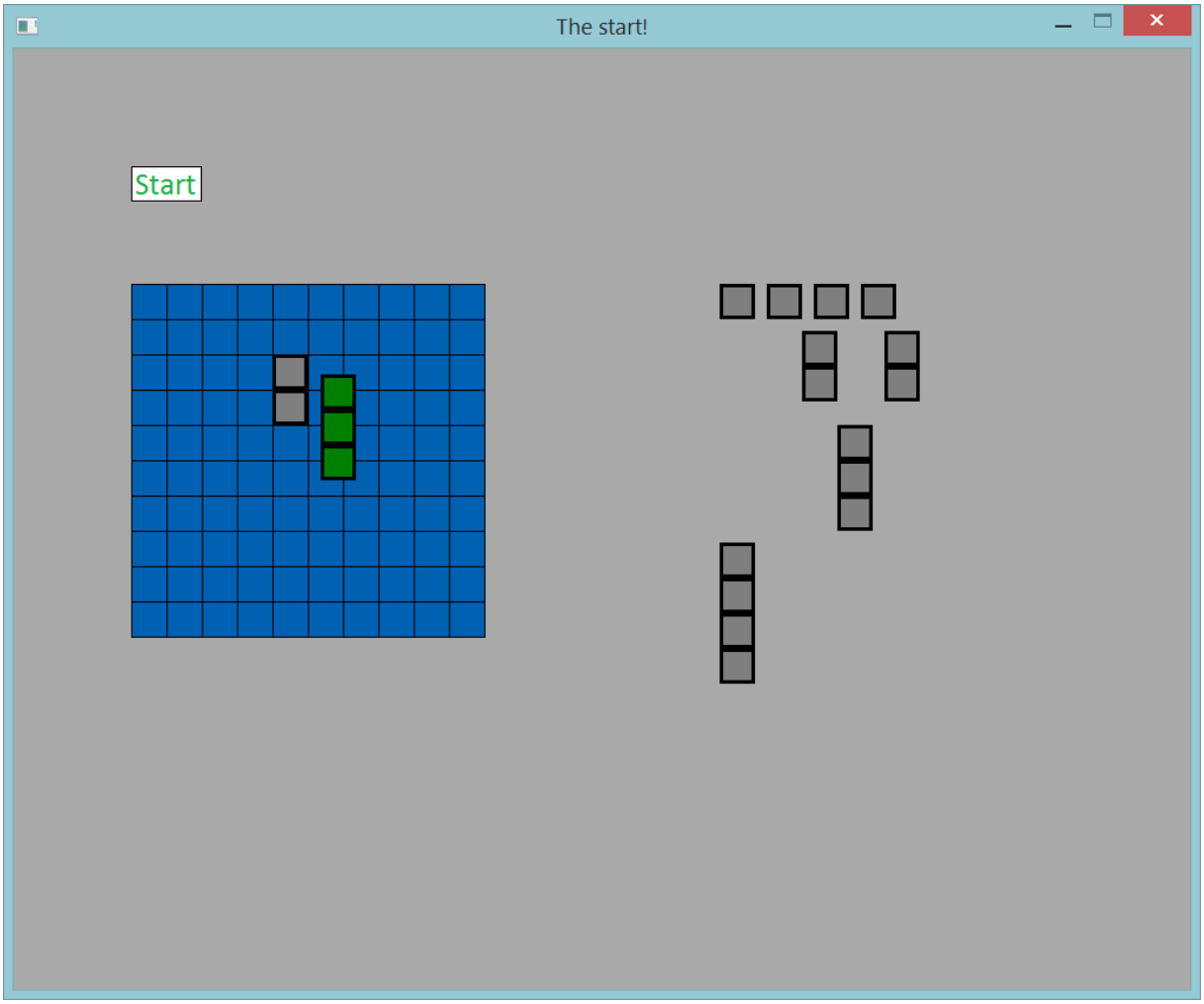

Рис. 4.1.1. Попытка поставить корабли рядом

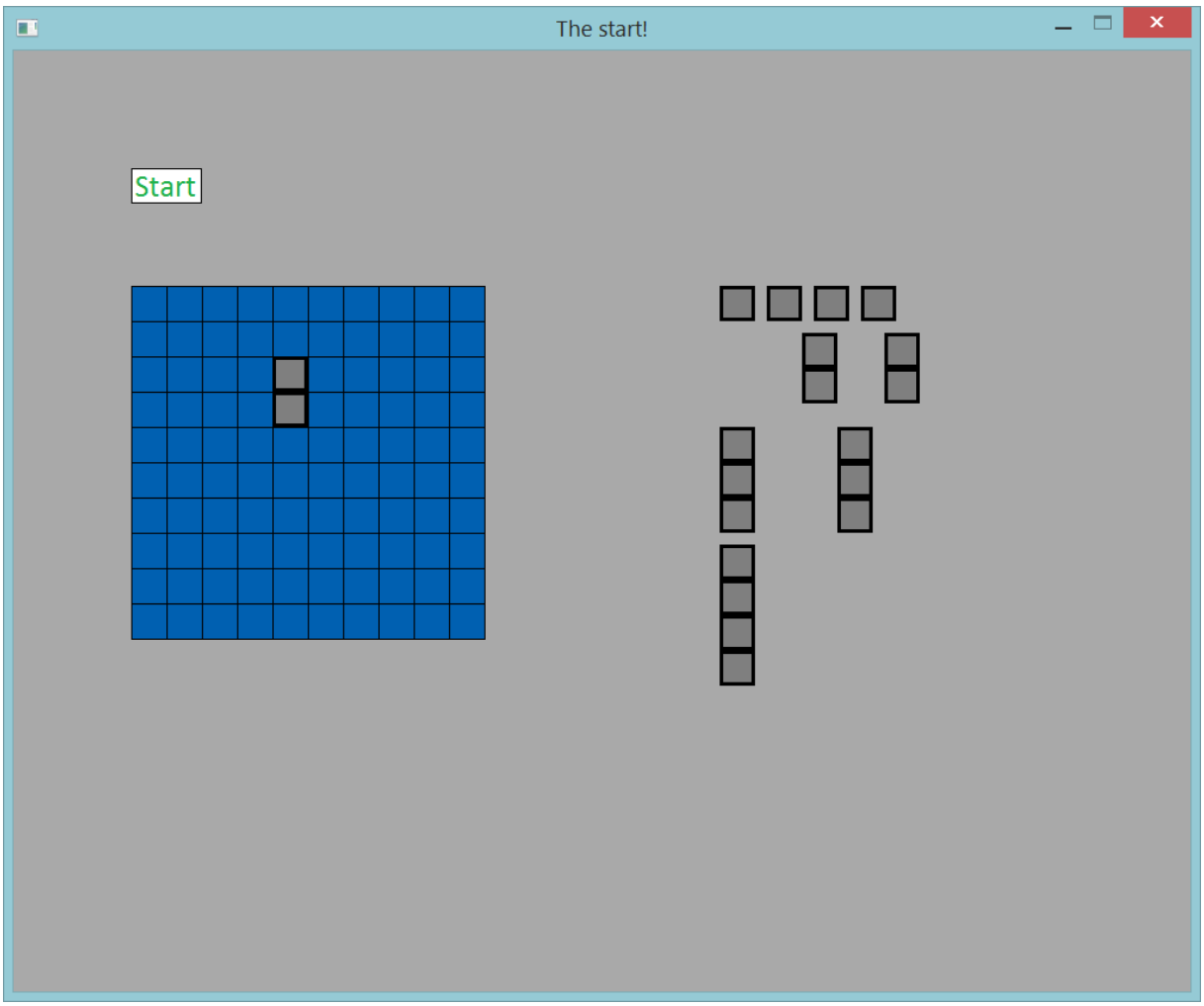

Рис. 4.1.2. Результат

При попытке поставить корабль рядом с другим кораблем, программа не дала этого сделать, поставив корабль, который хотели поставить неправильно, на его изначальное место.

4.2. Тест №2

Информационные сообщения.

На изображении ниже (Рис. 4.2.1) мы видим основное окно игры. Индикаторы рядом с надписями “YOU” и “BOT” горят зеленым и красным цветами соответственно, что говорит нам о том, что сейчас наш ход, а не компьютера.

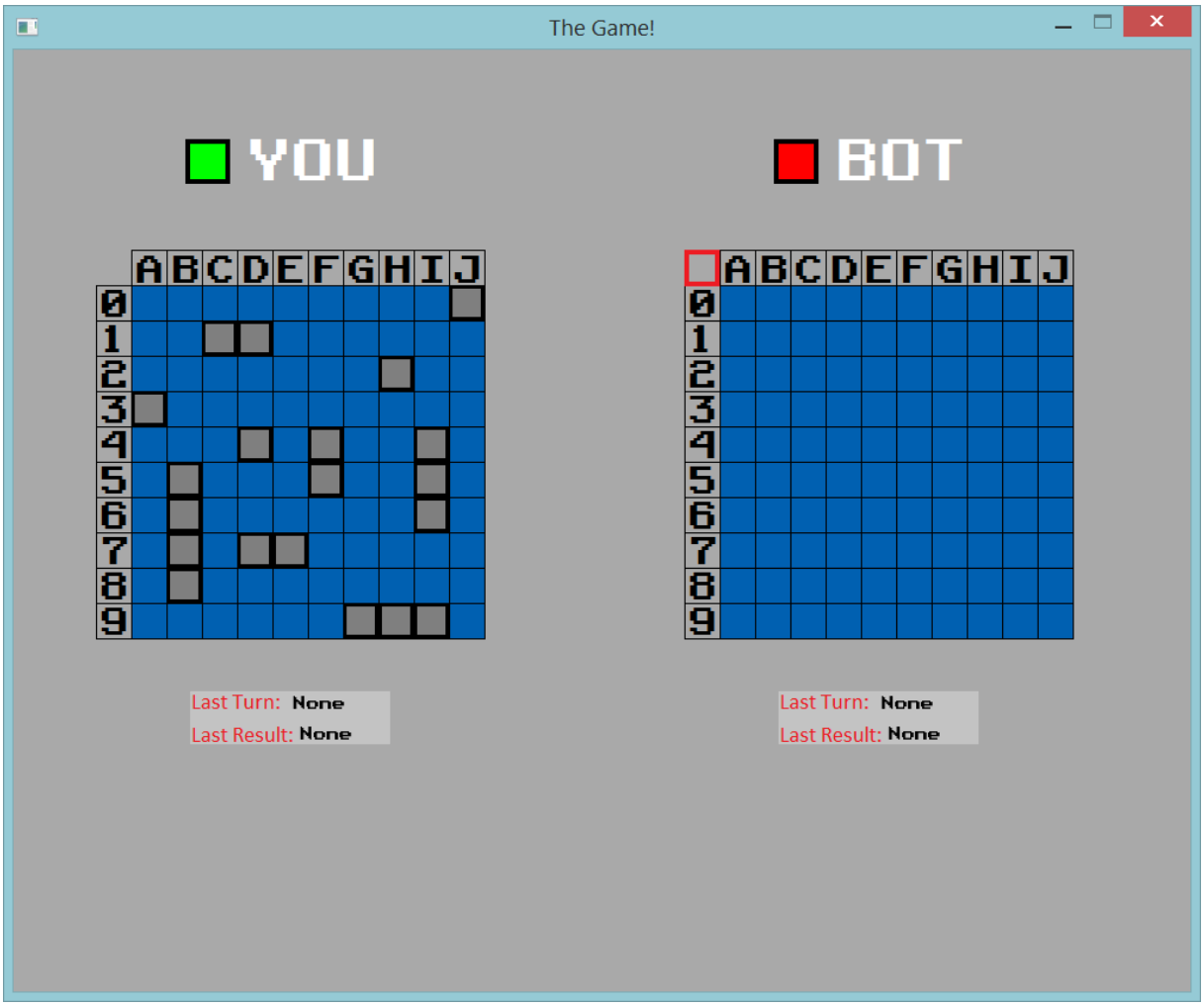

Рис. 4.2.1. Главное окно игры

Попробуем выстрелить в клетку 0A.

