МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

**КУРСОВА РОБОТА**

з основ програмування

на тему: Комп’ютерна гра «Стрілки»

Студента 1 курсу, групи ІП-11

Лисенка Андрія Юрійовича

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

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-11 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

**Лисенка Андрія Юрійовича**

1. Тема роботи Комп'ютерна гра "Стрілки"

2. Строк здачі студентом закінченої роботи 15.06.22

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: 9 сторінки, 40 рисунків, 18 таблиць, 2 посилання.

Об’єкт дослідження: гра "Стрілки".

Мета роботи: дослідження методів розробки програмного забезпечення

Виконана робота з дослідженням розробки ПЗ методом ООП та функціональних можливостей мови програмування та бібліотек для створення комп'ютерних ігор, а саме можливості ООП мови python та бібліотеки pygame. Приведена змістовна пояснювальна записка, з постановкою задачі та детальним описом об'єкто-оріентованого аналізу та об'єкто-оріентованої архітектури проекту.

Виконана програмна реалізація гри "Стрілки".

ЗМІСТ

[ВСТУП 7](#_Toc105272726)

[1 ПОСТАНОВКА ЗАДАЧІ 8](#_Toc105272727)

[2 ТЕОРЕТИЧНІ ВІДОМОСТІ 9](#_Toc105272728)

[2.1 Основні правила гри 9](#_Toc105272729)

[2.2 Інтерфейс гравця 9](#_Toc105272730)

[2.3 Задача гравця 9](#_Toc105272731)

[2.4 Діаграма прецендентів 10](#_Toc105272732)

[3 ОПИС АЛГОРИТМІВ 11](#_Toc105272733)

[3.1 Генерація стрілок 11](#_Toc105272734)

[3.2 Знаходження можливих напрямків стрілки 11](#_Toc105272735)

[3.3 Знаходження позиції стрілки 12](#_Toc105272736)

[3.4 Знаходження чисел до яких напрямлена стрілка 12](#_Toc105272737)

[3.5 Знаходження стрілок, що напрямлені до заданого числа 13](#_Toc105272738)

[3.6 Перевірка заповнення стрілок на правильність 13](#_Toc105272739)

[3.7 Генерація чисел 13](#_Toc105272740)

[4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 14](#_Toc105272741)

[4.1 Діаграма класів програмного забезпечення 14](#_Toc105272742)

[4.2 Опис методів частин програмного забезпечення 17](#_Toc105272743)

[4.2.1 Стандартні методи 17](#_Toc105272744)

[4.2.2 Користувацькі методи 18](#_Toc105272745)

[5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 23](#_Toc105272746)

[5.1 План тестування 23](#_Toc105272747)

[5.2 Приклади тестування 24](#_Toc105272748)

[6 ІНСТРУКЦІЯ КОРИСТУВАЧА 52](#_Toc105272749)

[6.1 Відкриття програми 52](#_Toc105272750)

[6.2 Складові ігрового поля 52](#_Toc105272751)

[6.3 Наповнення ігрового поля 53](#_Toc105272752)

[6.4 Завершення гри 56](#_Toc105272753)

[6.5 Завдання гравця 60](#_Toc105272754)

[6.6 Системні вимоги 60](#_Toc105272755)

[ВИСНОВКИ 62](#_Toc105272756)

[ПЕРЕЛІК ПОСИЛАНЬ 63](#_Toc105272757)

[ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ 64](#_Toc105272758)

[ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ 67](#_Toc105272759)

[arrows\_game.py 68](#_Toc105272760)

[collide\_rect\_evaluetor.py 74](#_Toc105272761)

[core.py 75](#_Toc105272762)

[colors.py 79](#_Toc105272763)

[grid\_position.py 79](#_Toc105272764)

[screen.py 80](#_Toc105272765)

[settings.py 81](#_Toc105272766)

[states.py 81](#_Toc105272767)

[time\_control.py 82](#_Toc105272768)

[add\_arrows\_button.py 82](#_Toc105272769)

[button.py 83](#_Toc105272770)

[delete\_arrow\_button.py 84](#_Toc105272771)

[end\_session\_button.py 85](#_Toc105272772)

[gen\_new\_board\_button.py 85](#_Toc105272773)

[arrow\_grid\_square.py 86](#_Toc105272774)

[grid\_square.py 87](#_Toc105272775)

[number\_grid\_square.py 88](#_Toc105272776)

[correct\_message.py 89](#_Toc105272777)

[message.py 90](#_Toc105272778)

[start\_message.py 90](#_Toc105272779)

[wrong\_message.py 91](#_Toc105272780)

[arrow.py 91](#_Toc105272781)

[board.py 92](#_Toc105272782)

[number.py 98](#_Toc105272783)

# **ВСТУП**

Дана робота присвячена написанню комп’ютерної гри «Стрілки» та вивченню розробки програмного забезпечення на основі ООП

Актуальність теми роботи: комп’ютерні ігри на разі мають велике значення не лише для підлітків та розваг дорослих, а й для більш масштабних розробок та проектів в різних сферах праці (архітектурний напрямок, промисловість) та загалом IT-індустрії. Дуже важливо, інноваційно та актуальне використання ООП в іграх.

Основне завдання роботи: розробка програмного забезпечення шляхом використання ООП на прикладі комп’ютерної гри «Стрілки».

ПОСТАНОВКА ЗАДАЧІ

Розробити комп'ютерну гру "Стрілки", яка складається з ігрового поля 8 на 8 клітинок, в кожній з яких розташоване число, яке вказує на те, скільки стрілок направлені в цю сторону. На підставі цих чисел гравець повинен розставити стрілки зверху, ліворуч, праворуч та знизу від поля.

Вхідними даними для даної роботи є ігрове поле 8 на 8 клітинок та числа, що розташовані в них випадковим чином.

Після того, як гравець розставить всі стрілки та підтвердить свої дії, програма повинна вивести на екран результат – чи правильно гравець розставив стрілки. Якщо так – вивести відповідне повідомлення та запропонувати повторити гру, або вийти з гри. Якщо ні – підкреслити, які цифри не збігаються з кількістю стрілок спрямованих до неї, та продовжити гру, поки гравець не змінить розташування стрілок та підтвердить, що хоче звершити гру.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Основні правила гри

Ігрове поле, розміром 8 х 8, складається з 64 квадратів. З усіх сторін (зверху, знизу, зліва та справа) є поля, в які необхідно вписати стрілки.

Стрілки можуть вказувати лише всередину квадрата. Наповнення квадрата – цифри, що вказують на кількість направлених на нього стрілок. Мінімальна цифра, яка може бути в квадраті, - 0 (якщо немає стрілки, що направлена в цей квадрат), максимальна – 8 (за умови, якщо всі можливі стрілки направлені в бік цього квадрату).

Інтерфейс гравця

1. Кнопка «Згенерувати нове поле»: якщо натиснути цю кнопку, буде створено нове поле із випадковим наповненням.
2. Кнопка «Видалити стрілку»: якщо її натиснути, виділена стрілка буде видалена, утвориться пуста клітинка, яку ще потрібно заповнити.
3. Кнопка «Змінити напрямок стрілки»: якщо натиснути цю кнопку, виділена стрілка змінить свій напрямок (об’єднання таких функцій як видалення старої стрілки та вписування нової).
4. Кнопка «Завершити розташування стрілок»: переводить стан гри у завершальний, перевіряє правильність розташувань стрілок та виводить відповідне повідомлення.

Задача гравця

Розставити стрілки навколо квадрату на підставі цифр всередині. Натиснути кнопку «Завершення розташування стрілки», таким чином відбувається перевірка правильно чи ні розташовані стрілки. У разі неправильності буде показано, де сталася помилка (цифри, що не відповідають розташуванню стрілок, будуть підсвічуватись). У разі правильності буде напис на екрані, що свідчить про перемогу гравця.

