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:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студента \_\_1\_\_ курсу, групи \_ІП-11\_\_\_

Панченка Сергія Віталійовича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(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

(назва вищого навчального закладу)

## Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-11 Семестр 2

### **ЗАВДАННЯ**

#### на курсову роботу студента

##### Панченка Сергія Віталійовича

(прізвище, ім’я, по батькові)

1. Тема роботи

2. Строк здачі студентом закінченої роботи

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

КАЛЕНДАРНИЙ ПЛАН

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

Студент

(підпис)

Керівник Муха І. П.

(підпис) (прізвище, ім’я, по батькові)

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

# **Анотація**

Пояснювальна записка до курсової роботи: 67 сторінок, 23 рисунки, 9 таблиць.

Об’єкт дослідження: гра «Морський бій»

Мета роботи: дослідження методів розробки програмного забезпечення

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».

Зміст

[Кафедра інформатики та програмної інженерії 2](#_Toc104629502)

[**ЗАВДАННЯ** 2](#_Toc104629503)

[на курсову роботу студента 2](#_Toc104629504)

[Студент 3](#_Toc104629505)

[Анотація 4](#_Toc104629506)

[Вступ 7](#_Toc104629507)

[1 ПОСТАНОВКА ЗАДАЧІ 8](#_Toc104629508)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 9](#_Toc104629509)

[Правила розміщення кораблів 9](#_Toc104629510)

[Пошук і потоплення кораблів супротивника 9](#_Toc104629511)

[3 ОПИС АЛГОРИТМІВ 11](#_Toc104629512)

[4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 13](#_Toc104629513)

[4.1. Діаграма класів програмного забезпечення 13](#_Toc104629514)

[4.2. Опис методів частин програмного забезпечення 13](#_Toc104629515)

[4.2.1. Користувацькі методи 13](#_Toc104629516)

[4.2.2. Стандартні методи 20](#_Toc104629517)

[5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 23](#_Toc104629518)

[5.1. План тестування 23](#_Toc104629519)

[5.2. Приклади тестування 23](#_Toc104629520)

[5.2.1. Випадок 1 23](#_Toc104629521)

[5.2.2. Випадок 2 26](#_Toc104629522)

[5.2.3. Випадок 3 28](#_Toc104629523)

[5.2.4. Випадок 4 31](#_Toc104629524)

[6 ІНСТРУКЦІЯ КОРИСТУВАЧА 34](#_Toc104629525)

[6.1 Робота з програмою 34](#_Toc104629526)

[6.2 Системні вимоги 38](#_Toc104629527)

[7 Аналіз результатів 40](#_Toc104629528)

[Висновки 41](#_Toc104629529)

[Перелік посилань 42](#_Toc104629530)

[Додаток А Технічне завдання 43](#_Toc104629531)

[ДОДАТОК Б Лістинг 46](#_Toc104629532)

Вступ

Дана робота присвячена вивченню розробки програмного забезпечення з використанням парадигми ООП, і стосується написання комп’ютерної гри «Морський бій». Актуальність теми полягає в тому, що сучасну IT-індустрію неможливо уявити без використання парадигми ООП при розробці програмного забезпечення. Основним завданням ООП є спрощення підтримки програм та їхнього написання, а саме: зменшення об’ємів коду і простота його розуміння.

# **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 знаходиться 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 почергово 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.

# **ТЕОРЕТИЧНІ ВІДОМОСТІ**

### Правила розміщення кораблів

Ігрове поле — квадрат 11 × 11 кожного гравця, на якому розміщений флот кораблів.

Загалом є п’ятнадцять кораблів:

* 1 корабель — ряд із 5
* 2 кораблі — ряд із 4 клітин
* 3 кораблі — ряд із 3 клітин
* 4 кораблі — ряд із 2 клітин
* 5 кораблів — 1 клітина

При розміщенні кораблі не можуть торкатися один одного кутами.

Поруч зі «своїм» квадратом креслять «чужий» такого самого розміру, лише порожній. Це ділянка моря, де ходять кораблі супротивника.

При влученні в корабель супротивника — на чужому полі ставлять хрестик. Той, хто влучив, стріляє ще раз.

### Пошук і потоплення кораблів супротивника

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, «К1!».

* Якщо постріл влучив у клітину, яку не займає жоден з кораблів супротивника, то гравець дістає відповідь «Мимо!» і той, хто стріляв, ставить на чужому квадраті в цьому місці крапку. Право ходу переходить до суперника.
* Якщо постріл влучив у клітину, де розташований багатопалубний корабель (розміром більше ніж 1 клітина), то гравець дістає відповідь «Поранив!» або «Влучив!», крім одного випадку (див. пункт 3). Гравець, що стріляв, ставить на чужому полі в цю клітину хрестик, а його супротивник ставить хрестик на своєму полі також у цю клітку. Гравець, який стріляв, дістає право на ще один постріл.
* Якщо постріл влучив у клітину, де розташований однопалубний корабель, або в останню неуражену клітину багатопалубного корабля, то гравець чує у відповідь «Потоплений!» або «Вбито!». Обидва гравці відзначають потоплений корабель на аркуші. Гравець, що стріляв, здобуває право на ще один постріл.

Переможцем є той, хто першим потопив усі 15 кораблів противника. Суперники вивчають ігрові поля один у одного і затверджують результат, якщо не було жодних порушень.

# **ОПИС АЛГОРИТМІВ**

Перелік всіх основних змінних та їхнє призначення наведено в таблиці

Таблиця 3.1 – Основні змінні та їхні призначення

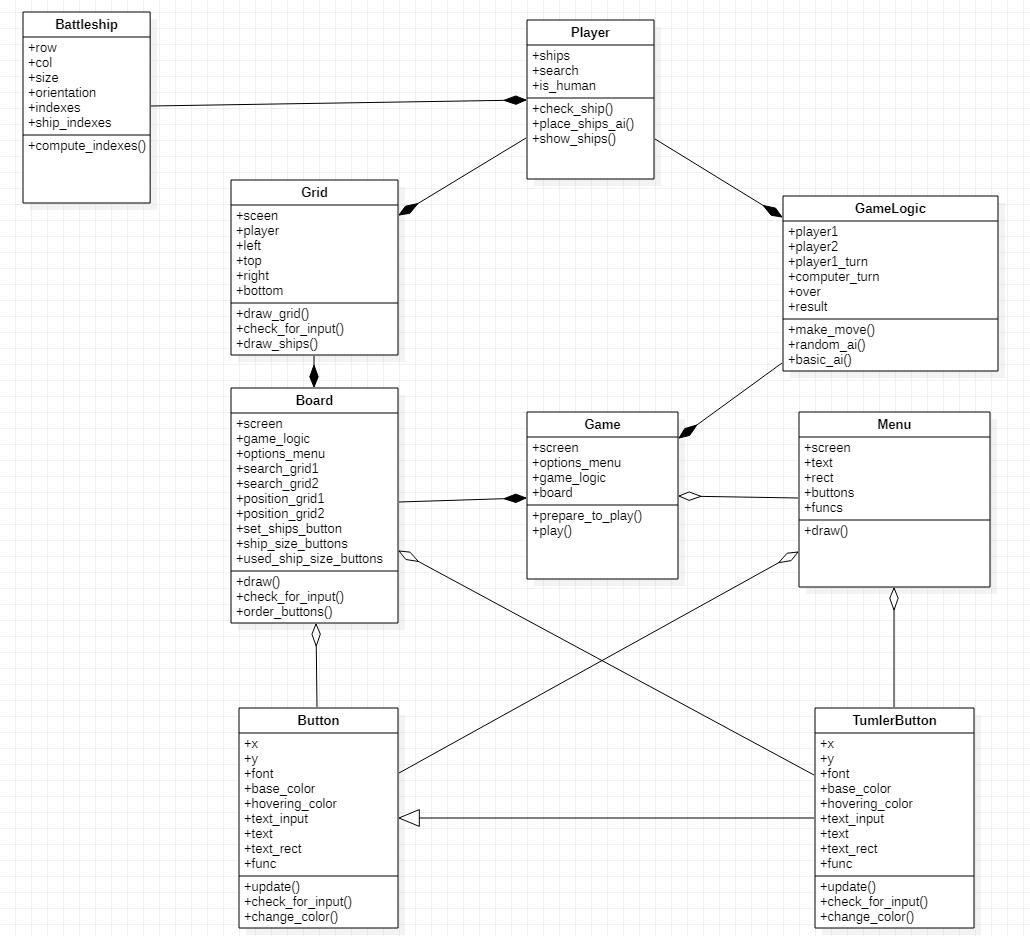
|  |  |
| --- | --- |
| Змінна | Призначення |
| search | Масив індексів клітинок пошукової таблиці гравця |
| unknown | Індекси, під якими невідомо, що знаходиться |
| hits | Індекси, під якими знаходяться підбиті кораблі противника |
| neighbours1 | Масив індексів з невідомим вмістом, що знаходяться на відстані однієї клітинки від індексів з підбитими кораблями противника |
| neighbours2 | Масив індексів з невідомим вмістом, що знаходяться на відстані двох клітинок від індексів з підбитими кораблями противника |
| checker\_board | Масив парних індексів з невідомим вмістом |
| row | рядок |
| col | колонка |
| MAP\_SIZE | Розмір мапи |
| is\_both\_neighbour | Булеве значення, що відповідає, чи є елемент і в neighbours1, і в neighbours2 |

3.1 Стандартний алгоритм

1. ПОЧАТОК
2. ЦИКЛ проходу по масиву unknown( u – поточний елемент )
   1. ЯКЩО u - 1 В hits АБО u + 1 В hits АБО u – MAP\_SIZE АБО u + MAP\_SIZE
      1. Включити елемент u до neighbours1
   2. ЯКЩО u - 2 В hits АБО u + 2 В hits АБО u – MAP\_SIZE\*2 АБО u + MAP\_SIZE\*2
      1. Включити елемент u до neighbours2
3. is\_both\_neighbour := False
4. ЦИКЛ проходу по unknown( u – поточний елемент )
   1. ЯКЩО u В neighbours1 І u В neighbours2
      1. Зробити хід по u
      2. is\_both\_neighbour :=True
5. ЯКЩО is\_both\_neighbour == True:
   1. ЯКЩО існують елементи в neighbours1:
      1. Зробити хід по випадковому елементу з neighbours1
   2. ІНАКШЕ
      1. ЦИКЛ проходу по масиву unknown( u – поточний елемент )
         1. row := u // MAP\_SIZE
         2. col := u % MAP\_SIZE
         3. ЯКЩО (row + col) % 2 == 0:
            1. Включити елемент u до checker\_board
      2. ЯКЩО checker\_board має елементи
         1. Зробити хід по випадковому елементу з checker\_board
      3. ІНАКШЕ
         1. ЯКЩО існують елементи в unknown
            1. Зробити хід у випадковий індекс з невідомим вмістом
6. КІНЕЦЬ

