

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з Основ програмування
(назва дисципліни)
на тему: гра «Морський бій»

Студента 1 курсу, групи ІІ-13
Дойчева Костянтина Миколайовича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник Головченко Максим Миколайович
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____

Національна оцінка: _____

Члени комісії

(підпис)

(вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

(вчене звання, науковий ступінь, прізвище та ініціали)

Київ- 2022 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Об'єктно-орієнтоване програмування

Спеціальність "Інженерія програмного забезпечення"

Курс 1 Група ІІІ-13

Семестр 2

ЗАВДАННЯ на курсову роботу студента

Дойчева Костянтина Миколайовича

(прізвище, ім'я, по батькові)

1. Тема роботи Гра "Морський бій"

2. Строк здачі студентом закінченої роботи 30.08.2022

3. Вихідні дані до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Дата видачі завдання 10.02.2022

КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
|----------|---|--------------------------------------|-----------------------------------|
| 1. | Отримання теми курсової роботи | 10.02.2022 | |
| 2. | Підготовка ТЗ | 02.05.2022 | |
| 3. | Пошук та вивчення літератури з питань курсової роботи | 03.05.2022 | |
| 4. | Розробка сценарію роботи програми | 04.05.2022 | |
| 5. | Узгодження сценарію роботи програми з керівником | 04.05.2022 | |
| 6. | Розробка (вибір) алгоритму рішення задачі | 04.05.2022 | |
| 7. | Узгодження алгоритму з керівником | 04.05.2022 | |
| 8. | Узгодження з керівником інтерфейсу користувача | 05.05.2022 | |
| 9. | Розробка програмного забезпечення | 06.05.2022 | |
| 10. | Налагодження розрахункової частини програми | 06.05.2022 | |
| 11. | Розробка та налагодження інтерфейсної частини програми | 07.05.2022 | |
| 12. | Узгодження з керівником набору тестів для контрольного прикладу | 25.05.2022 | |
| 13. | Тестування програми | 26.05.2022 | |
| 14. | Підготовка пояснювальної записки | 05.06.2022 | |
| 15. | Здача курсової роботи на перевірку | 12.06.2022 | |
| 16. | Захист курсової роботи | 15.06.2022 | |
| | | | |
| | | | |
| | | | |
| | | | |

Зміст

| | |
|--|-----------|
| Зміст | 4 |
| АНОТАЦІЯ | 4 |
| ВСТУП | 5 |
| 1 ПОСТАНОВКА ЗАДАЧІ | 6 |
| 2 ТЕОРЕТИЧНІ ВІДОМОСТІ | 7 |
| 2.1 Правила розміщення кораблів (2.1) | 7 |
| 2.2 Пошук і потоплення кораблів супротивника (2.1) | 8 |
| 3 ОПИС АЛГОРИТМІВ | 9 |
| 4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ | 11 |
| 4.1 Діаграма класів програмного забезпечення | 11 |
| 4.2 Опис методів частин програмного забезпечення | 12 |
| 5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ | 15 |
| 5.1 План тестування | 15 |
| 6 ІНСТРУКЦІЯ КОРИСТУВАЧА | 16 |
| 6.1 Робота з програмою | 16 |
| 6.3 Системні вимоги | 23 |
| ВИСНОВОК | 25 |
| ПЕРЕЛІК ПОСИЛАНЬ | 26 |

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: сторінок, рисунки, таблиць, 2 посилання.

Об'єкт дослідження: гра «Морський бій»

Мета роботи: дослідження методів розробки програмного забезпечення, дослідження методів обрахунку ходів на гральному полі.

Вивчено метод розробки програмного забезпечення з використанням принципів ООП. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв'язання кожної з них.

Виконана програмна реалізація гри «Морський бій».

ВСТУП

Дана робота присвячена вивченню розробки програмного забезпечення з використанням парадигми ООП, і стосується написання комп'ютерної гри «Морський бій». Задача полягає у графічному представленні даної гри та реалізації партії з комп'ютером.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде давати змогу грати в гру «Морський бій».

Дано поле 11*11 клітинок і набір кораблів: 5 однопалубних, 4 двопалубних, 3 трипалубних, 2 чотирипалубних, 1 п'ятипалубний.

Кожна палуба займає рівно 1 клітинку. Корабель може розташовуватися тільки по прямій лінії (горизонтально або вертикально). Кораблі не можуть стикатися навіть кутами. Знизу і праворуч від поля дано кількість палуб, які знаходяться в цьому ряду або стовпці. Частина полів може бути заповнена - поле позначено як пусте або містить частину корабля.

Розробити програму для гри в морський бій гравця з комп'ютером. Програма повинна дозволяти розставляти кораблі на ігровому полі, контролювати правильність їх розстановки, давати противникам можливість по чергово робити ходи і видавати відповідні інформаційні повідомлення. Коли в якості одного з гравців виступає комп'ютер, програма повинна аналізувати попередні ходи і наступний робити на основі проведеного аналізу. Ігрові об'єкти повинні бути реалізовані через класи.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Правила розміщення кораблів (2.1)

Ігрове поле — квадрат 11×11 кожного гравця, на якому розміщений флот кораблів. Загалом є п'ятнадцять кораблів:

- 1 корабель — ряд із 5 клітинок
- 2 кораблі — ряд із 4 клітинок
- 3 кораблі — ряд із 3 клітинок
- 4 кораблі — ряд із 2 клітинок
- 5 кораблів — 1 клітинка

При розміщенні кораблі не можуть торкатися один одного кутами.

Поруч зі «своїм» полем креслять «чуже» такогож розміру, лише порожнє. Це ділянка моря, де ходять кораблі супротивника.

2.2 Пошук і потоплення кораблів супротивника (2.1)

Перед початком бойових дій гравці кидають жереб чи домовляються, хто буде ходити першим.

Гравець, що виконує хід, здійснює постріл — називає вголос координати клітини, в якій, на його думку, перебуває корабель суперника, наприклад, «A1!».

Якщо постріл влучив у порожню клітину, то гравець отримає відповідь «Мимо!» і той, хто стріляв, ставить на чужому квадраті в цьому місці крапку, або інший знак, що означає промах. Право ходу переходить до суперника.