Діаграма прецендентів

Результати аналізу представлені у вигляді UML діаграми прецендентів *(Рисунок 2.1)*

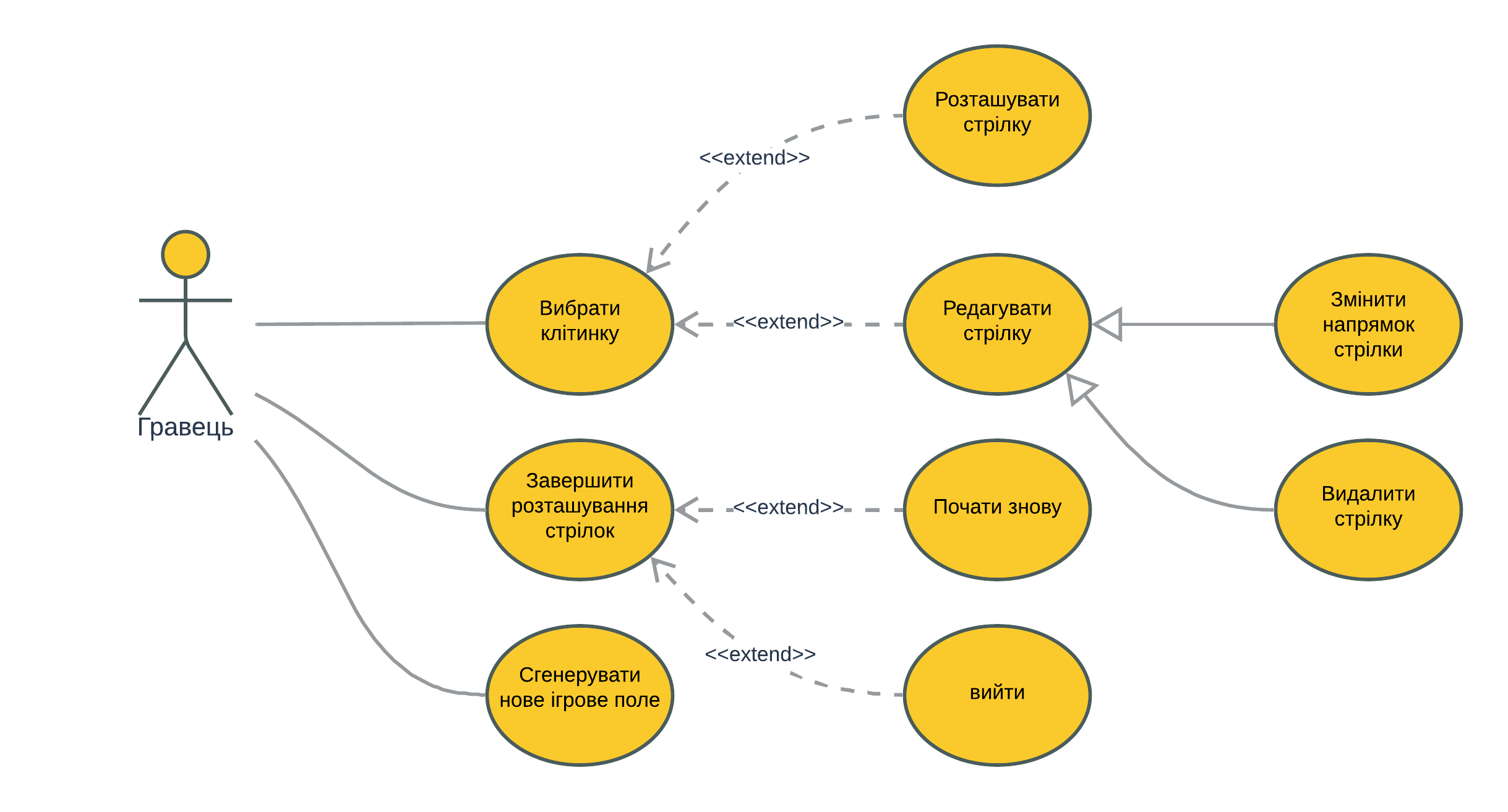


Рисунок 2.1 – Діаграма прецендентів

ОПИС АЛГОРИТМІВ

Опис усіх основних алгоритмів, що впливають на центральну логіку гри. Перелік всіх основних змінних та їхнє призначення наведено в *Таблиці 3.1*.

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
|  | Всі напрямки стрілок |
| *arrow\_sets* | Словник з ключами як напрямки розташування стрілок та можливих напрямків стрілки для цього розташування |
| *forbidden\_directions* | Неможливі напрямки стрілок у кутах сітки |
| *arrows* | Розташування та напрямки всіх стрілок |
| *numbers* | Розташування та значення числ сітки |
| *position* | Розташування відносно ігорового поля |
| *arrow* | Напрямок стрілки |
| *grid\_square* | Вектор розташування відносно ігрового поля |

Генерація стрілок

arrows = {}

for arrow\_set in arrow\_sets:

grid\_count = grid count along current set

for i from 0 to grid\_count:

possible\_directions = get\_possible\_directions(arrow\_set, i) (1.3)

choice = random among possible\_directions

add choice to arrows[arrow\_set]

Знаходження можливих напрямків стрілки

grid\_count = grid count along current set

possible\_directions = arrow\_sets[arrow\_set]

if arrow\_num == 0:

remove forbidden\_direction[arrow\_set][0] from possible\_directions

else if arrow\_num == grid\_count - 1:

remove forbidden\_direction[arrow\_set][1] from possible\_directions

Знаходження позиції стрілки

if arrows\_set\_direction == (0, -1):

position = (arrow\_num, -1)

else if arrows\_set\_direction == (1, 0):

position = grid\_count.x, arrow\_num

else if arrows\_set\_direction == (-1, 0):

position = -1, arrow\_num

else if arrows\_set\_direction == (0, 1):

position = arrow\_num, grid\_count.y

return position

Знаходження чисел до яких напрямлена стрілка

if arrow is empty:

return

grid\_squares = []

grid\_square = position

while True:

grid\_square += arrow

if grid\_square not inside gaming board:

break

add grid\_square to grid\_squares

return grid\_squares

Знаходження стрілок, що напрямлені до заданого числа

result = []

for arrow\_set\_direction and arrow\_set in arrows:

for arrow\_num and arrow in arrow\_set:

position = get\_position(arrow\_set\_direction, arrow\_num) (1.3)

if arrow not empty:

if grid\_square is in get\_span(position, arrow): (1.4)

arrow\_set\_direction and arrow\_num add to result

return result

**Перевірка заповнення стрілок на правильність**

wrong\_numbers = []

for number in numbers:

if number != count\_pointings(number.position) (1.5)

add number to wrong\_numbers

return wrong\_numbers

Генерація чисел

arrows = gen\_arrows() (1.1)

for col from 0 to grid\_count.x:

for row from 0 to grid\_count.y:

add length of count\_pointings(col, row) to numbers (1.5)

ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Діаграма класів програмного забезпечення

Короткий опис усіх класів, що були створені під час розробки ПЗ. Нижче розташована UML діаграма класів *(Рисунок 4.1).*

* **ArrowsGame**

Головний клас гри, який ініціалізує всі об'єкти гри та запускає цикл гри. Також відповідальний за обробки подій.

* **Board**

Клас ігрового поля, що містить у собі числа та стрілки. Керує подіями, що стосуються безпосередньо ігрового поля та об'єктів на ньому.

* **ArrowGridSquare**

Підклас GridSquare, що містить стрілки та пов'язані атрибути.

* **NumberGridSquare**

Підклас GridSquare, що містить числа та пов'язані атрибути.

* **GridSquare**

Абстрактний клас об'єктів клітинок ігрового поля, що містять числа або стрілки.

* **Number.**

Клас для рендерингу чисел, що використовуються, як атрибут зображення NumberGridSquare.

* **Arrow**

Клас для рендерингу стрілок, що використовуються, як атрибут зображення ArrowGridSquare.

* **GridPosition**

Контролючий клас, що керує розташуванням об'єктів на ігровому полі.

* **Button**

Базовий клас кнопок інтерфейсу гравця.

* **AddArrowButton**

Клас кнопки для додавання або зміни напрямку стрілки на ігровому полі.

