Лабораторная работа №6
«Морской бой»

Предварительные сведения

Лабораторная работа посвящена отработке следующих навыков:

- знакомство с базовыми принципами объектно-ориентированного программирования;
- написание программы в объектно-ориентированном языке программирования с динамической типизацией;
- оперирование базовыми приёмами программирования на языке Ruby.

В ходе выполнения данной работы необходимо выполнить 35 задания на языке Ruby. Описываемые конструкции связаны с реализацией игры «Морской бой», правила которой (с некоторыми изменениями относительно классической версии) изложены в разделе 1. Непосредственно задания приведены в разделе 4.

Прежде чем приступать к выполнению заданий, ознакомьтесь с замечаниями, изложенными в пункте 3 данного описания.

1. Правила игры «Морской бой»

Классические правила игры можно найти на странице Морской бой.

Изменения, относительно классической версии следующие:

- Размер игрового поля задаётся константой в программе.
- Координаты корабля по вертикали и по горизонтали задаются числами от 0 до размерности поля без единицы.
- Корабль, после попадания в него, не теряет одну из своих клеток, как это происходит в классическом морском бое, а теряет часть своего здоровья. Максимальное здоровье корабля $100 \times n$ единиц, где n — количество клеток у корабля. В случае одного попадания корабль теряет 70 единиц своего здоровья. Корабль тонет, если его здоровье меньше или равно $30 \times n$ единиц (то есть корабль, занимающий одну клетку тонет сразу после первого попадания в него). Таким образом противник может потопить корабль, состоящий из нескольких клеток, попав несколько раз в одну и ту же его точку.
- Перед очередным своим выстрелом игрок может поменять расположение одного из своих кораблей: сдвинуть корабль на одну клетку вперёд/назад или развернуть корабль на 90 или 180 градусов относительно какой либо своей точки.
- Перед очередным выстрелом игрока все его повреждённые корабли (неутонувшие корабли, здоровье которых меньше максимального) увеличивают своё здоровье на 30 единиц (или меньше, если кораблю не хватало до полного выздоровления меньше 30 единиц).
- Как и в классической версии игрок делает несколько выстрелов подряд до тех пор, пока не промахнётся. В одной такой серии выстрелов игрок не имеет права выстрелить в одну и ту же позицию. Но такой выстрел делать не запрещается. Если такой выстрел всё-таки происходит, то он расценивается как промах и ход передаётся второму игроку.

2. Общая схема реализации приложения

Игра оперирует объектами 5-ти классов. Основной класс — класс `Game`. Объект этого класса — конкретная игра между двумя игроками. Игроки — объекты класса `Player`. Каждый игрок (объект) имеет стратегию расстановки кораблей, стратегию выбора координат для выстрелов и стратегию выбора корабля для передвижения. Если имеется два игрока, то можем для них создать объект класса `Game`. Этот объект создаст для игроков экземпляры игровых полей — объекты классов `BattleField` и опросит игроков о расстановке кораблей на предназначенных для них полях.

Класс `BattleField` описывается как наследник класса `Field`, который определяет игровое поле как двумерный массив клеток: если корабль присутствует в какой-то клетке, то соответствующий элемент массива должен содержать ссылку на этот корабль. Класс `BattleField` добавляет информацию о живых кораблях игры: определяются конкретные корабли и хранятся их координаты. Кроме того, здесь же определены методы выстрела по игровому полю и получения результатов выстрелов.

Каждый корабль представляет из себя объект класса `Ship` и характеризуется своим размером, величиной своего здоровья и своими координатами. Установка корабля на поле, в частности, означает, что у корабля появляется характеристика — поле, на котором он установлен. Для корабля определяются, в частности, методы движения по полю. Движение корабля заключается в том, что он меняет свою позицию на своём поле.

Процесс игры ведётся объекта класса `Game`. После каждого хода он спрашивает очередного игрока какой корабль тот хочет «подвинуть» и куда, куда он хочет выстрелить, а после выбора координат выстрела опрашивает поле противника о том, что получится в результате такого выстрела. Результат выстрела сообщается стрелявшему игроку, а в зависимости от «успешности» выстрела выбирается игрок для следующего хода.

Таким образом игрок не имеет прямого доступа к своему игровому полю или полу своего противника, а общается с ними только посредством объекта класса `Game`.