Якщо стріляючий влучив у клітину, де розташований однопалубний корабель, то це означає, що корабель противника потоплено. Обидва гравців помічають клітинку спеціальним знаком і гравець, що стріляв, здобуває право на ще один постріл.

Якщо постріл влучив у клітину, де розташований багатопалубний корабель (розміром більше ніж 1 клітинка), то гравець отримує сигнал «Поранив!» або «Влучив!». Гравець, що стріляв, ставить на чужому полі в цю клітину хрестик, а його супротивник ставить хрестик на своєму полі також у цю ж клітку. Влучивший отримує право на ще один хід.

Переможцем вважається той, хто першим потопив усі 15 кораблів противника. Суперники перевіряють ігрові поля один у одного на вірність результатів.

3 ОПИС АЛГОРИТМІВ

3.1 Пошук клітинки для пострілу

Перелік всіх основних змінних та їхнє призначення наведено в таблиці

Таблиця 3.1 – Основні змінні, методи та їхні призначення

| Змінна або метод | Призначення |
|----------------------|---|
| flatened | Масив усіх клітинок дошки |
| availableCells | Масив доступних клітинок для пострілу |
| getIsHit | Метод перевіряючи чи клітинка стріляна |
| onLost | Функція, яка викликатиметься при програші |
| isPrevShotSuccessful | Булева змінна для збереження чи попередній постріл був успішний |
| prevShot | Координати останнього пострілу |
| neighbours | Масив сусідніх клітинок відносно координат попереднього пострілу (prevShot) |
| grid | Об'єкт гральної дошки |
| getNeighbours(cell) | Метод об'єкту grid, повертаючий сусідні клітинки відносно обраної |
| filtered | Масив не стріляних сусідніх клітинок |
| index | Випадково згенероване число для обрання елементу з масиву available cells |
| coordinates | Координати пострілу |

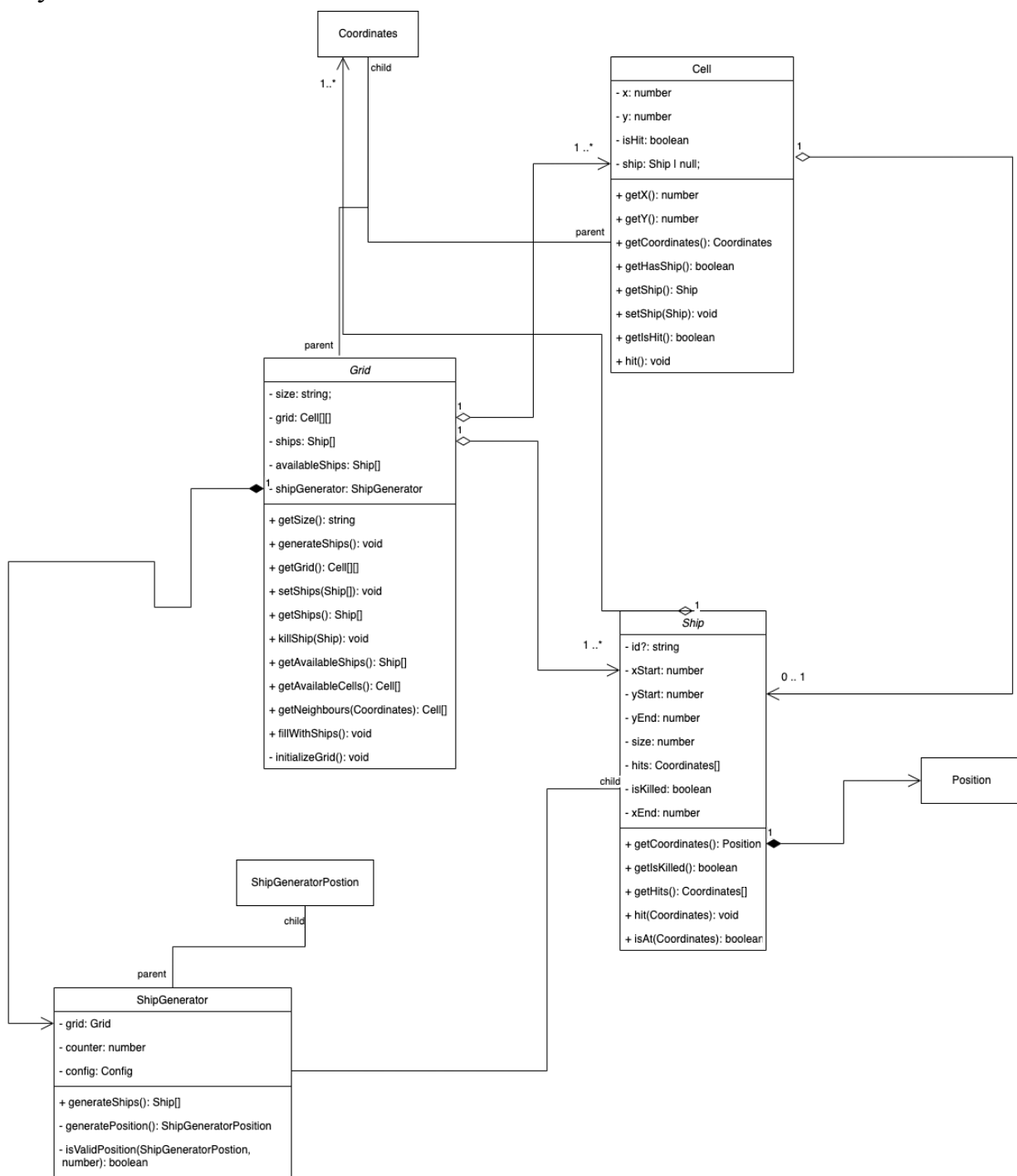
1. ПОЧАТОК
2. Цикл проходу по масиву **flatened**, де **cell** - поточний елемент
 - а. фільтрація масиву **ЯКЩО** анонімна функція(callback) **ПОВЕРТАЄ TRUE**, **ТО** клітинка додається до масиву **availableCells**. В середині цієї функції повертається значення повернуте викликом методу **getIsHit** в класі об'єкті **cell**
3. **ЯКЩО** довжина **availableCells** == 0, **ТО** викликається функція **onLost** **ІНАКШЕ**