* **DeleteArrowButton**

Клас кнопки для видалення стрілки на ігровому полі.

* **EndSessionButton**

Клас кнопки для завершення розташування стрілок та перевірки правильності їх розташування.

* **GenNewBoardButton**

Клас кнопки, що генерує нове ігрове поле та починає гру спочатку.

* **Settings**

Клас, що містить у собі усі ігрові константи.

* **Screen**

Містить ігрове вікно та пов'язані атрибути.

* **States**

Керуючий клас, що зберігає контролює стани гри.

* **Core**

Використовується для різних функцій та змінних основної логіки гри.

* **Message**

Базовий клас повідомлення.

* **CorrectMessage**

Повідомлення, що гравець бачить у кінці гри при правильному розташуванні стрілок.

* **StartMessage**

Повідомлення, що гравець бачить у перед початком гри.

* **WrongMessage**

Повідомлення, що гравець бачить у кінці гри при неправильному розташуванні стрілок.

* **CollideRectEvaluetor**

Клас для обчислення прямокутників зіткнення для повідомлень.

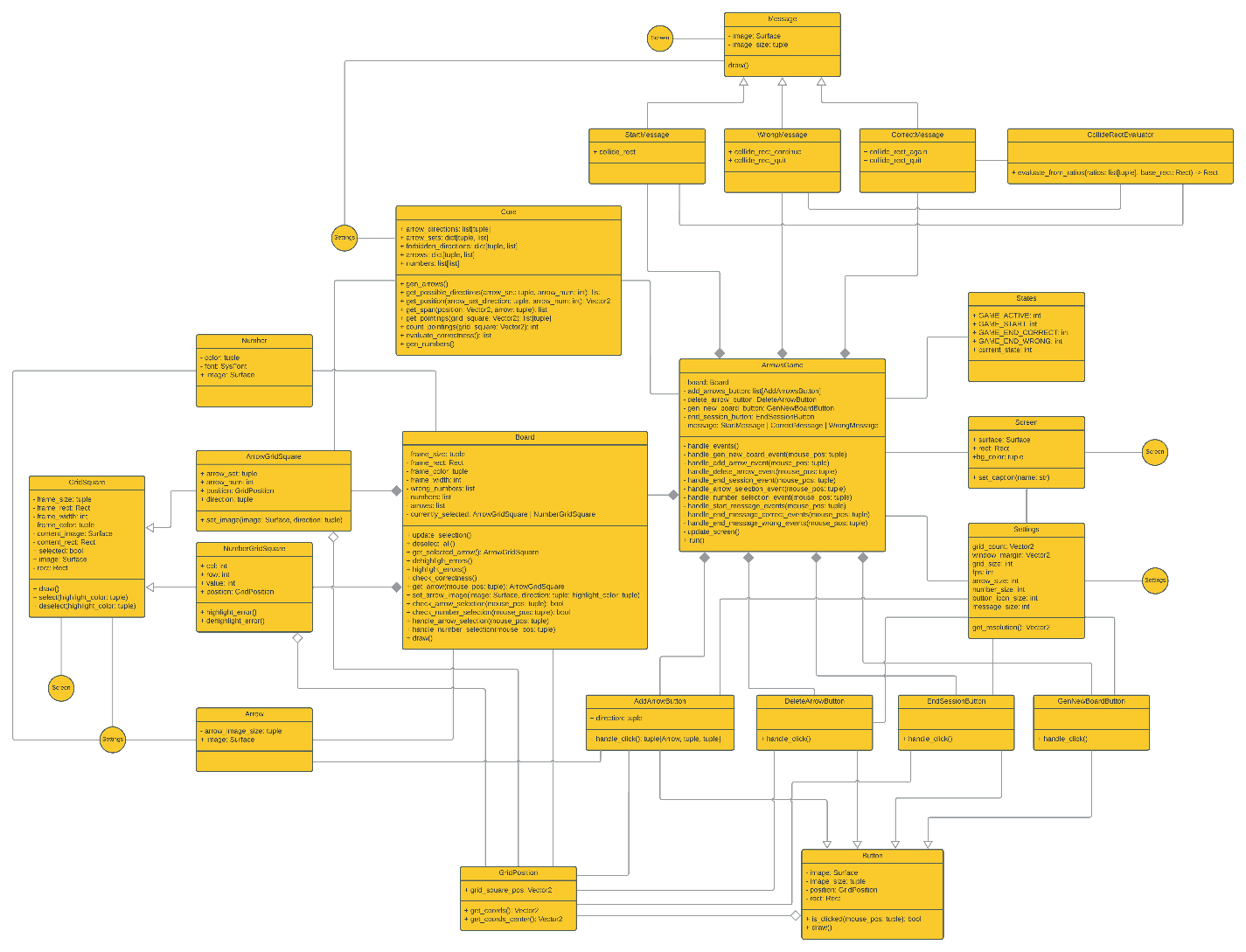


Рисунок 4.1 – Діаграма класів

Опис методів частин програмного забезпечення

### Стандартні методи

У *Таблиці 4.1* наведено повний опис стандартних методів, що використані у проекті.

Таблиця 4.1 – Стандартні методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. № п/п | 1. Назва классу | 1. Назва функції | 1. Призначення функції | 1. Опис вхідних параметрів | 1. Опис вихідних параметрів | 1. Заголовний файл |
| 1 | Vector2 | \_\_init\_ | Class Constructor | x - x coordinate  y - y coordinate | Vector2 object | pygame.math |
| 2 | Vector2 | length | get the length of the vector | - | length of the vector | pygame.math |
| 3 | Vector2 | copy | get the deep copy of the vector | - | copy of the vector | pygame.math |
| 4 | Rect | \_\_init\_\_ | Class Constractor | left - left edge coordinate  top - top edge coordinate  width - width of rectangle  height - height of rectagle | Rect object | pygame |
| 5 | Rect | collidepoint | test if a point is inside a rectangle | x - x coordinate  y - y coordinate | Returns true if the given point is inside the rectangle | pygame |
| 6 | Clock | \_\_init\_\_ | Class Constructor | - | Clock object | pygame.time |
| 7 | Clock | tick | update the clock | framerate - framerate at which the game should run | milliseconds from previous call | pygame.time |
| 8 | Surface | \_\_init\_\_ | Class Constructor | width - width of surface  height - height of surface | Surface object | pygame |
| 9 | Surface | blit | draw one image onto another | source - source surface to draw | - | pygame |
| 10 | Surface | fill | fills the surface with color | color - color to fill the surface with | - | pygame |
| 11 | Surface | get\_size | get dimetions of the surface | - | tuple with dimentions of the surface | pygame |
| 12 | Surface | get\_rect | get the rectangular area of the Surface | - | Rect object | pygame |
| 13 | Surface | convert\_alpha | change the pixel format of an image including per pixel alphas | - | Converted Surface | pygame |
| 14 | Font | \_\_init\_\_ | Class constructor | filename - name font file  size - size of font in points | Font object | pygame.font |

### Користувацькі методи

У *Таблиці 4.2* наведено повний опис методів, що були створені під час розробки ПЗ.