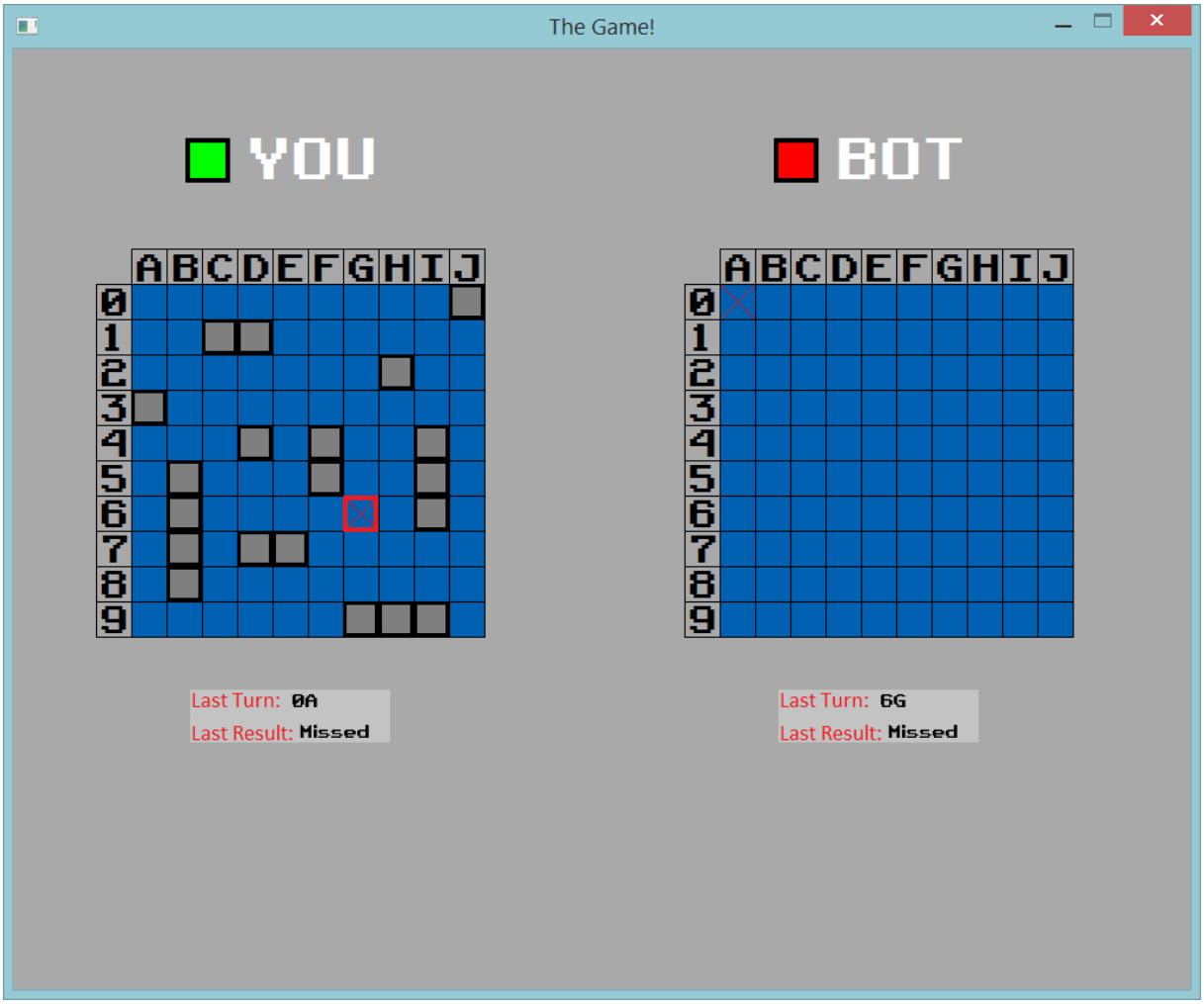

Рис. 4.2.2 Результат

На изображении (Рис. 4.2.2) мы видим, что на нашей стороне окна в строке “Last Turn” появилось сообщение о том, что последний наш ход был совершен в точку 0A, а в строке “Last Result” программа сообщила нам о том, что мы не попали. А так же после хода компьютера, мы видим, что на его стороне окна в тех же строках сообщения о том, куда он выстрелил и с каким результатом (Так же, для удобства пользователя, на поле отмечен последний ход компьютера красной рамкой).

5. РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ СИСТЕМЫ

1) Начальное окно (Рис. 5.1.1).
   1 – Кнопка для начала игры.
   Чтобы начать игру, наведитесь курсором на кнопку и нажмите «Левую кнопку мыши». Данную кнопку можно нажать только тогда, когда все корабли расставлены на поле.
   2 – Поле пользователя.
   3 – Корабли.
   Для того чтобы поставить корабль на поле, наведитесь курсором на корабль, который хотите поставить, зажмите «Левую кнопку мыши» и затем, не отпуская кнопку, передвиньте курсор туда, куда хотите поставить корабль. На изображении ниже (Рис. 5.1.2.) показано, как должен располагаться корабль относительно клеток поля. Чтобы повернуть корабль, наведитесь курсором на корабль, который хотите повернуть и нажмите «Правую кнопку мыши».

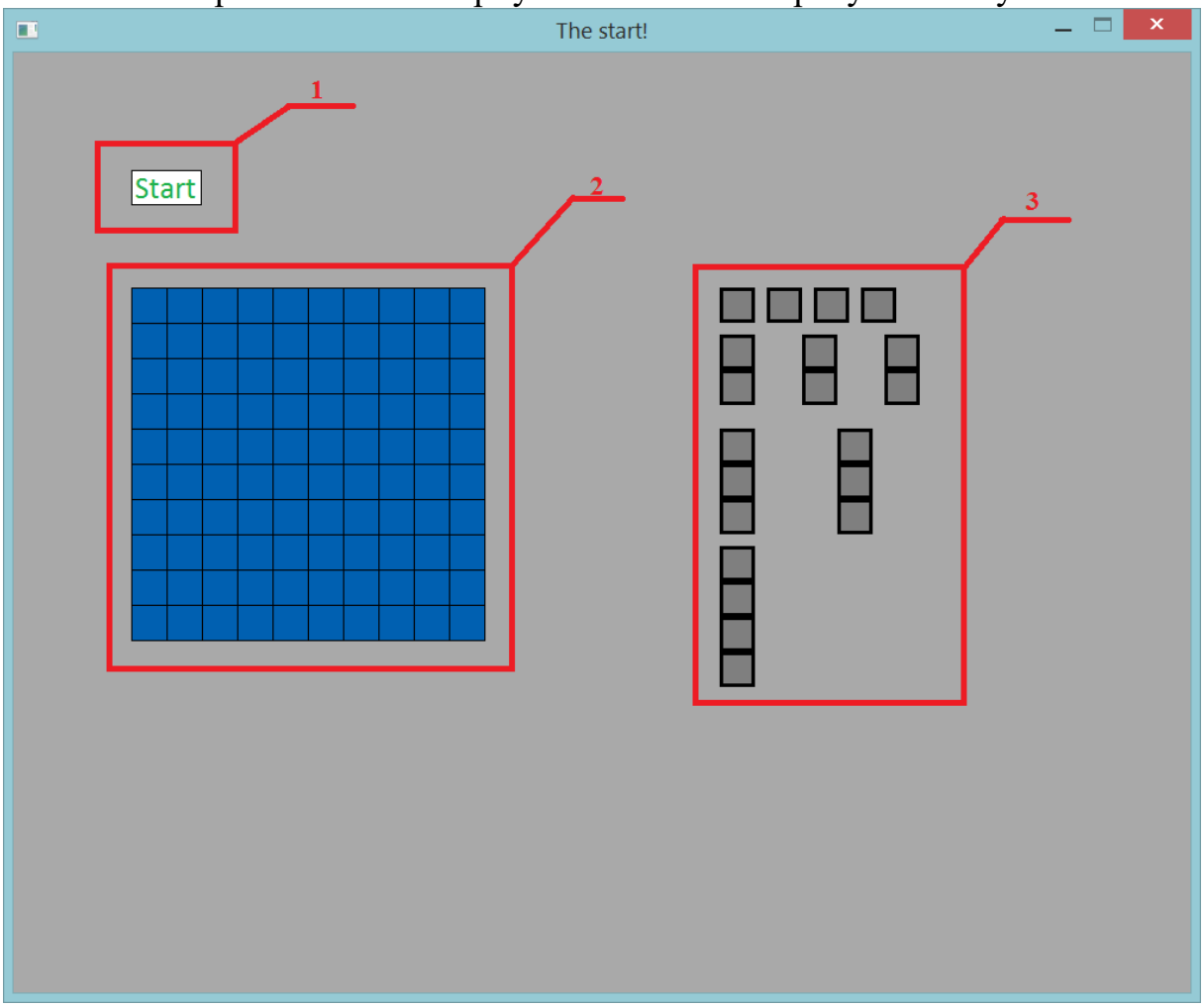

Рис. 5.1.1.