- перевіряємо чи кон'юнкція значень **isPrevShotSuccessful** і **prevShot** повертає TRUE.
4. ПРИ УМОВІ ЩО кон'юнкція значень **isPrevShotSuccessful** і **prevShot** повертає TRUE,
записуємо у змінну **neighbours** значення повернуте з методу **getNeighbours** у об'єкта **grid**. ФІЛЬТРУЄМО масив **neighbours** за допомогою анонімних ф-цій (callback-ів) перевіряючи чи поточний елемент масиву **neighbours** НЕ стріляний.
 5. ЯКЩО довжина масиву **filtered** БІЛЬША за 0, ПЕРЕЗАПИСУЄМО змінну **availableCells** значенням **neighbours**.
 6. ІНІЦІАЛІЗУЄМО змінну **index** за допомогою генерації випадкового числа в межах від 0 до довжини масиву **availableCells** (не включно).
 - а. ПОВЕРТАЄМО координати елемента, який знаходиться за індексом **index** і записуємо їх у змінну **coordinates**.
 7. ЗАПИСУЄМО значення змінної **coordinates** у змінну **prevShot**
 8. ПОВЕРТАЄМО змінну **coordinates** для подальшого вистрілу
 9. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Діаграма класів програмного забезпечення

Рисунок 4.1



4.2 Опис методів частин програмного забезпечення

4.2.1 Користувацькі методи

Таблиця 4.1 – Користувацькі методи

| № | Клас | Метод | Призначення методу | Вхідні параметри | Вихідні дані |
|----|------|----------------|--|------------------|---------------|
| 1 | Ship | constructor | конструктор | Position, id | |
| 2 | Ship | getCoordinates | Отримання координат корабля | - | Position |
| 3 | Ship | getIsKilled | Чи корабель вбито | - | boolean |
| 4 | Ship | getHits | Отримати влучання | - | Coordinates[] |
| 5 | Ship | hit | Влучити в корабель | coordinates | void |
| 6 | Ship | isAt | Чи корабель перебуває в координатах | coordinates | boolean |
| 7 | Cell | constructor | конструктор | x, y | - |
| 8 | Cell | getX | Отримати координату X | - | number |
| 9 | Cell | getY | Отримати координату Y | - | void |
| 10 | Cell | getHasShip | Визначити чи корабель перебуває в клітинці | - | boolean |
| 11 | Cell | getShip | Отримати корабель в клітинці | - | Ship null |
| 12 | Cell | setShip | Поставити | ship | void |

| | | | | | |
|----|---------------|-------------------|----------------------------------|--------------------|-----------------------|
| | | | корабель у клітинку | | |
| 13 | Cell | getIsHit | Чи корабель в клітинці поранений | - | boolean |
| 14 | Cell | hit | Влучити у клітинку | - | void |
| 15 | ShipGenerator | constructor | конструктор | grid | - |
| 16 | ShipGenerator | generateShips | Розташувати кораблі в сітці | - | Ship[] |
| 17 | ShipGenerator | generatePosition | Згенерувати випадкові координати | - | ShipGeneratorPosition |
| 18 | ShipGenerator | isValidPosition | Перевірити правильність позиції | position, shipSize | boolean |
| 19 | Grid | constructor | конструктор | size | - |
| 20 | Grid | getSize | Отримати розмір сітки | - | string |
| 21 | Grid | generateShips | Розташувати кораблі | - | void |
| 22 | Grid | getGrid | Отримати сітку | - | Cell[][] |
| 23 | Grid | setShips | Додати флот | ships | void |
| 24 | Grid | getShips | Отримати флот | - | Ship[] |
| 25 | Grid | killShip | Потопити корабль | ship | - |
| 26 | Grid | getAvailableShips | Отримати цілі кораблі | - | Ship[] |
| 27 | Grid | getAvailableCells | Отримати не стріляні | - | Cell[] |

| | | | | | |
|----|------|----------------|-----------------------------|-------------|--------|
| | | | клітинки | | |
| 28 | Grid | getNeighbours | Отримати сусідні клітинки | coordinates | Cell[] |
| 29 | Grid | fillWithShips | Розташувати збережений флот | - | void |
| 30 | Grid | initializeGrid | Створити сітку | - | void |

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 План тестування

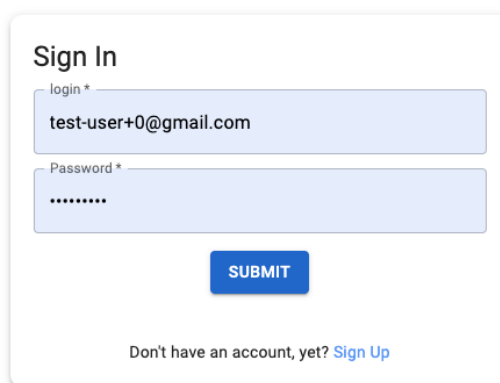
Складемо план тестування програмного забезпечення, за допомогою якого протестуємо весь основний функціонал

1. Тестування генерації та розташування кораблів
 - a. Зміна розміру сітки
 - b. Перерозташування кораблів
2. Тестування входу в кімнату очікування
 - a. По кліку на кнопку Play повинен виконуватися перехід на іншу сторінку
 - b. На сторінці має відображатися поле гравця з його флотом
3. Тестування гри з комп'ютером
 - a. По кліку на кнопку P Lay with Computer має здійснюватися перехід на сторінку з грою
 - b. Повинно відображатися 2 дошки. Дошка гравця має показувати його кораблі. Дошка суперника має бути "чиста"
 - c. При кліку на клітинку суперника, вона має змінити колір
 - i. Якщо гравець влучив, то має змінитися колір на зелений
 - ii. Якщо вбив - червоний
 - iii. Промазав - синій
 - d. При кліку суперника, клітинка гравця має змінити колір
 - i. Якщо противник влучив, то має змінитися колір на зелений
 - ii. Якщо вбив - червоний
 - iii. Промазав - синій
 - e. При умові що один з гравців програв/переміг має з'явитися відповідне повідомлення і переадресація на головну сторінку
4. Перевірка результатів
 - a. При відвідуванні сторінки Archive має бути видно результаті зустрічей
 - b. При натиску кнопки export as JSON має зберегтися файл у форматі JSON на комп'ютер користувача

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

Після запуску програми в браузері має відкритися сторінка Sign In (Рисунок 6.1)



The image shows a 'Sign In' form with a title 'Sign In' at the top. Below the title are two input fields: 'login *' containing the email 'test-user+0@gmail.com' and 'Password *' containing masked characters '*****'. A blue 'SUBMIT' button is positioned below the password field. At the bottom of the form, there is a link that reads 'Don't have an account, yet? [Sign Up](#)'.