# **ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

## Діаграма класів програмного забезпечення



## Опис методів частин програмного забезпечення

### Користувацькі методи

Таблиця 4.2.1.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | Button | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) pos - позиція миші 3) font - шрифт тексту 4) text\_input - текст кнопки 5) base\_color - основний колір 6) hovering\_color - другорядний колір 7) func - функція | ВІДСУТНІ | game\_classes.py |
| 2 | Button | update | Відображає об'єкт класу на екрані | 1) self - об'єкт класу 2) screen - екран | ВІДСУТНІ | game\_classes.py |
| 3 | Button | change\_color | Змінює колір при наведенні | 1) self - об'єкт класу 2) position - позиція миші | ВІДСУТНІ | game\_classes.py |
| 4 | TumblerButton | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) pos - позиція миші 3) font - шрифт тексту 4) text\_input - текст кнопки 5) base\_color - основний колір 6) hovering\_color - другорядний колір 7) func - функція | ВІДСУТНІ | game\_classes.py |
| 5 | TumblerButton | update | Відображає об'єкт класу на екрані | 1) self - об'єкт класу 2) screen - екран | ВІДСУТНІ | game\_classes.py |
| 6 | TumblerButton | check\_for\_input | Реєструє нитискання | 1) self - об'єкт класу 2) position - позиція миші | ВІДСУТНІ | game\_classes.py |
| 7 | TumblerButton | change\_color | Змінює колір при наведенні | 1) self - об'єкт класу 2) position - позиція миші | ВІДСУТНІ | game\_classes.py |
| 8 | Menu | \_\_init\_\_ | Конструктор класу | 1) self -об'єкт класу 2) screen - екран 3) text - основний текст 4) buttons - об'єкти класу Buttons 5) funcs - функції до кнопок | ВІДСУТНІ | game\_classes.py |
| 9 | Menu | draw | Відображає об'єкт класу на екрані | 1) self - об'єкт класу 2) \*\*kwargs - словник назв кнопок та їх функцій | ВІДСУТНІ | game\_classes.py |
| 10 | Grid | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) screen - екран 3) player - гравець 4) top - зміщення догори 5) left - зміщення наліво | ВІДСУТНІ | game\_classes.py |
| 11 | Grid | draw\_grif | Відображає об'єкт класу на екрані | 1) self - об'єкт класу 2) search - поле гравця | ВІДСУТНІ | game\_classes.py |
| 12 | Grid | check\_for\_input | Реєструє нитискання | 1) self - об'єкт класу 2) position - позиція миші | ВІДСУТНІ | game\_classes.py |
| 13 | Grid | draw\_ships | Відображає кораблі | 1) self - об'єкт класу | ВІДСУТНІ | game\_classes.py |
| 14 | Battleship | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) size - розмір 3) row - рядок 4) col - колонка 5) orientation - орієнтація | ВІДСУТНІ | game\_classes.py |
| 15 | Battleship | compute\_indexes | Розраховує розташування | 1) self - об'єкт класу | ВІДСУТНІ | game\_classes.py |
| 16 | Player | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) is\_human - значення, чи є гравець людиною | ВІДСУТНІ | game\_classes.py |
| 17 | Player | check\_ship | Перевіряє, чи є розташування корабля можливим | 1) self - об'єкт класу 2) ship - об'єкт класу Battleship | Posiible - bool | game\_classes.py |
| 18 | Player | place\_ships\_ai | Комп'ютер розставляє кораблі | 1) self - об'єкт класу 2) sizes - список розмірів кораблів | ВІДСУТНІ | game\_classes.py |
| 19 | GameLogic | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) is\_human1 - значення, чи є гравець 1 людиною 3) is\_human2 - значення, чи є гравець 2 людиною | ВІДСУТНІ | game\_classes.py |
| 20 | GameLogic | make\_move | Робить хід | 1) self - об'єкт класу 2) I - індекс клітинки | ВІДСУТНІ | game\_classes.py |
| 21 | GameLogic | computer\_algorithm | Комп'ютер ходить по клітинці | 1) self - об'єкт класу | ВІДСУТНІ | game\_classes.py |
| 23 | Board | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) screen - екран 3) options\_menu - об'єкт класу Menu 4) game\_logic - об'єкт класу GameLogic | ВІДСУТНІ | game\_classes.py |
| 24 | Board | draw | Відображає об'єкт класу на екрані | 1) self - об'єкт класу 2) draw\_buttons - значення, чи треба відображати об'єкти класу Button | ВІДСУТНІ | game\_classes.py |
| 25 | Board | order\_buttons | Розташовує кнопки у правильному порядку | 1) self - об'єкт класу | ВІДСУТНІ | game\_classes.py |
| 26 | Game | \_\_init\_\_ | Конструктор класу | 1) self - об'єкт класу 2) sceen - екран 3) options\_menu - об'єкт класу Menu | ВІДСУТНІ | game\_classes.py |
| 27 | Game | prepare\_to\_play | Підготовує гру до початку | 1) self - об'єкт класу | ВІДСУТНІ | game\_classes.py |
| 28 | Game | play | Розпочинає гру | 1) self - об'єкт класу | ВІДСУТНІ | game\_classes.py |
| 29 | Відсутній | get\_font | Отримати шрифт | 1) font\_size - розмір шрифта | Об'єкт класу Font | other\_funcs.py |
| 31 | Board | check\_for\_input | Реєструє нитискання | 1) self - об'єкт класу 2) position - позиція миші | ВІДСУТНІ | game\_classes.py |
| 32 | Button | check\_for\_input | Реєструє нитискання | 1) self - об'єкт класу 2) position - позиція миші | ВІДСУТНІ | game\_classes.py |

### Стандартні методи

Таблиця 4.2.2.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | pygame | \_\_init\_\_ | Ініціалізація pygame | ВІДСУТНІЙ | ВІДСУТНІЙ | Pygame |
| 2 | Pygame.font | \_\_Init\_\_ | Ініціалізація шрифта pygame | ВІДСУТНІЙ | ВІДСУТНІЙ | Pygame |
| 3 | Pygame.display | Set\_caption | Встановити заколовок | 1. Title - заголовок | ВІДСУТНІЙ | Pygame |
| 4 | Pygame.display | Set\_mode | Встановити розмір екрану | 1. Size – кортеж ширини та висоти екрана | ВІДСУТНІЙ | pygame |
| 5 | Pygame.Rect | \_\_init\_\_ | Конструктор | 1. X – позиція по горизонталі 2. Y –позиція по вертикалі 3. Width –ширина 4. Length - довжина | ВІДСУТНІЙ | pygame |
| 6 | Pygame.draw | Rect | Малює прямокутник | 1. Screen – екран 2. Color – колір 3. Rect – об1єкт класу Rect 4. Width – товщина прямокутника | ВІДСУТНІЙ | Pygame.draw |
| 7 | Pygame.draw | Circle | Малює коло | 1. Screen - екран 2. Color – колір 3. Center – позиція центра 4. Radius – радус кола | ВІДСУТНІЙ | pygame |
| 8 | Pygame.display | blit | Оновлення екрана | ВІДСУТНІЙ | ВІДСУТНІЙ | pygame |
| 9 | ВІДСУТНІЙ | choice | Випадково обирає елемент з певного списку | 1. List - список | ВІДСУТНІЙ | random |
| 10 | Pygame.font | render | Створює блок тексту | 1. Text\_input – вхідний текст 2. Pixelization – значення, яке вказує чи треба пікселізація 3. Base\_color – базовий колір 4. Hovering\_color – вторинний колір | Об’єкт класу Pygame.text | pygame |
| 11 | Pygame.text | Get\_rect | Отримує прямокутник, в якому знаходиться текст | ВІДСУТНІЙ | Об’єкт pygame.rect | pygame |

# **ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

## План тестування

Для тестування будемо змінювати опції гри та перевіряти результат. Тоді розглянемо випадки:

Таблиця 5.1.1

|  |  |  |
| --- | --- | --- |
| Випадки | Гравець 1 | Гравець 2 |
| 1 | Людина | Людина |
| 2 | Комп’ютер | Людина |
| 3 | Людина | Комп’ютер |
| 4 | Комп’ютер | Комп’ютер |

## Приклади тестування

### Випадок 1

Таблиця 5.2.1.1

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість двох людей грати між собою |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Гравець 1 –людина. Гравець 2 – людина. |
| Схема проведення тесту | Розпочати гру та спробувати пограти |
| Очікуваний результат | Виведення повідомлення про перемогу одного з гравців |
| Стан програми після проведення випробувань | Один з гравців переміг і повідомлення висвітилося на екрані |

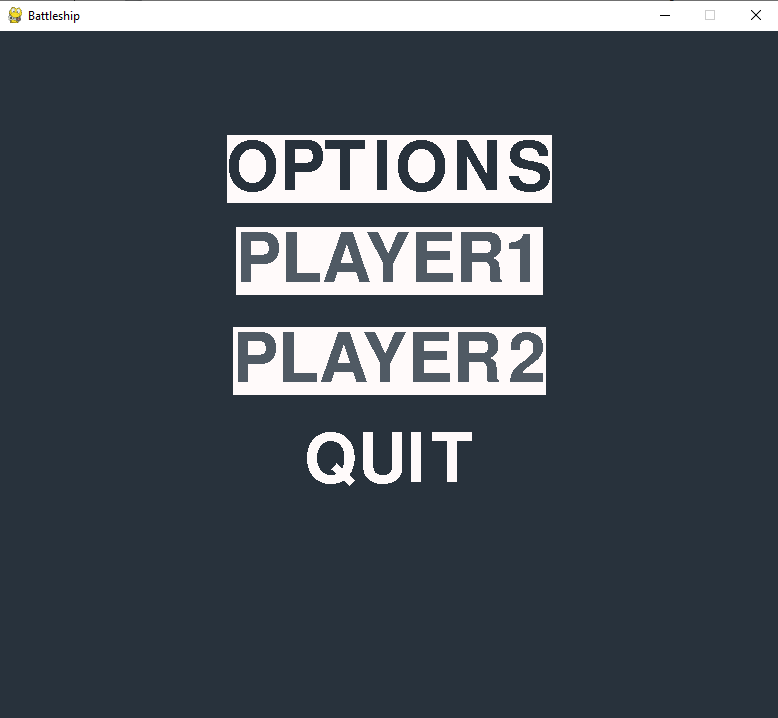


 Рисунок 5.2.1.1 – Виставлення параметрів

Рисунок 5.2.1.2 – Розставлення кораблів

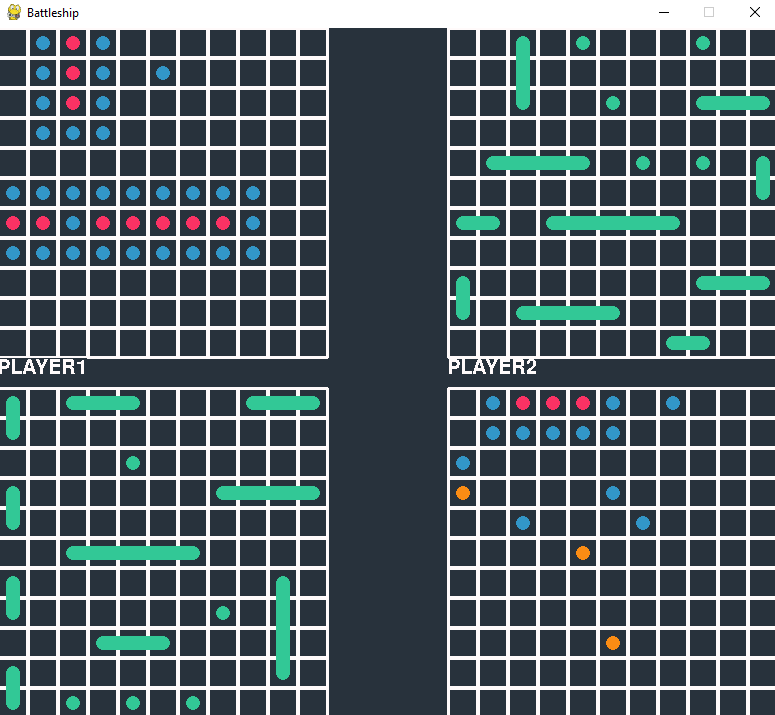


Рисунок 5.2.1.3 – Гра

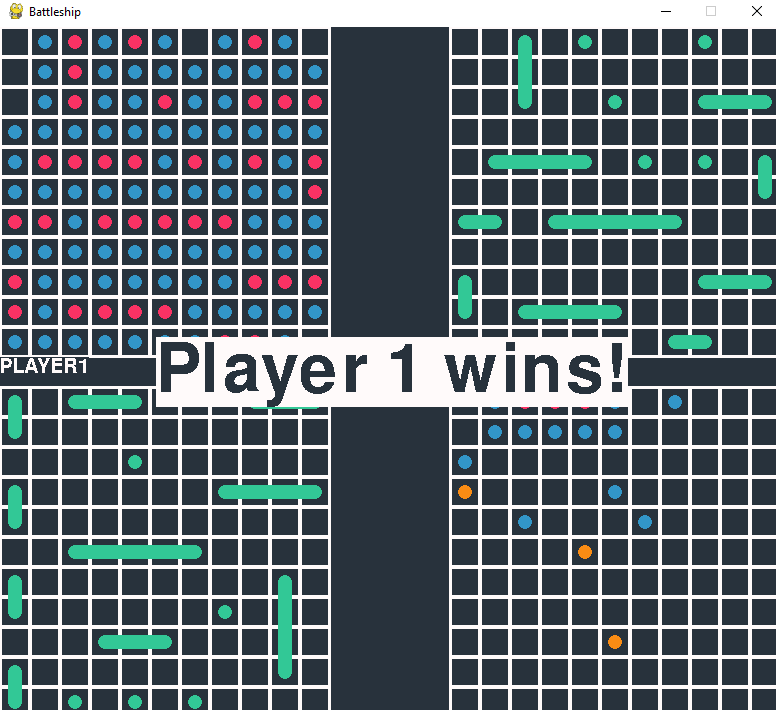


Рисунок 5.2.1.4 – Повідомлення про закінчення гри

### Випадок 2

Таблиця 5.2.2.1

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість двох людей грати між собою |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Гравець 1 – комп’ютер. Гравець 2 – людина. |
| Схема проведення тесту | Розпочати гру та спробувати пограти |
| Очікуваний результат | Виведення повідомлення про перемогу одного з гравців |
| Стан програми після проведення випробувань | Один з гравців переміг і повідомлення висвітилося на екрані |