3. Замечания по выполнению заданий

3.1. Необходимый минимум

Для выполнения работы потребуются сведения о следующих функциях, операциях и конструкциях:

- конструкция `class...end`
- конструкция `self`
- конструкция `super`
- конструкция `def...end`
- конструкция `return`
- конструкция `do...end`
- конструкция `while...end`
- конструкция `if...end`
- конструкция `begin...end until`
- конструкция `break`
- конструкции `attr_reader` и `attr_accessor`
- процедуры `print`, `puts`
- конструкция `...?...:...`
- методы для чисел `between?`, `to_s`, `to_f`, `round`, `times`
- методы для строк `to_i`, `strip`
- метод для перечислителей `to_a`,
- методы для массивов `map`, `each`, `zip`, `reduce`, `each_index`, `each_with_index`, `all?`, `any?`, `inject`, `find_all`, `empty?`, `include?`
- методы для массивов `reverse`, `reverse!`, `shuffle`, `shuffle!`, `sort`, `sort!`, `+`, `|`, `push`, `sample`

Информацию об этих и других конструкциях можно найти в конспекте, приведённом на портале.

3.2. Ограничения

При выполнении данной лабораторной работы не накладывается специальных ограничений (кроме ограничений, возможно, оговорённых в задании).

- Нельзя использовать конструкцию цикла `for`, конструкцию `begin`.
- Конструкцию `while` / `unless` можно использовать только там, где это использование наиболее естественно. Использовать `while` / `unless` там, где можно было бы использовать `for` нельзя.
- Разрешается вводить дополнительные вспомогательные функции или параметры/переменные, не оговорённые в условии заданий. Все дополнительные методы должны снабжаться развёрнутыми комментариями, так же как и дополнительные переменные состояний. В программе не должно присутствовать необоснованно введённых методов.
- Имена методов и переменных в решении должны совпадать с именами, указанными в условиях заданий (в частности, речь идёт о переменных, оговорённых в задании).
- Несмотря на то, что явно отсутствуют правила оформления, код должен быть ясным, читабельным, тщательно отформатированным в едином стиле.
- Набор возможных для использования методов и конструкций не ограничен теми, что перечислены в разделе 3.1. Хотя перечисленный набор и является достаточным для выполнения (включив, правда, элементарные методы для сложения/умножения/вычитания и т.п, не перечисленные в предыдущем разделе). Но если вы считаете, что для оптимального выполнения какого-то из заданий желательно использовать особую функцию/конструкцию, не упомянутую выше — поделитесь этим этом на форуме «Лабораторная работа №6».

3.3. Предостережения насчёт решения

Решением каждой задачи должен быть метод/методы/конструкции с указанным именем и возвращающий значение в той форме, в которой спрашивается в задании.

Не следует делать предположений насчёт задания, не сформулированных явно в условии. Если возникают сомнения — задайте вопрос на форуме «Лабораторная работа №6».

Решение данной лабораторной работы не предполагает наличия тестов. Поэтому будьте внимательны при оформлении и отладке решений.

4. Задания

Для создания игры потребуется описать 5 классов. Два из них `Field` и `BattleField` будут описывать игровое поле, класс `Ship` будет описывать корабль, класс `Player` — игрока с его стратегиями. Класс `Game` будет контролировать ход игры для заданных двух игроков.

1. Задание посвящено описанию вспомогательной функции.

   В классе `Array` задайте метод `add b`, который будет добавлять к массиву почленно элементы массива `b`. Таким образом результатом выражения

   ```
   [1,2,3].add [1,2,5]
   ```

   должен быть список `[2,4,8]`. Следует считать, что элементы массивов заданы корректно: длина их равна и к ним применима операция `+`. Выполнять проверку этого факта не следует.

   Решение должно быть без побочных эффектов.

   Типовое решение — 5 строк кода, включая строки `class ... end`.

Сначала опишем класс `Field` для игрового поля. Объект класса `Field` содержит лишь информацию о занятом и свободном пространстве на игровом поле. Его методы позволяют разместить ссылку на объект в клетках игрового поля, запросить информацию о свободном пространстве.

2. Опишите класс `Field`, в который добавьте константу `FieldSize` — размер игрового поля. Присвойте этой константе значение 10 (при загрузке решения на портал должно быть установлено именно это значение константы).

   Добавьте в описание класса метод `initialize` без аргументов, в котором создайте двумерный массив `@field` размерностью игрового поля (определённого константой), заполненный значением `nil`.

   Решение — 7-8 непустых строк кода.

