Проект "Разработка стратегии в стиле боя в игре "Герои Меча и Магии 3" против искусственного интеллекта". (PJ)

1. GeneratePresetImpl

Алгоритм

1. Сначала **сортируется** входной список `unitList` по убыванию эффективности.
2. Затем в цикле (do-while) многократно пробегают по всему списку, пытаясь добавить каждого юнита, пока есть очки и не достигнут лимит.
3. При добавлении нового юнита генерируется случайная позиция (x,y), и если клетка не занята, юнит ставится туда.

Сложность

1. **Сортировка** списка на n элементов:

   $$O(n \log n)$$

2. **Цикл** добавления (do-while):
   - В каждом полном проходе мы **итеративно** обходим весь список из n юнитов.
   - За один такой проход мы можем добавить некоторое количество юнитов (в худшем случае — несколько).
   - Общее **количество** добавленных юнитов ограничено:

- m — общее число, которое физически влезает в лимит очков и лимит по типам.

- В **худшем случае** мы можем сделать до m таких «успешных добавлений» (когда хотя бы один юнит на проходе добавился). Каждый проход обходится в $O(n)$, и таких проходов может быть порядка m.

Итого получаем:

$O(m \times n)$

3. **Поиск свободной клетки**:
   - Предположим, что эта операция (проверка `usedPositions.contains(...)` и добавление) занимает в среднем O(1) (операции с `HashSet` работают за O(1) в среднем).
   - Следовательно, влияние этой части на итоговую асимптотику не меняет результат.
4. **Итого получается:**

   $O(n \log n) \ + \ O(m \times n) \approx O(m \times n)$

2. SimulateBattleImpl

Алгоритм

1. Получаем список юнитов игрока и компьютера.
2. Пока и у игрока, и у компьютера есть живые юниты:
   - Сортируем **каждую** армию по убыванию атаки, где сложность будет $O(n \log n)$, где n — общее число юнитов в одной армии или суммарно — см. ниже).
   - Формируем список ходов (чередуем игрока и компьютер).
   - Каждый юнит в полученном списке (размер порядка n) совершает атаку.
   - Удаляем мёртвых ($O(n)$).

3. Повторяем раунды.

Сложность

- Пусть n — **суммарное** количество юнитов (игрок + компьютер). Тогда в худшем случае у нас примерно по $N/2$ юнитов у каждой стороны.
- **Сортировка** по убыванию атаки в начале каждого раунда:

  $$O((n/2)\log(n/2)) \approx O(n \log n)$$
- **Формирование** «чередующегося» списка:

  $O(n)$
- **Проход** по списку для атак:

  $O(n)$
- **Удаление** мёртвых:

  $O(n)$

Итого на **один раунд** получаем

$$O(n \log n) + O(n) + O(n) \;=\; O(n \log n)$$

В худшем случае, если за один раунд погибает очень мало юнитов (например, 1-2), максимальное число раундов пропорционально N. Поэтому общее время:

$$O(n) \;\times\; O(n \log n) \;=\; O(n^2 \log n)$$

3. SuitableForAttackUnitsFinderImpl

Алгоритм

1. Армия разбита на строки.
2. Для **каждой строки** либо идём **справа налево** (если атакует левая армия), либо **слева направо** (если атакует правая армия), пока не найдём первого живого юнита. Добавляем его и прерываем цикл по строке.

Сложность

- Пусть общее число строк — n, а в одной строке может быть до m юнитов.
- Каждый **ряд** мы сканируем **линейно** (может быть в худшем случае до m проверок, пока не найдём живого).
- Суммарно по всем n строкам получаем:

  $O(n \times m)$

4. UnitTargetPathFinderImpl

Алгоритм

1. Дана двумерная сетка `fieldWidth` × `fieldHeight`.
2. Используется Breadth First Search — **алгоритм обхода графа в ширину** от стартовой клетки (координаты атакующего) к целевой клетке (координаты атакуемого).
3. При обходе помечаем «занятые» клетки как недоступные.
4. Как только дошли до цели — восстанавливаем путь и завершаем.

Сложность

- Алгоритм обхода графа в ширину на сетке размера `fieldWidth` × `fieldHeight` в худшем случае пройдёт **все** клетки 1 раз.

  $O(fieldWidth \times fieldHeight)$
- Каждая клетка добавляется в очередь не более одного раза, и мы смотрим до 4 соседей.
- Таким образом, получается:

  $O(n \times m)$