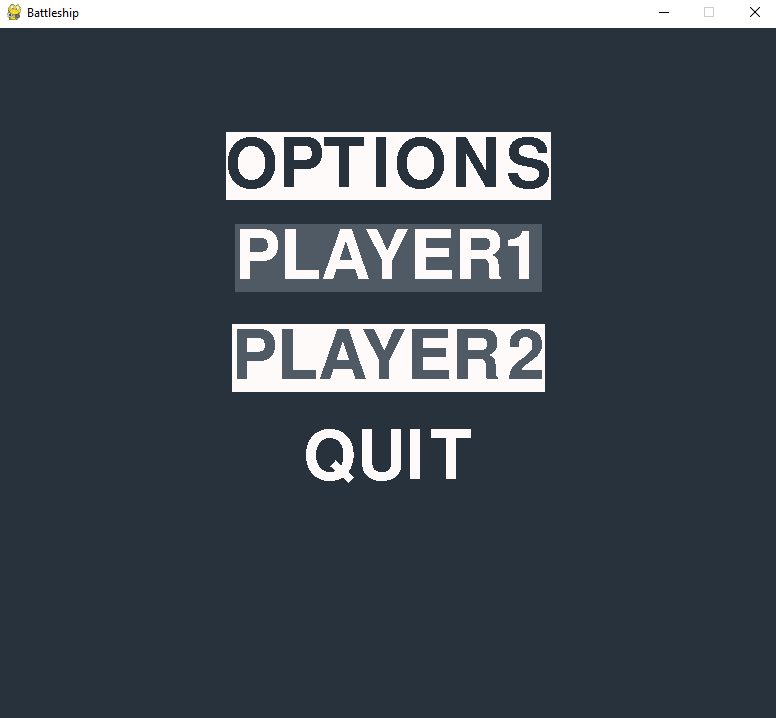


Рисунок 5.2.2.1 – Виставлення параметрів

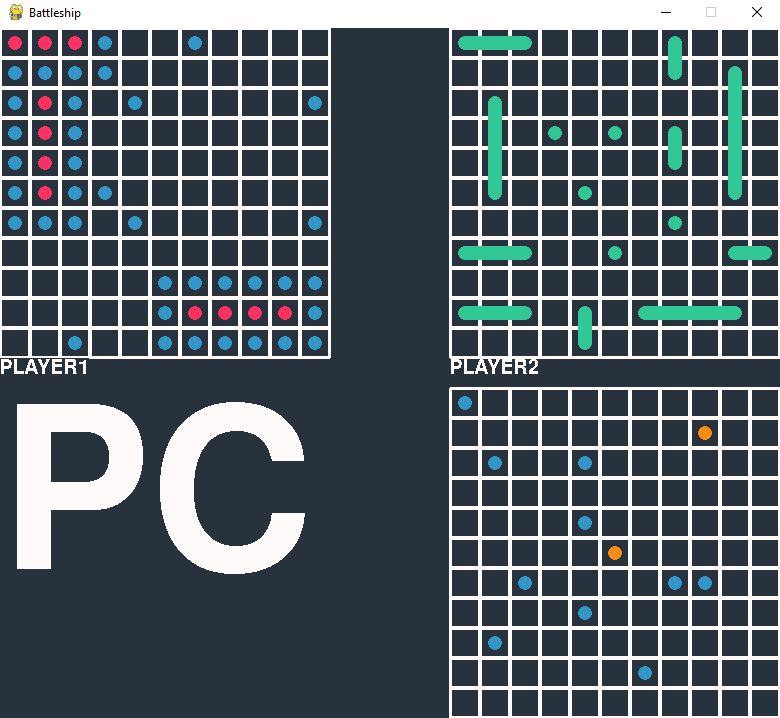
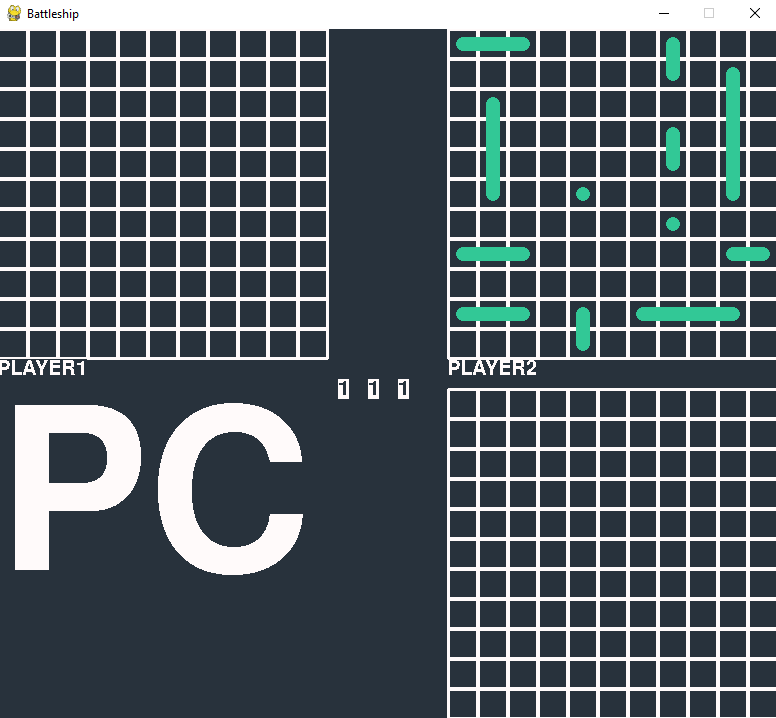


Рисунок 5.2.2.2 – Розставлення кораблів

Рисунок 5.2.2.3 – Гра

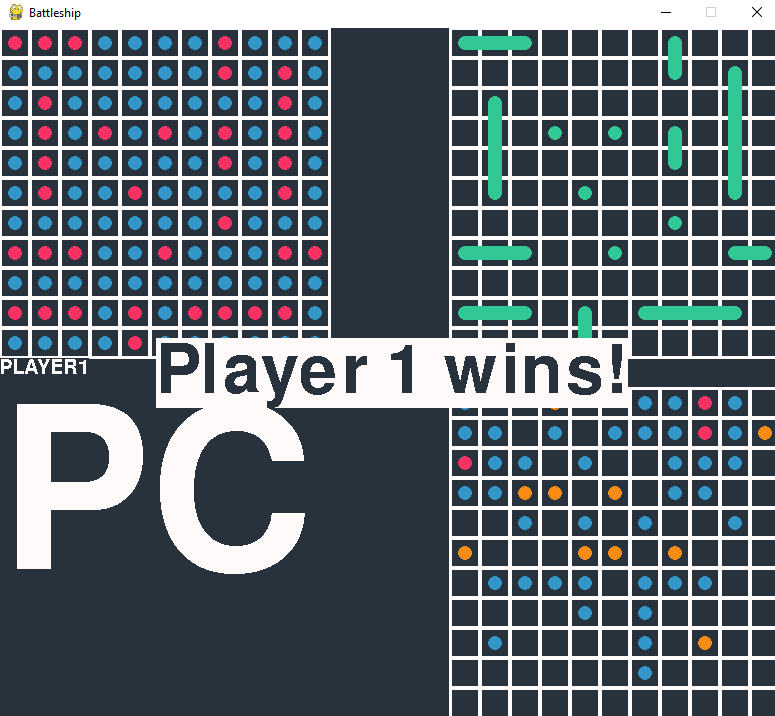
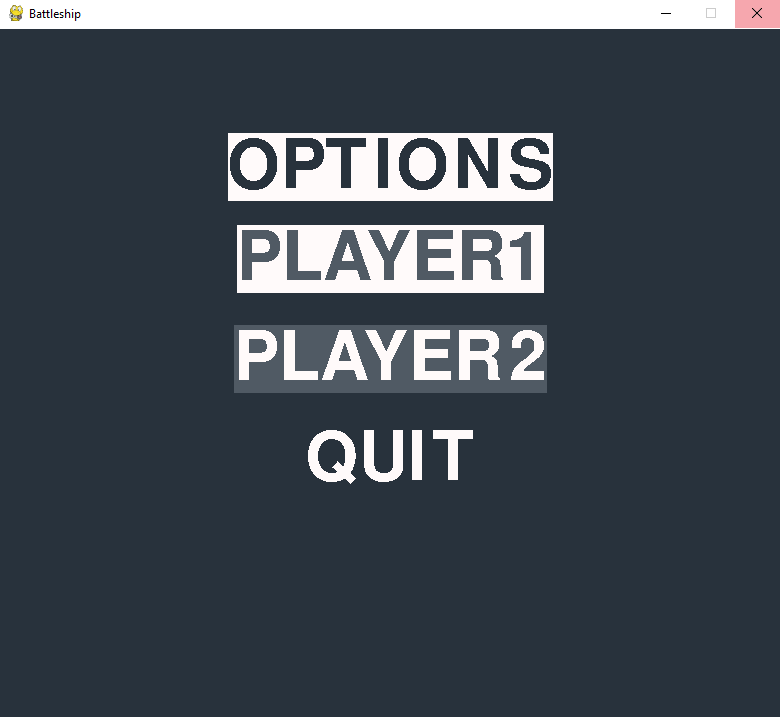


Рисунок 5.2.2.4 – Повідомлення про закінчення гри

### Випадок 3

Таблиця 5.2.3.1

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість двох людей грати між собою |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Гравець 1 – людина. Гравець 2 – комп’ютер. |
| Схема проведення тесту | Розпочати гру та дивитися за грою комп’ютерів |
| Очікуваний результат | Виведення повідомлення про перемогу одного з гравців |
| Стан програми після проведення випробувань | Один з гравців переміг і повідомлення висвітилося на екрані |