Рисунок 6.1 - Сторінка входу

При умові що користувач не має аккаунту, йому потрібно зареєструватися на сторінці Sign Up (Рисунок 6.2)

The image shows a 'Sign up' form with the following fields and values:

- First name ***: Empty text input field.
- Last name ***: Empty text input field.
- Email ***: Empty text input field.
- Username ***: Text input field containing the value `test-user+0@gmail.com`.
- Phone number**: Empty text input field.
- Password ***: Password input field with masked characters (dots).
- Confirm password ***: Empty text input field.

The form is titled 'Sign up' and has a light blue header bar. The 'Username' and 'Password' fields are highlighted with a light blue background.

Рисунок 6.2 - Сторінка реєстрації

Після входу в особистий кабінет має відкритися сторінка Dashboard (Рисунок 6.3)

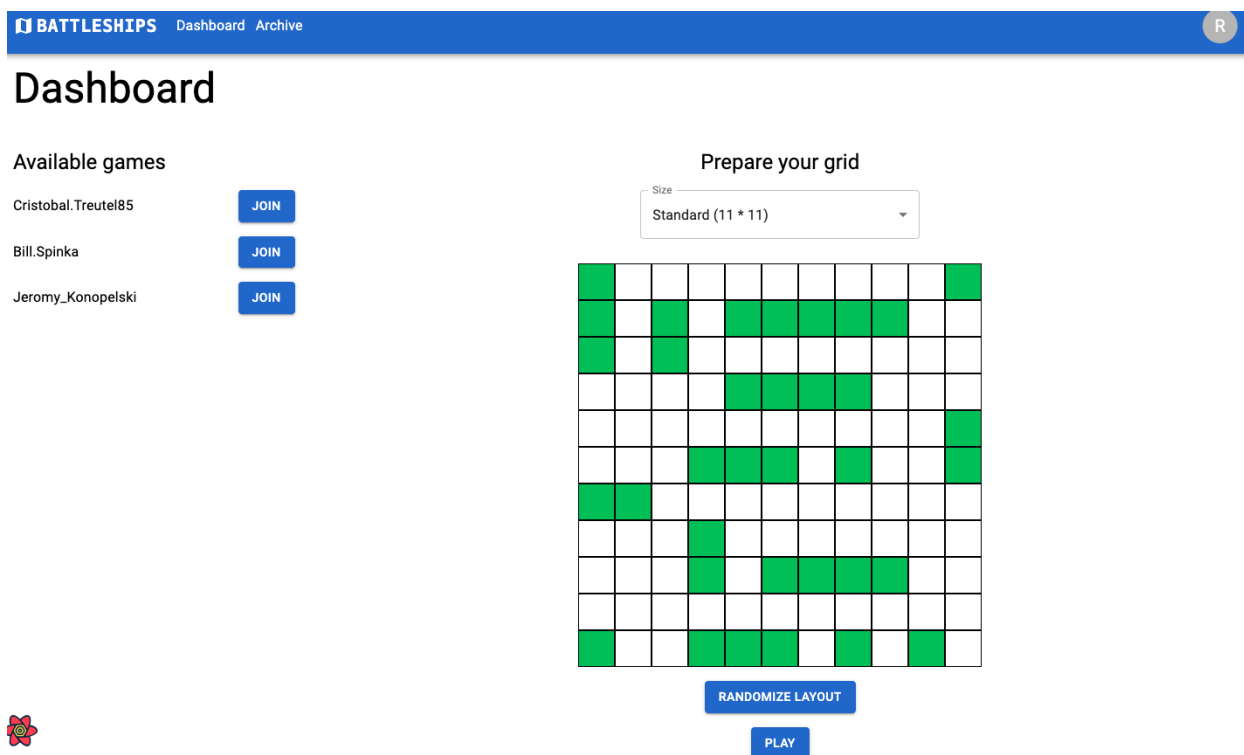


Рисунок 6.3 - Dashboard

Тут користувач може розташувати кораблі за допомогою кнопки Randomize layout і почати гру натиснувши кнопку Play

Після натискання кнопки його має перевести на сторінку Waiting room (Рисунок 6.4)

