БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

КУРСОВОЙ ПРОЕКТ

по дисциплине «Задача оптимизации расстановки кораблей и выбора координат для обстрела противника в игре “Морской бой”»

Тема: «Задача оптимизации расстановки кораблей и выбора координат для обстрела противника в игре “Морской бой”»

|  |  |
| --- | --- |
| Исполнитель: | студент 4 курса группа 10701217  Хоронеко А.Ю. |
| Руководитель проекта: | профессор, д.т.н.  Прихожий А. А. |

Минск 2020

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Моделирование и оптимальное проектирование технических систем»

Тема: «Задача оптимизации расстановки кораблей и выбора координат для обстрела противника в игре “Морской бой”»

|  |  |
| --- | --- |
| Исполнитель: | студент 4 курса группа 10701217  Хоронеко А.Ю. |
| Руководитель проекта: | профессор, д.т.н.  Прихожий А. А. |

Минск 2020

**ОГЛАВЛЕНИЕ**

[1 ВВЕДЕНИЕ 5](#_Toc58928716)

[2 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc58928717)

[3 МОДЕЛЬ РЕШЕНИЯ ЗАДАЧИ 7](#_Toc58928718)

[4 МЕТОД РЕШЕНИЯ ЗАДАЧИ 10](#_Toc58928719)

[5 АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ 11](#_Toc58928720)

[6 ИНФОРМАЦИОННАЯ МОДЕЛЬ 13](#_Toc58928721)

[7 АРХИТЕКТУРА ПРОГРАММЫ 15](#_Toc58928722)

[8 ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ 16](#_Toc58928723)

[9 ТЕСТОВЫЙ ПРИМЕР. ТЕСТИРОВАНИЕ ПРОГРАММЫ 19](#_Toc58928724)

[10 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ 21](#_Toc58928725)

[11 ЗАКЛЮЧЕНИЕ 25](#_Toc58928726)

[12 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc58928727)

[13 ПРИЛОЖЕНИЕ 27](#_Toc58928728)

# ВВЕДЕНИЕ

Игра «морской бой» достаточно хорошо известна и популярна. Практически каждый школьник в тот или иной период своей жизни играл в эту игру. В последние годы в связи с появлением компьютеров и новых обучающих и развивающих программ вновь возрос интерес к ней. Если набрать запрос о поиске игры в Интернет, то поисковая машина выдаст несколько тысяч ссылок. Здесь и реклама, и различные варианты игры, и качественные исследования оптимальных стратегий игры и т. д. Но мало кто знает о том, что эта игра имеет серьезное научное и практическое приложение, и для ее анализа могут быть использованы современные математические и компьютерные методы. В качестве при мера такого приложения можно привести проблему эффективного поиска записей в больших базах данных, обладающих сложной многоуровневой структурой.

В данной работе мы будем рассматривать только классические правила игры, которые более подробно будут раскрыты в главе 3 Модель решения задачи.

Целью курсового проекта стало нахождение оптимизации расстановки кораблей, для наиболее эффективного запутывания врага, и оптимизации выбора клеток для выстрелов, для наиболее эффективного поиска вражеских кораблей.

# ПОСТАНОВКА ЗАДАЧИ

В ходе выполнения курсового проекта необходимо разработать приложение, которое будет решать задачу об оптимизации расстановки кораблей и оптимизации выбора клеток обстрела.

Краткие требования к приложению:

— построение оптимизированной конфигурации кораблей на поле;

— возможность совершения оптимизированных выстрелов;

— возможность графически визуализировать конфигурацию и выстрелы;

— возможность сравнить действие оптимизированного и неоптимизированного алгоритмов расстановки и выстрела;

Условия разработки приложения:

― среда разработки – Microsoft Visual Studio 2019;

― язык программирования – C#.

# МОДЕЛЬ РЕШЕНИЯ ЗАДАЧИ

В данной работе мы будем рассматривать только классические правила игры.

Играют два игрока. У каждого из них имеется поле размером 10 на 10 клеток, на котором перед началом игры располагаются корабли.

Корабли бывают следующих видов:

* единичный (4 шт.). Состоит из одной клетки поля;
* двойной (3 шт.). Состоит из двух клеток поля;
* тройной ( 2 шт.). Состоит из трех клеток поля;
* четверной (1 шт. ). Состоит из четырех клеток поля.

Корабли представляют собой горизонтальную либо вертикальную конфигурацию клеток, расположенных по линии. Сгибать корабли запрещено. В процессе игры игроки делают выстрелы по полям расположения кораблей друг друга по очереди. Если один из игроков промахнулся – попал в поле, не занятое кораблем противника – то ходить начинает соперник. Пока промаха не произошло, стрельба игрока продолжается

При игре в «Морской бой» выигрыш одной из сторон обеспечивается уничтожением кораблей противника за меньшее число шагов, чем у противоположной стороны. Можно выделить четыре параметра игры: алгоритм атаки игрока 1, алгоритм расстановки кораблей игрока 1, алгоритм атаки игрока 2, алгоритм расстановки кораблей игрока 2. Цель работы – найти оптимальную стратегию одной из сторон. Поскольку игрок 1 не может влиять на выбор атакующего алгоритма игрока 2, а также на выбор расстановки кораблей игрока 2, и наоборот, то оптимизировать следует атакующие алгоритмы и расстановки одной из сторон без учета действий другой стороны. Поэтому методы теории игр в данном случае мало применимы. Тем не менее, возможно представить ситуацию, когда защищающийся игрок до конца игры не выставляет единичный корабль. И лишь на последнем ходе размещает его, тем самым определяя конфигурацию расстановки кораблей с максимальным количеством шагов для ее уничтожения при заданных расстановках остальных кораблей. Аналогичным образом, атакуемый игрок может размещать корабли в момент игры в тех клетках, по которым ведет стрельбу противник, – здесь определяется минимальное число шагов для данного алгоритма атаки. В работе необходимо получить усредненные результаты для множества алгоритмов защиты и атаки, поэтому используется аппарат математической статистики. Для удобства обозначения примем нумерацию строк матрицы поля расположения кораблей за X (X = 0,..,9), колонок матрицы поля за Y (Y = 0,...,9).

Для упрощения сравнения алгоритмов, мы будем использовать не полноценную игру, а рассматривать только процесс расстановки кораблей на поле и алгоритмы обстрела вражеского поля без ответа от противника.

# МЕТОД РЕШЕНИЯ ЗАДАЧИ

Существуют несколько популярных алгоритмов расстановки кораблей:

* расстановка кораблей по краям;
* группировка кораблей с одного бока поля;
* расположение кораблей случайно в одном направлении;
* расположение кораблей случайно в двух направлениях;
* полностью случайное расположение кораблей;
* группировка кораблей в центре поля.

*Расстановка кораблей по краям поля*.

Этот алгоритм расстановки выделяет полосу у края поля шириной в одну клетку. То есть координаты: X = 0, Y = 0,..,9; X = 9, Y = 0,...,9; X = 0,...,9, Y = 0; X = 0,...,9, Y = 9. В рамках выделенной полосы размещаются все корабли, кроме единичных, случайным образом. После размещения всех кораблей, кроме единичных, размещаются единичные корабли по всему полю в доступных клетках.

*Группировка кораблей с одной стороны поля.*

Этот алгоритм размещения заключается в выделении с произвольной стороны поля области шириной четыре клетки (например, X = 0,...,3, Y = 0,...,9). Затем в выделенной области произвольным образом размещаются все корабли, кроме единичных. Ограничение накладывается на ориентацию корабля (вертикальность или горизонтальность). Если выделены крайняя правая или левая области поля, то ориентация кораблей вертикальна, если верхняя или нижняя – то горизонтальна. Если корабли в указанных начальных ограничениях ширины в 4 клетки невозможно разместить на одном из шагов алгоритма размещения, то ширина поля увеличивается на единицу до тех пор, пока все корабли, кроме единичных, не будут размещены. В последнюю очередь размещаются единичные корабли на всем пространстве поля случайным образом.

*Расположение кораблей случайно в одном направлении.*

Этот алгоритм выбирает ориентацию всех кораблей (вертикальность или горизонтальность) и размещает корабли случайно по всему полю в заданной ориентации.

*Расположение кораблей случайно в двух направлениях.*

Алгоритм реализован так, чтобы после размещения на поле половина клеток кораблей, кроме единичных, принадлежала вертикальным кораблям, другая половина – горизонтальным. В результате при реализации алгоритма выбирается ориентация одного четверного и двух двойных кораблей, а остальные неединичные корабли имеют другую ориентацию.

*Полностью случайное расположение кораблей.*

В данном алгоритме размещения случайным образом выбирается как ориентация, так и координаты кораблей.