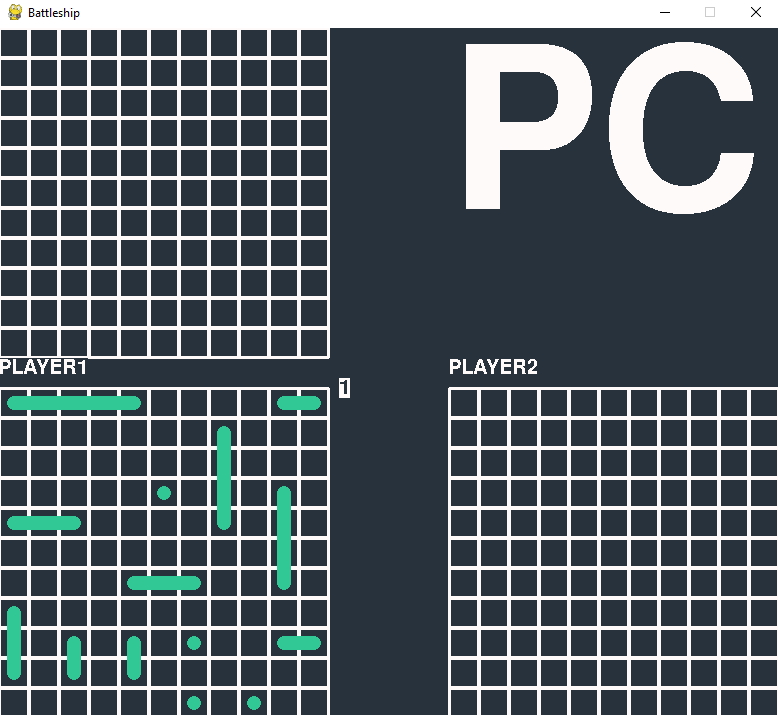
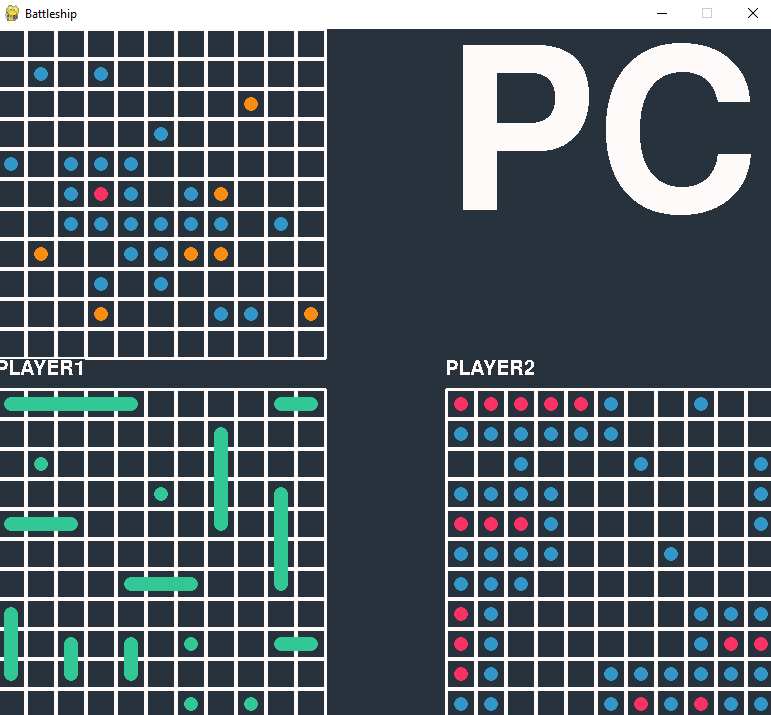
Рисунок 5.2.3.1 – Виставлення параметрів

Рисунок 5.2.3.2 – Розставлення кораблів



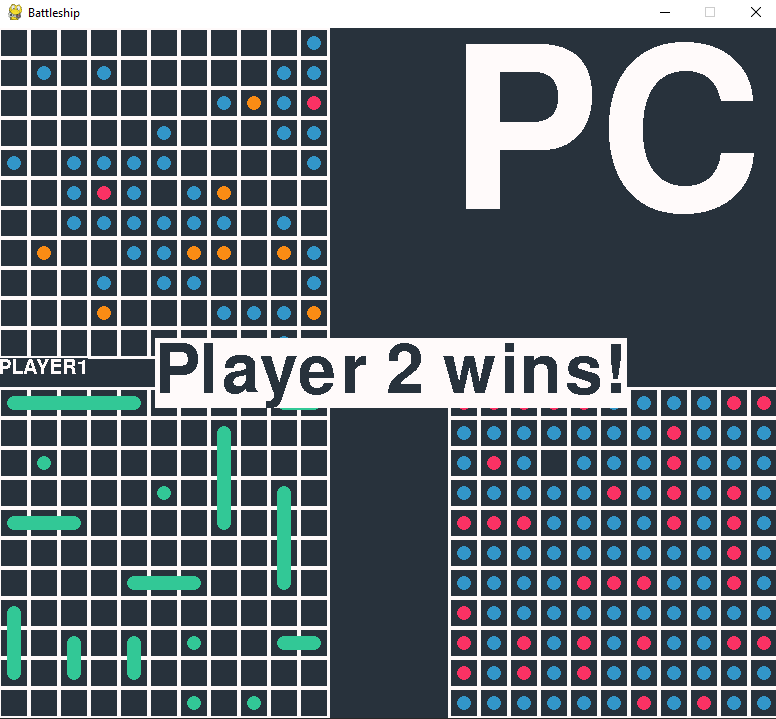
Рисунок 5.2.3.3 – Гра

Рисунок 5.2.3.4 – Повідомлення про закінчення гри

### Випадок 4

Таблиця 5.2.4.1

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість двох людей грати між собою |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Гравець 1 – комп’ютер. Гравець 2 – комп’ютер. |
| Схема проведення тесту | Розпочати гру та дивитися за грою комп’ютерів |
| Очікуваний результат | Виведення повідомлення про перемогу одного з гравців |
| Стан програми після проведення випробувань | Один з гравців переміг і повідомлення висвітилося на екрані |

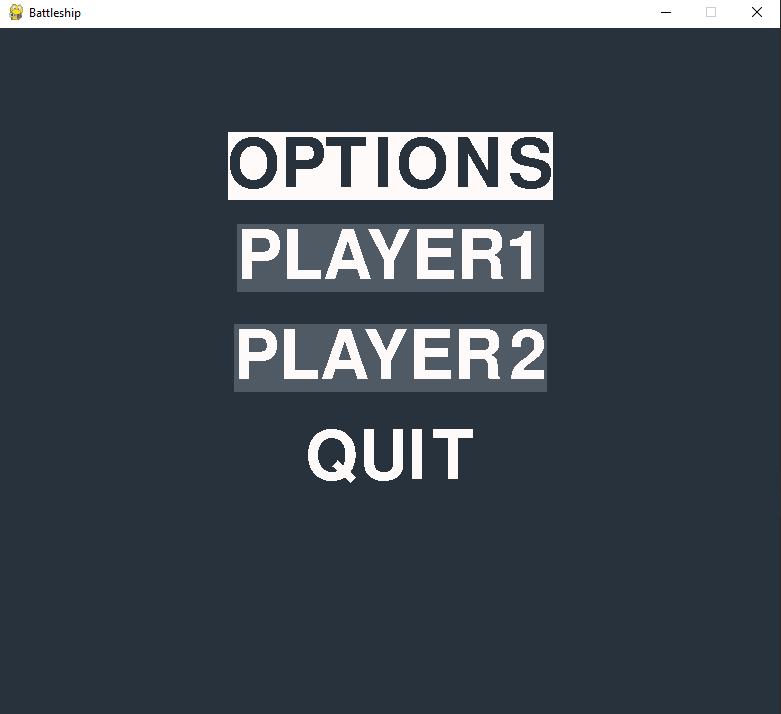
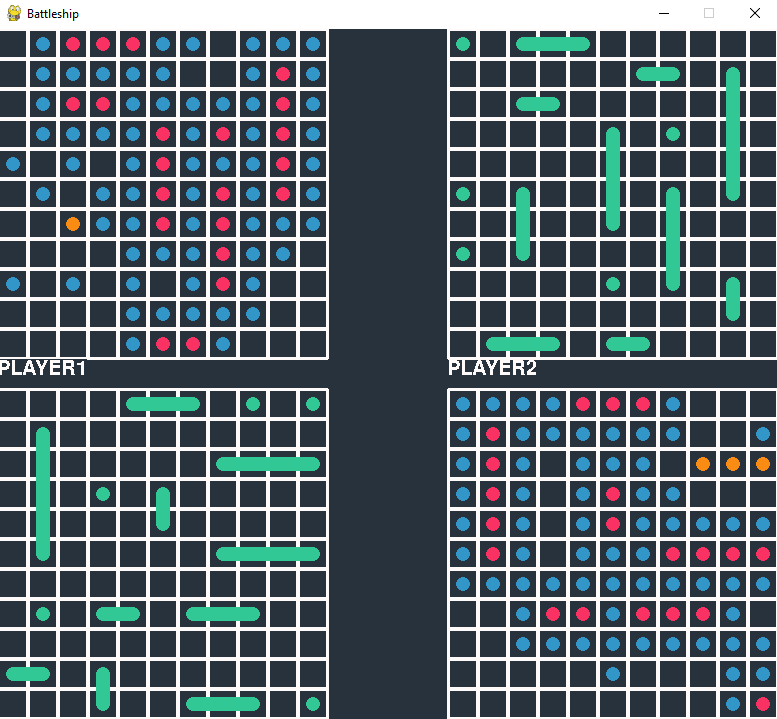
Рисунок 5.2.4.1 – Виставлення параметрів

Рисунок 5.2.4.2 – Гра

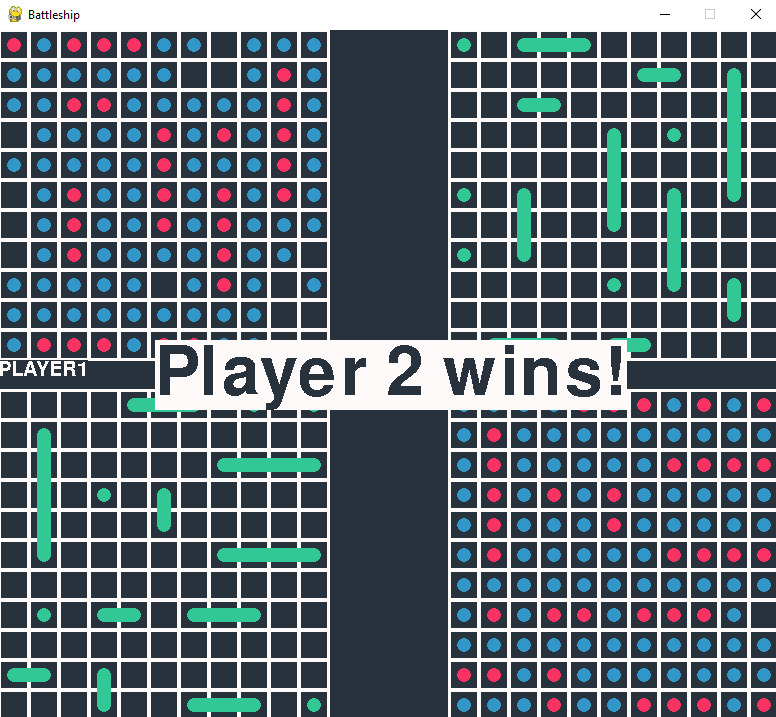


Рисунок 5.2.4.3 – Повідомлення про закінчення гри

# **ІНСТРУКЦІЯ КОРИСТУВАЧА**

## Робота з програмою

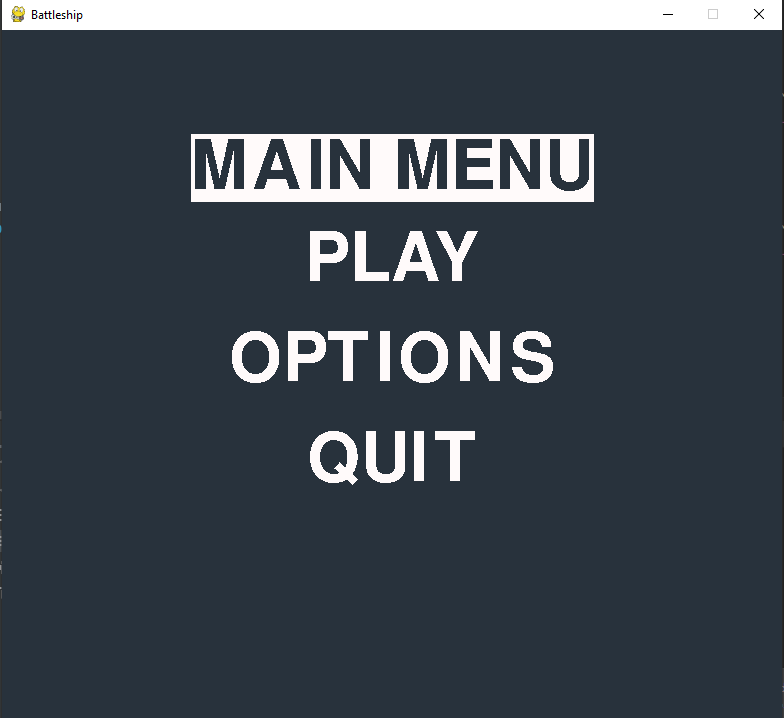
Після запуску виконавчого файлу з розширенням \*.exe, відкривається головне вікно програми (Рисунок 6.1.1).