3. Добавьте в `Field` метод класса `size`, выдающий размер игрового поля.

   Решение — 3 строки кода.

4. Добавьте в класс `Field` метод `set!(n, x, y, hor, ship)`, записывающий в массив игрового поля `@field` информацию о корабле, состоящем из `n` клеток. Клетки расположены в ряд. Если `hor==true`, то ряд параллелен первой размерности массива `@field` (оси Ox). Ряд начинается с клетки с координатами `(x,y)` и продолжается в сторону увеличения значений координат. В результате выполнения метода соответствующие клетки массива `@field` должны быть заполнены значением `ship`.

   Например, если для объекта `f` класса `Field` выполнить

   ```
   f.set!(3, 2, 4, false, 5)
   ```

   то элементы `@field[2][4]`, `@field[2][5]` и `@field[2][6]` должны принять значение `5`. А после выполнения

   ```
   f.set!(3, 2, 4, false, nil)
   ```

   эти же элементы массива должны стать равными `nil`.

   Значение, возвращаемое методом, не играет роли и может быть произвольным.

   Предполагаем, что координаты и параметры корабля заданы корректно, чтобы он полностью поместился на поле.

   Типовое решение — 6 строк кода.

Здесь и далее в примерах некоторых методов в качестве корабля используется целочисленное значение. На самом деле, в качестве значения аргумента `ship` может выступать любой объект. В разрабатываемом приложении предполагается, что значения этого параметра — объекты класса `Ship`.

5. Опишите метод `to_s`, отображающий содержание массива `@field` в виде строки. Вывод этой строки на экране должен происходить в виде обрамленной прямоугольной области на экране.

   Первая размерность массива `@field` должна соответствовать направлению сверху вниз (вертикальному), а вторая — слева направо (горизонтальному). Если `@field[i][j]` равно `nil`, то соответствующая позиция отмечается пробелом, а если значение `@field[i][j]` равно `v`, то соответствующая клетка на экране должна быть представлена значением `v.to_s` (предполагается, что это значение должно представлять один символ).

   Обрамление должно представлять линии сверху и снизу, составленные из знаков `-` (минус), линии по бокам, составленные из знаков `|` (вертикальная черта) и знаков `+` (плюс) по углам.

   Корректное выполнение задание должно приводить к получению строки, начинающейся и заканчивающейся знаками `+` (плюс).

   Например, после последовательности команд

```
f = Field.new
f.set!(4,3,3,false,1)
f.set!(3,5,4,true,2)
f.set!(1,9,9,true, 3)
f.to_s
```

   должна получаться строка

```
"+----------+\n|          |\n|          |\n|          |\n|   1111   |\n|          |\n|    2     |\n|    2     |\n|    2     |\n|          |\n|          |\n|         3|\n+----------+"
```

   где `\n` обозначает символ перевода строки.

   Типовое решение состоит из 14 строк кода.

6. Опишите метод `print_field`, выводящий на экран строковое представление массива `@field`. После последнего выведенного символа должен быть выведен символ конца строки.

   Значение, возвращаемое методом, должно быть равно `nil`.

   Например, после последовательности команд

```
f = Field.new
f.set!(4,3,3,false,1)
f.set!(3,5,4,true,2)
f.set!(1,9,9,true, 3)
f.print_field
```

   на экране должно выводиться

```
+----------+
|          |
|          |
|          |
|   1111   |
|          |
|    2     |
|    2     |
|    2     |
|          |
|          |
|         3|
+----------+
```

   Типовое решение состоит из 3 строк кода.

7. Опишите метод `free_space?(n, x, y, hor, ship)`, возвращающий `true`, если корабль с заданными параметрами помещается в игровое поле и не будет касаться других кораблей, т. е. на момент проверки в заданном поле и прилегающих клетках нет ничего, за исключением, возможно, корабля `ship`.

   Так, например, для поля из предыдущего примера, выражение

```
f.free_space?(4,0,3,true,1)
```

должно вернуть `true`, так как в заданной области и на прилегающих клетках нет ничего, кроме самого корабля `1`. Выражение

```
f.free_space?(4,1,3,true,1)
```

должно вернуть `false`, так как при таком расположении корабль будет касаться корабля `2`. Выражение

```
f.free_space?(4,7,2,true,1)
```

должно вернуть `false`, так как заданный корабль не умещается в игровом поле.

Типовое решение — 23 строки.