*Группировка кораблей в центре поля.*

Алгоритм сделан так, чтобы максимально сгруппировать корабли в центре поля. Для этого выбирается квадрат в центре поля с координатами X = {4,5}; Y = {4,5}. Делается попытка разместить корабли в данной области. Сперва размещаются единичные корабли, затем двойные, потом тройные и четверной. Если в выделенной области отсутствует место для размещения корабля, ее границы увеличиваются на единицу.

В данной работе для сравнения были выбраны алгоритмы полностью случайного расположения и расстановки кораблей по краям поля.

Так как наибольшую сложность нахождения представляют собой одиночные корабли, то для сравнения работы алгоритмов нам надо сравнить вероятности попадания в одиночные корабли в том или ном случае, т.е. количество свободных клеток на поле, в которые можно разместить одиночные корабли.

Среди наиболее популярных атакующих алгоритмов существуют:

* диагональные алгоритмы,
* линейные алгоритмы (обход змейкой),
* случайные алгоритмы,
* алгоритмы обхода по квадратам.

*Диагональный алгоритм* характеризуется выстрелами по диагоналям, параллельным главной диагонали поля расположения кораблей. В алгоритме используются параметры. Первый из них – Y координата серии выстрелов, второй – шаг для серии выстрелов. Для того чтобы обеспечить полный прострел поля, необходимо иметь несколько серий, характеризующихся параметрами Y координаты первой ячейки выстрела и шагом стрельбы. X координата первой ячейки выстрела всегда равна 0. Когда выбрана ячейка для выстрела, происходит – 9 – стрельба параллельно главной диагонали с постоянным шагом 1. После достижения края поля вновь выбирается первая ячейка для новой диагонали обстрела с учетом шага. Обход осуществляется справа налево. Если для заданного шага и начальной диагонали обстрела выстрелы более невозможны, производится переход к следующему шагу и новой начальной координате отчета диагоналей.

*Линейный алгоритм (обход по змейке)* характеризуется выстрелами снизу вверх с определенным шагом. В качестве варьируемых параметров используются начальная координата (например, (0, 1)) и шаг (например, 5). В дальнейшем идет обстрел первой колонки поля, затем второй, третьей с шагом пять, пока не достигнут конец поля. Например, первый выстрел будет (0, 1), второй (0,6), третий (1,2) и так далее. После достижения конца поля выбираются новые начальная координата и шаг и вновь происходит обстрел. Так до тех пор, пока все корабли не будут убиты.

*Случайный алгоритм* характеризуется полностью случайной стрельбой. Координаты (X,Y) выбираются произвольно в диапазоне 0,...,9.

*Квадратный алгоритм (обход по квадратам*) заключается в обстреле поля противника по квадратам. Сначала обстреливаются четыре угловых точки квадрата, затем четыре соседние к ним по часовой стрелке, после это соседние к обстрелянным на предыдущем шаге и так далее. Варьируемыми параметрами являются размер стороны квадрата и координаты первой точки угла.

В данной работе были использованы диагональный и случайный алгоритмы.

# АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

Для решения задачи оптимизации расстановки был составлен следующий алгоритм:

Так как оптимальных вариантов расположений кораблей на поле существует ограниченное число, то исходными данными для генерации являются варианты расположения кораблей и сочетания кораблей, которые мы можем расположить.

Существует 3 основных варианта расположения кораблей:

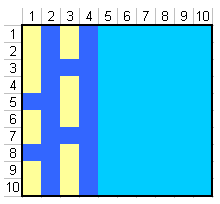
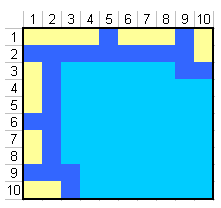
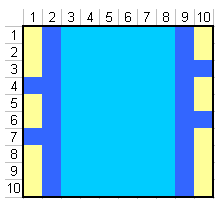


Рисунок 5.1 – Оптимальные расстановки кораблей

Первым шагом алгоритма является определение конфигурации: Г-образной или линейной.

Далее, если было выбрано Г-образное расположение, генерируется матрица из списка кораблей, а после случайно выбирается угол поворота матрицы (0, 90, 180, 270) и с помощью цикла матрица поворачивается на необходимый угол.

Если была выбрана линейная конфигурация, то случайным образом выбирается два сочетания кораблей (3, 3, 2 и 4, 2, 2) и их взаимная конфигурация, а также определяется, будут они размещены вертикально или горизонтально. После определяется местоположение наборов на самом поле и непосредственно размещаются на свои места на поле.

Для решения задачи оптимизации выстрелов был составлен следующий алгоритм:

Сначала выбираем клетку поля, с которой будет начинаться диагональ, далее поднимается от этой клетки диагонально вверх до того момента, пока не упремся в верхнюю или в правую боковую грань, при этом, если мы при диагональном обходе натыкаемся на какую-либо клетку, принадлежащую кораблю, то мы запускаем алгоритм добивания. Далее мы случайным образом выбираем следующую клетку для выстрела из оставшихся свободных клеток.

Добивание производится крестообразно. Сначала делается попытка добивать вверх от переданной точки попадания. Добивание идет последовательно клетка за клеткой, пока не происходит промах. Если при движении вверх было хотя бы одно попадание, то делается вывод о вертикальном расположении корабля. После промаха при движении вверх и в случае установления вертикальности корабля идет движения вниз до первого промаха. После промаха при движении вверх и в случае неопределенности ориентации корабля идет движение вправо. В случае промаха при движении вправо дальнейшее движение продолжается влево от начальной точки попадания. В случае промаха при движении прицела добивания влево начинается движение вниз. Таким образом будут обойдены все варианты расположения кораблей. На каждом шаге добивания делается проверка гипотезы о том, что корабль не уничтожен. Если в результате проверки устанавливается подбитие всех клеток корабля, добивание прекращается, а буферные клетки, то есть клетки, в которых вероятность содержания клеток корабля равна 0 (см. правила игры в модели решения задачи).

Когда мы прошли все диагонали, поле будет выглядеть как на рисунке 5.2.

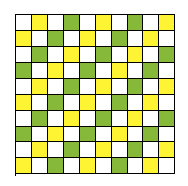


Рисунок 5.2 – Поле после всех диагональных проверок

# ИНФОРМАЦИОННАЯ МОДЕЛЬ

Для программной реализации оптимизации расстановки кораблей был разработан класс PlaygroundUtil, в котором были реализованы методы для:

* Генерации пустого поля размером 10 на 10;
* Генерации конфигурации поля с помощью полностью случайного расположения кораблей (с и без одиночных кораблей)
* Генерации конфигурации поля с помощью оптимального расположения кораблей (с и без одиночных кораблей)
* Добавления буферной зоны около корабля
* Подсчета свободных клеток для одиночных кораблей

Для программной реализации оптимизации выбора клеток выстрелов был разработан класс ShootingUtil, в котором были реализованы методы для:

* Вызова генерации нового поля
* Выбора клетки и выстрела в нее с помощью случайного алгоритма
* Выбор клетки и выстрела в нее с помощью диагонального алгоритма

# АРХИТЕКТУРА ПРОГРАММЫ

Для решения поставленной задачи было разработано оконное приложение реализующее решение задачи оптимизации расстановки кораблей и выбора координат для обстрела противника.

На рисунке 7.1 представлены программные файлы, реализующие разработанную программу.

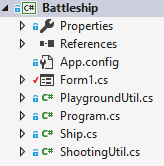
****

Рисунок 7.1 – Структура проекта

# ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

Графический интерфейс пользователя представлен одним окном с несколькими разделами. В первом разделе происходит генерирование различных конфигураций кораблей двумя алгоритмами: алгоритмом случайной расстановки и оптимизированным алгоритмом (рисунок 8.1).

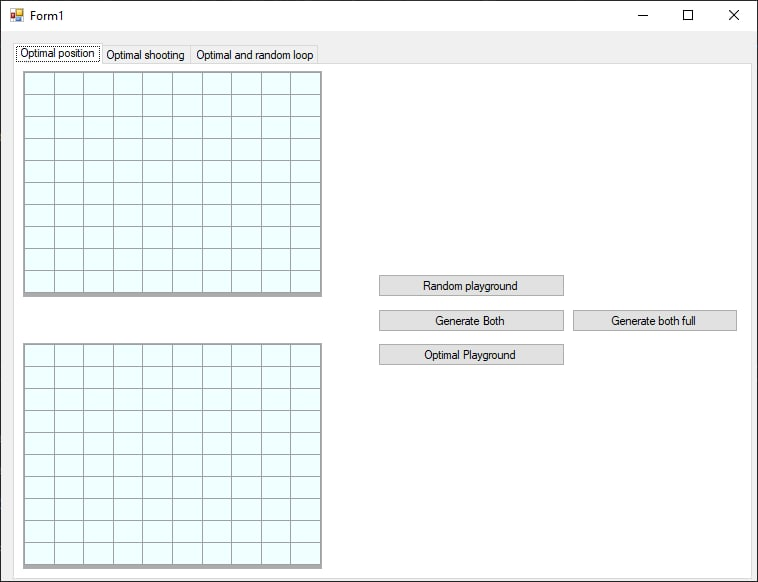


Рисунок 8.1 –Окно программы при загрузке.