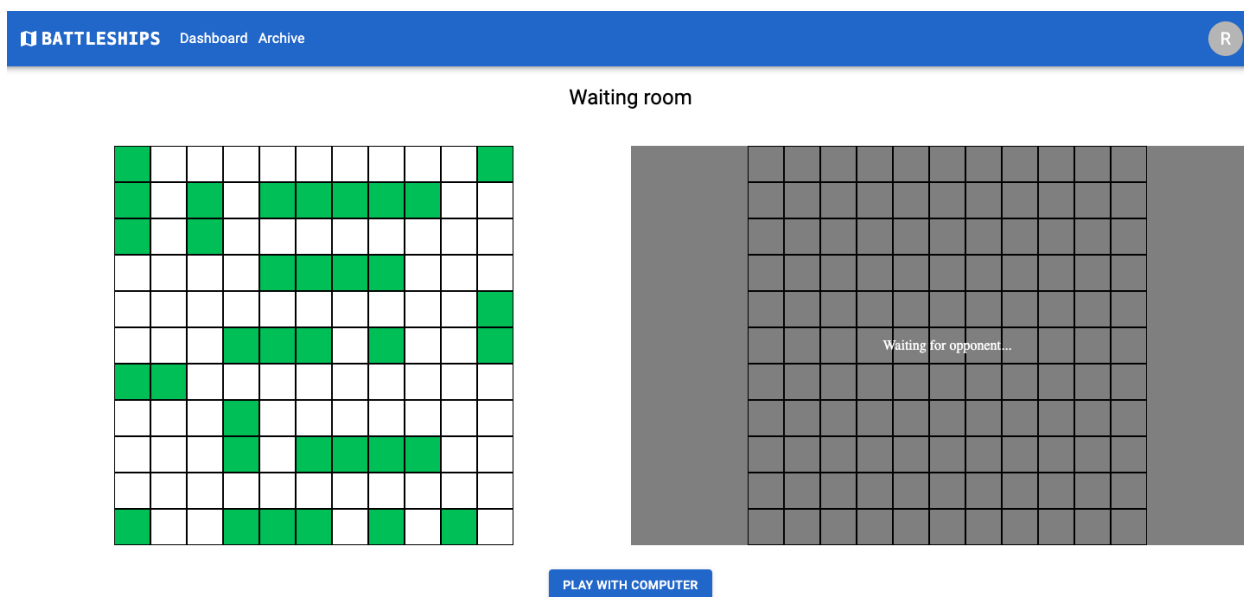


Рисунок 6.4 - Waiting room

Для початку гри з комп'ютером потрібно натиснути кнопку Play with computer. Тоді користувач перенесеться на сторінку гри (Рисунок 6.5)

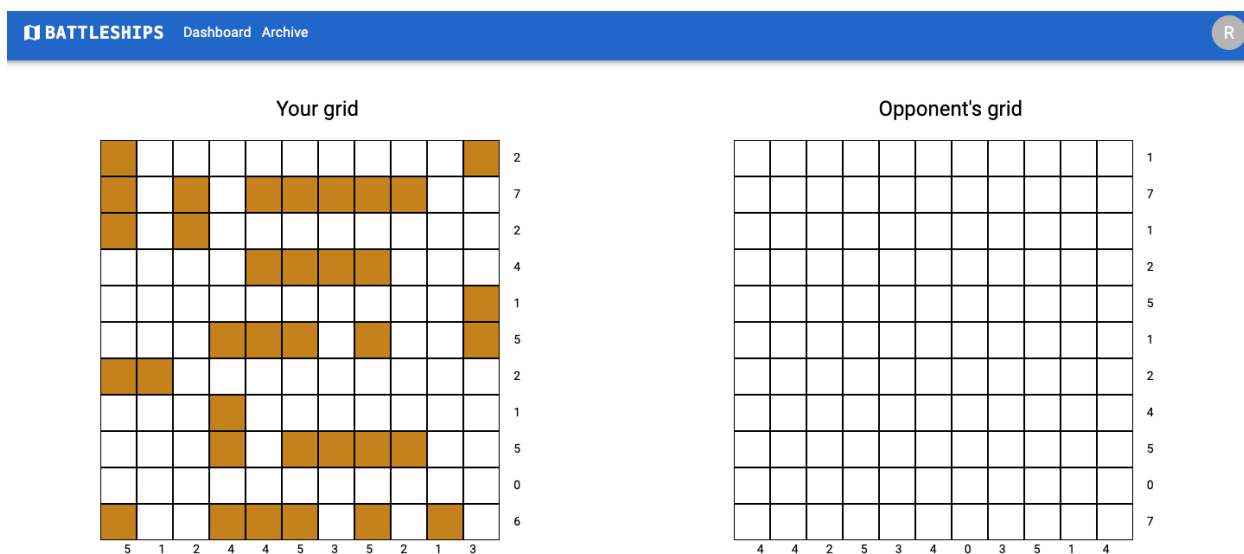


Рисунок 6.5 - Сторінка гри

Тут він може побачити свій флот та сітку противника. Після пострілів обох учасників сітка матиме різні кольори (Рисунок 6.6)

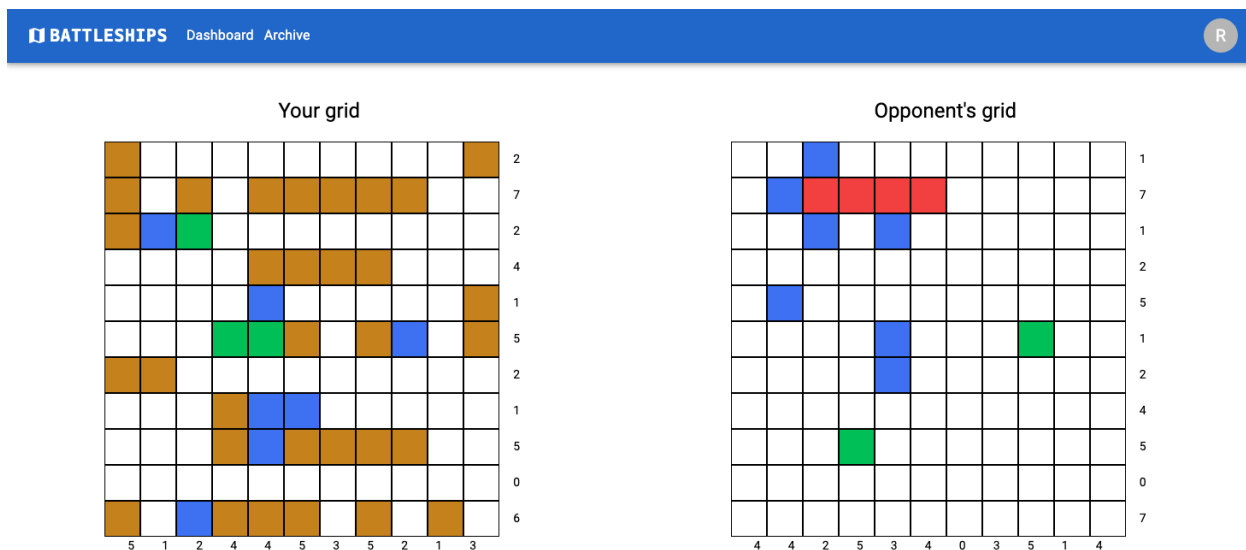


Рисунок 6.6 - Ігрові поля

Після завершення гри, гравець отримає інформацію про результати (Рисунок 6.7)