Таблиця 4.2 – Користувацькі методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний  файл |
| 1 | ArrowsGame | \_\_init\_\_ | Init game objects | - | - | arrows\_game |
| 2 | ArrowsGame | \_handle\_events | Handle pygame events queue | - | - | arrows\_game |
| 3 | ArrowsGame | \_handle\_gen\_  new\_board\_event | Generate new board if gen\_new\_board\_button is clicked, clear add and delete arrow buttons | mouse\_pos - Mouse position | - | arrows\_game |
| 4 | ArrowsGame | \_handle\_add\_  arrow\_event | Set arrow image and direction of select arrow grid square if add\_arrow\_button is clicked. Updates selection for  correct highlighting of numbers that the arrow points to. Creates delete arrow button. Creates and end session  button if every arrow grid square is filled and gets rid of error numbers highlighting. | mouse\_pos - Mouse position | - | arrows\_game |
| 5 | ArrowsGame | \_handle\_delete\_  arrow\_event | Sets selected image and direction to None. Updates selection for correct highlighting of numbers that the arrow  points to. Deletes delete arrow button and end session button, gets rid of error numbers highlighting. | mouse\_pos - Mouse position | - | arrows\_game |
| 6 | ArrowsGame | \_handle\_end\_  session\_event | Evaluates correctness of arrows and sets appropriate game state | mouse\_pos - Mouse position | - | arrows\_game |
| 7 | ArrowsGame | \_handle\_arrow\_  selection\_event | Highlights selected arrow and numbers that it points to. Adds button for adding arrows to selected square. | mouse\_pos - Mouse position | - | arrows\_game |
| 8 | ArrowsGame | \_handle\_number\_  selection\_event | Highlights selected number and arrows that point to it. Clears all arrow manipulation buttons | mouse\_pos - Mouse position | - | arrows\_game |
| 9 | ArrowsGame | \_handle\_start\_  message\_events | Sets game state to active and deletes message. | mouse\_pos - Mouse position | - | arrows\_game |
| 10 | ArrowsGame | \_handle\_end\_message  \_correct\_events | Handles end message button click when game is over and all numbers on grid match with number of arrows that  point to it. | mouse\_pos - Mouse position | - | arrows\_game |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11 | ArrowsGame | \_handle\_end\_message  \_wrong\_events | Handles end message button click when game is over and at least one number on grid don't with number of arrows that point to it. | mouse\_pos - Mouse position | - | arrows\_game |
| 12 | ArrowsGame | \_update\_screen | Render updated objects on screen and update screen | - | - | arrows\_game |
| 13 | ArrowsGame | run | Run main game loop | - | - | arrows\_game |
| 14 | CollideRect  Evaluetor | evaluate\_from\_ratios | Get collision rect for message buttons | ratios - Coordinates of collide rect box divided by image size  base\_rect - Rect of image that holds collide rect | Collision rect for message buttons | utils.collide\_rect  \_evaluator |
| 15 | Core | gen\_arrows | Generate arrows dict with keys as direction on game board and list of arrows as values | - | - | utils.core |
| 16 | Core | get\_possible\_directions | Get possible arrow directions for given arrow location | arrow\_set - Direction in which the arrow is located  arrow\_num - Sequence number of arrow on arrow set counting from up or left | List of possible directions that arrow can point to | utils.core |
| 17 | Core | get\_position | Get position relative to board of given arrow | arrows\_set\_direction - Direction in which the arrow is located  arrow\_num - Sequence number of arrow on arrow set counting from up or left | Position vector of arrow relative to game board | utils.core |
| 18 | Core | get\_span | Get all grid squares that given arrow points to | position - Position vector of arrow relative to game board  arrow - Direction in which arrow points to | List of grid square positions that given arrow points to | utils.core |
| 19 | Core | get\_pointings | Get all arrows that point to specified location on board | grid\_square - Position of grid square relative to board | List of arrow sets and arrow numbers that point to given grid square | utils.core |
| 20 | Core | count\_pointings | Count number of arrows that point to specified location on board | grid\_square - Position of grid square relative to board | Number of arrows that point to given grid square | utils.core |
| 21 | Core | evaluate\_correctness | Get all numbers that don't match with number of arrows that point to them | - | List of numbers, values of which don't match with number of arrows that point to them | utils.core |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 22 | Core | gen\_numbers | Generate numbers matrix based on previously generated arrows | - | - | utils.core |
| 23 | GridPosition | \_\_init\_\_ | Class constructor | grid\_square\_pos - Column and row of grid square relative to board | - | control.  grid\_position |
| 24 | GridPosition | get\_coords | Get exact pixel position on upper right corner of the object | - | Vector of pixel coordinates of topleft corner of grid square | control.  grid\_position |
| 25 | GridPosition | get\_coords\_center | Get center of specified grid square in pixels | - | Vector of pixel coordinates of the center of grid square | control.  grid\_position |
| 26 | Screen | set\_caption | Set caption for game window | name - Name for window caption | - | control.screen |
| 27 | Settings | get\_resolution | Get actual pixel resolution of entire screen | - | Actual pixel resolution of entire screen | control.settings |
| 28 | Board | \_\_init\_\_ | Class constructor | - | - | assets.board |
| 29 | Board | update\_selection | Deselect and select object for correct arrow adding and deletion | - | - | assets.board |
| 30 | Board | deselect\_all | Deselect any selection | - | - | assets.board |
| 31 | Board | get\_selected\_arrow | Return selected arrow object if any were selected | - | arrow grid  square  object | assets.board |
| 32 | Board | dehighlight\_errors | Get rid of highlighting on numbers that don't match | - | - | assets.board |
| 33 | Board | highlight\_errors | Highlight numbers that don't match | - | - | assets.board |
| 34 | Board | check\_correctness | Load current arrows and numbers to core class and evaluate correctness | - | - | assets.board |
| 35 | Board | get\_arrow | Get arrow by pixel position | pos - position to get arrow by | arrow grid  square  object | assets.board |
| 36 | Board | set\_arrow\_image | Set arrow image for selected arrow square if any | image - New image to set  direction - direcion of arrow on image to set direction attribute  highlight\_color - color that object was highlighted to restore it | - | assets.board |
| 37 | Board | check\_arrow\_selection | Check if arrow was selected | mouse\_pos - Mouse position in pixel coordinates | True if arrow under current mouse position is selected | assets.board |
| 38 | Board | check\_number\_selection | Check if number was selected | mouse\_pos - Mouse position in pixel coordinates | True if number under current mouse position is selected | assets.board |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 | Board | handle\_arrow\_selection | Select arrow and numbers it points to | mouse\_pos - Mouse position in pixel coordinates | - | assets.board |
| 40 | Board | handle\_number\_selection | Select number and arrows that point to it | mouse\_pos - Mouse position in pixel coordinates | - | assets.board |
| 41 | Board | draw | Draw object to given surface | - | - | assets.board |
| 42 | Arrow | \_\_init\_\_ | Class constructor | direction - Direction of arrow to render appropriate image | - | assets.arrow |
| 43 | Number | \_\_init\_\_ | Class constructor | value - Numeric value of number object | - | assets.number |
| 44 | Button | \_\_init\_\_ | Class constructor | image\_path - Filepath to button image  position - Grid square position relative to board  color - Color to fill the button image with  conform\_size - Scale image according to button icon size value in settings. If False - scaling\_nonconformal parameter is required  scaling\_nonconformal - Value to scale image with | - | assets.buttons.  button |
| 45 | Button | is\_clicked | Check if button is clicked | - | - | assets.buttons.  button |
| 46 | Button | draw | Draw object to given surface | - | - | assets.buttons.  button |
| 47 | Button | handle\_click | Abstract method for action that button makes | - | - | assets.buttons.  button |
| 48 | AddArrow  Button | \_\_init\_\_ | Class constructor | direction - Direction of arrow that it adds when button is clicked | - | assets.buttons.  add\_arrow\_  button |
| 49 | AddArrow  Button | handle\_click | Return new arrow image of given direction to set as image attribute of selected square | - | Needed attributes for changing the image of arrow grid square: new arrow image, its direction and color that it was highlighted with | assets.buttons.  add\_arrow\_  button |
| 50 | DeleteArrow  Button | \_\_init\_\_ | Class constructor | - | - | assets.buttons.  delete\_arrow\_  button |
| 51 | EndSession  Button | \_\_init\_\_ | Class constructor | - | - | assets.buttons.  end\_session  button |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 52 | GenNew  BoardButton | \_\_init\_\_ | Class constructor | - | - | assets.buttons.  gen\_new\_board  \_button |
| 53 | GridSquare | \_\_init\_\_ | Class constructor | content - Image to fill the square with | - | assets.  grid\_squares.  grid\_square |
| 54 | GridSquare | draw | Draw object to given surface | - | - | assets.  grid\_squares.  grid\_square |
| 55 | GridSquare | select | Add given color to image | highlight\_color - Color to restore highlighting with | - | assets.  grid\_squares.  grid\_square |
| 56 | GridSquare | deselect | Subtract given color from image | highlight\_color - Color to restore highlighting with | - | assets.  grid\_squares.  grid\_square |
| 57 | ArrowGrid  Square | \_\_init\_\_ | Class constructor | arrow\_set - Direction in which the arrow is located  arrow\_num - Sequence number of arrow on arrow set counting from up or left | - | assets.  grid\_squares.  arrow\_grid  \_square |
| 58 | ArrowGrid  Square | set\_image | Set arrow image for arrow grid square | image - Arrow image to fill the arrow grid square  direction - Direction that arrow points to | - | assets.  grid\_squares.  arrow\_grid  \_square |
| 59 | NumberGrid  Square | \_\_init\_\_ | Class constructor | col - Column of number position relative to board  row - Row of number position relative to board  value - Numeric value of number object | - | assets.  grid\_squares.  number\_grid  \_square |
| 60 | NumberGrid  Square | highlight\_error | Add red highlight color to image | - | - | assets.  grid\_squares.  number\_grid  \_square |
| 61 | NumberGrid  Square | dehighlight\_error | Subtract red highlight color from image | - | - | assets.  grid\_squares.  number\_grid  \_square |
| 62 | Message | \_\_init\_\_ | Class constructor | image\_path - Path to message image path | - | assets.messages.  message |
| 63 | Message | draw | Draw object to given surface | - | - | assets.messages.  message |
| 64 | StartMessage | \_\_init\_\_ | Class constructor | - | - | assets.messages.  start\_message |
| 65 | Correct  Message | \_\_init\_\_ | Class constructor | - | - | assets.messages.  correct\_message |
| 66 | Wrong  Message | \_\_init\_\_ | Class constructor | - | - | assets.messages.  wrong\_message |

ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

План тестування

Під час тестування будемо використовувати різні функціональні можливості програми та порівнювати графічний результат з очікуваним.

1. Тестування початкового екрану
   1. Натискання кнопки «старт»
2. Тестування виділення чисел та стрілок
   1. Виділення стрілок з пустою клітинкою
   2. Виділення стрілок з заповненою клітинкою
   3. Виділення чисел
3. Тестування розставлення стрілок
   1. Розставлення стрілки в пусту клітинку
   2. Зміна напрямку стрілки
   3. Видалення стрілки
4. Тестування кнопки перевірки результатів та генерації нового поля
   1. Кнопка генерації нового поля з частково заповненим полем
   2. Кнопка перевірки результатів з неправильним розташуванням стрілок
   3. Кнопка перевірки результатів з правильним розташуванням стрілок
5. Тестування повідомлень у кінці грі
   1. Повернення до гри
   2. Вихід з гри

Приклади тестування

Проведемо випробування та задокументуємо результати у таблицях та рисунках.

Таблиця 5.1 ‑ Приклад роботи програми на початку гри

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість почати гру |
| Початковий стан програми | Відкрите вікно з початковим повідомленням |
| Вхідні дані | - |
| Схема проведення тесту | Натискання кнопки «старт» |
| Очікуваний результат | Початок гри |
| Стан програми після проведення випробувань | Вікно програми з пустим ігровим полем |

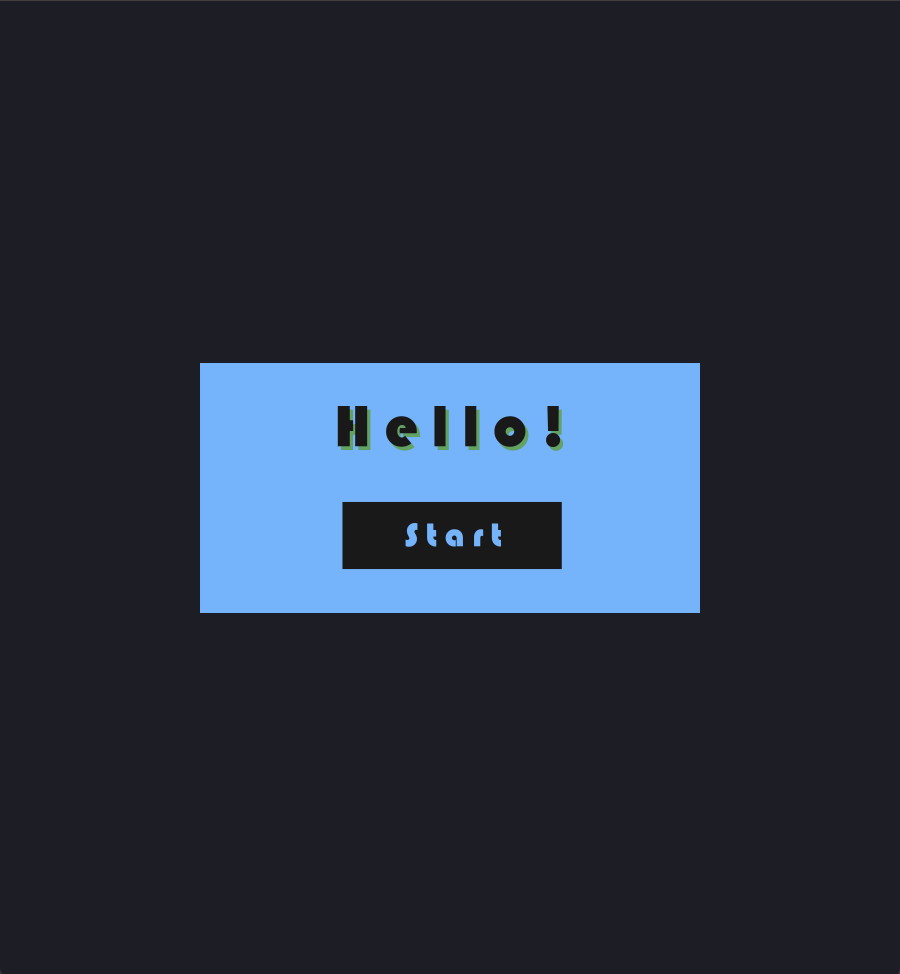


Рисунок 5.1 - до проведення тесту «Натискання кнопки «старт»»

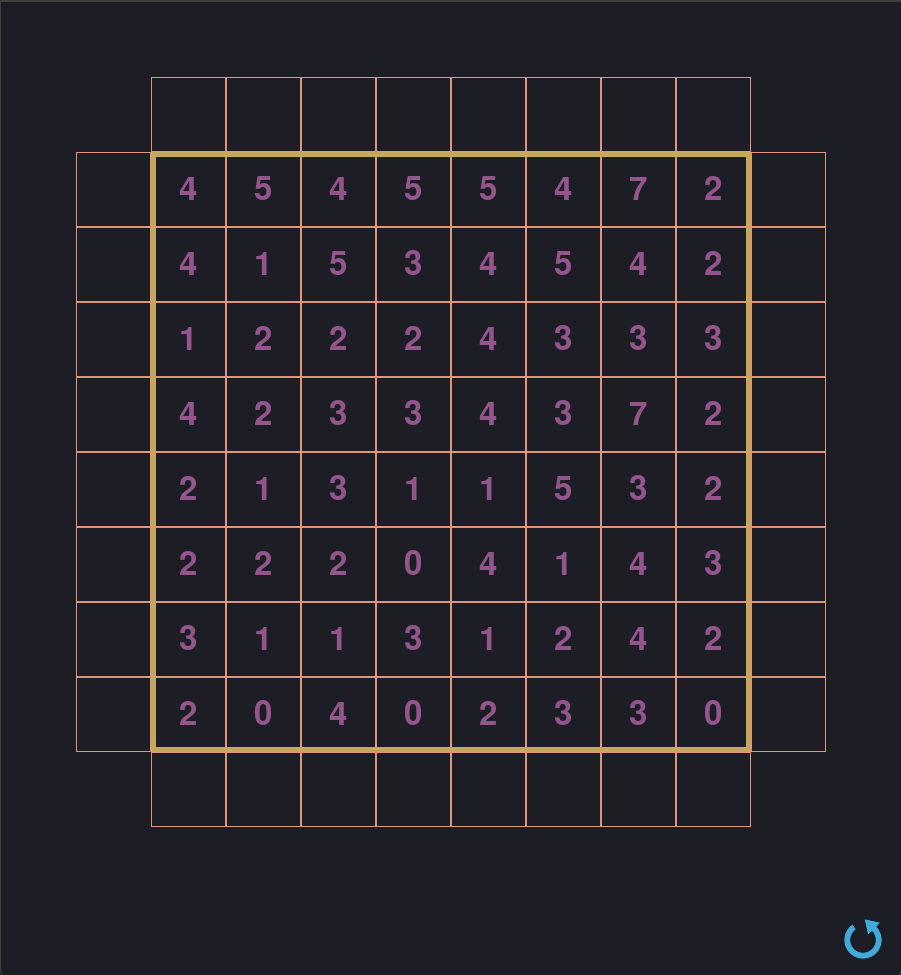


Рисунок 5.2 - після проведення тесту «Натискання кнопки «старт»»