Следующий набор из 10 заданий посвящён формированию класса `Ship`. Каждый представитель этого класса описывает корабль, состоящий из заданного количества клеток на заданном игровом поле. Корабль имеет информацию о своём здоровье и расположении на игровом поле (если он уже помещён на поле). Методы корабля позволяют получить информацию о нем и изменить некоторые его свойства, включая его расположение на игровом поле.

8. Опишите класс `Ship` включив в него метод `initialize(field, len)`, проводящий инициализацию корабля, занимающего `len` клеток и относящегося к игровому полю `field` (представитель класса `Field`). В этом методе должны создаваться переменные состояния, описывающие корабль:

   `@len` — длина корабля (количество клеток);

   `@myfield` — игровое поле корабля;

   `@maxhealth` — значение максимально-возможного здоровья корабля;

   `@minhealth` — значение здоровья, при котором данный корабль утонет;

   `@health` — текущее здоровье.

   Добавьте к классу getter-ы `len` и `coord`.

   Объем решения — 10 значащих строк.

9. Добавьте метод `to_s`, выдающий строку `"X"` для любого объекта класса `Ship`.

   Объем решения — 3 строки.

10. Добавьте метод `clear`, удаляющий корабль с игрового поля `@myfield`. Предполагается, что в случае вызова этого метода координаты корабля на игровом поле уже сохранены в массиве `@coord`, а параметр, указывающий направление корабля — в переменной `@hor`.

    Объем решения — 3 строки.

11. Опишите метод `set!(x, y, hor)`, устанавливающий корабль в своём игровом поле в заданных координатах и в заданном направлении. Параметры метода играют ту же роль и принимают те же значения, что и одноимённые параметры метода `set!` в классе `Field`. Метод должен сначала проверить, можно ли разместить корабль на игровом поле `@myfield` в заданной позиции. Если размещение возможно, то, в случае если корабль уже размещён в какой-то позиции поля (в случае существования массива `@coord`), удалить корабль с поля (с помощью `clear`). После этого разместить корабль на поле в заданной позиции, сохранить координаты начала корабля и координаты конца корабля в четырёх элементах массива `@coord`, направление корабля (значение `hor`) — в переменной `@hor`.

    Метод должен возвращать `true`, если корабль размещён успешно. Значением метода должно быть `false`, если размещение в заданной позиции невозможно.

    Объем решения — 13 строк.

12. Добавьте метод `kill`. Метод должен удалять корабль с игрового поля и удалять массив `@coord` (связав с этим именем значение `nil`).

    Объем решения — 4 строки.

13. Создайте метод `explode`, описывающий реакцию корабля на попадание в него выстрела противника. Метод должен уменьшать здоровье корабля на 70 единиц, после чего, если оно упало до критического уровня, «утопить» его с помощью метода `kill`.

    Если корабль утонул, метод должен вернуть длину корабля. В противном случае метод возвращает `nil`.

    Решение составит порядка восьми строк.

14. Опишите метод `cure`, «ремонтирующий» корабль. Метод должен увеличивать здоровье на 30 единиц до достижения максимальной величины.

    Объем решения составит четыре строки.

15. Опишите метод `health`, выдающий значение текущего здоровья корабля в процентах. Значение должно быть представлено как величина от 0 до 100 с точностью до двух цифр после запятой.

    Объем решения составит три строки.

16. Опишите метод `move(forward)`, передвигающий корабль на одну клетку вперёд или назад вдоль оси, согласно характеристике `@hor` корабля. Если `forward == true`, то движение производится вперёд, иначе — назад. Метод должен вычислять координаты корабля после перемещения и обращаться к своему методу `set!`.

    Возвращаемым значением должно быть `true` в случае, если движение произошло успешно и `false` — в противном случае.

    Объем типового решения — 7 строк.

17. Опишите метод `rotate(n, k)` разворачивающий корабль относительно его `n`-ой клетки. Здесь `n` номер клетки, отсчитанной от начальной. Считаем, что номер начальной клетки — 1. Целое число `k` задаёт угол поворота: если `k` равно 1 — поворот налево (на 90 градусов против часовой стрелки), если `k` равно 2 — поворот на 180 градусов, если `k` равно 3 — поворот направо (на 90 градусов по часовой стрелке).

    Метод должен высчитывать координаты новой начальной точки и новое направление, после чего обращаться к методу `set!`. В случае, если поворот невозможен или аргументы заданы некорректно, метод должен возвращать `false`, иначе — `true`.