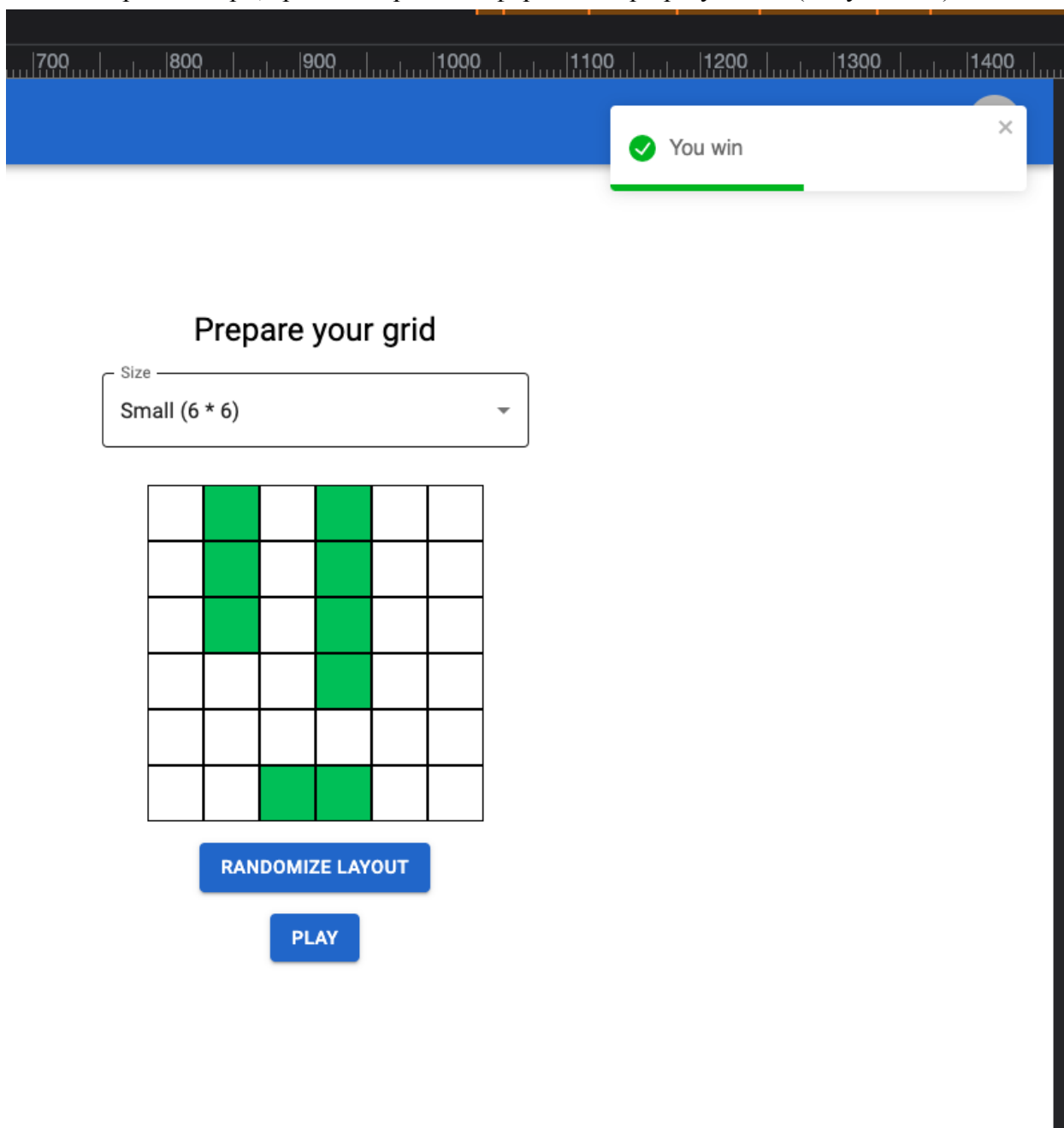


Рисунок 6.7 - результати гри

Також гравець може відвідати сторінку Archive для перегляду всіх результатів (Рисунок 6.8)

The screenshot shows a web application interface for 'BATTLESHIPS'. At the top is a blue navigation bar with the logo and links to 'Dashboard' and 'Archive'. A user profile icon with the letter 'R' is in the top right. Below the navigation bar is the title 'Archive'. A table displays game results with columns: Player 1, Player2, Date, and Winner. The first row shows a game between 'Computer' and 'Gregorio.Wolff18' on '2022-08-25T18:57:56.409Z', with 'Gregorio.Wolf...' as the winner. Below the table is a pagination bar indicating 'Rows per page: 100' and '1-1 of 1'. At the bottom left of the table area is a blue button labeled 'EXPORT AS JSON'.

| Player 1 | Player2 | Date | Winner |
|----------|------------------|--------------------------|------------------|
| Computer | Gregorio.Wolff18 | 2022-08-25T18:57:56.409Z | Gregorio.Wolf... |

Rows per page: 100 1-1 of 1

EXPORT AS JSON

Рисунок 6.8 - сторінка з усіма результатами

Після натискання кнопки Export as JSON гравець може експортувати свої результати в форматі JSON

6.3 Системні вимоги

Системні вимоги
програмного забезпечення

| | Мінімальні | Рекомендовані |
|--------------------|---|---|
| Операційна система | Windows 8/Windows 10/Windows 11 (з останніми оновленнями) | Windows 8/Windows 10/Windows 11 (з останніми оновленнями) |

| | | |
|--------------------|--|---------------|
| Процесор | Intel® Pentium® III 1.0 GHz | Intel Core I3 |
| Оперативна пам'ять | 4 ГБ | 8ГБ |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 800x600 | 1440x800 |

Рекоменовані браузері на базі Chromium (Google Chrome, Edge), WebKit (Safari) та SpiderMonkey (Firefox)

ВИСНОВОК

Під час курсової роботи було вивчено метод розробки програмного забезпечення з використанням ООП на прикладі гри «Морський бій». Практика зі створення Web-застосунків, розробки алгоритмів та проектування основних конструкцій. Зрозумів концепт роботи з PostgreSQL, Docker, Node.js. Здобув знання з побудови API на основі REST архітектури.

ПЕРЕЛІК ПОСИЛАНЬ

1. GitHub репозиторій з кодом - <https://github.com/doichev-kostia/battleships>
2. Node.js - <https://nodejs.org/en/>
3. Material UI - <https://mui.com/>
4. Tailwind css - <https://tailwindcss.com/>

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник Головченко М. М.