На данной форме у пользователя есть возможность сгенерировать поле только первым алгоритмом, только вторым, сразу оба поля двумя алгоритмами. После генерации каким-либо способом неполных матриц, рядом с соответствующим полем появится сообщение о количестве свободных клеток для размещения одинарных кораблей (рисунок 8.2).

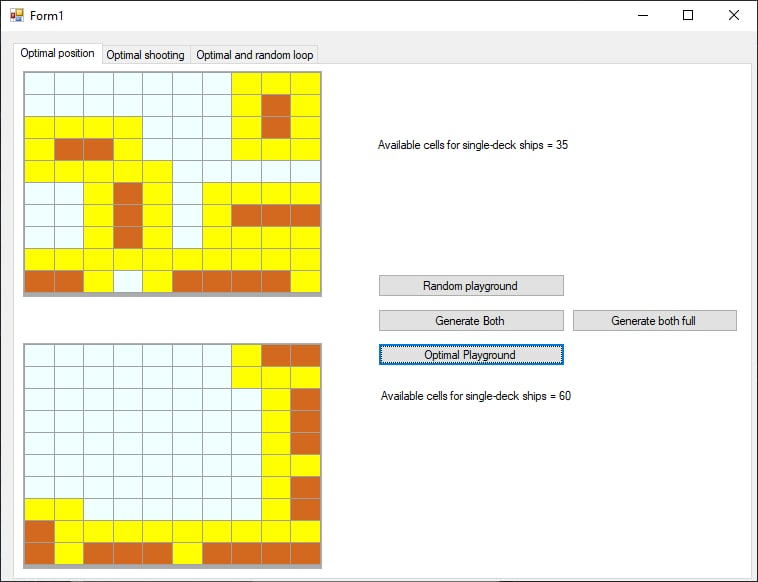


Рисунок 8.2 –Окно после генерации неполных полей.

Также у пользователя есть возможность сгенерировать поле, состоящее из 10 кораблей, т.е. полное поле и использовать их, например, для своей игры (рисунок 8.3).

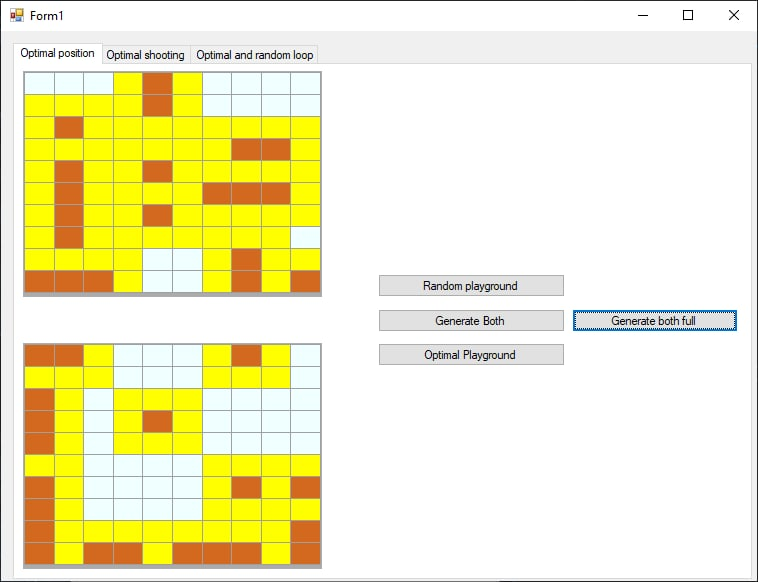


Рисунок 8.3 –Окно после генерации неполных полей.

Во втором разделе представлены два поля, на первом из которых есть возможность сгенерировать матрицу, после чего есть возможность сделать выстрел на второй матрице, так как это происходило бы при реальной игре, т.е. на поле после выстрела отображается клетка, в которую был произведен выстрел, определенным цветом, в зависимости от результата выстрела (голубой – ещё не открытая клетка, желтый – пустой выстрел или буферная зона около корабля, коричневый – клетка корабля) (рисунок 8.4).

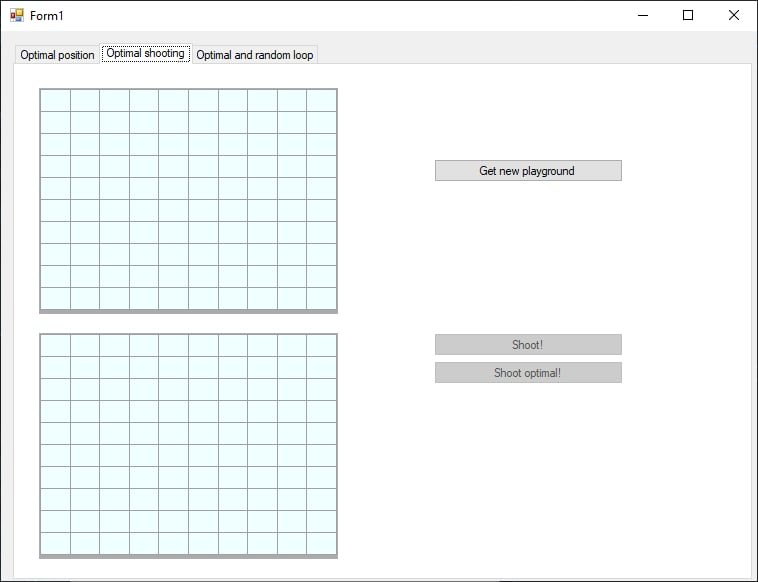


Рисунок 8.4 –Раздел выстрелов по полю.

Пока не будет сгенерирована исходное поле, кнопки для совершения выстрелов не будут доступны. Также, когда был совершен хотя бы один выстрел, кнопка выстрелов другим алгоритмом становится недоступна. (рисунок 8.5).

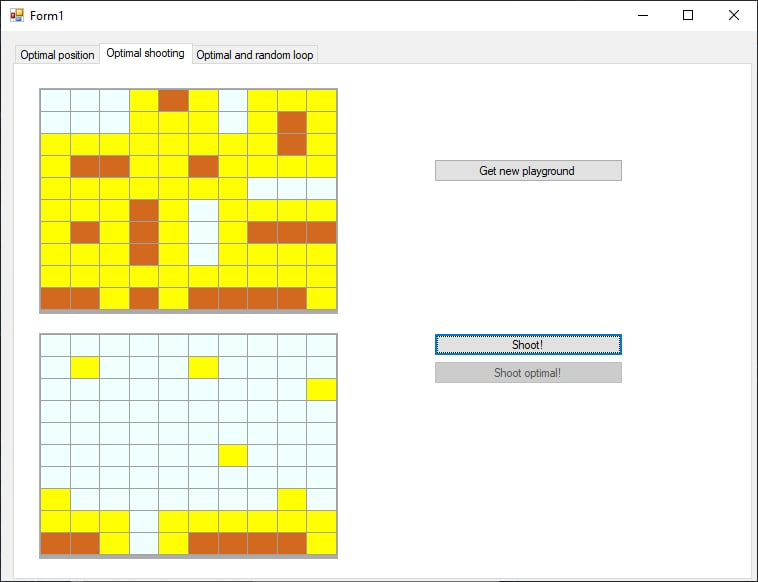


Рисунок 8.5 –Форма после выстрелов алгоритмом случайного выбора.

После того, как все корабли были потоплены, появляется сообщение о завершении игры и количестве выстрелов, понадобившихся для завершения игры (рисунок 8.6.)

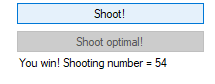


Рисунок 8.6 –Сообщение о завершении игры.

В третьем разделе у пользователя есть возможность запустить цикл с указываемым числом итераций, чтоб получить статистику по случайному и оптимизированному алгоритмам (рисунок 8.7).

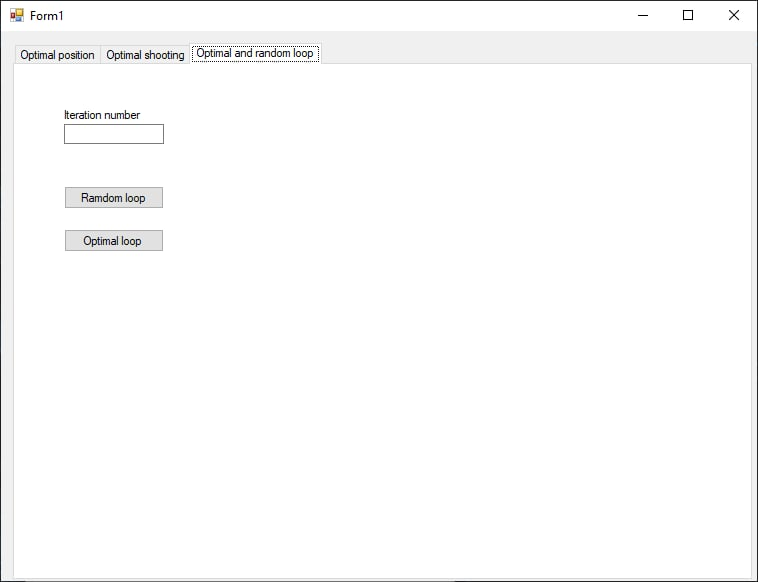


Рисунок 8.7 –Форма статистики для алгоритмов.