Рисунок 6.1.1 – Головне вікно програми

Далі за допомогою вкладки OPTIONS вибираємо, чи гравці 1 nта 2 живими чи комп’ютерами(рисунок 6.1.2):

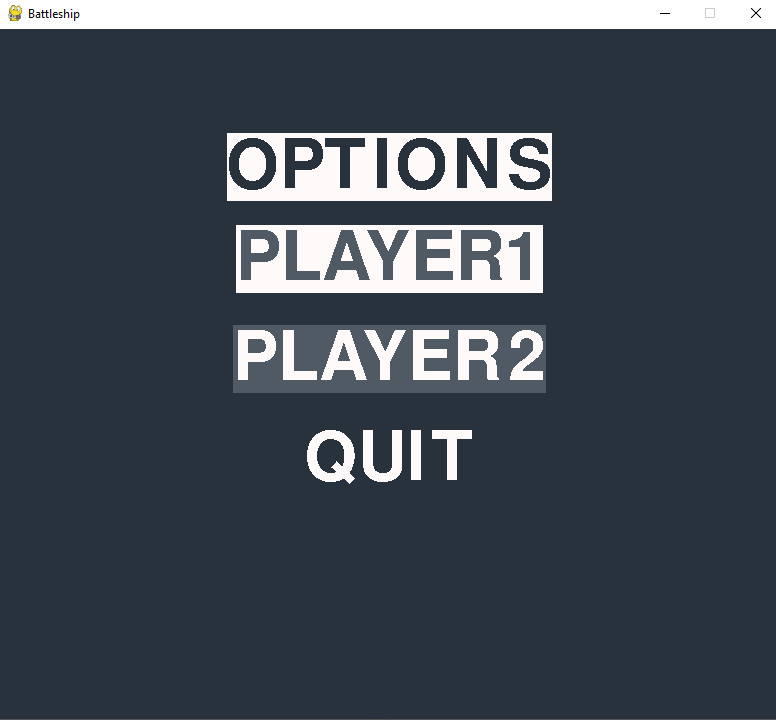
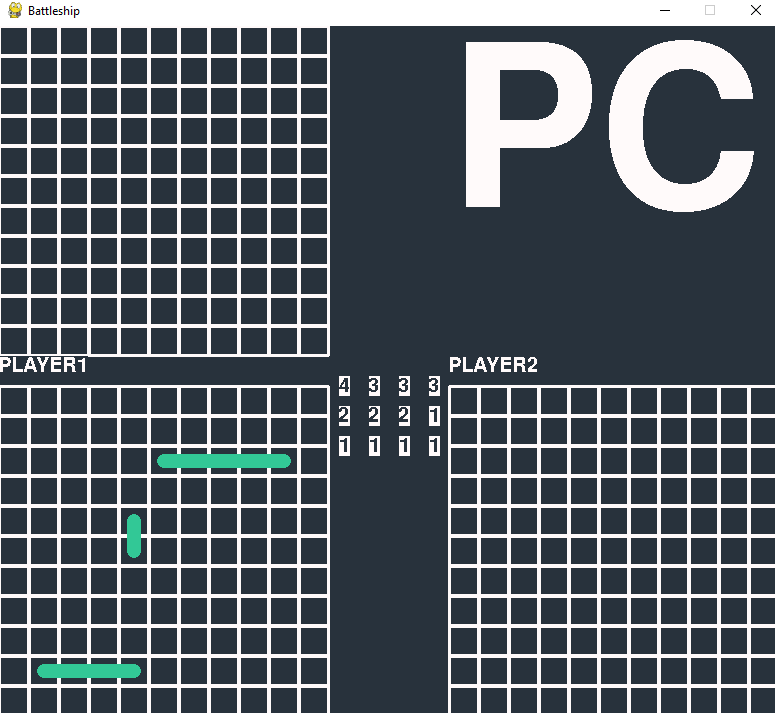
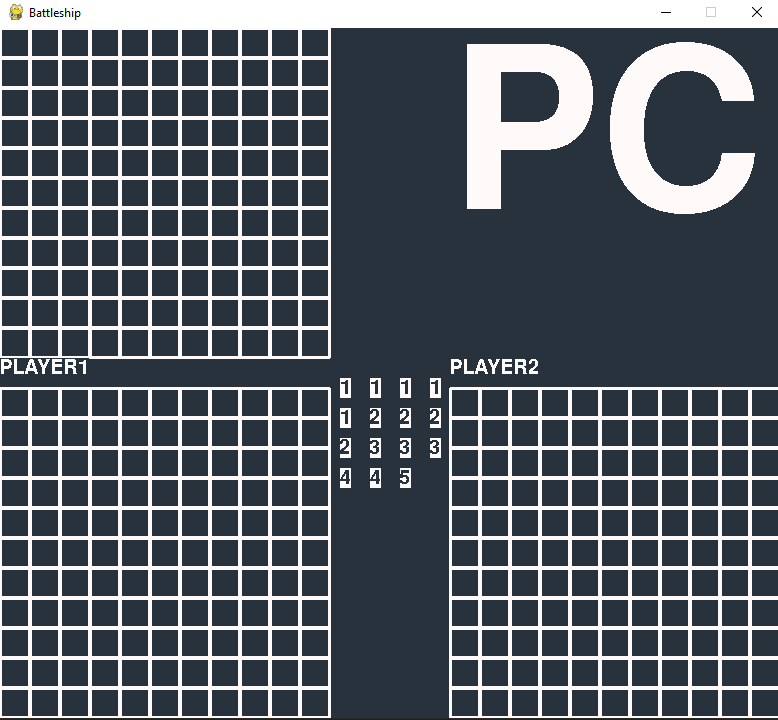


Рисунок 6.1.2 – Вибір гравців

Після цього натискаємо QUIT і PLAY, розставляємо кораблики, натискаючи на відповідний розмір корабля. Положення корабля залежить від ЛКМ( горизонтальне положення ) і ПКМ (вертикальне положення)(Рисунок 6.1.3 та 6.1.4):



Рисунки 6.1.3 та 6.1.4 – Розстановка кораблів

Натискаємо кнопку SET SHIPS( Рисунок 6.1.5) і розпочинаємо гру(Рисунок 6.1.6).

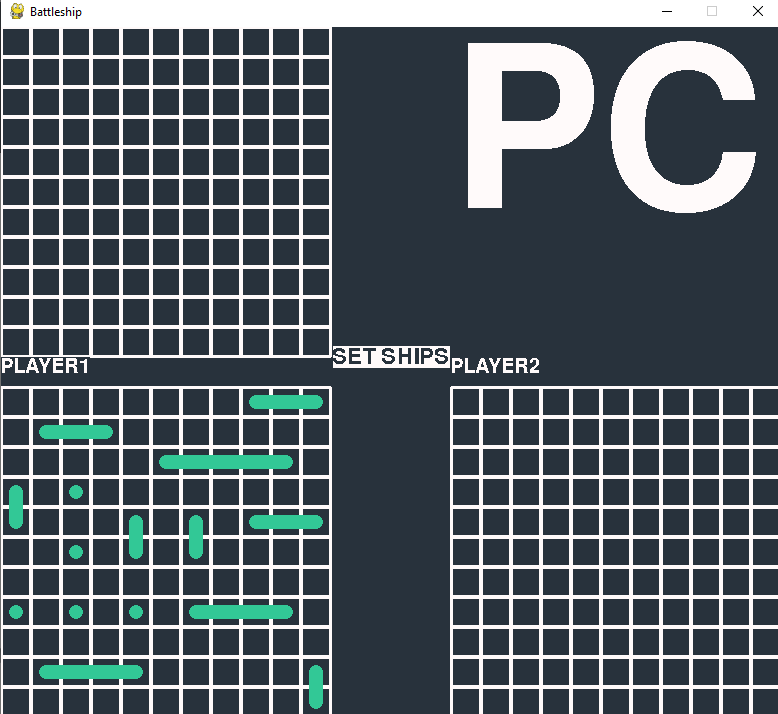


Рисунок 10.5 – Закінчити розстановку кораблі

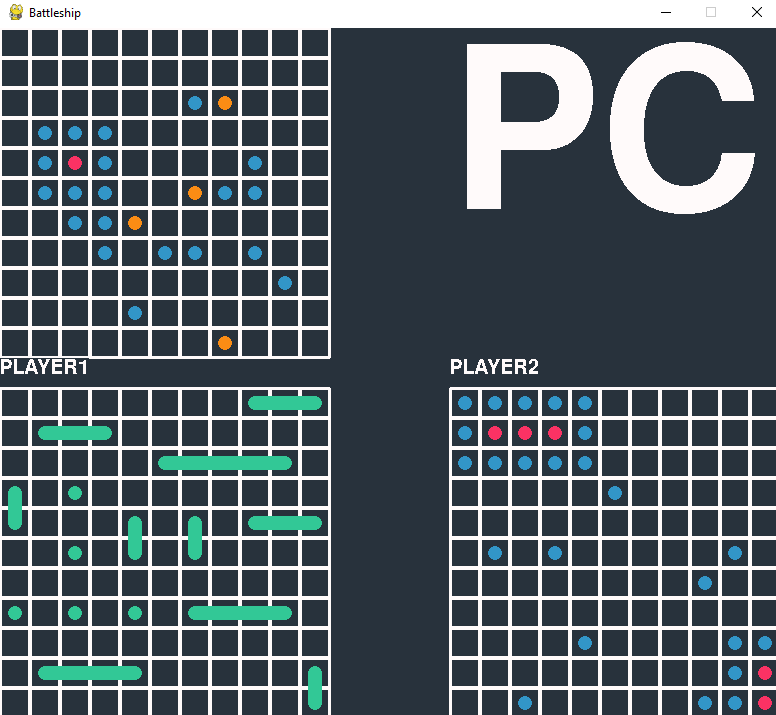


Рисунок 10.6 – Процес гри

Результатом виконання є повідомлення про те, який гравець переміг(Рисунок 10.7). Щоб розпочати гру заново, натисніть ESC.

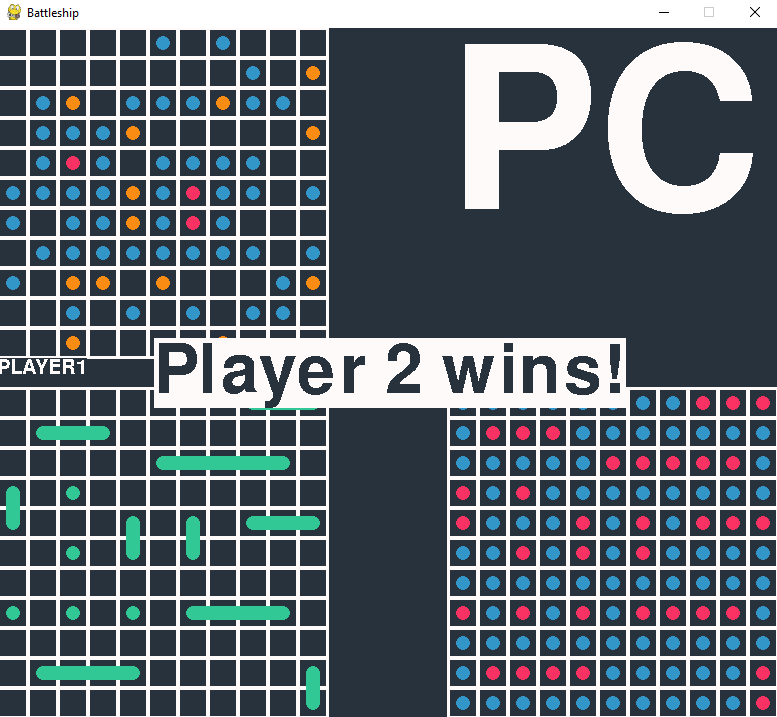


Рисунок 10.7 – Повідомлення про закінчення гри

## Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 10.1.