    Для получения координат новой начальной точки корабля можно воспользоваться формулами вычисления точки после поворота на заданный угол относительно другой точки. Если точка A с координатами (x, y) поворачивается против часовой стрелки на угол α относительно точки с координатами (x_c, y_c), то новые координаты (x', y') точки A вычисляются по формулам:

$$\begin{cases} x' = (x - x_c)\cos\alpha - (y - y_c)\sin\alpha + x_c, \\ y' = (x - x_c)\sin\alpha + (y - y_c)\cos\alpha + y_c. \end{cases}$$

    Стоит обратить внимание, при размещении корабля предполагается, что каждая координата его начальной точки не больше соответствующей координаты конечной точки. При некоторых поворотах (например, поворот на 180 градусов) новой начальной точкой корабля становится новое положение конечной точки корабля.

    При выполнении задания нельзя использовать вещественнозначные функции, включая `sin` и `cos`. Не возбраняется каждый из случаев поворота рассмотреть отдельно.

    Объем форматированного типового решения — порядка 34 строки.

Следующий набор заданий посвящён формированию класса `BattleField`. Представители этого класса представляют собой игровое поле с кораблями—объектами класса `Ship`. Объект класса `BattleField` кроме информации об игровом поле имеет доступ к информации о кораблях. Такой объект — основной объект, отвечающий за состояние поля в игре. Объект `BattleField` взаимодействует со связанными с ним кораблями: он может напрямую изменять их свойства, так же как и корабли, при желании, могут напрямую изменять свойства относящегося к ним объекта `BattleField`.

18. Создайте класс `BattleField`, наследник класса `Field`, включив в него константу `Ships` — список длин кораблей в начальный момент игры. Присвойте этой константе значение `[4,3,3,2,2,2,1,1,1,1]` (при загрузке решения на портал должно быть установлено именно это значение константы).

    Добавьте метод `newships`, формирующий массив `@allships` новых кораблей — представителей класса `Ship` для каждого элемента списка `Ships`. В качестве игрового поля конструктору класса `Ship` экземпляр класса `BattleField` должен передавать себя.

Опишите метод `initialize`, который помимо действий определённых в родительском классе, запускает метод `newships`.

Количество значащих строк в типовом решении — 10.

19. Добавьте в класс метод `fleet`, выдающий информацию об имеющихся кораблях в виде массива пар `[i, len]`, где `i` — номер корабля в массиве `@allships`, а `len` — его длина.

    Типовое решение — три строки.

20. Опишите метод `place_fleet pos_list`, размещающий корабли массива `@allships` на игровом поле. Метод должен получать массив четвёрок `[i, x, y, hor]`, в котором `i` — номер соответствующего корабля в массиве `@allships`, а величины `x`, `y` и `hor` — параметры его размещения для передачи методу `set!` экземпляра класса `Ship`.

    Если в соответствии с переданным массивом какой-то корабль разместить нельзя, или после обработки массива остались неразмещенными какие-то корабли, то метод должен убрать все корабли с игрового поля (с помощью `kill` экземпляра `Ship`) и выдать `false` или `nil`.

    Если метод возвращает значение не равное `false` или `nil`, то это должно сигнализировать о том, что все корабли успешно размещены на игровом поле.

    Объем метода в типовом решении — 12 строк.

21. Создайте метод `remains`, выдающий информацию об имеющихся кораблях в массиве `@allships` в виде четвёрок `[i, coord, len, health]`, где `i` — номер корабля в массиве `@allships`, `coord` — четвёрка координат его размещения, `len` — его длина и `health` — его здоровье в процентах.

    Типовое решение — три строки.

22. Опишите метод `refresh`, который обновляет содержимое массива `@allships`, оставляя в нем только те корабли, которые размещены в игровом поле `@field`.

    Объем решения — три строки.

23. Опишите метод `shoot c`, где `c` — пара координат `[x, y]`. Метод должен описывать реакцию на выстрел по точке игрового поля с заданными координатами.

    Метод должен обратиться к элементу массива `@field` с заданными координатами. Если значение в нем отсутствует, то это означает промах и метод должен вернуть строку `"miss"` в качестве результата.

    Если по заданным координатам в массиве `@field` находится корабль, то для него должен запуститься метод `explode`. Если этот метод вернёт числовое значение n, т. е. корабль убит, то массив `@allships` должен быть обновлен и возвращена строка `"killed` n`"`. Если `explode` вернёт `nil`, то это означает, что корабль ранен и нужно вернуть строку `"wounded"`.