На форме есть поле ввода количества итераций и две кнопки для алгоритмов. После ввода какого-то числа и нажатия на любую кнопку, рядом с кнопкой появится сообщение с количеством выстрелов для того или иного алгоритма (рисунок 8.8).

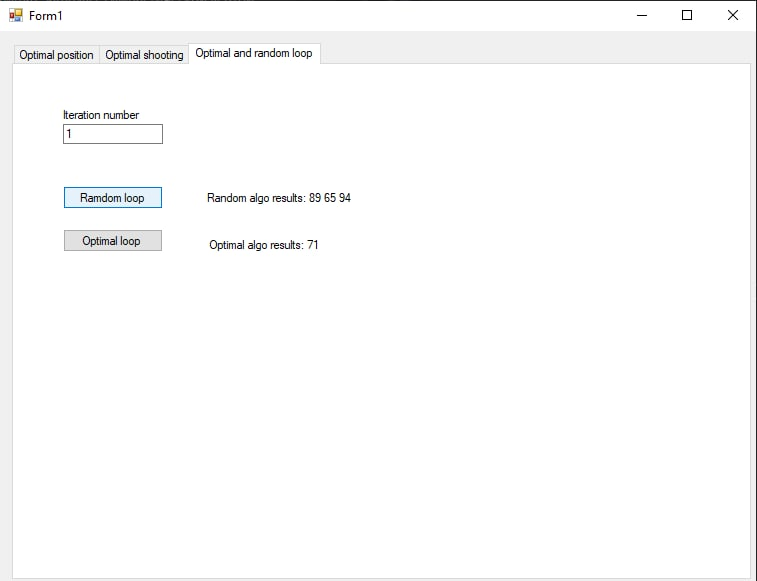


Рисунок 8.8 –Статистика алгоритмов.

# ТЕСТОВЫЙ ПРИМЕР. ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тестирование созданного программного продукта производилось на следующей конфигурации компьютера: AMD Ryzen 3550H, RAM 16 Гб, SSD 256 Гб, ОС Windows 10 Pro.

Для проверки работоспособности данной программы будут проделаны следующие действия:

* Введено неверное значение в разделе статистики;
* Попытка произвести выстрел, когда поле ещё не сгенерировано;

В третьем разделе нашей программы есть поле ввода только для целых чисел, то есть нам надо сообщать пользователю об неверных значениях. Данная проверка совершается при попытке разбора введенной строки в целое число. Если пользователь ввел нечисловые данные, то на экране появится соответствующее сообщение (рисунок 9.1).

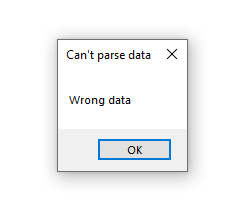


Рисунок 9.1 – Сообщение о неверных данных.

Во втором разделе, при попытке нажать на кнопки выстрелов до генерирования поля ничего не происходит, что означает, что программа отрабатывает корректно (рисунок 9.2). Но как только мы нажимаем на кнопку генерации поля, то кнопки для совершения выстрелов становятся доступными, что опять говорит о корректности работы программы. И при совершении выстрелов, одна из кнопок вновь становится недоступной, что и должно быть при верном выполнении.

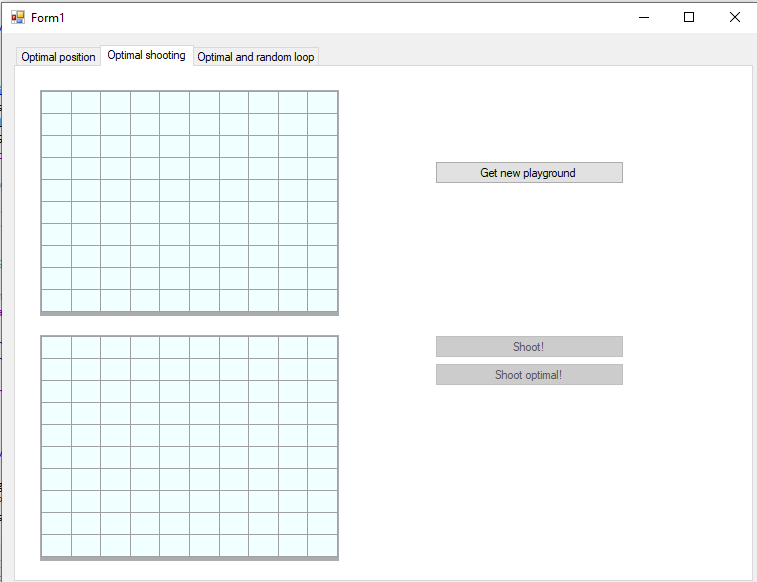


Рисунок 9.2 – Форма алгоритмов оптимизации выстрела до нажатия на кнопку генерации поля.

# ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

После запуска программы отобразится окно с двумя полями. Результат выполнения представлен на рисунке 10.1.

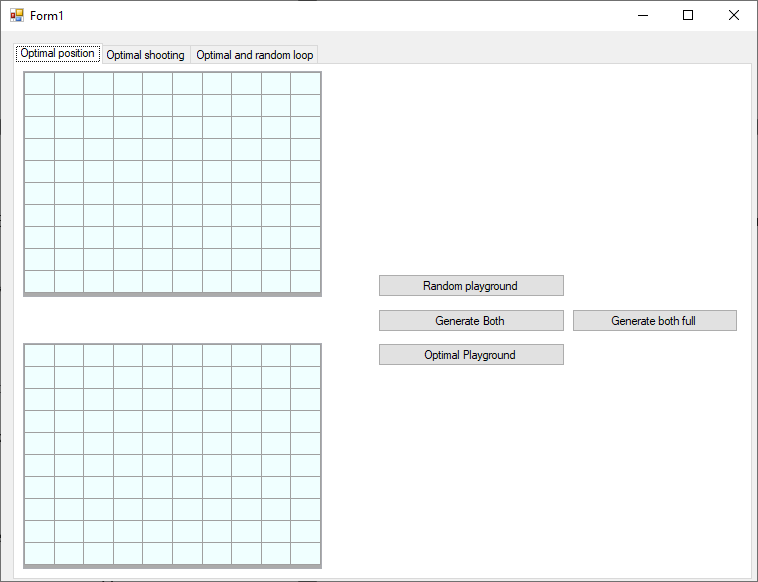


Рисунок 10.1 – Окно программы после запуска

Для простоты использования, на форме есть кнопка, которая вызывает генерацию сразу двух полей. После нажатия на нее отображаются оба поля. Результат выполнения представлен на рисунке 10.2.

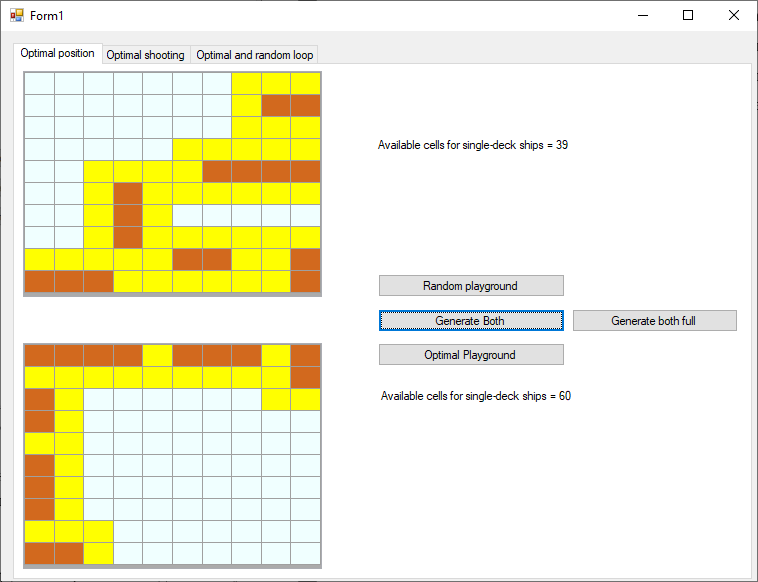


Рисунок 10.2 – Результат построения обоих полей

В результате выполнения видно, что при оптимальной расстановке у нас остается 60 клеток для размещения одинарных кораблей, а при случайной - 39.

При переключении на второй раздел отобразится окно с двумя полями. Результат представлен на рисунке 10.3.