Таблиця 10.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10/Windows 11 (з останніми обновленнями) | Windows 7/ Windows 8/Windows 10/Windows 11  (з останніми обновленнями) |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/  Windows 8/Windows 10) | 2 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 800х600 | 1024х768 або краще |

# **АНАЛІЗ РЕЗУЛЬТАТІВ**

Під час курсової роботи було розроблено гру «Морський бій» з використанням ООП. Були описані алгоритми, програмне забезпечення з таблицею усіх методів класів та діаграма класів. Тестування програми було виконано успішно при всіх можливих налаштування, і програма видає очікуваний результат, що підтверджує правильність її роботи. Наведені системні вимоги щодо використання програми.

Висновки

Перелік посилань

Додаток А Технічне завдання

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_ р.

Виконавець:

Студент \_\_\_Панченко Сергій Віталійович*\_\_\_\_\_\_\_\_\_\_*

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_ р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: <*Тема курсової роботи*>

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*>.

1. 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.

* 1. *Стадії та етапи розробки*:

1. Об'єктно-орієнтований аналіз предметної області задачі (до\_\_.\_\_.202\_ р.)
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.).
   1. *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.

ДОДАТОК Б Лістинг

GAME\_CLASSES.PY

"""

Модуль, що описує ігрові класи

"""

import random

import inspect

import sys

import pygame

from game\_colors import WHITE, GREEN, GREY, LIGHT\_GREY, COLORS

from game\_constants import MAP\_SIZE, SHIPS\_SIZES, SQ\_SIZE, \

INDENT, FONT\_SIZE, WIDTH, HEIGHT, H\_MARGIN, V\_MARGIN

from other\_functions import get\_font

class Grid:

"""

Клас, що використовується для представлення градки

...

Атрибути

--------

screen : pygame.display.Surface

екран, на який буде намальована градка

player : Player

гравець, який володіє даною градкою

left : int

відступ від лівого боку екрана

top : int

відступ від верху екрана

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\_\_init\_\_(self, screen, player, left=0, top=0)

Конструктор класу Grid

Аргументи:

1) self - об'єкт класу

2) screen - екран

3) player - гравець

4) top - зміщення догори

5) left - зміщення наліво

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def draw\_grid(self, search=False)

Відображає об'єкт класу на екрані

Аргументи:

1) self - об'єкт класу

2) search - поле гравця

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def draw\_grid(self, search=False)

Відображає об'єкт класу на екрані

Аргументи:

1) self - об'єкт класу

2) search - поле гравця

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def check\_for\_input(self, position)

Реєструє нитискання на градку

Аргументи:

1) self - об'єкт класу

2) position - позиція миші

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def draw\_ships(self)

Відображає кораблі

Аргументи:

1) self - об'єкт класу

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, screen, player, left=0, top=0):

self.screen = screen

self.player = player

self.left = left

self.top = top

self.right = left + MAP\_SIZE \* SQ\_SIZE

self.bottom = top + MAP\_SIZE \* SQ\_SIZE

def draw\_grid(self, search=False):

for i in range(MAP\_SIZE \*\* 2):

x = self.left + i % MAP\_SIZE \* SQ\_SIZE

y = self.top + i // MAP\_SIZE \* SQ\_SIZE

square = pygame.Rect(x, y, SQ\_SIZE, SQ\_SIZE)

pygame.draw.rect(self.screen, WHITE, square, width=3)

if search:

x += SQ\_SIZE // 2

y += SQ\_SIZE // 2

pygame.draw.circle(self.screen,

COLORS[self.player.search[i]],

(x, y), radius=SQ\_SIZE // 4)

def check\_for\_input(self, position):

x1, y1 = self.left, self.top

x2 = self.left + (MAP\_SIZE \*\* 2 - 1) % MAP\_SIZE \* SQ\_SIZE

y2 = self.top + (MAP\_SIZE \*\* 2 - 1) // MAP\_SIZE \* SQ\_SIZE

top\_left\_square = pygame.Rect(x1, y1, SQ\_SIZE, SQ\_SIZE)

bottom\_right\_square = pygame.Rect(x2, y2, SQ\_SIZE, SQ\_SIZE)

row, col, index = None, None, None

if position[0] in range(top\_left\_square.left, bottom\_right\_square.right) \

and position[1] in range(top\_left\_square.top, bottom\_right\_square.bottom):

row = (position[1] - self.top) // SQ\_SIZE

col = (position[0] - self.left) // SQ\_SIZE

index = row \* MAP\_SIZE + col

return row, col, index

def draw\_ships(self):

for ship in self.player.ships:

x = self.left + ship.col \* SQ\_SIZE + INDENT

y = self.top + ship.row \* SQ\_SIZE + INDENT

if ship.orientation == 'h':

width = ship.size \* SQ\_SIZE - INDENT \* 2

height = SQ\_SIZE - INDENT \* 2

elif ship.orientation == 'v':

width = SQ\_SIZE - INDENT \* 2

height = ship.size \* SQ\_SIZE - INDENT \* 2

rectangle = pygame.Rect(x, y, width, height)

pygame.draw.rect(self.screen, GREEN, rectangle, border\_radius=15)

class Battleship:

"""

Клас, що представляє корабель

...

Атрибути

--------

row : int

Рядок, у якому розташовується ніс корабля

col : int

Колонка, у якій розташовується ніс корабля

size : int

Розмір корабля

orientation : str

Оріентація корабля

indexes : list

Список індексів, які займає корабель повністю

ship\_indexes : list

Список індексів, які власне займає лише корабель

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, size, row, col, orientation)

Конструктор класу

Аргументи:

1) self - об'єкт класу

2) size - розмір

3) row - рядок

4) col - колонка

5) orientation - орієнтація

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def compute\_indexes(self)

Розраховує розташування корабля

Аргументи:

1) self - об'єкт класу

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, size, row, col, orientation):

self.row = row

self.col = col

self.size = size

self.orientation = orientation

self.indexes = self.compute\_indexes()

self.ship\_indexes = self.indexes[:size]

def compute\_indexes(self):

start\_index = self.row \* MAP\_SIZE + self.col

result\_indexes = []

if self.orientation == 'h':

indexes = [start\_index + i for i in range(self.size)]

top\_left\_padding = [indexes[0] - 1 - MAP\_SIZE]

left\_padding = [indexes[0] - 1]

bottom\_left\_padding = [indexes[0] - 1 + MAP\_SIZE]

bottom\_padding = [i + MAP\_SIZE for i in indexes]

bottom\_right\_padding = [indexes[-1] + 1 + MAP\_SIZE]

right\_padding = [indexes[-1] + 1]

top\_right\_padding = [indexes[-1] + 1 - MAP\_SIZE]

top\_padding = [i - MAP\_SIZE for i in indexes]

result\_indexes += indexes

elif self.orientation == 'v':

indexes = [start\_index + i \* MAP\_SIZE for i in range(self.size)]

top\_left\_padding = [indexes[0] - 1 - MAP\_SIZE]

left\_padding = [i - 1 for i in indexes]

bottom\_left\_padding = [indexes[-1] - 1 + MAP\_SIZE]

bottom\_padding = [indexes[-1] + MAP\_SIZE]

bottom\_right\_padding = [indexes[-1] + 1 + MAP\_SIZE]

right\_padding = [i + 1 for i in indexes]

top\_right\_padding = [indexes[0] + 1 - MAP\_SIZE]

top\_padding = [indexes[0] - MAP\_SIZE]

result\_indexes += indexes

if left\_padding[0] // MAP\_SIZE == indexes[0] // MAP\_SIZE:

result\_indexes += left\_padding

if top\_padding[0] >= 0:

result\_indexes += top\_padding

if left\_padding[0] // MAP\_SIZE == indexes[0] // MAP\_SIZE and top\_padding[0] >= 0:

result\_indexes += top\_left\_padding

if bottom\_padding[0] < MAP\_SIZE \*\* 2:

result\_indexes += bottom\_padding

if left\_padding[0] // MAP\_SIZE == indexes[0] // MAP\_SIZE and bottom\_padding[0] < MAP\_SIZE \*\* 2:

result\_indexes += bottom\_left\_padding

if right\_padding[0] // MAP\_SIZE == indexes[0] // MAP\_SIZE:

result\_indexes += right\_padding

if right\_padding[0] // MAP\_SIZE == indexes[0] // MAP\_SIZE and bottom\_padding[0] < MAP\_SIZE \*\* 2:

result\_indexes += bottom\_right\_padding

if right\_padding[0] // MAP\_SIZE == indexes[0] // MAP\_SIZE and top\_padding[0] >= 0:

result\_indexes += top\_right\_padding

return result\_indexes

class Player:

"""

Клас, що представляє гравця

...

Атрибути

--------

is\_human : bool

Значення, чи є гравець людиною

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, is\_human)

Конструктор класу

Аргументи:

1) self - об'єкт класу

2) is\_human - значення, чи є гравець людиною

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

check\_ship(self, ship)

Перевіряє, чи є розташування корабля можливим

Аргументи:

1) self - об'єкт класу

2) ship - об'єкт класу Battleship

Повертає:

possible : bool

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def place\_ships\_ai(self, sizes)

Комп'ютер розставляє кораблі

Аргументи:

1) self - об'єкт класу

2) sizes - список розмірів кораблів

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, is\_human):

self.ships = []

self.search = ['U' for \_ in range(MAP\_SIZE \*\* 2)] # 'U' for unknown

self.is\_human = is\_human

if not self.is\_human:

print('lala')

self.place\_ships\_ai(sizes=SHIPS\_SIZES)

self.indexes = [

ind for sub in list(map(lambda x: x.ship\_indexes, self.ships))

for ind in sub

]

def check\_ship(self, ship):

possible = True

size = ship.size

for i in ship.indexes[:size]:

# indexes must be less than MAP\_SIZE\*\*2

if i >= MAP\_SIZE \*\* 2:

possible = False

break

# ships cannot go beyond grid

new\_row = i // MAP\_SIZE

if ship.orientation == 'h' and new\_row != ship.row:

possible = False

break

# ships cannot intersect

for other\_ship in self.ships:

if i in other\_ship.indexes:

possible = False

break

if not possible:

break

return possible

def place\_ships\_ai(self, sizes):

all\_placed = False

while not all\_placed:

for size in sizes[::-1]:

placed = False

counter = 0

while not placed:

# create new ship

ship = Battleship(size=size, row=random.randint(0, MAP\_SIZE - 1),

col=random.randint(0, MAP\_SIZE - 1),

orientation=random.choice(["h", "v"]))

# place the ship

if self.check\_ship(ship):

self.ships.append(ship)

placed = True

counter += 1

if counter > 100:

break

if not placed:

self.ships.clear()

break

if self.ships:

all\_placed = True

class GameLogic:

"""

Клас, що представляє об'єкт ігрової логіки

...