Таблиця 5.2 ‑ Приклад роботи програми під час виділення стрілок з пустою клітинкою

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість виділення стрілки |
| Початковий стан програми | Відкрите вікно з програми з пустим ігровим полем |
| Вхідні дані | - |
| Схема проведення тесту | ЛКМ на один з квадратів, де розташовуються стрілки |
| Очікуваний результат | Виділена клітинка з додатковими кнопками розташування стрілки з поміж можливих напрямків |
| Стан програми після проведення випробувань | Виділена клітинка з кнопками розташування стрілок з правильним набором напрямків |

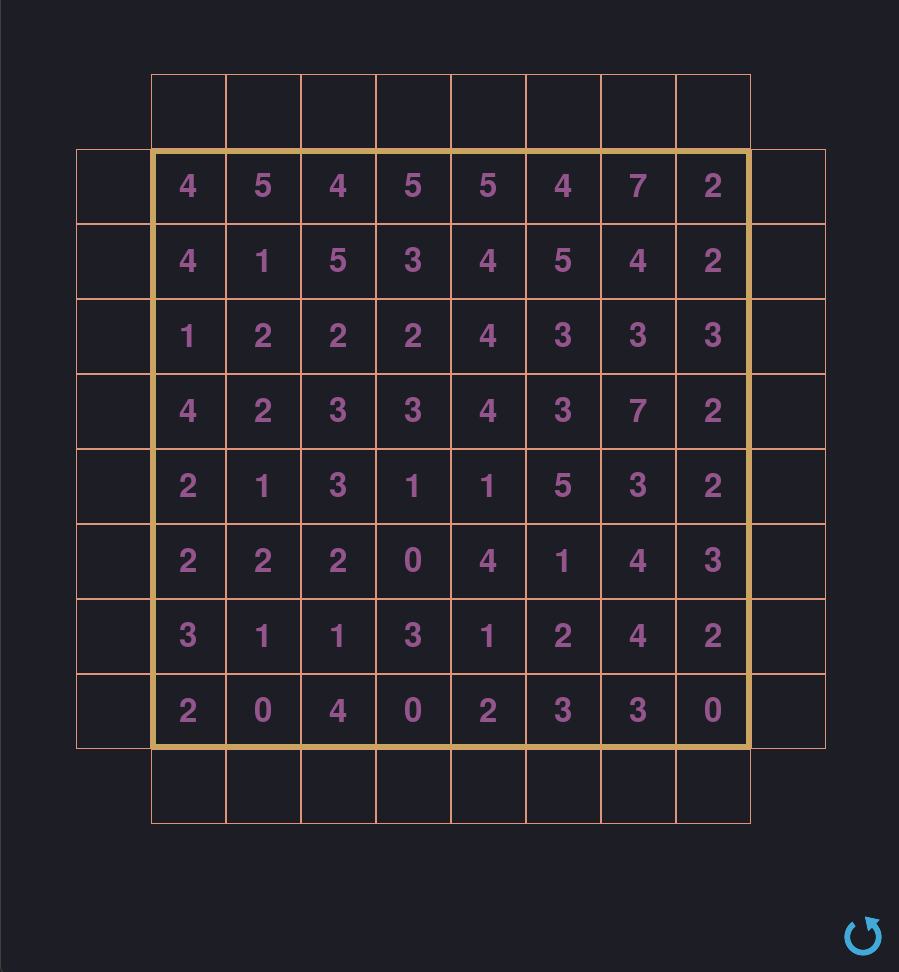


Рисунок 5.3 - до проведення тесту «Виділення стрілок з пустою клітинкою»

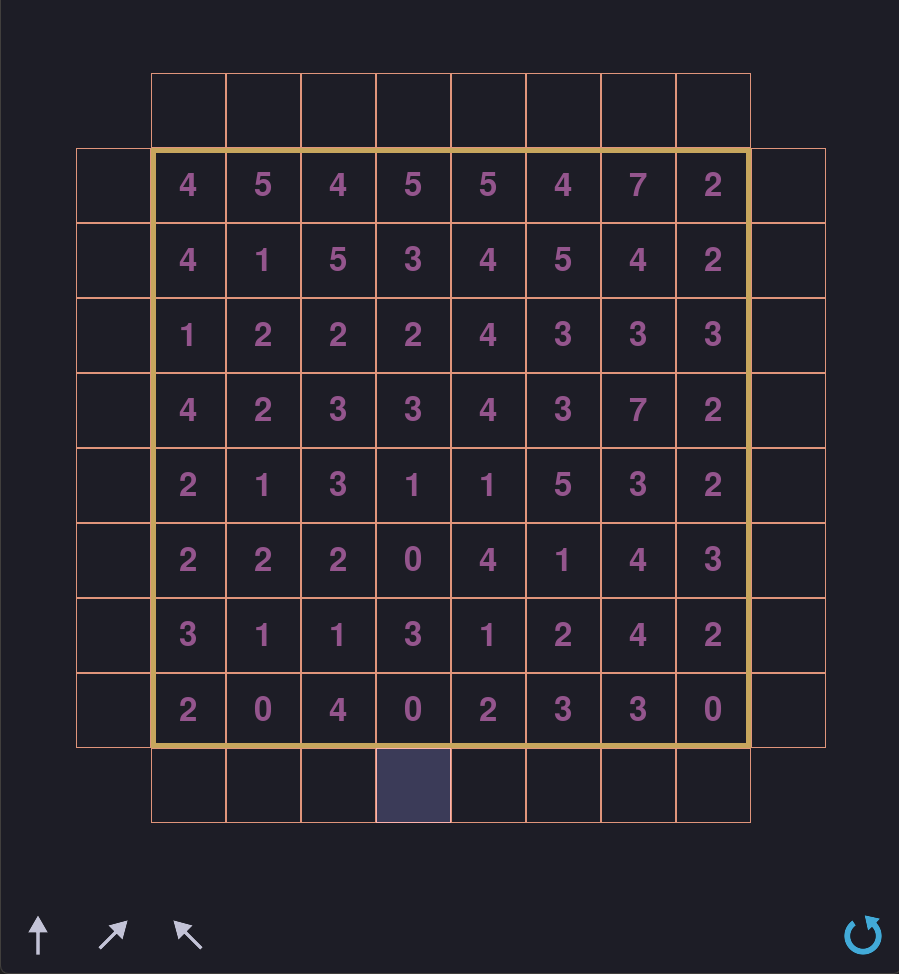


Рисунок 5.4 - після проведення тесту «Виділення стрілок з пустою клітинкою» (не кутове положення стрілки)