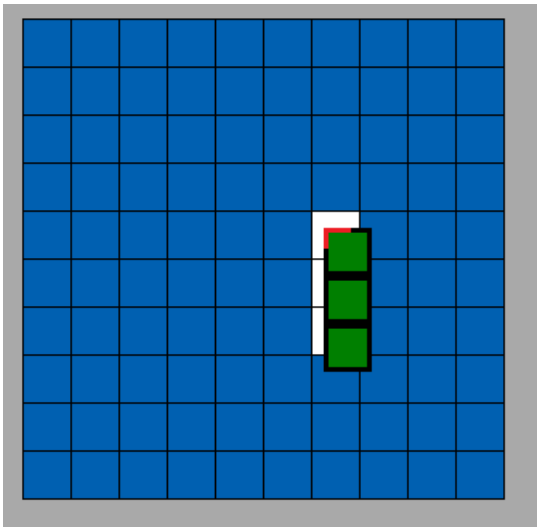

Рис. 5.1.2.

Корабль должен располагаться левым верхним углом (на изображении выделено красным цветом) в крайней клетке, тогда он будет занимать на поле клетки, выделенные белым цветом на изображении.

2) Окно игры (Рис. 5.2.1.).
1 – 1.Половина окна пользователя. 2.Половина окна компьютера.
2 – 1.Индикатор хода пользователя. 2. Индикатор хода компьютера. Если данный индикатор горит зеленым сейчас ход того игрока, чей индикатор горит зеленым.
3 – 1.Информация о последнем ходе игрока 2.Информация о последнем ходе компьютера. В строке “Last Turn” отображается информация о том, в какую клетку был совершен последний ход. В строке “Last Result” отображается информация о том, к какому результату привел последний ход: Missed – не попал, Damaged – попал, Killed – убил.
4 – 1.Поле игрока. 1.1. Индикатор последнего хода компьютера. Этот индикатор показывает куда в последний раз был совершен ход компьютера. 1.2.Ваш корабль. 1.3.Клетка, в которую стрелял компьютер, но не попал по кораблю. 2.Поле компьютера. 2.1.Клетка, в которую стрелял пользователь, но не попал по кораблю. 2.2. Клетка, в которую стрелял игрок и попал по кораблю или убил его. Ход игрока заключается в том, чтобы нажать «Левой кнопкой мыши» на одну из клеток на поле компьютера.

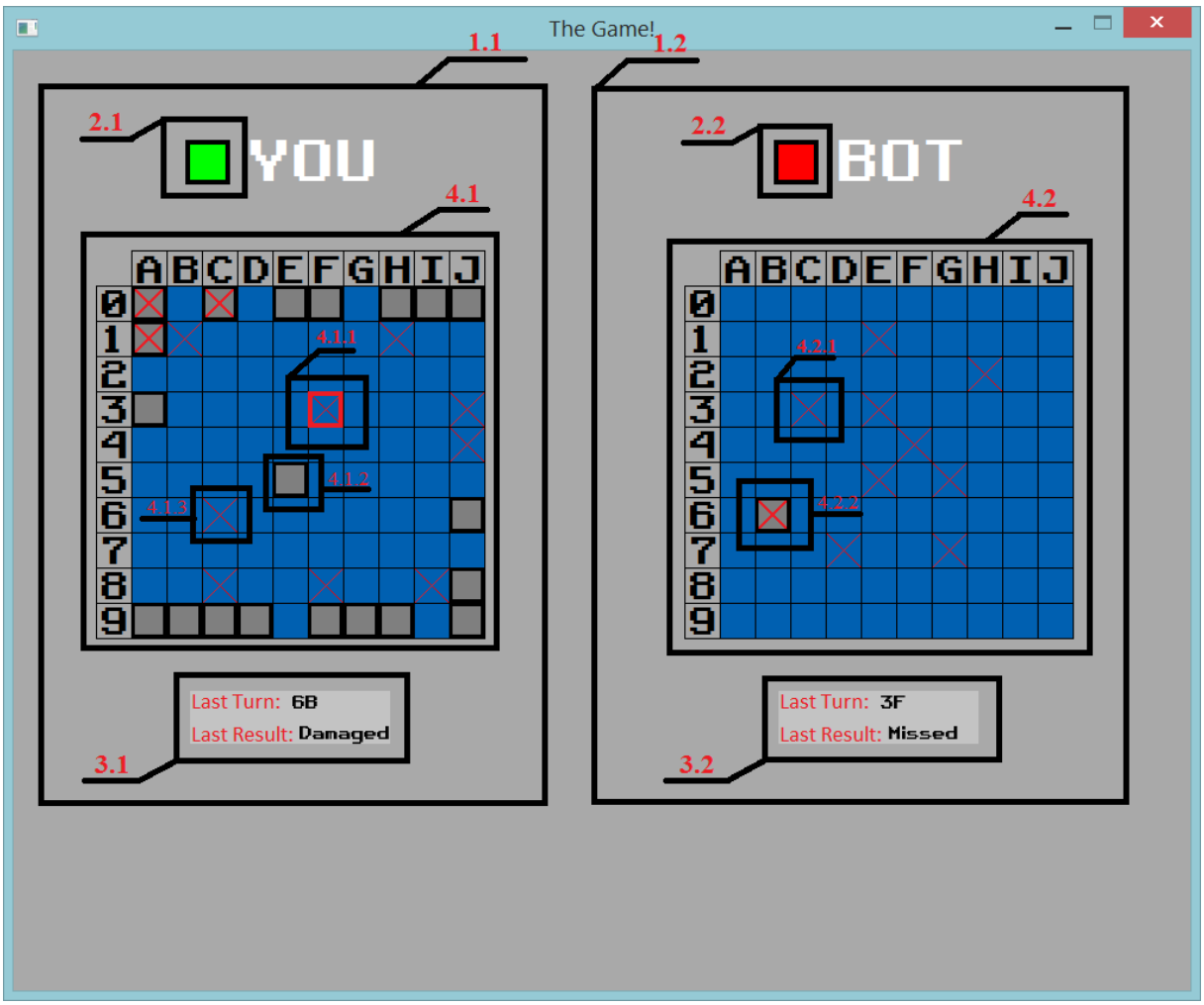

Рис. 5.2.1.

3) Окно результата (Рис. 5.3.1.).
   1 – Сообщение с результатом. (“You won!” – победа, “You lost:(“ – проигрыш)
   2 – Кнопка для повторной игры. Наведите курсором на кнопку и нажмите «Левую кнопку мыши».

Рис. 5.3.1.

6. ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были достигнуты все цели курсовой работы и была написана игра «Морской бой» на языке C++ с использованием классов в среде разработки Microsoft Visual Studio 2017.

Для достижения требуемого результата были решены следующие задачи:

- разработано четыре основных класса игры (Pos,Ship,Player,TheGame);
- разработана UML диаграмма классов;
- разработан графический интерфейс, текстуры для различных окон программы и шрифт для надписей;
- проведено тестирование системы с устранением лишних недочётов и внесением необходимых правок в алгоритм работы программы и графический интерфейс.

7. СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы [Текст]. – Взамен ГОСТ 24.201-85; введ. 1990-01-01. М.: Стандартинформ, 2009. – 21 с.

Зорина, Н.В. Методические указания по выполнению курсовой работы для бакалавров, обучающихся по направлениям 09.03.02 «Информационные системы и технологии», 09.03.04 «Программная инженерия» [Текст] / Н.В. Зорина, Л.Б. Зорин, О.В. Соболев. – Москва: Московский технологический университет, 2016. – 42 с.

Лафоре Р. Объектно-ориентированное программирование C++. Классика Computer Science. 4-е изд. – СПБ.: Питер, 2018. – 928 с.: ил. – (Серия «Классика Computer Science»).

Tutorials for SFML 2.5 [Электронный ресурс] // www.sfml-dev.org: SFML. URL: https://www.sfml-dev.org/tutorials/2.5 09.04.2019

SFML (разработка 2D игр), язык C++ [Электронный ресурс] // http://kychka-pc.ru/ 21.04.2019

ПРИЛОЖЕНИЕ А

Техническое задание

1. ОБЩИЕ ПОЛОЖЕНИЯ

1.1 Полное наименование системы и ее условное обозначение

1.1.1 Полное наименование системы

Полное наименование: Компьютерная игра «Морской бой».

1.1.2 Краткое наименование системы

Краткое наименование: Система.

1.2 Номер договора (контракта)

Работа выполняется на основании Задания на выполнение курсовой работы.

1.3 Наименования организации-заказчика и организаций-участников работ

1.3.1 Заказчик

Заказчик: РТУ МИРЭА

1.3.2 Исполнитель

Исполнитель: студент группы ИНБО-04-18 Дмитриев Алексей Романович

1.4 Перечень документов, на основании которых создается система

Документы, на основании которых создается система:

− Учебный план (№1382.9 09.03.03 ПИ Очн, пс, 4 года (8 сессий) УМУ_09.03.03_ИО_ИИТ_2018).

1.5 Плановые сроки начала и окончания работы по созданию системы

Плановый срок начала работы по созданию Системы – 26 февраля 2019 года.

Плановый срок окончания работы по созданию Системы – 20 мая 2019 года.

1.6 Источники и порядок финансирования работ

Разработка Системы финансируется РТУ МИРЭА, а, в частности, Федеральным бюджетом Российской Федерации.

1.7 Порядок оформления и предъявления заказчику результатов работ по созданию системы

Система передаётся в виде готового программного обеспечения на базе средств вычислительной техники Заказчика в сроки, установленные в п. 2.5 данного ТЗ. Приёмка осуществляется в составе Исполнителя и уполномоченных представителей Заказчика.

Порядок предъявления системы, её испытаний и окончательной приёмки определён в п.7 настоящего ТЗ. Совместно с предъявлением Системы Исполнителем производится сдача разработанного комплекта документации согласно п. 9 данного ТЗ.

1.8 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ

При разработке Системы Исполнитель должен руководствоваться требованиями следующих документов:

− ГОСТ 34.601-90 Комплекс стандартов Автоматизированные системы. Стадии создания;

− Методические указания по выполнению курсовой работы для бакалавров.

1.9 Определения, обозначения и сокращения

Определения, обозначения и сокращения отсутствуют.

2. НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ

2.1 Назначение системы

Система предназначена для развлечения.

2.2 Цели создания системы

Цели создания Системы:

- сдача курсовой работы;
- приобретение опыта разработки приложений средней сложности;
- изучение объектно-ориентированного программирования;
- приобретение навыков написания технической документации.

3. ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ

Объектом автоматизации для данной Системы является машина, которая играет с пользователем в игру «Морской бой» по ниже приведённым правилам:

1) У каждого игрока есть 10 кораблей: 4 однопалубных, 3 двухпалубных, 2 трехпалубных, 1 четырехпалубный, а также свое поле 10х10 клеток (каждая палуба – 1 клетка), на котором он должен расставить свои корабли.
2) Для того, чтобы начать игру, все корабли должны быть расставлены на полях так, чтобы никакие из них не соприкасались гранями или углами.
3) После того, как все корабли расставлены, начинается игра. Цель игры – убить все корабли соперника. Ходы делаются поочередно, первый ход делает пользователь.
4) Ход заключается в том, что игрок должен «выстрелить» в одну из 100 клеток на поле соперника. Если в клетке, в которую «выстрелил» игрок, нет палубы какого-либо корабля, то игроку выводится сообщение о том, что он «не попал», а право хода переходит к сопернику. Если же в клетке оказывается палуба какого-либо корабля, то игроку выводится сообщение о том, что он «попал» или «убил», и ход этого игрока продолжается. Корабль считается убитым, если игрок попал во все его палубы.
5) Игра заканчивается, когда все корабли одного из игроков убиты.