    Типовое решение занимает 13 строк.

24. Опишите метод `cure`, который активирует метод `cure` для всех кораблей массива `@allships`.

    Решение — три строки.

25. Опишите метод `game_over?`, выдающий `true`, если на поле (в массиве `@allships`) не осталось кораблей.

    Решение — три строки.

26. Опишите метод `move l_move`, меняющий положение корабля на поле. Параметр `l_move` которого представляет тройку целых чисел `[i, move_t, dir]`, в которой `i` — номер корабля в массиве `@allships`, `move_t` указывает, какое движение должен сделать корабль, а `dir` — параметр этого движения. В частности, если значение `move_t` — от 1 до 3, то заданное движение — поворот в соответствующем направлении, а `dir` — номер клетки, относительно которой осуществляется поворот. Если значение `move_t` иное, то заданное движение — сдвиг. Если при этом `dir` равно единице, то — сдвиг вперед. В противном случае — назад.

    Метод должен для заданного корабля активировать метод, отвечающий за соответствующее перемещение и вернуть его реакцию.

    Типовое решение занимает 9 строк.

Следующий набор заданий посвящён формированию класса `Player`. Представитель этого класса отвечает за поведение игрока в ходе игры. Основные методы класса представляют собой стратегию действия игрока: стратегию размещения кораблей на поле перед началом игры, стратегию хода в части перемещения кораблей на игровом поле, стратегию выбора координат для очередного выстрела. Объект класса может накапливать статистику о ходе игры, которую он может использовать при выборе стратегии. Объекты класса `Player` не могут вносить изменения в игровое поле напрямую, а лишь могут отвечать на запросы извне.

27. Опишите класс `Player`, в который добавьте accessor `manual`.

    Добавьте к классу метод `reset`, который инициализирует две переменные `@allshots` и `@lastshots` пустыми массивами. Массив `@lastshots` будет содержать в себе информацию о текущей серии выстрелов игрока. Массив `@allshots` будет представлять собой список прошедших серий выстрелов игрока.

    Добавьте к классу метод `initialize(name, manual)`. Метод должен присвоить переменным `@name` и `@manual` значения соответствующих параметров, присваивать переменной `@lastsample` значение `[1, 0]` и запускать метод `reset`. Параметр `manual` должен быть необязательным. Его значением по умолчанию должно быть `true`.

    Переменная `@name` должна содержать строку — имя игрока. Значение `@manual` определяет, в каком режиме играет игрок: если `@manual` истина, то игра осуществляется в ручном (диалоговом) режиме, игрок задаёт каждый свой ход через консоль. В случае, если `@manual` — ложь, игра ведётся в автоматическом режиме.

    Переменная `@lastsample` отвечает за смещение координат выстрела в автоматическом режиме игры. Здесь этой переменной присваивается начальное значение.

    Объем решения — 13 строк.

28. Опишите метод `to_s`, который для игрока выдаёт строку, представляющую его имя. Объем решения — 3 строки.

29. Опишите метод `random_point`, возвращающий пару `[x,y]` координат случайной точки игрового поля.

30. Опишите метод `place_strategy ship_list`, выдающий план размещения кораблей из списка `ship_list`. Параметр метода — список пар `[i, len]`, где `i` — номер корабля, а `len` — его длина.

    Метод должен выдавать массив четвёрок `[i, x, y, hor]`, где `i` — номер соответствующего корабля, а величины `x`, `y` и `hor` — параметры его размещения. Размещение должно быть случайным, но корректным.

    Считаем, что длины заданных кораблей таковы, что их последовательное размещение на поле в порядке от самого длинного до самого короткого не вызывает трудностей и не требует шагов отката.

    Типовое решение занимает 20 строк.

31. Создадим методы `hit message` и `miss`, которые будут изменять статистику выстрелов в переменных `@allshots` и `@lastshots`.

    Предполагаем, что каждый элемент массива `@lastshots` представляет пару `[[x,y], message]`. Кроме того, считаем, что переменная `@shot` будет содержать пару координат `[x, y]` последнего выстрела игрока.

    Запуск метода `hit message` будет производиться, когда последний выстрел попал в какой-то корабль, а `message` — сообщение об этом. Метод должен добавить пару `[@shot, message]` в конец списка `@lastshots`.