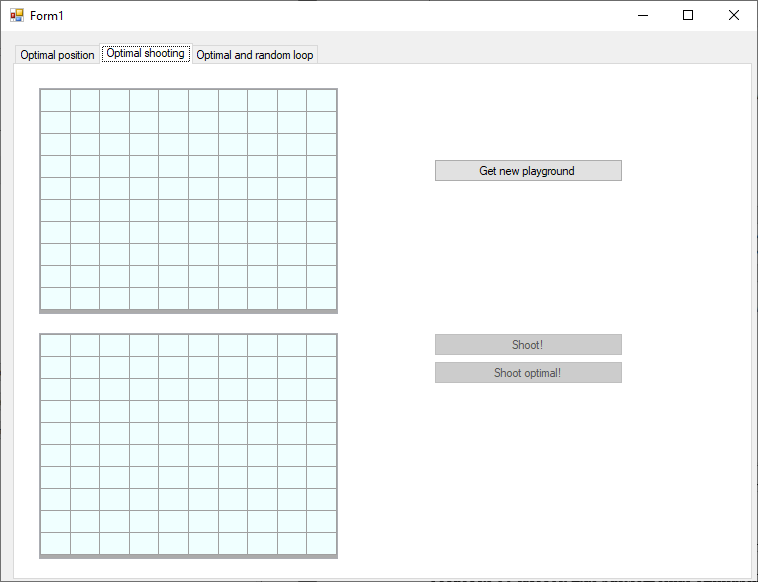


Рисунок 10.3 – Второй раздел

После нажатия на кнопку генерации поля окно программы будет выглядеть, как показано на рисунке 10.4.

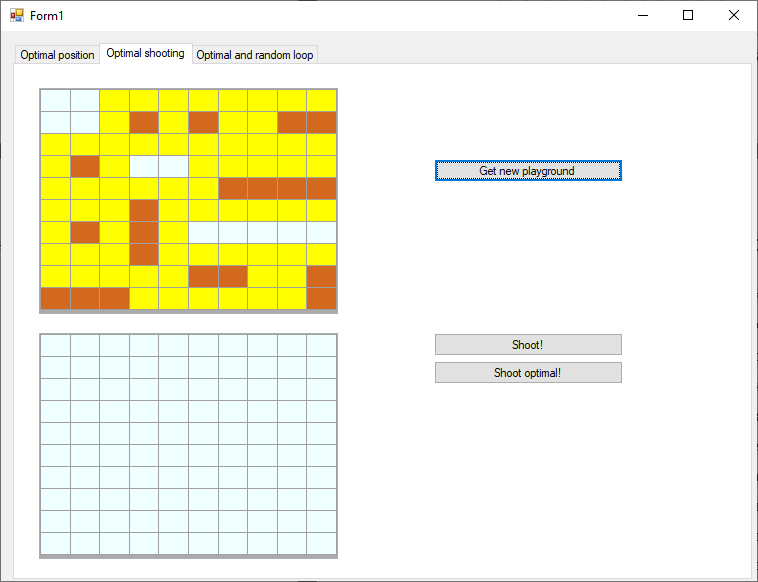


Рисунок 10.4 – Окно после генерации поля

Далее у пользователя есть выбор, каким алгоритмом совершать выстрелы. В данном случае мы выберем оптимальный алгоритм. Результат представлен на рисунке 10.5.

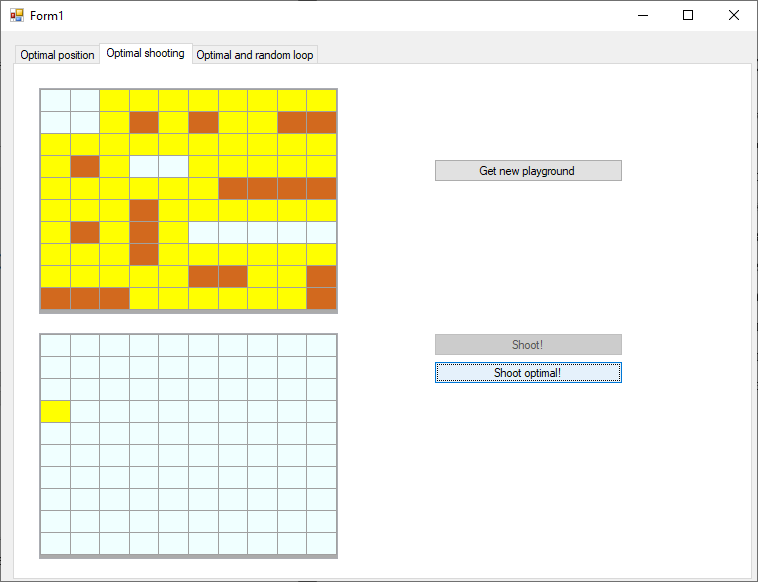


Рисунок 10.5 – Окно после первого выстрела оптимальным алгоритмом

После попадания во все корабли, появляется надпись о завершении игры и количестве затраченных выстрелов для этого (рисунок 10.6)

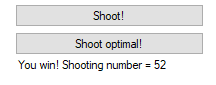


Рисунок 10.6 – Окончание работы алгоритма

# ЗАКЛЮЧЕНИЕ

Результатом выполнения данного курсового проекта является разработанная программа, которая позволяет решить задачу оптимизации расстановки кораблей на поле и выбора оптимальной клетки для выстрела.

Тестирование разработанного приложения на начальных этапах выявило ряд ошибок в реализации, которые были исправлены в последствие. В ходе тестирования было установлено, что программа успешно работает в стандартном режиме эксплуатации, а также при различных нестандартных действиях пользователя. Таким образом, поставленная задача выполнена в полном объеме.

В дальнейшем программа может быть модифицирована добавлением новых алгоритмов, а так же различных вариантов задачи.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гик Е.Я. Занимательные математические игры. –М: Знание, 1987. – 160 с.
2. Дидыч Я.О., Малинецкий Г.Г. Поиск оптимальных алгоритмов действий противников в игре «Морской бой». –М: Препринты ИПМ им. М.В.Келдыша, 2015. № 98. – 16 с.
3. Ляхов А.Ф., Ляхов Ф.А Как правильно играть в «Морской бой». – М: Компьютерные инструменты в образовании, 2002. – 8 с.
4. Веб-сайт Хабрахабр [Электронный ресурс] Режим доступа : https://habr.com

# ПРИЛОЖЕНИЕ

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Battleship

{

public partial class Form1 : Form

{

int[,] playgroundRandom;

int[,] playgroundOptimal;

int[,] shootedPlayground;

int[,] playground;

PlaygroundUtil PlaygroundUtil = new PlaygroundUtil(10);

ShootingUtil shootingUtil = new ShootingUtil();

public Form1()

{

InitializeComponent();

playgroundRandom = PlaygroundUtil.CreateEmptyPlayground();

playgroundOptimal = PlaygroundUtil.CreateEmptyPlayground();

shootedPlayground = PlaygroundUtil.CreateEmptyPlayground();

playground = PlaygroundUtil.CreateEmptyPlayground();

DataGridViewInit(dataGridView1, playgroundRandom);

DataGridViewInit(dataGridView2, playgroundOptimal);

DataGridViewInit(dataGridView3, shootedPlayground);

DataGridViewInit(dataGridView4, playground);

DrawPlayground(dataGridView1, playgroundRandom);

DrawPlayground(dataGridView2, playgroundOptimal);

DrawPlayground(dataGridView3, shootedPlayground);

DrawPlayground(dataGridView4, playground);

Shoot.Enabled = false;

ShootOptimal.Enabled = false;

}

private void dataGridView1\_SelectionChanged(object sender, EventArgs e)

{

dataGridView1.ClearSelection();

}

private void dataGridView2\_SelectionChanged(object sender, EventArgs e)

{

dataGridView2.ClearSelection();

}

private void dataGridView3\_SelectionChanged(object sender, EventArgs e)

{

dataGridView3.ClearSelection();

}

private void dataGridView4\_SelectionChanged(object sender, EventArgs e)

{

dataGridView4.ClearSelection();

}

private void DrawPlayground(DataGridView dataGridView, int[,] playground)

{

int i = 0;

int j = 0;

foreach (DataGridViewRow row in dataGridView.Rows)

{

j = 0;

foreach (DataGridViewCell cell in row.Cells)

{

if (playground[i, j] == 1)

{

cell.Style.BackColor = Color.Chocolate;

}

else if (playground[i, j] == -1)

{

cell.Style.BackColor = Color.Yellow;

}

else

{

cell.Style.BackColor = Color.Azure;

}

j++;

}

i++;

}

}

private void Random\_playground\_Click(object sender, EventArgs e)

{

playgroundRandom = PlaygroundUtil.GenerateRandomPlaygroundWithoutOnes();

DrawPlayground(dataGridView1, playgroundRandom);

label1.Text = "Available cells for single-deck ships = " + PlaygroundUtil.GetCellNumberForOnesShip(playgroundRandom);