4. ТРЕБОВАНИЯ К СИСТЕМЕ

4.1 Требования к системе в целом

4.1.1 Требования к структуре и функционированию системы

4.1.1.1 Перечень подсистем, их назначение и основные характеристики

Разрабатываемая Система должна состоять из нескольких классов, а именно:

- Pos - координата;
- Ship - корабль;
- Player - игрок;
- TheGame - основной класс, реализующий основные циклы и инициализирующий все окна Системы;

4.1.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы

Связь между компонентами Системы обеспечивается функциями, определёнными в компонентах Системы.

4.1.2 Требования к численности и квалификации персонала системы

Для работы разрабатываемой Системы необходимо и достаточно одного человека. Данная Система не требует какой-либо квалификации или подготовки пользователей для работы с ней.

4.1.3 Показатели назначения

Показателями назначения системы являются результаты, совпадающие с ожидаемыми во время тестирования.

4.1.4 Требования к надежности

Данная Система должна хранить все данные в динамической памяти Исключения составляют каталог с элементами графического интерфейса, файлы расширения openal32.dll, sfml-graphics-2.dll, sfml-system-2.dll, sfml-window-2.dll, и файл шрифта 14722.ttf, которые должны храниться в том же каталоге, что и Система.

4.1.5 Требования к безопасности

Работу данной Системы следует завершать только по нажатии кнопки «Закрыть» в правом верхнем углу окна.

4.1.6 Требования к эргономике и технической эстетике

Графический интерфейс Системы должен разрабатываться на основе офтальмологических исследований, с использованием сочетаний цветов, комфортных для глаз человека.

4.1.7 Требования к транспортабельности для подвижных АС

Разрабатываемая Система должна запускаться на любых устройствах под управлением операционной системы не старше чем Windows XP без процедуры установки или повторной компиляции данной Системы.

4.1.8 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

Для корректной работы разрабатываемая Система должна храниться в отдельном каталоге со всеми своими компонентами. Особых требований по обслуживанию данная Система не имеет.

4.1.9 Требования к защите информации от несанкционированного доступа

Данная Система должна обеспечивать только динамическое хранение информации.

4.1.10 Требования по сохранности информации при авариях

В случае аварийной ситуации текущее состояние Системы можно получить, введя те же самые входные данные.

4.1.11 Требования к защите от влияния внешних воздействий

Разрабатываемая Система должна обеспечивать безопасную работу без случайных или умышленных изменений, внесённых при переключении между состояниями. Для того, чтобы изменить состояние Системы вручную, требуется предварительно переключить её в режим ожидания ввода информации и лишь за тем дополнять или изменять состояние поля.

4.1.12 Требования к патентной чистоте

Данная Система должна являться интеллектуальной собственностью Исполнителя и быть патентно чистой по отношению ко всем странам мира.

4.1.13 Требования по стандартизации и унификации

Система также должна быть реализована методами объектно-ориентированного программирования.

4.1.14 Дополнительные требования

Разрабатываемая Система дополнительных требований не имеет.

4.2 Требования к функциям (задачам), выполняемым системой

Данная Система должна соответствовать данным условиям:

- Класс TheGame должен представлять собой основное управление Системой. В нём должны быть реализованы функции создания и поддержки корректной работы всех окон Системы;
- Класс Pos является координатой клетки на поле каждого из игроков;
- Класс Ship является кораблем. Он должен хранить координаты своего положения на поле игрока, а также о его состоянии;

– Класс Player является игроком. Он должен хранить данные о всех своих кораблях, а также данные о состоянии хода игрока.

4.3 Требования к видам обеспечения

4.3.1 Требования к математическому обеспечению системы

Разрабатываемая Система должна уметь контролировать расстановку кораблей пользователем, расставить корабли компьютера случайным образом, соблюдая правило расстановки, контролировать количество кораблей игроков, а также компьютер должен уметь анализировать свои ходы и делать следующие на основе результатов анализа

4.3.2 Требования информационному обеспечению системы

Также Система должна получать сигналы, контролирующие её работу.

4.3.3 Требования к лингвистическому обеспечению системы

Для создания Системы должен быть использован низкоуровневый язык программирования C++, а также подключаемая графическая библиотека SFML. Для взаимодействия Системы с пользователем, руководства по использованию Системы и документации Системы должен использоваться русский и английский языки.

4.3.4 Требования к программному обеспечению системы

Данная Система должна являться кроссплатформенным приложением и работать на всех операционных системах с поддержкой графического интерфейса. Для корректной работы Системы не требуется загрузки сторонних приложений или библиотек.

4.3.5 Требования к техническому обеспечению

Для корректной работы Системы, она должна запускаться на устройствах с подключённым экраном, мышью и клавиатурой. Также ширина экрана должна быть строго больше чем его высота.

4.3.6 Требования к организационному обеспечению

Разрабатываемая Система не должна начинать работу при неверно введённых входных данных.

4.3.7 Требования к методическому обеспечению

Данная система должна поставляться с определённым пакетом документации, состоящем из:

– технического задания (ГОСТ 34.602-89);
– отчёта по выполнению курсовой работы;
– задания на выполнение курсовой работы.

5. СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ (РАЗВИТИЮ) СИСТЕМЫ

- Сбор необходимой информации

  Результат: определение целей, задач проектируемой системы.
- Анализ предметной области

  Результат: введение базовых требований к решению задач и целей.
- Разработка ТЗ

  Результат: готовое техническое задание с определёнными сроками выполнения курсовой работы.
- Разработка модели программы

  Результат: полностью спроектированные классы, готовая UML диаграмма.
- Разработка готового проекта

  Результат: готовая рабочая программа.
- Тестирование программного продукта

  Результат: устранение ошибок и недочётов в работе программного продукта.
- Сдача системы в эксплуатацию с описанием алгоритмов и готовой технической документацией

6. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ СИСТЕМЫ

6.1 Виды, состав, объем и методы испытаний системы

Для разрабатываемой Системы будут созданы десять тестов, проверяющих её работу на предельных и промежуточных значениях. Для каждого теста будет составлен полный перечень промежуточных и конечного состояний.

6.2 Общие требования к приему работ по стадиям

Разработка данной Системы делится на шесть стадий:

– получение задания на выполнение курсовой работы;
– составление и согласование технического задания;
– создание и тестирование Системы Исполнителем;
– написание технической документации для Системы;
– демонстрация Системы Заказчику;
– защита курсовой работы.

ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ

Для корректной работы разрабатываемой Системы необходимо использовать экраны, ширина которых строго больше их высоты.

В начальном окне Системы надо правильно расставить все 10 кораблей.

Для работы с Системой в основном окне требуется нажимать на левую клавишу компьютерной мыши тогда, когда курсор на экране указывает на клетку, на которую пользователь хочет нажать.

7. ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ

Кроме создания работоспособной Системы Исполнитель должен составить пакет документации, состоящий из:

– технического задания;

– пояснительной записки.

8. ИСТОЧНИКИ РАЗРАБОТКИ

В качестве источников разработки использовались данные ресурсы:

– Зорина, Н.В. Методические указания по выполнению курсовой работы для бакалавров, обучающихся по направлениям 09.03.02 «Информационные системы и технологии», 09.03.04 «Программная инженерия» / Н.В. Зорина, Л.Б. Зорин, О.В. Соболев,- Москва, 2017 - 41 с.

ПРИЛОЖЕНИЕ Б

Листинг программы

Файл Pos.h

```
#pragma once
class Pos
{
public:
	int x;
	int y;
	Pos(int,int);
	Pos();
	void setPos(int, int);
	~Pos();
};
```

Файл Pos.cpp

```
#include "Pos.h"
Pos::Pos(int x,int y)
{
	this->x = x;
	this->y = y;
}
Pos::Pos()
{
}
void Pos::setPos(int x, int y) {
	this->x = x;
	this->y = y;
}
Pos::~Pos()
{
}
```

Файл Ship.h

```
#pragma once
#include <vector>
#include "Pos.h"
using namespace std;

class Ship
{
protected:
	int heads;//Количество целых палуб
public:
	bool isDamaged;//Подбит
	bool isAlive;//Жив
	vector<Pos> shipSurrounding;//Точки в окружении корабля
	vector<Pos> posShip;//Координаты палуб корабля
	Ship();//Создает живой корабль без палуб

	void setHeads(int);//Задать количество палуб
	void setPos(Pos,bool);//Задать координаты корабля и его окружения
	int getHeads();//Получить количество оставшихся палуб
	void Damage();//Подбить/убить корабль
	~Ship();
};
```

Файл Ship.cpp

```
#include "Ship.h"
Ship::Ship()
{
	isDamaged = false;
	isAlive = true;
}
void Ship::Damage() {
	isDamaged = true;
	heads--;
	if (heads == 0)
		isAlive = false;
}
void Ship::setHeads(int heads) {
	this->heads = heads;
}
void Ship::setPos(Pos posMain,bool direct) {
	if (direct)
		for (int i = -1; i != heads + 1; i++)
			if ((i != -1) && (i != heads)) {
				posShip.push_back(Pos(posMain.x + i, posMain.y));
				shipSurrounding.push_back(Pos(posMain.x + i, posMain.y + 1));
				shipSurrounding.push_back(Pos(posMain.x + i, posMain.y - 1));
			}
			else {
				shipSurrounding.push_back(Pos(posMain.x + i, posMain.y));
				shipSurrounding.push_back(Pos(posMain.x + i, posMain.y + 1));
				shipSurrounding.push_back(Pos(posMain.x + i, posMain.y - 1));
			}
	else
		for (int i = -1; i != heads + 1 ; i++)
			if ((i != -1) && (i != heads)) {
				posShip.push_back(Pos(posMain.x, posMain.y + i));
				shipSurrounding.push_back(Pos(posMain.x + 1, posMain.y + i));
				shipSurrounding.push_back(Pos(posMain.x - 1, posMain.y + i));
			}
			else {
				shipSurrounding.push_back(Pos(posMain.x , posMain.y + i));
				shipSurrounding.push_back(Pos(posMain.x + 1, posMain.y + i));
				shipSurrounding.push_back(Pos(posMain.x - 1, posMain.y + i));
			}


}
int Ship::getHeads() {
	return heads;
}
Ship::~Ship()
{
}
```