    Запуск метода `miss` будет производиться, когда последний выстрел не попал в корабль. Метод должен добавить пару `[@shot, "miss"]` в конец списка `@lastshots`, получившийся список добавить в конец списка `@allshots` и присвоить переменной `@lastshots` новый пустой массив.

32. Метод `shot_strategy` должен выдавать и сохранять в переменной `@shot` координаты нового выстрела. Часть этого метода, отвечающая за ручной ввод координат для выстрела, уже приведена в шаблоне решения. Вам необходимо «дописать» метод, добавив выбор координат для выстрела в автоматическом режиме.

    В нашей стратегии координаты будут подбираться только за счёт анализа последних выстрелов в текущей серии (выстрелов после последнего промаха), т. е. выстрелов, информация о которых хранится в `@lastshots`. Будем действовать по следующим правилам.

    1) Если выстрел является первым выстрелом в серии или предыдущий выстрел потопил корабль, то делаем выстрел по случайной точке (полученной с помощью `random_point`).

2) Если предыдущий выстрел ранил корабль, и это был первый выстрел по этому кораблю в данной серии выстрелов (т. е. это был первый выстрел в серии или выстрел после потопления другого корабля), то в `@shot` сохранить координаты случайным образом выбранной соседней клетки. При этом в переменной `@lastsample` сохранить пару `[` δ_x `,` δ_y `]`, где δ_x и δ_y — величины смещения по осям относительно предыдущей точки.
3) Если предыдущий выстрел уже второй раз ранил корабль в этой серии выстрелов, то в качестве `@shot` выбрать новую точку, сместившись от предыдущего значения `@shot` на величины, сохраненные в `@lastsample`.
4) Если на шаге 2) или шаге 3) в переменной `@shot` была сохранена точка вне границ игрового поля, то следует поменять значения в массиве `@lastsample` на значения с обратным знаком (т. е. начать смещаться по той же линии, но в другую сторону). После этого в `@shot` записать координаты, полученные из координат последнего выполненного выстрела с сохраненным смещением.
5) Если в результате предыдущих шагов в `@shot` сохранились координаты, по которым в этой серии уже был сделан выстрел, то запустить метод `shot_strategy` рекурсивно.

Следует обратить внимание, что результатом метода должно быть значение `@shot`.

Часть необходимого куска кода в типовом решении занимает 17 строк.

33. Метод `ship_move_strategy remains` получает список всех «живых» кораблей, представленных четвёрками `[i, coord, len, health]`, где `i` — номер корабля, `coord` — четвёрка координат его размещения на игровом поле, `len` — длина корабля и `health` — его здоровье в процентах.

    Метод должен решить, какой корабль и как необходимо переместить. Результат должен выдаваться в формате, принимаемом методом `move` объекта класса `BattleField` в качестве аргумента. То есть результат должен быть представлен в виде тройки `[i, move_t, dir]`, в которой `i` — номер корабля, `move_t` указывает, какое движение должен сделать корабль, а `dir` — параметр этого движения. В частности, если значение `move_t` — от 1 до 3, то заданное движение — поворот в соответствующем направлении, а `dir` — номер клетки, относительно которой осуществляется поворот. Если значение `move_t` иное, то заданное движение — сдвиг. Если при этом `dir` равно единице, то — сдвиг вперёд. В противном случае — назад.

    В шаблоне решения уже приведена часть метода, отвечающая за ручной выбор перемещения. Вам нужно реализовать часть метода, отвечающую за автоматический подбор.

    Реализуем подбор перемещения очень наивным методом. Из списка `remains` выберем корабль с наименьшим здоровьем. После этого случайным образом сгенерируем параметры перемещения: в качестве `move_t` возьмём случайное целое от 0 до 3, а в качестве `dir` — случайное целое от 1 до длины корабля. Сформируем получившуюся тройку значений и выдадим ее в качестве результата.

    Объем описанного куска кода в типовом решении — 4 строки.

Последний блок заданий касается описания класса `Game`. Объект этого класса является связующим звеном между участниками игры и следит за ходом ее выполнения. Объект `Game` отправляет запросы игрокам и игровым полям и перенаправляет ответы на них.

34. Опишите класс `Game`, в котором разместите методы **`initialize`**`(player_1, player_2)` и `reset p`.