«__» _____ 2022 р

.

Виконавець:

Студент Дойчев Костянтин Миколайович

«__» _____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання курсової роботи
на тему: Гра «Морський бій»
з дисципліни:
«Основи програмування»

Київ 2022

1. Мета: Метою курсової роботи є розробка гри "Морський бій"

2. Дата початку роботи: «___»_____202_ р.

3. Дата закінчення роботи: «___»_____202_ р.

4. Вимоги до програмного забезпечення.

1) Функціональні вимоги:

розставляти кораблі на ігровому полі

контролювати правильність їх розстановки

давати противникам можливість почергово робити ходи

видавати інформаційні повідомлення про завершення гри(хто переміг)

в якості одного з гравців виступає комп'ютер

малювати мапу

позначати пошкоджені кораблі

< вимоги до функціональних характеристик>.

2) Нефункціональні вимоги:

Програма повинна мати головне меню з опціями, де

користувач може обрати, чи він грає з іншою людиною чи комп'ютером.

Ігрове поле має мати 4 мапи, де позначені власні кораблі та

противника; по 2 мапи на кожного гравця

Все програмне забезпечення та супроводжуюча технічна

документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

5. Стадії та етапи розробки:

1) Об'єктно-орієнтований аналіз предметної області задачі

(до __.__.202_ р.)

33

2) Об'єктно-орієнтоване проектування архітектури програмної

системи (до __.__.202_ р.)

3) Розробка програмного забезпечення (до __.__.202_ р.)

4) Тестування розробленої програми (до __.__.202_ р.)

5) Розробка пояснювальної записки (до __.__.202_ р.).

6) Захист курсової роботи (до __.__.202_ р.).

6. Порядок контролю та приймання. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

Тексти програмного коду програмного забезпечення
Гра “Морський бій”

CD-RW
(Обсяг програми (документа), арк., Кб)
арк, Кб

студента I курсу групи ІІІ-13
Дойчева Костянтина Миколайовича

Для перегляду всього коду, можна перейти на [GitHub](#)

1 cell.ts

```
import Ship from "../ship";
import { Coordinates } from "../types";

export class Cell {
  private readonly x: number;
  private readonly y: number;
  private isHit = false;
  private ship: Ship | null = null;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }

  public getX(): number {
    return this.x;
  }

  public getY(): number {
    return this.y;
  }

  public getCoordinates(): Coordinates {
    return { x: this.x, y: this.y };
  }

  public getHasShip(): boolean {
    return !!this.ship;
  }

  public getShip(): Ship | null {
```

```
        return this.ship;
    }

    public setShip(ship: Ship): void {
        this.ship = ship;
    }

    public getIsHit(): boolean {
        return this.isHit;
    }

    public hit(): void {
        this.isHit = true;
        if (this.ship) {
            this.ship.hit({ x: this.x, y: this.y });
        }
    }
}
```


2. ship.ts

```

import { Coordinates } from "app/utils/types";

export interface Position {
  xStart: number;
  yStart: number;
  xEnd: number;
  yEnd: number;
}

class Ship {
  private readonly id?: string;
  private readonly size: number;
  private xStart: number;
  private yStart: number;
  private xEnd: number;
  private yEnd: number;
  private isKilled = false;
  private hits: Coordinates[] = [];

  constructor({ xStart, yStart, xEnd, yEnd }: Position, id?: string) {
    this.id = id;
    this.xStart = xStart;
    this.yStart = yStart;
    this.xEnd = xEnd;
    this.yEnd = yEnd;
    const isHorizontal = this.yStart === this.yEnd;
    const isVertical = this.xStart === this.xEnd;

    if (!isVertical && !isHorizontal) {

```

```

        this.size = 1;
    } else if (isVertical) {
        this.size = Math.abs(this.yEnd - this.yStart) + 1;
    } else if (isHorizontal) {
        this.size = Math.abs(this.xEnd - this.xStart) + 1;
    }
}

public getCoordinates(): Position {
    return {
        xStart: this.xStart,
        yStart: this.yStart,
        xEnd: this.xEnd,
        yEnd: this.yEnd,
    };
}

public getIsKilled() {
    return this.isKilled;
}

public getHits() {
    return this.hits;
}

public hit(coordinates: Coordinates) {
    const isUnique = this.hits.every((hit) => hit.x !== coordinates.x || hit.y
    !== coordinates.y);
    if (!isUnique) {
        return;
    }
    this.hits = [...this.hits, coordinates];
    if (this.hits.length === this.size) {
        this.isKilled = true;
    }
}

```

```

    }

    public isAt(coordinates: Coordinates) {
        return (
            this.xStart <= coordinates.x &&
            this.xEnd >= coordinates.x &&
            this.yStart <= coordinates.y &&
            this.yEnd >= coordinates.y
        );
    }
}

export default Ship;

```

3. ship-generator.ts

```

import { GAME_CONFIG } from "app/constants/game-config";
import Ship, { Position as ShipPosition } from "./ship";
import { Grid } from "./grid";
import { Coordinates } from "../types";

type ShipGeneratorPosition = {
    isHorizontal: boolean;
    coordinates: Coordinates;
};

export class ShipGenerator {

```

```

private grid: Grid;
private counter = 0;
private config = GAME_CONFIG.standard;

constructor(grid: Grid) {
    this.grid = grid;
    this.config = GAME_CONFIG[grid.getSize()];
}

public generateShips() {
    this.counter = 0;

    type TShip = {
        size: number;
    };

    const placedShips: Ship[] = [];

    /**
     * Sort from largest to smallest
     */
    const ships = this.config.ships.sort((a, b) => b.size - a.size);

    const allShips = ships.reduce<TShip[]>((acc, ship) => {
        const total: TShip[] = [];
        for (let i = 0; i < ship.quantity; i++) {
            total.push({ size: ship.size });
        }

        return [...acc, ...total];
    }, []);

    allShips.forEach((ship) => {
        let position: ShipGeneratorPosition;
        do {