Файл Player.h

```
#pragma once
#include<vector>
#include "Ship.h"
#include <iostream>
using namespace std;

class Player
{
protected:
	int numberOfShips;//Количество оставшихся кораблей
public:
	vector<Ship> pships;//Корабли
	bool turn;// Ход
	Player();//Создает игрока с 10 кораблями
	void setShip(Pos,int,bool);//Задать координаты корабля
	void killShip();//Убить корабль
	int getNumberOfShips();//Получить количество оставшихся кораблей
```

```
	~Player();
};
```

Файл Player.cpp

```
#include "Player.h"

Player::Player()
{
	turn = false;
	numberOfShips = 10;
}
void Player::setShip(Pos mainPos,int number,bool isRotated) {
	Ship ship;
	if (number < 4) {
		ship.setHeads(1);
		ship.setPos(mainPos, isRotated);
	}
	else
		if (number < 7) {
			ship.setHeads(2);
			ship.setPos(mainPos, isRotated);
		}
		else
			if (number < 9) {
				ship.setHeads(3);
				ship.setPos(mainPos, isRotated);
			}
			else {
				ship.setHeads(4);
				ship.setPos(mainPos, isRotated);
			}
	pships.push_back(ship);
}


void Player::killShip() {
	numberOfShips--;
}
int Player::getNumberOfShips() {
	return numberOfShips;
}
Player::~Player()
{
}
```

Файл TheGame.h

```
#pragma once
#include "Player.h"
#include <SFML/Graphics.hpp>
#include<cmath>
#include<ctime>
using namespace std;
using namespace sf;
class TheGame
{
protected:
	Player player;//Игрок-пользователь
	Player playerCPU;//Игрок-компьютер
	vector<FloatRect> cpuShips;//Корабли компьютера
	vector<Sprite> ships;//Корабли пользователя
	Sprite plain1;//Поле пользователя
	Sprite plain2;//Поле компьютера
	Texture markText;//Текстура "красного крестика"
	Texture shipText;//Тектура корабля
	Texture Text;//Основная текстура
	Texture TextInformation;//Основная текстура текстовой информации
	Image markImg;//Изображение "красного крестика"
	vector<bool> canRotate;//Можно ли повернуть n-ый корабль пользователя
	vector<bool> isRotated;//Повернут ли n-ый корабль
public:
	TheGame();
```

```
	bool restart;//Играть заново
	bool win;//Победа или поражение
	bool goOn;//Продолжить
	void Texturising();//Создает основные текстуры
	void StageOne();//Первый этап: расстановка кораблей
	void StageTwo();//Второй этап: запись координат кораблей в соответствии с тем,
как расставил корабли пользователь. Расстановка кораблей компьютера и запись их
координат.
	void StageThree();//Третий этап: сама игра
	void StageFinal();//Последний этап: окно с результатом и кнопкой "Restart"
	~TheGame();
};
```