    Метод **`initialize`**`(player_1, player_2)` должен получать в качестве аргументов два представителя класса `Player`. В методе должна инициализироваться переменная `@game_over` значением **`false`**. Кроме того, должен формироваться рабочий массив `@players` из двух элементов, в котором для каждого игрока содержится значение вида `[player_i, bfield_i, count_i]`, где

    `player_i` — ссылка на соответствующего игрока (один из аргументов **`initialize`**);

    `bfield_i` — представитель класса `BattleField` — игровое поле игрока `player_i` (создается при инициализации);

    `count_i` — количество выстрелов, сделанных игроком. При инициализации должно равняться нулю.

    После формирования массива `@players` для каждого его элемента `p` запускается `reset p`. Перед окончанием метода **`initialize`** следует случайным образом перемешать элементы массива `@players`.

    В методе `reset p` должна производиться начальная установка игрока и соответствующего игрового поля. Параметр `p` представляет массив из трёх элементов `[player, bfield, count]`, в котором на нулевой позиции стоит игрок, а на первой — его игровое поле. Алгоритм работы метода следующий:

1) Выдать на экран сообщение `" `$name$` game setup"`, где $name$ — имя игрока `player`;
2) Для игрока `player` запустить метод `reset`;
3) Получить список кораблей игрового поля `bfield` с помощью метода `fleet`.
4) Передать список, полученный на предыдущем шаге, методу `place_strategy` игрока `player` и получить от него план размещения этих кораблей на игровом поле.
5) Передать план размещения методу `place_fleet` игрового поля `bfield`.
6) Если размещение произошло успешно, то выдать на экран сообщение `"Ships placed"`. В противном случае поднять исключение с помощью команды

```
raise "Illegal ship placement"
```

Возвращаемый результат метода роли не играет.

Объем решения — порядка 20 строк.

35. Опишите метод `start`, управляющий ходом игры. Метод должен работать с массивом `@players`, относительно которого считает, что на начальной позиции в массиве располагается информация, касающаяся игрока, чей ход является очередным (будем этого игрока называть очередным игроком).

    Метод должен представлять собой цикл, выполняющийся до тех пор, пока `@game_over` не примет истинное значение. Перед выполнением цикла следует инициализировать переменную `lastshots` пустым массивом. В этом массиве будем сохранять координаты выстрелов игрока в течение одной серии выстрелов. В теле цикла должны выполняться следующие действия:

    1) Увеличить на единицу счётчик выстрелов очередного игрока.
    2) Вывести на экран сообщение `"Step `n` of player `$name$`"`, в котором $name$ — имя очередного игрока, а n — номер его очередного выстрела.
    3) Добавить здоровье всем кораблям очередного игрока, запустив метод `cure` его игрового поля.
    4) Получить список `remains` кораблей игрового поля.
    5) Передать список, полученный на предыдущем шаге, методу `ship_move_strategy` игрока, тем самым получить описание перемещения.
    6) Передать описание, полученное на предыдущем шаге, методу `move` игрового поля (не заботясь о результате).
    7) Получить от метода `shot_strategy` очередного игрока координаты нового выстрела.
    8) Добавить координаты выстрела в массив `lastshots` и передать эти координаты методу `shoot` игрового поля второго игрока.

       Если координаты выстрела уже есть в массиве `lastshots`, то выдать сообщение `"Illegal shot"`, после чего, не запуская метод `shoot`, считать, что он выдал результат `"miss"`.
    9) Выдать на экран сообщение `"[ `x`, `y`] `res`"`, где x и y — координаты выстрела, res — результат, выданный методом `shoot` на предыдущем шаге.
    10) Если результат метода `shoot` — `"miss"`, то запустить для игрока метод `miss`, поменять местами элементы в массиве `@players` и заменить значение переменной `lastshots` на пустой массив.

        В противном случае запустить метод `hit` res для игрока, обновить значение переменной `@game_over` на результат `game_over?` игрового поля второго игрока и, в случае, если этот результат не ложь, вывести сообщение `"Player `$name$` wins!"`, где $name$ — имя очередного игрока.

Общий объем решения, включая строки, предварительно заданные в файле, должен составить порядка 500 строк.

5. Запуск игры

Для начала игры нужно создать два объекта для игроков, которые нужно передать методу `new` класса `Game`, после чего для созданной игры запустить метод `start`.

Например, следующий код должен начинать игру, в которой один из игроков — компьютер, а второй — человек:

```
p1 = Player.new("Ivan",false)
p2 = Player.new("Feodor")
g = Game.new(p1, p2)
g.start
```