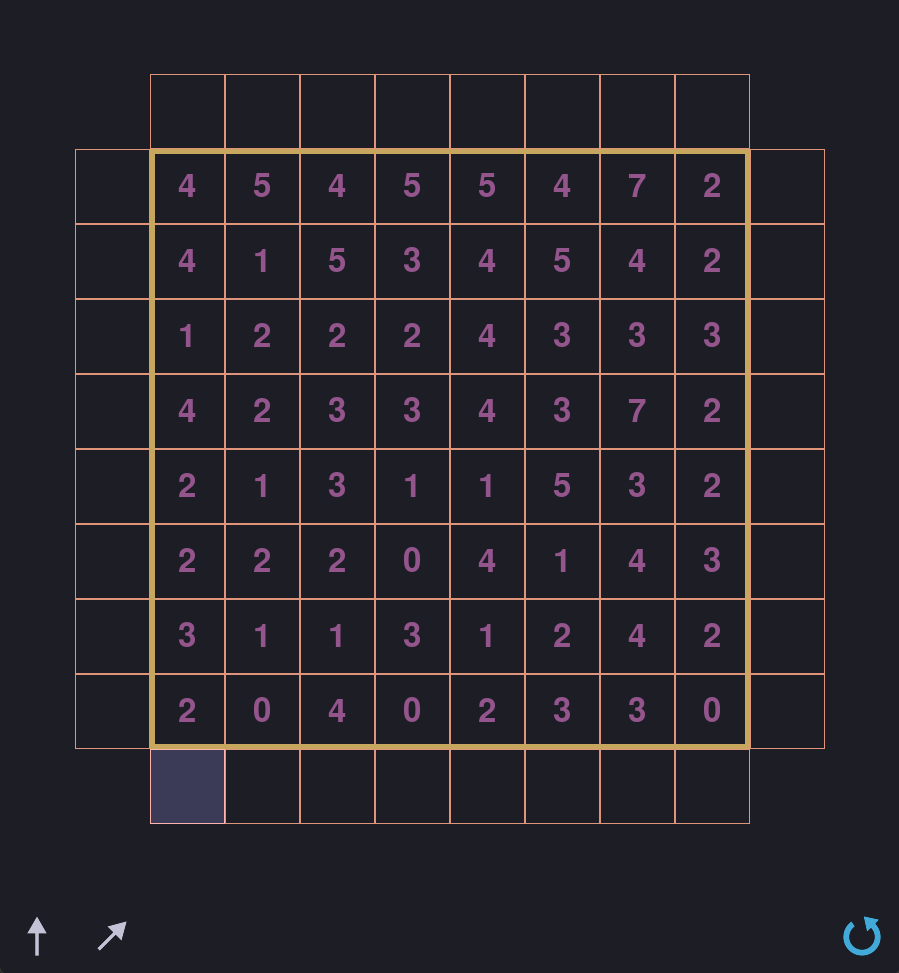


Рисунок 5.5 - після проведення тесту «Виділення стрілок з пустою клітинкою» (кутове положення стрілки)

Таблиця 5.3 ‑ Приклад роботи програми під час виділення стрілок з заповненою клітинкою

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість виділення стрілки |
| Початковий стан програми | Відкрите вікно з програми з часткового заповненим ігровим полем |
| Вхідні дані | - |
| Схема проведення тесту | ЛКМ на один з квадратів, де розташована стрілка |
| Очікуваний результат | Виділена клітинка з додатковими кнопками розташування стрілки з поміж можливих напрямків, кнопка видалення стрілки виділення відповідних клітинок з числами, до яких напрямлена стрілка |
| Стан програми після проведення випробувань | Виділена клітинка з кнопками розташування стрілок з правильним набором напрямків та кнопка видалення стрілки, виділення відповідних клітинок з числами, до яких напрямлена стрілка |

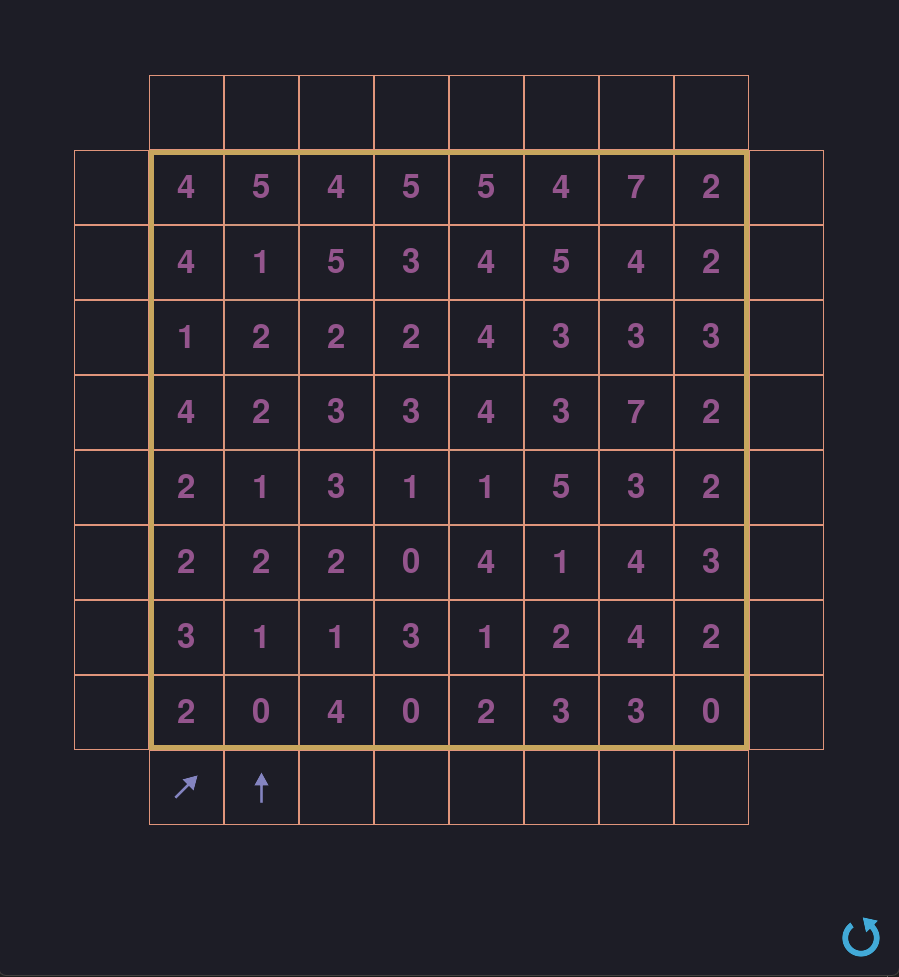


Рисунок 5.6 - до проведення тесту «Виділення стрілок з заповненою клітинкою»