Файл TheGame.cpp

```
#include "TheGame.h"
TheGame::TheGame()
{
	Text.loadFromFile("image/Text.png");
	shipText.loadFromFile("image/Text.png", IntRect(301, 0, 30, 30));
	shipText.setRepeated(true);
	markImg.loadFromFile("image/Text.png");
	markImg.createMaskFromColor(Color::Color(0, 96, 177));
	markText.loadFromImage(markImg);
	TextInformation.loadFromFile("image/TextInf.png");
	goOn = false;
	win = false;
	restart = false;
	for (int i = 0; i < 7; i++) {
		isRotated.push_back(false);
		canRotate.push_back(true);
	}
}

void TheGame::StageOne()
{//////////////////////////////////////////////////////////////////////////////////
//////Этап 1 - расстановка кораблей
	int lastShip = -1;
	bool isMove = false;
	float dX = 0, dY = 0;
	int k = -1;
	bool shipsLocked = false;
	Sprite startButton;
	startButton.setTexture(Text);
	startButton.setTextureRect(IntRect(301, 61, 60, 30));
	startButton.setPosition(100, 100);
	//////////////////////////////////////////////////////////////////////////////
///////////////Начальное окно,где игрок расставляет корабли
	sf::RenderWindow startWin(sf::VideoMode(1000, 800), "The start!",
sf::Style::Close);
	startWin.setFramerateLimit(100);
	while (startWin.isOpen())
	{
```

```
		Vector2i pixelPos = Mouse::getPosition(startWin);//забираем коорд курсора

		sf::Event sEvent;
		while (startWin.pollEvent(sEvent))
		{
			if (sEvent.type == sf::Event::Closed)
				startWin.close();
			if (sEvent.type == Event::MouseButtonPressed)
				if (sEvent.key.code == Mouse::Left) {
					if (shipsLocked)
						if
(startButton.getGlobalBounds().contains(pixelPos.x, pixelPos.y)) {
							goOn = true;
							startWin.close();
						}

	////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////Перетаскивание кораблей и постановка их на
поле
					for (int i = 0; i < ships.size(); i++)
						if
(ships[i].getGlobalBounds().contains(pixelPos.x, pixelPos.y))
						{
							k = i;
							lastShip = i;
							dX = pixelPos.x -
ships[i].getPosition().x;
							dY = pixelPos.y -
ships[i].getPosition().y;
							isMove = true;

						}
				}
			if (sEvent.type == Event::MouseButtonReleased)
				if (sEvent.key.code == Mouse::Left)
					for (int i = 0; i < ships.size(); i++)
						if
(ships[i].getGlobalBounds().contains(pixelPos.x, pixelPos.y))
						{
							if ((pixelPos.x > 100) && (pixelPos.x <
400) && (pixelPos.y > 200) && (pixelPos.y < 500))
							{

	ships[i].setPosition((int(abs((ships[i].getPosition().x - 100) / 30))) * 30 +
100, (int(abs((ships[i].getPosition().y - 200) / 30))) * 30 + 200);
								if (i > 3)
									if (i < 7) {
										if (!isRotated[i -
3]) {
```

```
                                                                        if
(int(abs((ships[i].getPosition().y - 200) / 30)) > 8)
        ships[i].setPosition(600 + (i - 4) * 70, 240);
                                                                        canRotate[i
- 3] = true;
                                                                }
                                                                else {
                                                                        if
(int(abs((ships[i].getPosition().x - 100) / 30)) > 8) {
        ships[i].setPosition(600 + (i - 4) * 70, 240);
        ships[i].rotate(-90);
        canRotate[i - 3] = true;
        isRotated[i - 3] = false;
        ships[i].setOrigin(Vector2f(0, 0));
                                                                        }
                                                                }
                                                                if
((int(abs((ships[i].getPosition().y - 200) / 30)) > 8) ||
(int(abs((ships[i].getPosition().x - 100) / 30)) > 8))
                                                                        canRotate[i
- 3] = false;
                                                                else
                                                                        canRotate[i
- 3] = true;
                                                        }
                                                        else
                                                                if (i < 9) {
                                                                        if
(!isRotated[i - 3])
                                                                                if
(int(abs((ships[i].getPosition().y - 200) / 30)) > 7) {
        ships[i].setPosition(600 + (i - 7) * 100, 320);
        canRotate[i - 3] = true;
                                                                                }
                                                                                else
{}
                                                                        else
                                                                                if
(int(abs((ships[i].getPosition().x - 100) / 30)) > 7) {
```

```
        ships[i].setPosition(600 + (i - 7) * 100, 320);

        ships[i].rotate(-90);

        canRotate[i - 3] = true;

        isRotated[i - 3] = false;

        ships[i].setOrigin(Vector2f(0, 0));

                                                                    }
                                                                    else
{}
                                                                    if
((int(abs((ships[i].getPosition().y - 200) / 30)) > 7) ||
(int(abs((ships[i].getPosition().x - 100) / 30)) > 7))

        canRotate[i - 3] = false;
                                                                    else

        canRotate[i - 3] = true;
                                                            }
                                                            else {
                                                                    if
(!isRotated[i - 3])
                                                                    if
(int(abs((ships[i].getPosition().y - 200) / 30)) > 6) {

        ships[i].setPosition(600, 420);

        canRotate[i - 3] = true;
                                                                    }
                                                                    else
{}
                                                                    else
                                                                    if
(int(abs((ships[i].getPosition().x - 100) / 30)) > 6) {

        ships[i].setPosition(600, 420);

        ships[i].rotate(-90);

        canRotate[i - 3] = true;

        isRotated[i - 3] = false;

        ships[i].setOrigin(Vector2f(0, 0));

                                                                    }
```

```
                                                                        if
((int(abs((ships[i].getPosition().y - 200) / 30)) > 6) ||
(int(abs((ships[i].getPosition().x - 100) / 30)) > 6))

        canRotate[i - 3] = false;
                                                                        else

        canRotate[i - 3] = true;
                                                                }
                                                }
                                                else {
                                                        if (i < 4)
                                                                ships[i].setPosition(600 +
i * 40, 200);
                                                        else
                                                                if (i < 7)

        ships[i].setPosition(600 + (i - 4) * 70, 240);

                                                                else
                                                                        if (i < 9)

        ships[i].setPosition(600 + (i - 7) * 100, 320);
                                                                        else

        ships[i].setPosition(600, 420);
                                                        if (i > 3)
                                                                if (isRotated[i - 3]) {
                                                                        ships[i].rotate(-
90);
                                                                        canRotate[i - 3] =
true;
                                                                        isRotated[i - 3] =
false;

        ships[i].setOrigin(Vector2f(0, 0));
                                                                }
                                                                else
                                                                        canRotate[i - 3] =
true;
                                                }

        ////////////////////////////////////////////////////////////////////////////
////////////////////////////Проверка на правильность постановки кораблей
                                                for (int j = 0; j < 10; j++)
                                                {
                                                        if
(plain1.getGlobalBounds().contains(ships[lastShip].getPosition())) {
                                                                FloatRect lastbox =
ships[lastShip].getGlobalBounds();
```

```
FloatRect box = ships[j].getGlobalBounds();
box.top -= 30; box.left -= 30;
box.width += 60; box.height += 60;

                                                                    if (j != lastShip)
                                                                        if
(lastbox.intersects(box)) {
                                                                                if (lastShip
< 4) {

        ships[lastShip].setPosition(600 + lastShip * 40, 200);

        canRotate[lastShip - 3] = true;
                                                                                }
                                                                                else
                                                                                        if
(lastShip < 7) {

        ships[lastShip].setPosition(600 + (lastShip - 4) * 70, 240);

        canRotate[lastShip - 3] = true;
                                                                                }

                                                                                else

        if (lastShip < 9) {

                ships[lastShip].setPosition(600 + (lastShip - 7) * 100, 320);

                canRotate[lastShip - 3] = true;

        }

        else {

                ships[lastShip].setPosition(600, 420);

        }


        if (lastShip > 3)

                if (isRotated[lastShip - 3]) {

                        ships[lastShip].rotate(-90);

                        canRotate[lastShip - 3] = true;

                        isRotated[lastShip - 3] = false;
```

```
                        ships[lastShip].setOrigin(Vector2f(0, 0));

                }
                                                                        }
                                                                }
                                                        }

        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////Конец проверка
                                                        isMove = false;
                                                        k = -1;
                                                }
                        for (int i = 0; i < ships.size(); i++)
                                ships[i].setColor(Color::White);

        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////Конец перетаскивания кораблей


        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////Поворот корабля
                        if (isMove == false)
                                if (sEvent.type == Event::MouseButtonPressed)
                                        if ((sEvent.key.code == Mouse::Right) &&
(sEvent.key.code != Mouse::Left))
                                                for (int i = 4; i < ships.size(); i++)
                                                        if
(ships[i].getGlobalBounds().contains(pixelPos.x, pixelPos.y))
                                                        {
                                                                lastShip = i;
                                                                if (canRotate[i - 3])
                                                                        if (!isRotated[i - 3]) {

        ships[i].rotate(90);
                                                                                isRotated[i - 3] =
true;
                                                                                if (i < 7)

        ships[i].setOrigin(Vector2f(0, 60));
                                                                                else
                                                                                        if (i < 9)

        ships[i].setOrigin(Vector2f(0, 90));
                                                                                        else

        ships[i].setOrigin(Vector2f(0, 120));
                                                                        }
                                                                        else {
```

```
                                                                        ships[i].rotate(-
90);
                                                                        isRotated[i - 3] =
false;

        ships[i].setOrigin(Vector2f(0, 0));
                                                                }

        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////Проверка на правильность постановки кораблей
                                                                for (int j = 0; j < 10;
j++)
                                                                {
                                                                        if
(plain1.getGlobalBounds().contains(ships[lastShip].getPosition())) {
                                                                                FloatRect
lastbox = ships[lastShip].getGlobalBounds();
                                                                                FloatRect
box = ships[j].getGlobalBounds();
                                                                                box.top -=
30; box.left -= 30;
                                                                                box.width +=
60; box.height += 60;
                                                                                if (j !=
lastShip)
                                                                                        if
(lastbox.intersects(box)) {

        if (lastShip < 4) {

                ships[lastShip].setPosition(600 + lastShip * 40, 200);

                canRotate[lastShip - 3] = true;

        }

        else

                if (lastShip < 7) {

                        ships[lastShip].setPosition(600 + (lastShip - 4) * 70, 240);

                        canRotate[lastShip - 3] = true;

                }


                else

                        if (lastShip < 9) {
```

```
                        ships[lastShip].setPosition(600 + (lastShip - 7) * 100, 320);
                        canRotate[lastShip - 3] = true;
                    }
                    else {
                        ships[lastShip].setPosition(600, 420);
                        canRotate[lastShip - 3] = true;
                    }
                    if (lastShip > 3)
                        if (isRotated[lastShip - 3]) {
                            ships[lastShip].rotate(-90);
                            canRotate[lastShip - 3] = true;
                            isRotated[lastShip - 3] = false;
                            ships[lastShip].setOrigin(Vector2f(0, 0));
                        }
                                                                        }
                                                                }
                                                        }
        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////Конец проверка
                                                }

        ////////////////////////////////////////////////////////////////////////////////
////////////////////////////Конец поворота корабля
                    ////////////////////////////////////////////Проверка,стоят ли все
корабли на поле
                    shipsLocked = true;
                    for (int i = 0; i < ships.size(); i++)
                        if
(!plain1.getGlobalBounds().contains(ships[i].getPosition()))
                            shipsLocked = false;
                    ////////////////////////////////////////////////Конец проверки
                }
                startWin.clear(sf::Color::Color(169, 169, 169));
                startWin.draw(plain1);
                if (isMove) {
```

```
                        ships[k].setColor(Color::Green);
                        ships[k].setPosition(pixelPos.x - dX, pixelPos.y - dY);
                }
                for (int i = 0; i < ships.size(); i++)
                        startWin.draw(ships[i]);
                startWin.draw(startButton);
                startWin.display();
        }
        ////////////////////////////////////////////////////////////////////////////
//////////////////////////Конец начального окна
}

void TheGame::StageTwo() {
        vector<FloatRect> cpuShipsBox;
        vector<Pos> CpuPos;
        vector<bool> CpuRotation;
        srand(time(NULL));
        int k = 0;
        for (int i = 0; i < 10; i++) {
                if (i > 3)
                        player.setShip(Pos((int(abs(ships[i].getPosition().x - 100))) / 30,