}

private void OptimalPlayground\_Click(object sender, EventArgs e)

{

playgroundOptimal = PlaygroundUtil.GenerateOptimalPlaygroundWithoutOnes();

DrawPlayground(dataGridView2, playgroundOptimal);

label2.Text = "Available cells for single-deck ships = " + PlaygroundUtil.GetCellNumberForOnesShip(playgroundOptimal);

}

private void GenerateBoth\_Click(object sender, EventArgs e)

{

Random\_playground\_Click(sender, e);

OptimalPlayground\_Click(sender, e);

}

private void DataGridViewInit(DataGridView dataGridView, int[,] playground)

{

for (int i = 0; i < playground.GetLength(0); i++)

{

string[] row = new string[playground.GetLength(1)];

for (int j = 0; j < playground.GetLength(1); j++)

{

row[j] = "";

}

dataGridView.Rows.Add(row);

}

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void GenerateBothFull\_Click(object sender, EventArgs e)

{

playgroundRandom = PlaygroundUtil.GenerateFullRandomPlayground();

DrawPlayground(dataGridView1, playgroundRandom);

label1.Text = "";

playgroundOptimal = PlaygroundUtil.GenerateFullOptimalPlayground();

DrawPlayground(dataGridView2, playgroundOptimal);

label2.Text = "";

}

private void GetNewPlayground\_Click(object sender, EventArgs e)

{

shootedPlayground = shootingUtil.GetNewShootingPlayground();

DrawPlayground(dataGridView3, shootedPlayground);

Shoot.Enabled = true;

ShootOptimal.Enabled = true;

playground = shootingUtil.GetPlayground();

DrawPlayground(dataGridView4, playground);

}

private void Shoot\_Click(object sender, EventArgs e)

{

playground = shootingUtil.Shoot();

DrawPlayground(dataGridView4, playground);

if (shootingUtil.result == 3)

{

label3.Text = "You win! Shooting number = " + shootingUtil.shootingNumber;

}

ShootOptimal.Enabled = false;

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

int count = Int32.Parse(textBox1.Text);

shootingUtil.LoopGenerationRandom(count);

label4.Text = "Random algo results: " + GetStringFromList(shootingUtil.randomNumber);

Shoot.Enabled = false;

} catch(FormatException)

{

MessageBox.Show("Wrong data", "Can't parse data", MessageBoxButtons.OK);

}

}

private String GetStringFromList(List<int> list)

{

String result = "";

foreach(var i in list)

{

result += i;

result += " ";

}

return result;

}

private void ShootOptimal\_Click(object sender, EventArgs e)

{

playground = shootingUtil.ShootOptimal();

DrawPlayground(dataGridView4, playground);

if (shootingUtil.result == 3)

{

label3.Text = "You win! Shooting number = " + shootingUtil.shootingNumber;

ShootOptimal.Enabled = false;

}

Shoot.Enabled = false;

}

private void OptimalLoop\_Click(object sender, EventArgs e)

{

try

{

int count = Int32.Parse(textBox1.Text);

shootingUtil.LoopGenerationOptimal(count);

label6.Text = "Optimal algo results: " + GetStringFromList(shootingUtil.optimalNumber);

}

catch (FormatException)

{

MessageBox.Show("Wrong data", "Can't parse data", MessageBoxButtons.OK);

}

}

}

}

Ship.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Battleship

{

class Ship

{

public int X { get; set; }

public int Y { get; set; }

public bool IsVertical { get; set; }

public int Size { get; set; }

public int Position { get; set; }

public List<int[]> Coordinates { get; set; }

public Ship(int size)

{

Size = size;

Coordinates = new List<int[]>();

}

public Ship(int size, int position)

{

Size = size;

Position = position;

}

public Ship(int size, int x, int y, bool isVertical)

{

Size = size;

X = x;

Y = y;

IsVertical = isVertical;

}

}

}

PlaygroundUtil.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Battleship

{

class PlaygroundUtil

{

public static int playground\_size;

int[] ships = { 4, 3, 3, 2, 2, 2, 1, 1, 1, 1 };

int[] shipsWithoutOnes = { 4, 3, 3, 2, 2, 2 };

int[,] linePositions = { { 7, 9 }, { 9, 7 }, { 0, 2 }, { 2, 0 }, { 9, 0 }, { 0, 9 } };

int[] ones = { 1, 1, 1, 1 };

Ship[,] firsLine = {{new Ship(3, 0), new Ship(3, 4), new Ship(2, 8)},

{new Ship(3, 0), new Ship(2, 4), new Ship(3, 7)},

{new Ship(2, 0), new Ship(3, 3), new Ship(3, 7)}};

Ship[,] secondLine = {{new Ship(2, 0), new Ship(2, 3), new Ship(4, 6)},

{new Ship(2, 0), new Ship(4, 3), new Ship(2, 8)},

{new Ship(4, 0), new Ship(2, 5), new Ship(2, 8)}};

Ship[] firstAngle = {new Ship(4, 0, 0, false), new Ship(3, 0, 5, false), new Ship(2, 0, 9, true),

new Ship(3, 2, 0, true), new Ship(2, 6, 0, true), new Ship(2, 9, 0, false)};

Ship[] secondAngle = {new Ship(4, 0, 0, false), new Ship(3, 0, 5, false), new Ship(2, 0, 9, true),

new Ship(2, 2, 0, true), new Ship(3, 5, 0, true), new Ship(2, 9, 0, false)};

Random random = new Random();

public PlaygroundUtil(int size)

{

playground\_size = size;

}

public int[,] CreateEmptyPlayground()

{

int[,] playground = new int[playground\_size, playground\_size];

for (int i = 0; i < playground\_size; i++)

{

for (int j = 0; j < playground\_size; j++)

{

playground[i, j] = 0;

}

}

return playground;

}

public int[,] GenerateFullRandomPlayground()

{

int[,] playground = GenerateRandomPlaygroundWithoutOnes();

GenerateOnesShip(playground);

return playground;

}

private void GenerateOnesShip(int[,] playground)

{

for (int i = 0; i < ones.Length; i++)

{

Ship ship = new Ship(ones[i]);

GenerateCoordinates(playground, ship);

AddShipRegion(playground, ship);

PlaceShipOnPlayground(playground, ship);

}

}

public int[,] GenerateRandomPlaygroundWithoutOnes()

{

int[,] playground = CreateEmptyPlayground();

for (int i = 0; i < shipsWithoutOnes.Length; i++)

{

Ship ship = new Ship(ships[i]);

GenerateCoordinates(playground, ship);

AddShipRegion(playground, ship);

PlaceShipOnPlayground(playground, ship);

}

return playground;

}

private void GenerateCoordinates(int[,] playground, Ship ship)

{

GetCoordinates(ship);

while (CheckPosition(playground, ship))

{

GetCoordinates(ship);

}

}

private void GetCoordinates(Ship ship)

{

ship.IsVertical = random.Next(2) == 0;

if (ship.IsVertical)

{

ship.X = random.Next(playground\_size - ship.Size + 1);

ship.Y = random.Next(playground\_size);

}

else

{

ship.Y = random.Next(playground\_size - ship.Size + 1);

ship.X = random.Next(playground\_size);

}

}

private bool CheckPosition(int[,] playground, Ship ship)

{

int[] region = GetShipRegionCoordinates(ship);

int fromX = region[0];

int toX = region[1];

int fromY = region[2];

int toY = region[3];

for (int i = fromX; i <= toX; i++)

{

for (int j = fromY; j <= toY; j++)

{

if (playground[i, j] == 1)

{

return true;

}

}

}

return false;

}

private void PlaceShipOnPlayground(int[,] playground, Ship ship)

{

if (ship.IsVertical)

{

for (int i = ship.X; i < ship.Size + ship.X; i++)

{

playground[i, ship.Y] = 1;

}

}

else

{

for (int i = ship.Y; i < ship.Size + ship.Y; i++)

{

playground[ship.X, i] = 1;

}

}

}

public void AddShipRegion(int[,] playground, Ship ship)

{

int[] region = GetShipRegionCoordinates(ship);

int fromX = region[0];

int toX = region[1];

int fromY = region[2];

int toY = region[3];

if (toY == 10)

{

toY -= 1;

}

for (int i = fromX; i <= toX; i++)

{

for (int j = fromY; j <= toY; j++)

{

if (playground[i, j] == 0)

{

playground[i, j] = -1;

}

}

}

}

private int[] GetShipRegionCoordinates(Ship ship)

{

int[] region = new int[4];

int kx = ship.IsVertical ? 1 : 0;

int ky = !ship.IsVertical ? 1 : 0;

int fromX = ship.X == 0 ? ship.X : ship.X - 1;

int toX = fromX - kx + ship.Size \* kx + 2;

if (ship.X == playground\_size - 1 || ship.X == 0 || ship.X == (playground\_size - ship.Size \* kx) || toX == playground\_size)