Атрибути

--------

player1 : Player

Перший гравець

player2 : Player

Другий гравець

player1\_turn : bool

Значення, чи ходить перший гравець на даному ході

computer\_turn : bool

Значення, чи ходить комп'ютер на даному ході

over : bool

Значення, чи закінчена гра

result : int

Значення, який гравець переміг

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, is\_human1, is\_human2)

Конструктор класу

Аргументи:

1) self - об'єкт класу

2) is\_human1 - значення, чи є гравець 1 людиною

3) is\_human2 - значення, чи є гравець 2 людиною

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def make\_move(self, i)

Робить хід

Аргументи:

1) self - об'єкт класу

2) I - індекс клітинки

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def computer\_algorithm(self)

Комп'ютер ходить по клітинці

Аргументи:

1) self - об'єкт класу

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, is\_human1, is\_human2):

self.player1 = Player(is\_human1)

self.player2 = Player(is\_human2)

self.player1\_turn = True

self.computer\_turn = True if not self.player1.is\_human else False

self.over = False

self.result = None

def make\_move(self, i):

player = self.player1 if self.player1\_turn else self.player2

opponent = self.player2 if self.player1\_turn else self.player1

hit = False

if player.search[i] != 'U':

hit = True

elif i in opponent.indexes:

player.search[i] = 'H'

hit = True

# check if ship is sunk

for ship in opponent.ships:

sunk = True

for i in ship.ship\_indexes:

if player.search[i] == 'U':

sunk = False

break

if sunk:

for i in ship.indexes:

player.search[i] = 'M'

for i in ship.ship\_indexes:

player.search[i] = 'S'

else:

player.search[i] = "M"

# check if game is over

game\_over = True

for i in opponent.indexes:

if player.search[i] == 'U':

game\_over = False

self.over = game\_over

self.result = 1 if self.player1\_turn else 2

# change the active player

if not hit:

self.player1\_turn = not self.player1\_turn

if (self.player1.is\_human and not self.player2.is\_human) or (

not self.player1.is\_human and self.player2.is\_human):

self.computer\_turn = not self.computer\_turn

def computer\_algorithm(self):

search = self.player1.search if self.player1\_turn else self.player2.search

unknown = [i for i, square in enumerate(search) if square == 'U']

hits = [i for i, square in enumerate(search) if square == 'H']

neighbours1 = []

neighbours2 = []

for u in unknown:

if u + 1 in hits or u - 1 in hits or u - MAP\_SIZE in hits or u + MAP\_SIZE in hits:

neighbours1.append(u)

if u + 2 in hits or u - 2 in hits or u - MAP\_SIZE \* 2 in hits or u + MAP\_SIZE \* 2 in hits:

neighbours2.append(u)

is\_both\_neighbour = False

for u in unknown:

if u in neighbours1 and u in neighbours2:

is\_both\_neighbour = True

self.make\_move(u)

break

if not is\_both\_neighbour:

if neighbours1:

self.make\_move(random.choice(neighbours1))

else:

# checker board pattern

checker\_board = []

for u in unknown:

row = u // MAP\_SIZE

col = u % MAP\_SIZE

if (row + col) % 2 == 0:

checker\_board.append(u)

if checker\_board:

self.make\_move(random.choice(checker\_board))

else:

if unknown:

random\_index = random.choice(unknown)

self.make\_move(random\_index)

return

class Button:

"""

Клас, що представляє кнопку

...

Атрибути

--------

x : int

Позиція кнопки по горизонталі

y : int

Позиція кнопки по вертикалі

font : pygame.Font

Шрифт тексту

base\_color : tuple

Базовий колір кнопки

hovering\_color : tuple

Другорядний колір кнопки

text\_input : str

Текст кнопки

text : pygame.SysFont

Власне рендерений текст кнопки

text\_rect : pygame.Rect

Прямокутник тексту

func : FunctionType

Функція, прив'язана до кнопки

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, pos, font, text\_input, base\_color, hovering\_color, func)

Конструктор класу

Аргументи:

1) self - об'єкт класу

2) pos - позиція миші

3) font - шрифт тексту

4) text\_input - текст кнопки

5) base\_color - основний колір

6) hovering\_color - другорядний колір

7) func - функція

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def update(self, screen)

Відображає об'єкт класу на екрані

Аргументи:

1) self - об'єкт класу

2) screen - екран

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def check\_for\_input(self, position)

Відображає об'єкт класу на екрані

Аргументи:

1) self - об'єкт класу

2) position - позиція миші

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def change\_color(self, position)

Змінює колір при наведенні

Аргументи:

1) self - об'єкт класу

2) position - позиція миші

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, pos, font, text\_input, base\_color, hovering\_color, func):

self.x = pos[0]

self.y = pos[1]

self.font = font

self.base\_color = base\_color

self.hovering\_color = hovering\_color

self.text\_input = text\_input

self.text = self.font.render(self.text\_input, False,

self.base\_color, self.hovering\_color)

self.text\_rect = self.text.get\_rect(center=(self.x, self.y))

self.func = func

def update(self, screen):

screen.blit(self.text, self.text\_rect)

def check\_for\_input(self, position):

if position[0] in range(self.text\_rect.left, self.text\_rect.right) \

and position[1] in range(self.text\_rect.top, self.text\_rect.bottom):

return True

return False

def change\_color(self, position):

if position[0] in range(self.text\_rect.left, self.text\_rect.right) \

and position[1] in range(self.text\_rect.top, self.text\_rect.bottom):

self.text = self.font.render(self.text\_input, False, self.hovering\_color)

else:

self.text = self.font.render(self.text\_input, False, self.base\_color)

class TumblerButton(Button):

"""

Клас, що представляє кнопку-тумблер

...

Батьки класу

------------

Button

Атрибути

--------

x : int

Позиція кнопки по горизонталі

y : int

Позиція кнопки по вертикалі

font : pygame.Font

Шрифт тексту

base\_color : tuple

Базовий колір кнопки

hovering\_color : tuple

Другорядний колір кнопки

text\_input : str

Текст кнопки

text : pygame.SysFont

Власне рендерений текст кнопки

text\_rect : pygame.Rect

Прямокутник тексту

func : FunctionType

Функція, прив'язана до кнопки

switched : bool

Значення, чи є кнопка-тумблер увімкненою

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, pos, font, text\_input, base\_color, hovering\_color, func)

Конструктор класу

Аргументи:

1) self - об'єкт класу

2) pos - позиція миші

3) font - шрифт тексту

4) text\_input - текст кнопки

5) base\_color - основний колір

6) hovering\_color - другорядний колір

7) func - функція

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def update(self, screen)

Відображає об'єкт класу на екрані

Аргументи:

1) self - об'єкт класу

2) screen - екран

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def check\_for\_input(self, position)

Відображає об'єкт класу на екрані

Аргументи:

1) self - об'єкт класу

2) position - позиція миші

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def change\_color(self, position)

Змінює колір при наведенні

Аргументи:

1) self - об'єкт класу

2) position - позиція миші

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, pos, font, text\_input, base\_color, hovering\_color, func):

Button.\_\_init\_\_(self, pos, font, text\_input, base\_color, hovering\_color, func)

self.switched = False

def change\_color(self, position):

if self.switched:

self.text = self.font.render(self.text\_input, False, self.hovering\_color, self.base\_color)

else:

self.text = self.font.render(self.text\_input, False, self.base\_color, self.hovering\_color)

class Menu:

"""

Клас, що представляє меню

...

Атрибути

--------

screen : pygame.display.Surface

Екран, на якому малюється меню

text : pygame.SysFont

Текст меню

rect : pygame.Rect

Прямокутник тексту

buttons : list

Список кнопок меню

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, screen, text, buttons, funcs)

Конструктор класу

Аргументи:

1) self -об'єкт класу

2) screen - екран

3) text - основний текст

4) buttons - об'єкти класу Buttons

5) funcs - функції до кнопок

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def draw(self, \*\*kwargs)

Відображає об'єкт класу на екрані

Аргументи:

1) self - об'єкт класу

2) \*\*kwargs - словник назв кнопок та їх функцій

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, screen, text, buttons, funcs):

self.screen = screen

self.text = get\_font(FONT\_SIZE).render(text, False, GREY, WHITE)

self.rect = self.text.get\_rect(center=(WIDTH // 2, HEIGHT // 5))

self.buttons = []

for button, func in zip(enumerate(buttons), funcs):

self.buttons.append(button[1][1](pos=(WIDTH // 2, HEIGHT // 3 + FONT\_SIZE \* button[0]),

text\_input=button[1][0],

font=get\_font(FONT\_SIZE), base\_color=WHITE,

hovering\_color=LIGHT\_GREY, func=func))

self.buttons.append(Button(pos=(WIDTH // 2, HEIGHT // 3 + FONT\_SIZE \* (button[0] + 1)),

text\_input='QUIT',

font=get\_font(FONT\_SIZE), base\_color=WHITE,

hovering\_color=LIGHT\_GREY, func=None))

def draw(self, \*\*kwargs):

while True:

self.screen.fill(GREY)

self.screen.blit(self.text, self.rect)

mouse\_pos = pygame.mouse.get\_pos()

for button in self.buttons:

button.change\_color(mouse\_pos)

button.update(self.screen)

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.MOUSEBUTTONDOWN:

for button in self.buttons:

if button.check\_for\_input(mouse\_pos):

if button.text\_input == 'QUIT':

return

elif button.\_\_class\_\_.\_\_name\_\_ == Button.\_\_name\_\_:

if inspect.signature(button.func).parameters:

button.func(\*\*(kwargs[button.text\_input]))

else:

button.func()

elif button.\_\_class\_\_.\_\_name\_\_ == TumblerButton.\_\_name\_\_:

button.switched = not button.switched

pygame.display.update()

class Board:

"""

Клас, що представляє дошку

...

Атрибути

--------

screen : pygame.Surface

Екран, на який малюється дошка

game\_logic : GameLogic

Об'єкт ігрової логіки

options\_menu : Menu

Об'єкт меню опцій

search\_grid1 : Grid

Пошукова градка першого гравця

search\_grid2 : Grid

Пошукова градка другого гравця

position\_grid1 : Grid

Позиційна градка першого гравця

position\_grid2 : Grid

Позиційна градка другого гравця

set\_ships\_button : TumblerButton

Кнопка дя закінчення розставлення кораблів

ship\_size\_buttons : list

Кнопки, які позначають кораблі

used\_ship\_size\_buttons : list

Використані кнопки, що позначають кораблі

Методи

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, screen, text, buttons, funcs)

Конструктор класу

Аргументи:

1) self - об'єкт класу

2) screen - екран

3) options\_menu - об'єкт класу Menu

4) game\_logic - об'єкт класу GameLogic

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

draw(self, draw\_buttons)

Відображає дошку на екрані

Аргументи:

1) self - об'єкт класу

2) draw\_buttons - значення, чи треба відображати об'єкти класу Button

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def check\_for\_input(self, position, player1\_turn)

Реєструє нитискання на дошку

Аргументи:

1) self - об'єкт класу

2) position - позиція миші

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def order\_buttons(self)

Розташовує кнопки у правильному порядку

Аргументи:

1) self - об'єкт класу

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, screen, options\_menu, game\_logic):

self.screen = screen

self.game\_logic = game\_logic

self.options\_menu = options\_menu

self.search\_grid1 = Grid(screen=self.screen, player=self.game\_logic.player1)

self.search\_grid2 = Grid(screen=self.screen, player=self.game\_logic.player2,

left=(WIDTH - H\_MARGIN) // 2 + H\_MARGIN,

top=(HEIGHT - V\_MARGIN) // 2 + V\_MARGIN)

self.position\_grid1 = Grid(screen=self.screen, player=self.game\_logic.player1,

top=(HEIGHT - V\_MARGIN) // 2 + V\_MARGIN)

self.position\_grid2 = Grid(screen=self.screen, player=self.game\_logic.player2,

left=(WIDTH - H\_MARGIN) // 2 + H\_MARGIN)

self.set\_ships\_button = TumblerButton(pos=(WIDTH // 2, SQ\_SIZE \* MAP\_SIZE),

text\_input='SET SHIPS', font=get\_font(FONT\_SIZE // 3),

base\_color=GREY, hovering\_color=WHITE, func=None)

self.ship\_size\_buttons = []

self.used\_ship\_size\_buttons = []

for i, size in enumerate(SHIPS\_SIZES):

x, y = i % 4, i // 4

print(x, y)

self.ship\_size\_buttons.append(TumblerButton(pos=(self.search\_grid1.right + x \* SQ\_SIZE + SQ\_SIZE // 2,

self.search\_grid1.bottom + y \* SQ\_SIZE + SQ\_SIZE),

font=get\_font(SQ\_SIZE), text\_input=str(size), base\_color=GREY,

hovering\_color=WHITE, func=None))

def draw(self, draw\_buttons):

self.screen.fill(GREY)