(int(abs((ships[i].getPosition().y - 200))) / 30)), i, isRotated[i - 3]);
                else
                        player.setShip(Pos((int(abs(ships[i].getPosition().x - 100))) / 30,
(int(abs((ships[i].getPosition().y - 200))) / 30)), i, false);
        }

        while (k < 10)
        {
                bool add = true;
                bool rotation = false;
                FloatRect box;
                Pos randPos;
                rotation = rand() % 2;
                if (k < 4)
                        randPos.setPos(rand() % 10, rand() % 10);
                else
                        if (k < 7)
                                if (rotation)
                                        randPos.setPos(rand() % 9, rand() % 10);
                                else
                                        randPos.setPos(rand() % 10, rand() % 9);
                        else
                                if (k<9)
                                        if (rotation)
                                                randPos.setPos(rand() % 8, rand() % 10);
                                        else
                                                randPos.setPos(rand() % 10, rand() % 8);
                                else
                                        if (rotation)
```

```
                                        randPos.setPos(rand() % 7, rand() % 10);
                                else
                                        randPos.setPos(rand() % 10, rand() % 7);
                        if (k < 4) {
                                box.left = 600 + randPos.x * 30; box.top = 200 + randPos.y * 30;
                                box.width += 30; box.height += 30;
                        }
                        else
                                if (k < 7)
                                        if (!rotation) {
                                                box.left = 600 + randPos.x * 30; box.top = 200 +
randPos.y * 30;
                                                box.width += 30; box.height += 60;
                                        }
                                        else {
                                                box.left = 600 + randPos.x * 30; box.top = 200 +
randPos.y * 30;
                                                box.width += 60; box.height += 30;
                                        }
                                else
                                        if (k < 9)
                                                if (!rotation) {
                                                        box.left = 600 + randPos.x * 30; box.top = 200
+ randPos.y * 30;
                                                        box.width += 30; box.height += 90;
                                                }
                                                else {
                                                        box.left = 600 + randPos.x * 30; box.top = 200
+ randPos.y * 30;
                                                        box.width += 90; box.height += 30;
                                                }
                                        else
                                                if (!rotation) {
                                                        box.left = 600 + randPos.x * 30; box.top = 200
+ randPos.y * 30;
                                                        box.width += 30; box.height += 120;
                                                }
                                                else {
                                                        box.left = 600 + randPos.x * 30; box.top = 200
+ randPos.y * 30;
                                                        box.width += 120; box.height += 30;
                                                }


                                        if (k != 0)
                                                for (int i = 0; i < cpuShipsBox.size(); i++)
                                                        if (cpuShipsBox[i].intersects(box))
                                                                add = false;

                                        if (add)
```

```
                                        {
                                                cpuShips.push_back(box);
                                                box.left -= 30; box.top -= 30;
                                                box.width += 60; box.height += 60;
                                                cpuShipsBox.push_back(box);
                                                CpuPos.push_back(randPos);
                                                CpuRotation.push_back(rotation);
                                                k++;
                                        }

        }
        for (int i = 0; i<10; i++) {
                if (i > 3)
                        playerCPU.setShip(CpuPos[i], i, CpuRotation[i]);
                else
                        playerCPU.setShip(CpuPos[i], i, false);
        }

}



void TheGame::StageThree()
{////////////////////////////////////////////////////////////////////////////////////////
//////////////////Этап 3 - сама игра
        srand(time(NULL));

        goOn = false;

        Sprite Sym1, Num1;
        Sprite Sym2, Num2;
        Sym1.setTexture(TextInformation);
        Sym1.setTextureRect(IntRect(0, 45, 301, 30));
        Sym1.setPosition(100, 170);
        Sym2.setTexture(TextInformation);
        Sym2.setTextureRect(IntRect(0, 45, 301, 30));
        Sym2.setPosition(600, 170);
        Num1.setTexture(TextInformation);
        Num1.setTextureRect(IntRect(0, 75, 301, 30));
        Num1.setRotation(-90);
        Num1.setPosition(70, 501);
        Num2.setTexture(TextInformation);
        Num2.setTextureRect(IntRect(0, 75, 301, 30));
        Num2.setRotation(-90);
        Num2.setPosition(570, 501);

        Sprite textInformation1;
        Sprite textInformation2;
        Sprite TextYOU;
        Sprite TextCPU;
```

```
	textInformation1.setTexture(TextInformation);
	textInformation1.setTextureRect(IntRect(0, 0, 170, 45));
	textInformation1.setPosition(150, 545);

	textInformation2.setTexture(TextInformation);
	textInformation2.setTextureRect(IntRect(0, 0, 170, 45));
	textInformation2.setPosition(650, 545);

	TextYOU.setTexture(TextInformation);
	TextYOU.setTextureRect(IntRect(171, 1, 106, 36));
	TextYOU.setPosition(200, 75);

	TextCPU.setTexture(TextInformation);
	TextCPU.setTextureRect(IntRect(280, 1, 106, 37));
	TextCPU.setPosition(700, 75);

	RectangleShape indicator, indicator2;
	indicator.setSize(Vector2f(30, 30));
	indicator.setOutlineThickness(4);
	indicator.setOutlineColor(Color::Black);
	indicator.setPosition(150, 80);

	indicator2.setSize(Vector2f(30, 30));
	indicator2.setOutlineThickness(4);
	indicator2.setOutlineColor(Color::Black);
	indicator2.setPosition(650, 80);

	Texture IndicatorShoot;
	IndicatorShoot.loadFromImage(markImg);
	Sprite shootIndicator;
	shootIndicator.setTexture(IndicatorShoot);
	shootIndicator.setTextureRect(IntRect(331, 30, 30, 31));
	shootIndicator.setPosition(570, 170);

	sf::Text lastTurnP, lastTurnC, lastResultP, lastResultC;
	sf::Font font; font.loadFromFile("14722.ttf");
	lastTurnP.setFont(font); lastTurnC.setFont(font); lastResultP.setFont(font);
lastResultC.setFont(font);
	lastTurnP.setCharacterSize(15); lastTurnC.setCharacterSize(15);
lastResultP.setCharacterSize(15); lastResultC.setCharacterSize(15);
	lastTurnP.setPosition(236, 545); lastTurnC.setPosition(736, 545);
lastResultP.setPosition(242, 571); lastResultC.setPosition(742, 571);
	lastTurnP.setFillColor(Color::Black); lastTurnC.setFillColor(Color::Black);
lastResultP.setFillColor(Color::Black); lastResultC.setFillColor(Color::Black);
	lastTurnP.setString("None"); lastTurnC.setString("None");
lastResultP.setString("None"); lastResultC.setString("None");

	string lastTurn;
	bool Aimed = false;
	Pos lastAim, lastPos1, lastPos2;
```

```
	Sprite mark;

	vector<Pos> playerPlainPos;
	vector<bool> playerPlain;
	vector<bool> cpuPlain;

	player.turn = true;
	for (int i = 0; i < 100; i++) {
		cpuPlain.push_back(true);
		playerPlain.push_back(true);
	}
	for (int i = 0; i < 10; i++)
		for (int j = 0; j < 10; j++)
			playerPlainPos.push_back(Pos(i, j));

	int number, CPUnumber, randPoint;

	mark.setTexture(markText);

	bool dX = false, dY = false, dInc = false, dDec = false, shootMiss = false;

	float sec = 2, time = 0;

	sf::RenderWindow mainWin(sf::VideoMode(1000, 800), "The Game!",
sf::Style::Close);
	mainWin.setFramerateLimit(100);

	vector<Sprite> xmark;
	bool isSet = false, CPUisSet = false, timer = false;

	while (mainWin.isOpen())
	{

		if (player.turn)
			indicator.setFillColor(Color::Green);
		else
			indicator.setFillColor(Color::Red);
		if (playerCPU.turn)
			indicator2.setFillColor(Color::Green);
		else
			indicator2.setFillColor(Color::Red);
		Vector2i pixelPos = Mouse::getPosition(mainWin);

		sf::Event event;
		while (mainWin.pollEvent(event))
		{
			if (event.type == sf::Event::Closed) {
				mainWin.close();

			}
```

```
		}

		if (playerCPU.getNumberOfShips() == 0) {
			win = true;
			goOn = true;
			mainWin.close();
		}
		if (player.getNumberOfShips() == 0) {
			goOn = true;
			mainWin.close();
		}


	////////////////////////////////////////////////////////////////////////////
///////Ход игрока
		if (player.turn)
			if (event.type == sf::Event::MouseButtonPressed) {
				if (event.mouseButton.button == sf::Mouse::Left) {
					if ((pixelPos.x > 600) && (pixelPos.x < 900) &&
(pixelPos.y > 200) && (pixelPos.y < 500))
						if (cpuPlain[(pixelPos.x - 600) / 30 +
((pixelPos.y - 200) / 30) * 10])
						{

	//////////////////////////////////////////Проверка,куда кликнул игрок
							for (int i = 0; i < cpuShips.size();
i++)
								if
(cpuShips[i].contains(pixelPos.x, pixelPos.y))
								{

	mark.setTextureRect(IntRect(331, 0, 30, 30));
									isSet = true;
									number = i;
									break;
								}
							if (!isSet) {
								mark.setTextureRect(IntRect(300,
30, 31, 31));
								player.turn = false;
								playerCPU.turn = true;
								lastResultP.setString("Missed");
							}

	//////////////////////////////////////////Конец проверки
							if (isSet)
							{

	playerCPU.pships[number].Damage();
```

```
                                                        if
(!playerCPU.pships[number].isAlive) {
                                                                playerCPU.killShip();

        lastResultP.setString("Killed");
                                                        }
                                                        else

        lastResultP.setString("Damaged");
                                                }

                                                char sym = char(((pixelPos.x - 600) /
30) + 65), num = char(((pixelPos.y - 200) / 30) + 48);
                                                lastTurn = num; lastTurn += sym;
                                                lastTurnP.setString(lastTurn);



                                                mark.setPosition(((event.mouseButton.x -
600) / 30) * 30 + 600, ((event.mouseButton.y - 200) / 30) * 30 + 200);
                                                xmark.push_back(mark);
                                                if (cpuPlain[(pixelPos.x - 600) / 30 +
((pixelPos.y - 200) / 30) * 10]) {
                                                        cpuPlain[(pixelPos.x - 600) / 30
+ ((pixelPos.y - 200) / 30) * 10] = false;
                                                        if (isSet)
                                                                if
(!playerCPU.pships[number].isAlive)
                                                                        for (int i = 0; i <
player.pships[number].shipSurrounding.size(); i++) {
                                                                                if
((playerCPU.pships[number].shipSurrounding[i].x > -1) &&
(playerCPU.pships[number].shipSurrounding[i].y > -1) &&
(playerCPU.pships[number].shipSurrounding[i].x < 10) &&
(playerCPU.pships[number].shipSurrounding[i].y < 10))
                                                                                {
                                                                                        if
(cpuPlain[playerCPU.pships[number].shipSurrounding[i].x +
playerCPU.pships[number].shipSurrounding[i].y * 10]) {

        mark.setTextureRect(IntRect(300, 30, 31, 31));

        mark.setPosition(playerCPU.pships[number].shipSurrounding[i].x * 30 + 600,
playerCPU.pships[number].shipSurrounding[i].y * 30 + 200);

        xmark.push_back(mark);

        cpuPlain[playerCPU.pships[number].shipSurrounding[i].x +
playerCPU.pships[number].shipSurrounding[i].y * 10] = false;
                                                                                        }
                                                                                }
```

```
                                                                        }
                                                }
                                                isSet = false;
                                        }
                                }
                        }

        ////////////////////////////////////////////////////////////////////////////
/////Конец хода игрока
                if (player.turn)
                        indicator.setFillColor(Color::Green);
                else
                        indicator.setFillColor(Color::Red);
                if (playerCPU.turn)
                        indicator2.setFillColor(Color::Green);
                else
                        indicator2.setFillColor(Color::Red);

                mainWin.clear(sf::Color::Color(169, 169, 169));
                mainWin.draw(plain1);
                mainWin.draw(plain2);
                for (int i = 0; i < ships.size(); i++)
                        mainWin.draw(ships[i]);
                mainWin.draw(textInformation1);
                mainWin.draw(textInformation2);
                mainWin.draw(lastTurnP); mainWin.draw(lastTurnC);
mainWin.draw(lastResultP); mainWin.draw(lastResultC);
                mainWin.draw(TextYOU);
                mainWin.draw(TextCPU);
                mainWin.draw(Sym1);
                mainWin.draw(Sym2);
                mainWin.draw(Num1);
                mainWin.draw(Num2);
                mainWin.draw(indicator);
                mainWin.draw(indicator2);
                if (!xmark.empty())
                        for (int i = 0; i < xmark.size(); i++)
                                mainWin.draw(xmark[i]);
                mainWin.draw(shootIndicator);
                mainWin.display();

        ////////////////////////////////////////////////////////////////////////////
/////Ход компьютера
                if (playerCPU.turn)
                        if (!Aimed)
                        {
                                randPoint = rand() % playerPlainPos.size();
                                Pos randPos = playerPlainPos[randPoint];
                                if (playerPlain[randPos.x + randPos.y * 10])
                                {
```

```
                                for (int i = 0; i < ships.size(); i++)
                                        if
(ships[i].getGlobalBounds().contains(randPos.x * 30 + 103, randPos.y * 30 + 203))
                                        {
                                                Aimed = true;
                                                CPUnumber = i;
                                                CPUisSet = true;
                                                lastPos1 = randPos;
                                        }
                                if (CPUisSet)
                                {
                                        mark.setTextureRect(IntRect(331, 0, 30, 30));
                                        player.pships[CPUnumber].Damage();
                                        if (!player.pships[CPUnumber].isAlive) {
                                                player.killShip();
                                                lastResultC.setString("Killed");
                                                Aimed = false;
                                                for (int i = 0; i <
player.pships[CPUnumber].shipSurrounding.size(); i++) {
                                                        if
((player.pships[CPUnumber].shipSurrounding[i].x > -1) &&
(player.pships[CPUnumber].shipSurrounding[i].y > -1) &&
(player.pships[CPUnumber].shipSurrounding[i].x < 10) &&
(player.pships[CPUnumber].shipSurrounding[i].y < 10))

        playerPlain[player.pships[CPUnumber].shipSurrounding[i].x +
player.pships[CPUnumber].shipSurrounding[i].y * 10] = false;
                                                }
                                        }
                                        else
                                                lastResultC.setString("Damaged");
                                        timer = true;
                                }
                                if (!CPUisSet)
                                {
                                        mark.setTextureRect(IntRect(300, 30, 31, 31));
                                        playerCPU.turn = false;
                                        player.turn = true;
                                        lastResultC.setString("Missed");
                                        timer = true;