{

toX -= 1;

}

int fromY = ship.Y == 0 ? ship.Y : ship.Y - 1;

int toY = fromY - ky + ship.Size \* ky + 2;

if (ship.Y == 0 || ship.Y == playground\_size - 1 || ship.Y == (playground\_size - ship.Size \* ky) || toY == playground\_size)

{

toY -= 1;

}

region[0] = fromX;

region[1] = toX;

region[2] = fromY;

region[3] = toY;

return region;

}

public int GetCellNumberForOnesShip(int[,] playground)

{

int number = 0;

for (int i = 0; i < playground.GetLength(0); i++)

{

for (int j = 0; j < playground.GetLength(1); j++)

{

if (playground[i, j] == 0)

{

number++;

}

}

}

return number;

}

public int[,] GenerateOptimalPlaygroundWithoutOnes()

{

int[,] playground = CreateEmptyPlayground();

if (IsLineOptimal())

{

GenerateLineOptimalPlaygroung(playground);

}

else

{

GenerateAngleOptimalPlayground(playground);

}

return playground;

}

private bool IsLineOptimal()

{

return random.Next(2) == 0;

}

private void GenerateLineOptimalPlaygroung(int[,] playground)

{

bool isVertical = random.Next(2) == 0;

int firstLineNumber = random.Next(firsLine.GetLength(0));

int secondLineNumber = random.Next(secondLine.GetLength(0));

int positions = random.Next(linePositions.GetLength(0));

int firstLinePosition = linePositions[positions, 0];

int secondLinePosition = linePositions[positions, 1];

PlaceLine(playground, firsLine, firstLineNumber, firstLinePosition, isVertical);

PlaceLine(playground, secondLine, secondLineNumber, secondLinePosition, isVertical);

}

private void PlaceLine(int[,] playground, Ship[,] line, int lineNumber, int linePosition, bool isVertical)

{

for (int i = 0; i < line.GetLength(1); i++)

{

Ship ship = line[lineNumber, i];

ship.IsVertical = isVertical;

if (isVertical)

{

ship.X = ship.Position;

ship.Y = linePosition;

}

else

{

ship.Y = ship.Position;

ship.X = linePosition;

}

AddShipRegion(playground, ship);

PlaceShipOnPlayground(playground, ship);

}

}

private void GenerateAngleOptimalPlayground(int[,] playground)

{

bool isFirstAgngle = random.Next(2) == 0;

PlaceAngle(playground, isFirstAgngle ? firstAngle : secondAngle);

FlipOverPlayground(playground, random.Next(4));

}

private void PlaceAngle(int[,] playground, Ship[] angle)

{

for (int i = 0; i < angle.Length; i++)

{

AddShipRegion(playground, angle[i]);

PlaceShipOnPlayground(playground, angle[i]);

}

}

private void FlipOverPlayground(int[,] playground, int flipNumber)

{

for (int i = 0; i < flipNumber; i++)

{

FlipOver90(playground);

}

}

private void FlipOver90(int[,] playground)

{

int[,] rotated = Copy(playground);

for (int i = 0; i < rotated.GetLength(0); i++)

{

for (int j = 0; j < rotated.GetLength(1); j++)

{

playground[playground.GetLength(0) - 1 - j, i] = rotated[i, j];

}

}

}

public int[,] Copy(int[,] playground)

{

int[,] copy = CreateEmptyPlayground();

for (int i = 0; i < playground.GetLength(0); i++)

{

for (int j = 0; j < playground.GetLength(1); j++)

{

copy[i, j] = playground[i, j];

}

}

return copy;

}

public int[,] GenerateFullOptimalPlayground()

{

int[,] playground = GenerateOptimalPlaygroundWithoutOnes();

GenerateOnesShip(playground);

return playground;

}

}

}

ShootingUtil.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Battleship

{

class ShootingUtil

{

PlaygroundUtil playgroundUtil = new PlaygroundUtil(10);

int[,] shootingPlayground;

int[,] playground;

List<Ship> ships;

List<int[]> emptyCoord;

public int result = 0;

List<int[]> shipList;

bool flag;

int isVertical = 0;

List<int[]> neighbour;

int[] cell;

public int shootingNumber = 0;

public List<int> randomNumber = new List<int>();

public List<int> optimalNumber = new List<int>();

int[,] startList = new int[,] { { 3, 0 }, { 7, 0 }, { 9, 2 }, { 9, 6 }, { 1, 0 }, { 5, 0 }, { 9, 0 }, { 9, 4 }, { 9, 8 } };

int startPosition = 0;

int[] optimalCell;

Random random = new Random();

private void GenerateNewRandomPlayground()

{

shootingPlayground = playgroundUtil.GenerateFullRandomPlayground();

playground = playgroundUtil.CreateEmptyPlayground();

ships = GenerateShipList();

GenerateEmptyCellList();

shipList = new List<int[]>();

shootingNumber = 0;

result = 0;

startPosition = 0;

startPosition = 0;

flag = false;

isVertical = 0;

optimalCell = null;

cell = null;

}

private void CleanPlayground()

{

playground = playgroundUtil.CreateEmptyPlayground();

ships = GenerateShipList();

GenerateEmptyCellList();

shipList = new List<int[]>();

shootingNumber = 0;

result = 0;

}

#region gettind playgrounds

public int[,] GetShootingPlayground()

{

if (shootingPlayground == null) { GenerateNewRandomPlayground(); }

return shootingPlayground;

}

public int[,] GetNewShootingPlayground()

{

GenerateNewRandomPlayground();

return shootingPlayground;

}

public int[,] GetPlayground()

{

return playground;

}

#endregion

#region server part

private int IsHit(int x, int y)

{

int result = 0;

if (shootingPlayground[x, y] == 1)

{

result = 1;

bool flag = false;

for (int i = 0; !flag && i < ships.Count; i++)

{

for (int j = 0; !flag && j < ships[i].Coordinates.Count; j++)

{

if (ships[i].Coordinates[j][0] == x && ships[i].Coordinates[j][1] == y)

{

ships[i].Coordinates.RemoveAt(j);

flag = true;

break;

}

}

if (ships[i].Coordinates.Count == 0)

{

result = 2;

ships.RemoveAt(i);

}

}

if (ships.Count == 0)

{

result = 3;

}

}

Console.WriteLine(result);

return result;

}

private List<Ship> GenerateShipList()

{

int[,] copy = playgroundUtil.Copy(shootingPlayground);

List<Ship> ships = new List<Ship>();

for (int i = 0; i < copy.GetLength(0); i++)

{

for (int j = 0; j < copy.GetLength(1); j++)

{

if (copy[i, j] == 1)

{

ships.Add(GenerateShip(copy, i, j));

}

}

}

return ships;

}

private Ship GenerateShip(int[,] playground, int x, int y)

{

Ship ship = new Ship(0);

ship.X = x;

ship.Y = y;

if (isHorizontal(x, y))

{

while (y < 10 && playground[x, y] == 1)

{

ship.Coordinates.Add(new int[] { x, y });

ship.Size++;

playground[x, y] = 0;

y++;

}

}

else

{

while (x < 10 && playground[x, y] == 1)

{

ship.Coordinates.Add(new int[] { x, y });

ship.Size++;

playground[x, y] = 0;

x++;

}

}

return ship;

}

private bool isHorizontal(int x, int y)

{

if (y != 9)

{

return shootingPlayground[x, y + 1] == 1;

}

else

{

return false;

}

}

#endregion

#region random algo

private void GenerateEmptyCellList()

{

emptyCoord = new List<int[]>();

for (int i = 0; i < playground.GetLength(0); i++)

{

for (int j = 0; j < playground.GetLength(1); j++)

{

if (playground[i, j] == 0)

{

emptyCoord.Add(new int[] { i, j });

}

}

}

}

private int[] GenerateShotingCell()

{

int index = random.Next(emptyCoord.Count);

int[] result = emptyCoord[index];

emptyCoord.RemoveAt(index);

return result;

}

public int[,] Shoot()

{

shootingNumber++;

if (result != 3)

{

CheckNeighborhood();

result = IsHit(cell[0], cell[1]);

CheckResult();

}

return playground;

}

private int[] CheckNeighborhood()

{

int position;

bool fl = false;

if (result == 0 && flag == false && !fl)

{

Console.WriteLine("res 0 fl 0");

cell = emptyCoord[random.Next(emptyCoord.Count)];

fl = true;

}

if (result == 1 && flag == false && isVertical == 0 && !fl)

{

Console.WriteLine("res 1 fl 0 iV 0");

flag = true;

shipList.Add(new int[] { cell[0], cell[1] });

Console.WriteLine(cell[0] + " " + cell[1]);

AddNeighbour();

position = random.Next(neighbour.Count);

cell = neighbour[position];

neighbour.RemoveAt(position);

fl = true;

}

if (result == 0 && flag == true && isVertical == 0 && !fl)