# draw\_grid search grids

self.search\_grid1.draw\_grid(search=True)

self.search\_grid2.draw\_grid(search=True)

# draw labels for players

text1 = "PLAYER1"

textbox1 = get\_font(SQ\_SIZE).render(text1, False, WHITE, GREY)

self.screen.blit(textbox1, (self.search\_grid1.left, self.search\_grid1.bottom))

text2 = "PLAYER2"

textbox2 = get\_font(SQ\_SIZE).render(text2, False, WHITE, GREY)

self.screen.blit(textbox2, (self.position\_grid2.left, self.position\_grid2.bottom))

# draw\_grid buttons

if draw\_buttons:

if self.ship\_size\_buttons:

for button in self.ship\_size\_buttons:

button.update(self.screen)

else:

self.set\_ships\_button.update(self.screen)

if self.game\_logic.player1.is\_human ^ self.game\_logic.player2.is\_human:

text = "PC"

textbox = get\_font(SQ\_SIZE \* MAP\_SIZE).render(text, False, WHITE, GREY)

if self.game\_logic.player1.is\_human:

self.position\_grid1.draw\_grid()

self.position\_grid1.draw\_ships()

self.screen.blit(textbox, (self.position\_grid2.left, self.position\_grid2.top))

else:

self.position\_grid2.draw\_grid()

self.position\_grid2.draw\_ships()

self.screen.blit(textbox, (self.position\_grid1.left, self.position\_grid1.top))

else:

# draw\_grid position grids

self.position\_grid1.draw\_grid()

self.position\_grid2.draw\_grid()

# draw\_grid ships onto position grids

self.position\_grid1.draw\_ships()

self.position\_grid2.draw\_ships()

def check\_for\_input(self, position, player1\_turn):

row, col, index = None, None, None

if player1\_turn:

row, col, index = self.search\_grid1.check\_for\_input(position)

elif not player1\_turn:

row, col, index = self.search\_grid2.check\_for\_input(position)

return row, col, index

def order\_buttons(self):

self.ship\_size\_buttons.sort(key=lambda x: int(x.text\_input), reverse=True)

for i, button in enumerate(self.ship\_size\_buttons):

x, y = i % 4, i // 4

button.text\_rect = button.text.get\_rect(

center=(SQ\_SIZE \* MAP\_SIZE + x \* SQ\_SIZE + SQ\_SIZE // 2,

SQ\_SIZE \* MAP\_SIZE + y \* SQ\_SIZE + SQ\_SIZE))

class Game:

"""

Клас, що представляє гру

...

Атрибут

-------

screen : pygame.Surface

Екран, на який малюються об'єкти гри

options\_menu : Menu

Об'єкт меню опцій

game\_logic : GameLogic

Об'єкт ігрової логіки

board : Board

Об'єкт дошки

Мутоди

------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def \_\_init\_\_(self, screen, options\_menu)

Розташовує кнопки у правильному порядку

Аргументи:

1) self - об'єкт класу

2) sceen - екран

3) options\_menu - об'єкт класу Menu

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def prepare\_to\_play(self)

Підготовує гру до початку

Аргументи:

1) self - об'єкт класу

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def play(self)

Підготовує гру до початку

Аргументи:

1) self - об'єкт класу

Повертає: None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

def \_\_init\_\_(self, screen, options\_menu):

self.screen = screen

self.options\_menu = options\_menu

self.game\_logic = None

self.board = None

def prepare\_to\_play(self):

# set game\_logic and board for game

self.game\_logic = GameLogic(self.options\_menu.buttons[0].switched,

self.options\_menu.buttons[1].switched)

self.board = Board(self.screen, self.options\_menu, self.game\_logic)

# check if player is human

for player\_number, player in enumerate((self.game\_logic.player1, self.game\_logic.player2)):

# if human then player must place its ships onto the board

if player.is\_human:

# while all player`s ships are not placed, it can not start game

while not self.board.set\_ships\_button.switched:

placing\_ships = False

GAME\_MOUSE\_POSITION = pygame.mouse.get\_pos()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

# escape key to close the animation

if event.type == pygame.K\_ESCAPE:

return

if event.type == pygame.MOUSEBUTTONDOWN:

# click on ship\_size\_buttons and chose size of a ship

for i1, button in enumerate(self.board.ship\_size\_buttons):

if button.check\_for\_input(GAME\_MOUSE\_POSITION):

button.switched = not button.switched

button.change\_color(GAME\_MOUSE\_POSITION)

for i2, other\_button in enumerate(self.board.ship\_size\_buttons):

if i2 != i1:

other\_button.switched = False

other\_button.change\_color(GAME\_MOUSE\_POSITION)

break

# chose square to put ship onto

for i1, button in enumerate(self.board.ship\_size\_buttons):

if button.switched:

placing\_ships = True

row, col, index = self.board.\_\_dict\_\_[f'position\_grid{player\_number + 1}']. \

check\_for\_input(GAME\_MOUSE\_POSITION)

if index is not None:

orientation = 'h' if pygame.mouse.get\_pressed()[0] else 'v'

ship = Battleship(size=int(button.text\_input), row=row,

col=col, orientation=orientation)

if player.check\_ship(ship):

player.ships.append(ship)

self.board.used\_ship\_size\_buttons. \

append(self.board.ship\_size\_buttons.pop(i1))

self.board.order\_buttons()

# chose ship to be replaced

if not placing\_ships:

for i1, ship in enumerate(player.ships):

row, col, index = self.board.\_\_dict\_\_[f'position\_grid{player\_number + 1}']. \

check\_for\_input(GAME\_MOUSE\_POSITION)

if index in ship.ship\_indexes:

for i2, used\_button in enumerate(self.board.used\_ship\_size\_buttons):

if ship.size == int(used\_button.text\_input):

used\_button.switched = False

used\_button.change\_color(GAME\_MOUSE\_POSITION)

self.board.ship\_size\_buttons. \

append(self.board.used\_ship\_size\_buttons.pop(i2))

break

self.board.order\_buttons()

player.ships.pop(i1)

# check if set\_button is pressed

if self.board.set\_ships\_button.check\_for\_input(GAME\_MOUSE\_POSITION) \

and not self.board.ship\_size\_buttons:

self.board.set\_ships\_button.switched = not self.board.set\_ships\_button.switched

self.board.draw(draw\_buttons=True)

pygame.display.update()

self.board.ship\_size\_buttons, self.board.used\_ship\_size\_buttons = \

self.board.used\_ship\_size\_buttons, self.board.ship\_size\_buttons

self.board.set\_ships\_button.switched = not self.board.set\_ships\_button.switched

# player`s ships being placed, program sets player`s indexes

player.indexes = [

ind for sub in list(map(lambda x: x.ship\_indexes, player.ships))

for ind in sub

]

self.play()

def play(self):

pause = False

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

# user mouse clicks

if event.type == pygame.MOUSEBUTTONDOWN and not pause:

mouse\_pos = pygame.mouse.get\_pos()

if not self.game\_logic.over:

row, col, index = self.board.check\_for\_input(mouse\_pos, self.game\_logic.player1\_turn)

if index is not None:

self.game\_logic.make\_move(index)

if event.type == pygame.KEYDOWN:

# escape key to close the animation

if event.key == pygame.K\_ESCAPE:

return

# space bar to pause and unpause the animation

if event.key == pygame.K\_SPACE and not self.game\_logic.over:

pause = not pause

if not pause:

self.board.draw(draw\_buttons=False)

# computer moves

if not self.game\_logic.over and self.game\_logic.computer\_turn:

self.game\_logic.computer\_algorithm()

# game over message

if self.game\_logic.over:

text = "Player " + str(self.game\_logic.result) + " wins!"

textbox = get\_font(FONT\_SIZE).render(text, False, GREY, WHITE)

self.screen.blit(textbox, (WIDTH // 5, HEIGHT // 2.22))

elif pause and not self.game\_logic.over:

text = "Pause"

textbox = get\_font(FONT\_SIZE).render(text, False, GREY, WHITE)

self.screen.blit(textbox, (WIDTH // 2.7, HEIGHT // 2.22))

pygame.time.delay(0)

pygame.display.update()

GAME\_CONSTANTS.PY

"""

Модуль для зберігання ігрових констант

"""

#Назва

TITLE = 'Battleship'

#Розмір пами

MAP\_SIZE = 11

#Кораблі

SHIPS\_SIZES = [1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 5]

#Шрифт

FONT\_NAME = 'fresansttf'

#Розмір шрифта

FONT\_SIZE = 100

#Розмір клітинки

SQ\_SIZE = 30

#Ширина градки

INDENT = 8

#Відступ між градками по горизонталі

H\_MARGIN = SQ\_SIZE \* 4

#Відступ між градками по вертикалі

V\_MARGIN = SQ\_SIZE

#Ширина екрана

WIDTH = SQ\_SIZE \* MAP\_SIZE \* 2 + H\_MARGIN

#Висота екрана

HEIGHT = SQ\_SIZE \* MAP\_SIZE \* 2 + V\_MARGIN

GAME\_COLORS.PY

"""

Модуль, що описує ігрові кольори

"""

GREY = (40, 50, 60)

LIGHT\_GREY = (80, 90, 100)

WHITE = (255, 250, 250)

GREEN = (50, 200, 150)

BLUE = (50, 150, 200)

RED = (250, 50, 100)

ORANGE = (250, 140, 20)

COLORS = {"U": GREY, "M": BLUE, 'H': ORANGE, "S": RED}

OTHER\_FUNCS.PY

"""

Модуль, що описує додаткові функції

"""

import pygame

from game\_constants import FONT\_NAME

def get\_font(font\_size):

"""

Функція, що видає шрифт потрібного розміру

:param font\_size: int

:return: pygame.SysFont

"""

return pygame.font.SysFont(FONT\_NAME, font\_size)

MAIN.PY

"""

Основний файл, у якому створюються усі об'єкти

та ініціалізуються налаштування гри

"""

import pygame

from game\_constants import TITLE, WIDTH, HEIGHT

from game\_classes import Button, TumblerButton, Menu, Game

pygame.init()

pygame.font.init()

pygame.display.set\_caption(TITLE)

is\_human1 = True

is\_human2 = True

SCREEN = pygame.display.set\_mode((WIDTH, HEIGHT))

OPTIONS\_MENU\_TEXT = 'OPTIONS'

OPTIONS\_MENU\_BUTTONS = [['PLAYER1', TumblerButton],

['PLAYER2', TumblerButton]]

OPTIONS\_MENU\_FUNCS = [None, None]

options\_menu = Menu(screen=SCREEN, text=OPTIONS\_MENU\_TEXT,

buttons=OPTIONS\_MENU\_BUTTONS,

funcs=OPTIONS\_MENU\_FUNCS)

options\_menu.buttons[0].switched = False

options\_menu.buttons[1].switched = False

game = Game(screen=SCREEN, options\_menu=options\_menu)

MAIN\_MENU\_TEXT = 'MAIN MENU'

MAIN\_MENU\_BUTTONS = [['PLAY', Button],

['OPTIONS', Button]]

MAIN\_MENU\_FUNCS = [game.prepare\_to\_play, options\_menu.draw]

main\_menu = Menu(screen=SCREEN, text=MAIN\_MENU\_TEXT,

buttons=MAIN\_MENU\_BUTTONS,

funcs=MAIN\_MENU\_FUNCS)

main\_menu.draw(PLAY={'None': None}, OPTIONS={'None': None})