                                }
                                char sym = char(randPos.x + 65), num = char(randPos.y
+ 48);
                                lastTurn = num; lastTurn += sym;
                                lastTurnC.setString(lastTurn);


                                mark.setPosition(randPos.x * 30 + 100, randPos.y * 30
+ 200);
```

```
                                shootIndicator.setPosition(randPos.x * 30 + 100,
randPos.y * 30 + 200);
                                xmark.push_back(mark);
                                CPUisSet = false;
                                if (playerPlain[randPos.x + randPos.y * 10]) {
                                        playerPlain[randPos.x + randPos.y * 10] =
false;
                                }
                                playerPlainPos.erase(playerPlainPos.begin() +
randPoint);
                                CPUisSet = false;
                        }
                        else
                                playerPlainPos.erase(playerPlainPos.begin() +
randPoint);
                }

        ////////////////////////////////////////////////////////////////////////Алго
ритм убийства 2х и более -палубных кораблей
                else {
                        if (player.pships[CPUnumber].isAlive)
                        {
                                if (!dX && !dY) {
                                        randPoint = rand() % 4;
                                        switch (randPoint) {
                                        case 0:
                                                lastPos2.x = lastPos1.x - 1;
                                                lastPos2.y = lastPos1.y;
                                                break;
                                        case 1:
                                                lastPos2.x = lastPos1.x + 1;
                                                lastPos2.y = lastPos1.y;
                                                break;
                                        case 2:
                                                lastPos2.x = lastPos1.x;
                                                lastPos2.y = lastPos1.y - 1;
                                                break;
                                        case 3:
                                                lastPos2.x = lastPos1.x;
                                                lastPos2.y = lastPos1.y + 1;
                                                break;
                                        default:
                                                break;
                                        }
                                }
                                else
                                {
                                        if (dX&&dDec)
                                                lastPos2.x = lastPos2.x - 1;
                                        if (dX&&dInc)
```

```
                                        lastPos2.x = lastPos2.x + 1;
                                if (dY&&dDec)
                                        lastPos2.y = lastPos2.y - 1;
                                if (dY&&dInc)
                                        lastPos2.y = lastPos2.y + 1;
                                }
                                if (plain1.getGlobalBounds().contains(lastPos2.x * 30
+ 115, lastPos2.y * 30 + 215)) {
                                        if (playerPlain[lastPos2.x + lastPos2.y * 10])
{
                                        for (int i = 0; i < ships.size(); i++)
                                                if
(ships[i].getGlobalBounds().contains(lastPos2.x * 30 + 103, lastPos2.y * 30 + 203))
                                                {
                                                        CPUisSet = true;
                                                }
                                        if (CPUisSet)
                                        {
                                                mark.setTextureRect(IntRect(331,
0, 30, 30));

        player.pships[CPUnumber].Damage();
                                                if
(!player.pships[CPUnumber].isAlive) {
                                                        player.killShip();

        lastResultC.setString("Killed");
                                                        Aimed = false;
                                                        dX = false; dY = false;
dDec = false; dInc = false; shootMiss = false;
                                                        for (int i = 0; i <
player.pships[CPUnumber].shipSurrounding.size(); i++) {
                                                                if
((player.pships[CPUnumber].shipSurrounding[i].x > -1) &&
(player.pships[CPUnumber].shipSurrounding[i].y > -1) &&
(player.pships[CPUnumber].shipSurrounding[i].x < 10) &&
(player.pships[CPUnumber].shipSurrounding[i].y < 10))

        playerPlain[player.pships[CPUnumber].shipSurrounding[i].x +
player.pships[CPUnumber].shipSurrounding[i].y * 10] = false;
                                                        }
                                                }
                                                else {

        lastResultC.setString("Damaged");
                                                        if (!dX && !dY)
                                                                switch (randPoint)
{
                                                                case 0:
```

```
dDec = true;

dInc = true;

dDec = true;

dInc = true;

30, 31, 31));

char(lastPos2.y + 48);

lastPos2.y * 30 + 200);

30 + 100, lastPos2.y * 30 + 200);

* 10]) {
```

```
                                dX = true;
                                break;
                        case 1:
                                dX = true;
                                break;
                        case 2:
                                dY = true;
                                break;
                        case 3:
                                dY = true;
                                break;
                        default:
                                break;
                        }

                }
                timer = true;
        }
        if (!CPUisSet)
        {
                mark.setTextureRect(IntRect(300,

                playerCPU.turn = false;
                player.turn = true;
                lastResultC.setString("Missed");
                timer = true;
                if (dX || dY) {
                        swap(dDec, dInc);
                        shootMiss = true;
                }
        }
        char sym = char(lastPos2.x + 65), num =

        lastTurn = num; lastTurn += sym;
        lastTurnC.setString(lastTurn);


        mark.setPosition(lastPos2.x * 30 + 100,

        shootIndicator.setPosition(lastPos2.x *

        xmark.push_back(mark);
        CPUisSet = false;
        if (playerPlain[lastPos2.x + lastPos2.y
```

```
                                                playerPlain[lastPos2.x +
lastPos2.y * 10] = false;
                                        }
                                        if (shootMiss) {
                                                lastPos2.x = lastPos1.x;
                                                lastPos2.y = lastPos1.y;
                                                shootMiss = false;
                                        }

                                }
                                else {
                                        if (dX || dY) {
                                                swap(dDec, dInc);
                                                lastPos2.x = lastPos1.x;
                                                lastPos2.y = lastPos1.y;
                                        }
                                }
                        }
                        else
                        {
                                if (dX || dY) {
                                        swap(dDec, dInc);
                                        lastPos2.x = lastPos1.x;
                                        lastPos2.y = lastPos1.y;
                                }
                                else {}
                        }

                }
        }

        ////////////////////////////////////////////////////////////////////////////
//Конец алгоритма
                if (player.getNumberOfShips() != 0)
                        if (!player.pships[0].isAlive && !player.pships[1].isAlive &&
!player.pships[2].isAlive && !player.pships[3].isAlive) {
                                for (int i = 0; i < 100; i++) {
                                        if (playerPlain[i])
                                                if (i < 10)
                                                        if (i = 0)
                                                                if (!playerPlain[i + 1] &&
!playerPlain[i + 10])
                                                                        playerPlain[i] =
false;
                                                                else {}
                                                        else
                                                                if (i = 9)
                                                                        if (!playerPlain[i
- 1] && !playerPlain[i + 10])
```

```
        playerPlain[i] = false;
                                                                    else {}
                                                                else
                                                                    if (!playerPlain[i
- 1] && !playerPlain[i + 1] && !playerPlain[i + 10])

        playerPlain[i] = false;
                                                                    else {}
                                                            else
                                                        if (i > 89)
                                                                if (i = 90)
                                                                    if (!playerPlain[i
+ 1] && !playerPlain[i - 10])

        playerPlain[i] = false;
                                                                    else {}
                                                                else
                                                                    if (i = 99)
                                                                        if
(!playerPlain[i - 1] && !playerPlain[i - 10])

        playerPlain[i] = false;
                                                                        else {}
                                                                    else
                                                                        if
(!playerPlain[i - 1] && !playerPlain[i + 1] && !playerPlain[i - 10])

        playerPlain[i] = false;
                                                                        else {}
                                                                else
                                                                    if (i % 10 == 0)
                                                                    if (!playerPlain[i
- 10] && !playerPlain[i + 10] && !playerPlain[i + 1])

        playerPlain[i] = false;
                                                                    else {}
                                                                else
                                                                    if (i % 10 == 9)
                                                                        if
(!playerPlain[i - 10] && !playerPlain[i + 10] && !playerPlain[i - 1])

        playerPlain[i] = false;
                                                                        else {}
                                                                    else
                                                                        if
(!playerPlain[i - 10] && !playerPlain[i + 10] && !playerPlain[i - 1] && !playerPlain[i +
1])

        playerPlain[i] = false;
```

```
                                }
                        }

        ////////////////////////////////////////////////////////////////////////////
/////Конец хода компьютера


        ////////////////////////////////////////////////////////////////////////////Задержка
хода компьютера
                    if (timer) {
                            sf::Clock clock;
                            while (time < sec)
                                    time = clock.getElapsedTime().asSeconds();
                            timer = false;

                    }

                    time = 0;

        ////////////////////////////////////////////////////////////////////////////Конец
задержки

                    mainWin.clear(sf::Color::Color(169, 169, 169));
                    mainWin.draw(plain1);
                    mainWin.draw(plain2);
                    for (int i = 0; i < ships.size(); i++)
                            mainWin.draw(ships[i]);
                    mainWin.draw(textInformation1);
                    mainWin.draw(textInformation2);
                    mainWin.draw(lastTurnP); mainWin.draw(lastTurnC);
mainWin.draw(lastResultP); mainWin.draw(lastResultC);
                    mainWin.draw(TextYOU);
                    mainWin.draw(TextCPU);
                    mainWin.draw(Sym1);
                    mainWin.draw(Sym2);
                    mainWin.draw(Num1);
                    mainWin.draw(Num2);
                    mainWin.draw(indicator);
                    mainWin.draw(indicator2);
                    if (!xmark.empty())
                            for (int i = 0; i < xmark.size(); i++)
                                    mainWin.draw(xmark[i]);
                    mainWin.draw(shootIndicator);
                    mainWin.display();
        }
}


void TheGame::StageFinal() {
```

```
	sf::RenderWindow finalWin(sf::VideoMode(500, 250), "The Final!",
sf::Style::Close);
	finalWin.setFramerateLimit(100);

	Image winlose;
	winlose.loadFromFile("image/WinLose.png");
	Texture losewin;
	Texture restartButton;
	Sprite buttonRestart;
	restartButton.loadFromFile("image/TextInf.png");
	buttonRestart.setTexture(restartButton);
	buttonRestart.setTextureRect(IntRect(300, 45, 88, 31));
	Sprite wonlost;
	winlose.createMaskFromColor(Color::White);
	losewin.loadFromImage(winlose);
	wonlost.setTexture(losewin);
	if (win) {
		wonlost.setTextureRect(IntRect(0, 0, 336, 47));
	}
	else
	{
		wonlost.setTextureRect(IntRect(0, 48, 404, 47));
	}

	wonlost.setPosition(80, 95);
	buttonRestart.setPosition(200, 200);

	while (finalWin.isOpen())
	{
		Vector2i pixelPos = Mouse::getPosition(finalWin);
		sf::Event event;

		while (finalWin.pollEvent(event))
		{
			if (event.type == sf::Event::Closed) {
				goOn = false;
				finalWin.close();
			}

		}
		if (event.type == Event::MouseButtonPressed)
			if (event.key.code == Mouse::Left)
				if (buttonRestart.getGlobalBounds().contains(pixelPos.x,
pixelPos.y))
				{
					goOn = true;
					finalWin.close();

				}
		if (win) {
```

```
                finalWin.clear(Color::Red);
            }
            else
            {
                finalWin.clear(Color::Color(0, 0, 139));
            }
            finalWin.draw(buttonRestart);
            finalWin.draw(wonlost);
            finalWin.display();
        }
}

void TheGame::Texturising() {

        plain1.setTexture(Text);
        plain1.setTextureRect(IntRect(0, 0, 301, 301));
        plain1.setPosition(100, 200);

        plain2.setTexture(Text);
        plain2.setTextureRect(IntRect(0, 0, 301, 301));
        plain2.setPosition(600, 200);


        ////////////////////////////////////////////////////////////////////////////
///////////////Спрайты кораблей
        for (int i = 0; i < 10; i++)
        {
            Sprite ship;
            if (i < 4) {
                ship.setTexture(shipText);
                ship.setPosition(600 + i * 40, 200);
            }
            else
                if (i < 7) {
                    ship.setTexture(shipText);
                    ship.setTextureRect(IntRect(0, 0, 30, 60));
                    ship.setPosition(600 + (i - 4) * 70, 240);
                }
                else {
                    if (i < 9) {
                        ship.setTexture(shipText);
                        ship.setTextureRect(IntRect(0, 0, 30, 90));
                        ship.setPosition(600 + (i - 7) * 100, 320);
                    }
                    else {
                        ship.setTexture(shipText);
                        ship.setTextureRect(IntRect(0, 0, 30, 120));
                        ship.setPosition(600, 420);
                    }
                }
```

```
                        ships.push_back(ship);
        }
        ////////////////////////////////////////////////////////////////////////////////
///////////////Конец спрайтов кораблей
}

TheGame::~TheGame()
{
}
```

Файл Main.cpp

```
#include "TheGame.h"
int main()
{
        bool game = true;//Отвечает за начало и "рестарт" игры
        while (game) {
                TheGame seabattoru;

                seabattoru.Texturising();
                seabattoru.StageOne();
                if (seabattoru.goOn) {
                        seabattoru.StageTwo();
                        seabattoru.StageThree();
                }
                if (seabattoru.goOn) {
                        seabattoru.StageFinal();
                }
                game = seabattoru.goOn;
        }
}
```