```

```

        position = this.generatePosition();
        if (this.counter > 5000) {
            throw new Error("Could not generate ships");
        }
    } while (!this.isValidPosition(position, ship.size));

    const { isHorizontal, coordinates } = position;

    const shipPosition = (() => {
        const shipPosition: Partial<ShipPosition> = {
            xStart: coordinates.x,
            yStart: coordinates.y,
        };

        if (isHorizontal) {
            shipPosition.xEnd = coordinates.x + ship.size - 1;
            shipPosition.yEnd = coordinates.y;
        } else {
            shipPosition.xEnd = coordinates.x;
            shipPosition.yEnd = coordinates.y + ship.size - 1;
        }

        return shipPosition as ShipPosition;
    })();

    const createdShip = new Ship(shipPosition);

    placedShips.push(createdShip);
    this.grid.getGrid().forEach((row) => {
        row.forEach((cell) => {
            if (createdShip.isAt(cell.getCoordinates())) {
                cell.setShip(createdShip);
            }
        });
    });
});

```

```

    });
    return placedShips;
}

private generatePosition() {
    this.counter++;
    const isHorizontal = Math.random() > 0.5;
    const coordinates = {
        x: Math.floor(Math.random() * this.config.size),
        y: Math.floor(Math.random() * this.config.size),
    };

    return {
        isHorizontal,
        coordinates,
    } as ShipGeneratorPosition;
}

private isValidPosition(position: ShipGeneratorPosition, shipSize: number) {
    const cells = this.grid.getGrid();

    const { isHorizontal, coordinates } = position;
    const { x, y } = coordinates;
    const isInGridRange = isHorizontal
        ? x + shipSize <= this.config.size - 1
        : y + shipSize <= this.config.size - 1;

    if (!isInGridRange) {
        return false;
    }

    /* Check whether there is a ship in the current position and the
    neighbouring cells*/
    for (let i = 0; i < shipSize + 1; i++) {
        if (isHorizontal) {

```

```

const currentRow = cells[y];
const prevRow = cells[y - 1];
const nextRow = cells[y + 1];
if (currentRow[x + i] && currentRow[x +
i].getHasShip()) {
    return false;
}
if (prevRow) {
    if (prevRow[x + i] && prevRow[x +
i].getHasShip()) {
        return false;
    }
}

if (nextRow) {
    if (nextRow[x + i] && nextRow[x +
i].getHasShip()) {
        return false;
    }
}

if (i === 0) {
    if (currentRow[x - 1] && currentRow[x -
1].getHasShip()) {
        return false;
    }

    if (prevRow) {
        if (prevRow[x - 1] && prevRow[x -
1].getHasShip()) {
            return false;
        }
    }

    if (nextRow) {

```

```

        if (nextRow[x - 1] && nextRow[x -
1].getHasShip()) {
            return false;
        }
    }
}
}
if (!isHorizontal) {
    if (i === 0) {
        const prevRow = cells[y - 1];
        if (prevRow) {
            if (prevRow[x].getHasShip()) {
                return false;
            }

            if (prevRow[x - 1] && prevRow[x -
1].getHasShip()) {
                return false;
            }

            if (prevRow[x + 1] && prevRow[x +
1].getHasShip()) {
                return false;
            }
        }
    }

    const currentRow = cells[y + i];
    if (currentRow[x] && currentRow[x].getHasShip()) {
        return false;
    }

    if (currentRow[x - 1] && currentRow[x -
1].getHasShip()) {
        return false;
    }
}

```



```
        }  
        if (currentRow[x + 1] && currentRow[x +  
1].getHasShip()) {  
            return false;  
        }  
    }  
    }  
    return true;  
}  
}
```

4. grid.ts

```

import { GAME_CONFIG, GridSize } from "app/constants/game-config";
import { Cell } from "app/utils/game/cell";
import Ship from "../ship";
import { ShipGenerator } from "../ship-generator";
import { Coordinates } from "../types";

export class Grid {
  private size: GridSize;
  private grid: Cell[][] = [];
  private ships: Ship[] = [];
  private availableShips: Ship[] = [];
  private shipGenerator: ShipGenerator;

  constructor(size: GridSize = "standard") {
    this.size = size;
    this.initializeGrid();
  }

  public getSize = () => this.size;

  public generateShips(): void {
    // To prevent some caching issues, we need to generate new ships
    every time
    this.ships = [];
    this.availableShips = [];
    this.initializeGrid();
    this.shipGenerator = new ShipGenerator(this);
    this.setShips(this.shipGenerator.generateShips());
  }

  public getGrid(): Cell[][] {

```

```

        return this.grid;
    }

    public setShips(ships: Ship[]): void {
        this.ships = ships;
        this.availableShips = ships;
    }

    public getShips(): Ship[] {
        return this.ships;
    }

    public killShip(ship: Ship): void {
        this.availableShips = this.availableShips.filter((s) => s !== ship);
    }

    public getAvailableShips(): Ship[] {
        return this.availableShips;
    }

    public getAvailableCells(): Cell[] {
        return this.grid.flat(2).filter((cell) => !cell.getIsHit());
    }

    public getNeighbours({ x, y }: Coordinates): Cell[] {
        const neighbours: Cell[] = [];
        if (x > 0) {
            neighbours.push(this.grid[y][x - 1]);
        }
        if (x < this.grid[y].length - 1) {
            neighbours.push(this.grid[y][x + 1]);
        }
        if (y > 0) {
            neighbours.push(this.grid[y - 1][x]);
        }
    }

```

```

        if (y < this.grid.length - 1) {
            neighbours.push(this.grid[y + 1][x]);
        }
        return neighbours;
    }

    public fillWithShips(): void {
        this.ships.forEach((ship) => {
            const { xStart, yStart, xEnd, yEnd } = ship.getCoordinates();
            for (let row = yStart; row <= yEnd; row++) {
                for (let col = xStart; col <= xEnd; col++) {
                    this.grid[row][col].setShip(ship);
                }
            }
        });
    }

    private initializeGrid(): void {
        this.grid = [];
        const { size } = GAME_CONFIG[this.size];
        for (let row = 0; row < size; row++) {
            this.grid[row] = [];
            for (let col = 0; col < size; col++) {
                this.grid[row][col] = new Cell(col, row);
            }
        }
    }
}

```