{

Console.WriteLine("res 0 fl 1 iV 0");

position = random.Next(neighbour.Count);

cell = neighbour[position];

neighbour.RemoveAt(position);

fl = true;

}

if (result == 1 && flag == true && isVertical == 0 && !fl)

{

Console.WriteLine("res 1 fl 1 iV 0");

shipList.Add(new int[] { cell[0], cell[1] });

Console.WriteLine(cell[0] + " " + cell[1]);

GetDimension();

GetNextShipCell();

fl = true;

}

if (result == 0 && flag == true && isVertical != 0 && !fl)

{

Console.WriteLine("res 0 fl 1 iV 1");

cell = shipList[0];

GetNextShipCell();

fl = true;

}

if (result == 1 && flag == true && isVertical != 0 && !fl)

{

Console.WriteLine("res 1 fl 1 iV 1");

shipList.Add(new int[] { cell[0], cell[1] });

Console.WriteLine(cell[0] + " " + cell[1]);

GetNextShipCell();

fl = true;

}

return cell;

}

private void CheckResult()

{

if (result == 0)

{

playground[cell[0], cell[1]] = -1;

}

if (result == 2)

{

playground[cell[0], cell[1]] = 1;

shipList.Add(new int[] { cell[0], cell[1] });

int[] first = GetLeftAndHighCell();

playgroundUtil.AddShipRegion(playground, new Ship(shipList.Count, first[0], first[1], isVertical == 1 ? true : false));

neighbour = null;

shipList = new List<int[]>();

isVertical = 0;

flag = false;

result = 0;

GenerateEmptyCellList();

}

if (result == 3)

{

playground[cell[0], cell[1]] = 1;

shipList.Add(new int[] { cell[0], cell[1] });

int[] first = GetLeftAndHighCell();

playgroundUtil.AddShipRegion(playground, new Ship(shipList.Count, first[0], first[1], isVertical == 1 ? true : false));

}

if (result == 1)

{

playground[cell[0], cell[1]] = 1;

}

}

private void AddNeighbour()

{

neighbour = new List<int[]>();

if (cell[0] != 0)

{

neighbour.Add(new int[] { cell[0] - 1, cell[1] });

}

if (cell[0] != 9)

{

neighbour.Add(new int[] { cell[0] + 1, cell[1] });

}

if (cell[1] != 0)

{

neighbour.Add(new int[] { cell[0], cell[1] - 1 });

}

if (cell[1] != 9)

{

neighbour.Add(new int[] { cell[0], cell[1] + 1 });

}

}

private void GetDimension()

{

if (cell[0] == shipList[0][0])

{

isVertical = 2;

}

else

{

isVertical = 1;

}

}

private void GetNextShipCell()

{

if (isVertical == 1)

{

if (shipList[0][0] > shipList[shipList.Count - 1][0])

{

if (cell[0] - 1 != -1 && playground[cell[0] - 1, cell[1]] == 0)

{

cell[0] -= 1;

}

else

{

cell = shipList[0];

cell[0] += 1;

}

}

else

{

if (cell[0] + 1 != 10 && playground[cell[0] + 1, cell[1]] == 0)

{

cell[0] += 1;

}

else

{

cell = shipList[0];

cell[0] -= 1;

}

}

}

else

{

if (shipList[0][1] > shipList[shipList.Count - 1][1])

{

if (cell[1] - 1 != -1 && playground[cell[0], cell[1] - 1] == 0)

{

cell[1] -= 1;

}

else

{

cell = shipList[0];

cell[1] += 1;

}

}

else

{

if (cell[1] + 1 != 10 && playground[cell[0], cell[1] + 1] == 0)

{

cell[1] += 1;

}

else

{

cell = shipList[0];

cell[1] -= 1;

}

}

}

}

private int[] GetLeftAndHighCell()

{

int[] leftAndHightCell = { 9, 9 };

for (int i = 0; i < shipList.Count; i++)

{

if (shipList[i][0] < leftAndHightCell[0] || shipList[i][1] < leftAndHightCell[1])

{

leftAndHightCell = shipList[i];

}

}

return leftAndHightCell;

}

#endregion

public void LoopGenerationRandom(int count)

{

for (int i = 0; i < count; i++)

{

GenerateNewRandomPlayground();

while (result != 3)

{

Shoot();

}

randomNumber.Add(shootingNumber);

}

}

public void LoopGenerationOptimal(int count)

{

for (int i = 0; i < count; i++)

{

GenerateNewRandomPlayground();

while (result != 3)

{

ShootOptimal();

}

optimalNumber.Add(shootingNumber);

}

}

#region optimal algo

public int[,] ShootOptimal()

{

if (result != 3)

{

CheckNeighborhoodOptimal();

result = IsHit(cell[0], cell[1]);

CheckOptimalResult();

}

return playground;

}

private int[] GetNextOptimalCell()

{

if (optimalCell == null)

{

optimalCell = new int[] { startList[startPosition, 0], startList[startPosition, 1] };

}

else

{

if (playground[optimalCell[0], optimalCell[1]] != 0)

{

if (optimalCell[0] != 0 && optimalCell[1] != 9)

{

optimalCell[0] -= 1;

optimalCell[1] += 1;

GetNextOptimalCell();

}

else

{

if (startPosition < startList.GetLength(0) - 1)

{

startPosition++;

optimalCell = new int[] { startList[startPosition, 0], startList[startPosition, 1] };

}

else

{

GenerateEmptyCellList();

optimalCell = emptyCoord[random.Next(emptyCoord.Count)];

}

}

}

}

return optimalCell;

}

private void CheckOptimalResult()

{

shootingNumber++;

if (result == 0)

{

playground[cell[0], cell[1]] = -1;

}

if (result == 2)

{

playground[cell[0], cell[1]] = 1;

shipList.Add(new int[] { cell[0], cell[1] });

int[] first = GetLeftAndHighCell();

playgroundUtil.AddShipRegion(playground, new Ship(shipList.Count, first[0], first[1], isVertical == 1 ? true : false));

neighbour = null;

shipList = new List<int[]>();

isVertical = 0;

flag = false;

result = 0;

GenerateEmptyCellList();

}

if (result == 3)

{

playground[cell[0], cell[1]] = 1;

shipList.Add(new int[] { cell[0], cell[1] });

int[] first = GetLeftAndHighCell();

playgroundUtil.AddShipRegion(playground, new Ship(shipList.Count, first[0], first[1], isVertical == 1 ? true : false));

}

if (result == 1)

{

playground[cell[0], cell[1]] = 1;

}

}

private int[] CheckNeighborhoodOptimal()

{

int position;

bool fl = false;

if (result == 0 && flag == false && !fl)

{

Console.WriteLine("res 0 fl 0");

cell = GetNextOptimalCell();

fl = true;

}

if (result == 1 && flag == false && isVertical == 0 && !fl)

{

Console.WriteLine("res 1 fl 0 iV 0");

flag = true;

shipList.Add(new int[] { cell[0], cell[1] });

Console.WriteLine(cell[0] + " " + cell[1]);

AddNeighbour();

position = random.Next(neighbour.Count);

cell = neighbour[position];

neighbour.RemoveAt(position);

fl = true;

}

if (result == 0 && flag == true && isVertical == 0 && !fl)

{

Console.WriteLine("res 0 fl 1 iV 0");

position = random.Next(neighbour.Count);

cell = neighbour[position];

neighbour.RemoveAt(position);

fl = true;

}

if (result == 1 && flag == true && isVertical == 0 && !fl)

{

Console.WriteLine("res 1 fl 1 iV 0");

shipList.Add(new int[] { cell[0], cell[1] });

Console.WriteLine(cell[0] + " " + cell[1]);

GetDimension();

GetNextShipCell();

fl = true;

}

if (result == 0 && flag == true && isVertical != 0 && !fl)

{

Console.WriteLine("res 0 fl 1 iV 1");

cell = shipList[0];

GetNextShipCell();

fl = true;

}

if (result == 1 && flag == true && isVertical != 0 && !fl)

{

Console.WriteLine("res 1 fl 1 iV 1");

shipList.Add(new int[] { cell[0], cell[1] });

Console.WriteLine(cell[0] + " " + cell[1]);

GetNextShipCell();

fl = true;

}

return cell;

}

#endregion

}

}

**ГРАФИЧЕСКАЯ ЧАСТЬ**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Структура программы оптимизации | | | | | | | | | | |
|  |  |  |  |  | КП–1070121705–2020-01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Хоронеко |  |  | Задача оптимизации расстановки кораблей и выбора координат для обстрела противника в игре “Морской бой”» | Лит | | | Лист | Листов |
| Руковод. | | Прихожий |  |  |  | У |  | 1 | 1 |
| Консульт. | | Прихожий |  |  | 1-40 01 01  БНТУ, г. Минск | | | | |
| Н.контр. | |  |  |  |
| Зав.каф. | | Полозков |  |  |