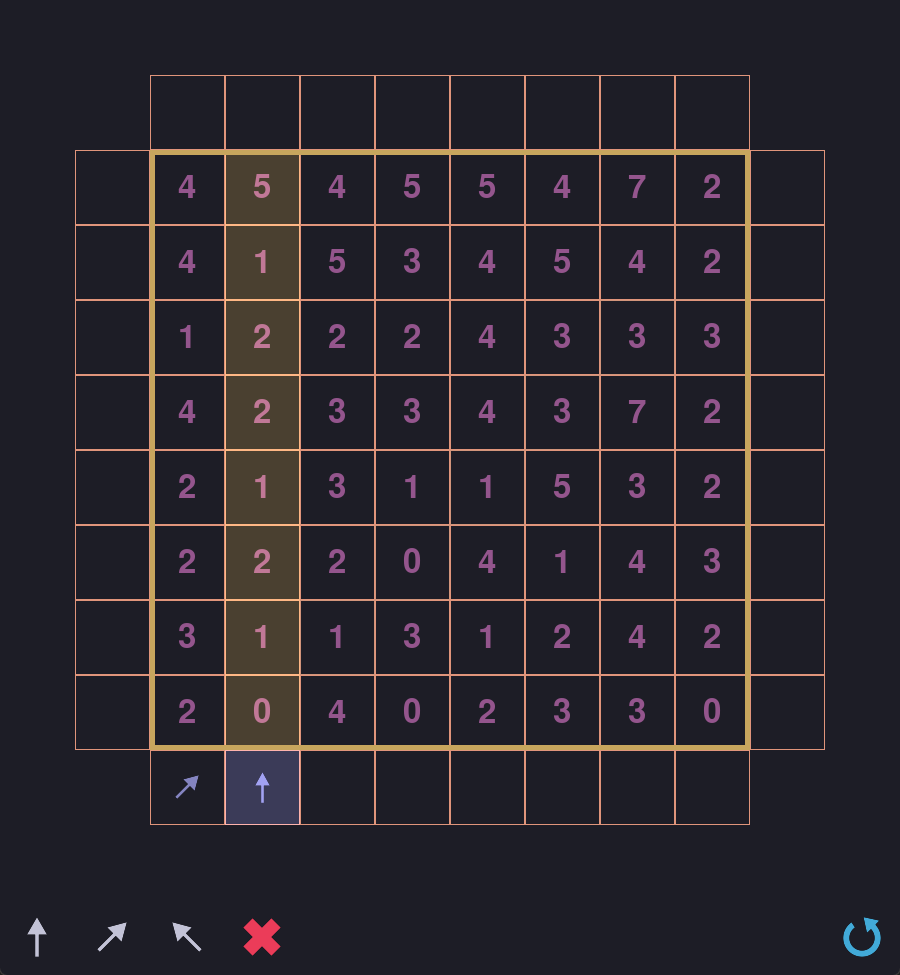


Рисунок 5.7 - після проведення тесту «Виділення стрілок з заповненою клітинкою» (виділення не діагональної стрілки)

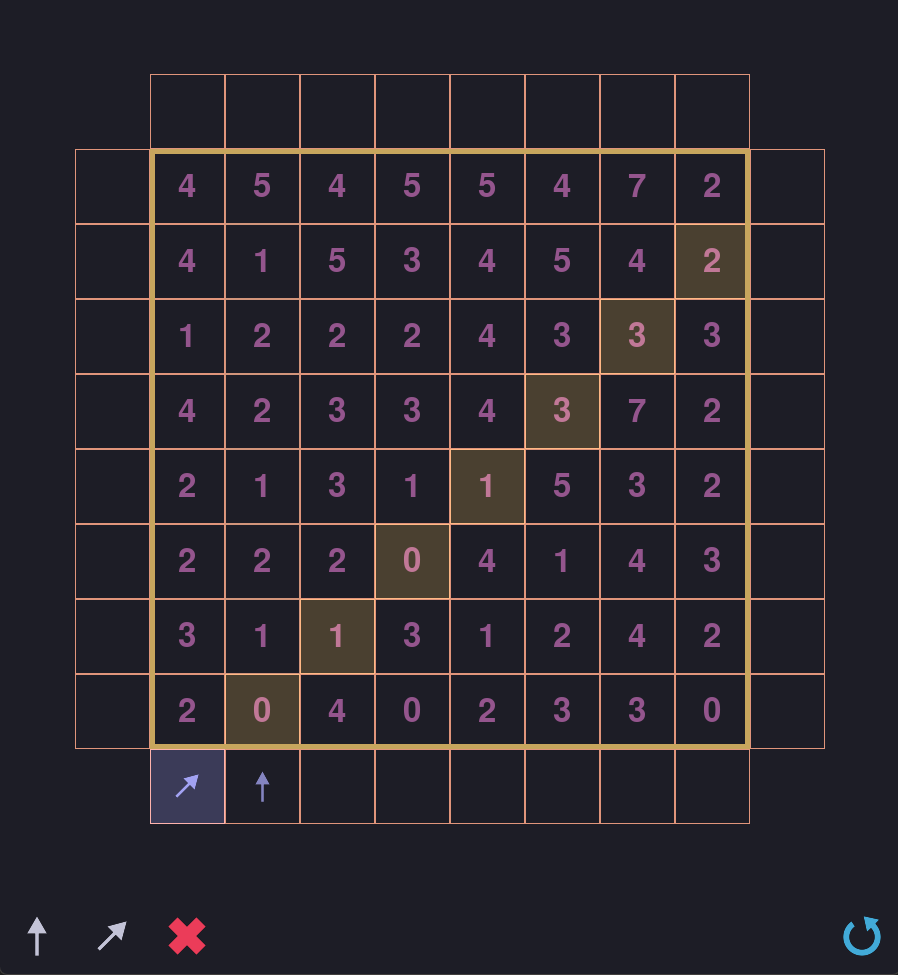


Рисунок 5.8 - після проведення тесту «Виділення стрілок з заповненою клітинкою» (виділення діагональної стрілки)

Таблиця 5.4 ‑ Приклад роботи програми під час виділення клітинки з числом

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість виділення чисел |
| Початковий стан програми | Відкрите вікно з програми з часткового заповненим ігровим полем |
| Вхідні дані | - |
| Схема проведення тесту | ЛКМ на один з квадратів, де розташоване число |
| Очікуваний результат | Виділена клітинка числа з додатковим виділенням стрілок, що напрямлені до цього числа |
| Стан програми після проведення випробувань | Виділена клітинка числа з додатковим виділенням стрілок, що напрямлені до цього числа |

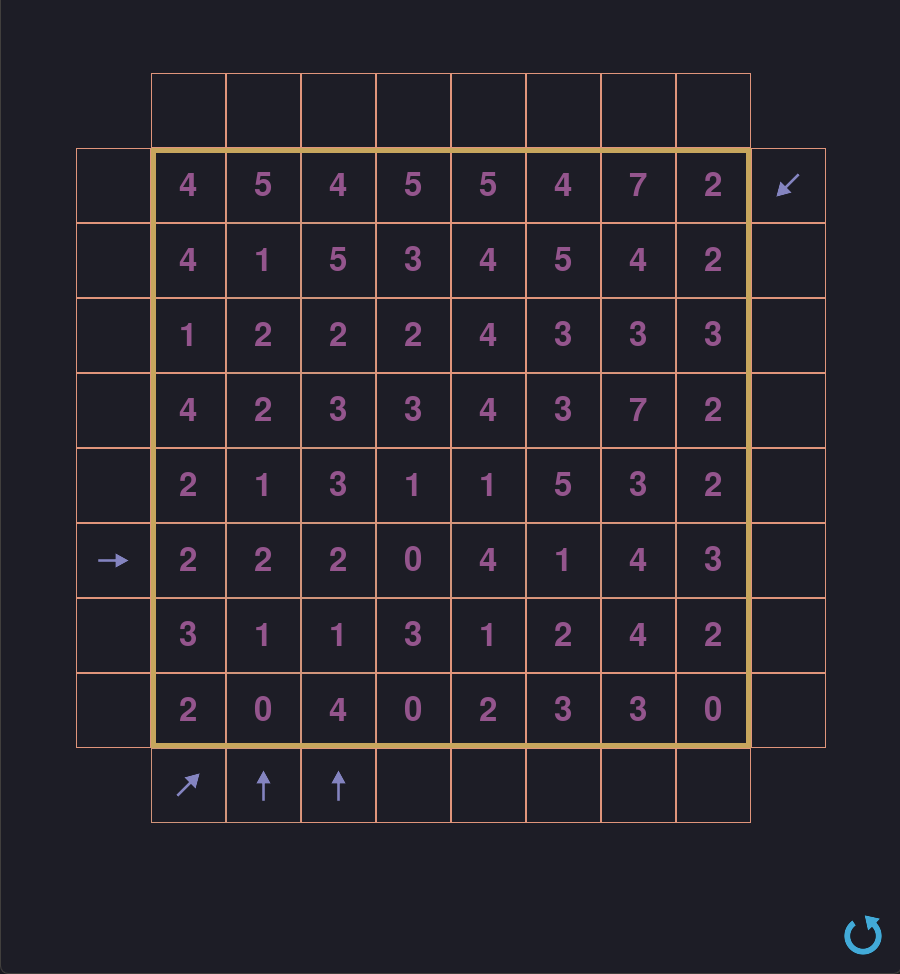


Рисунок 5.9 - до проведення тесту «Виділення чисел»

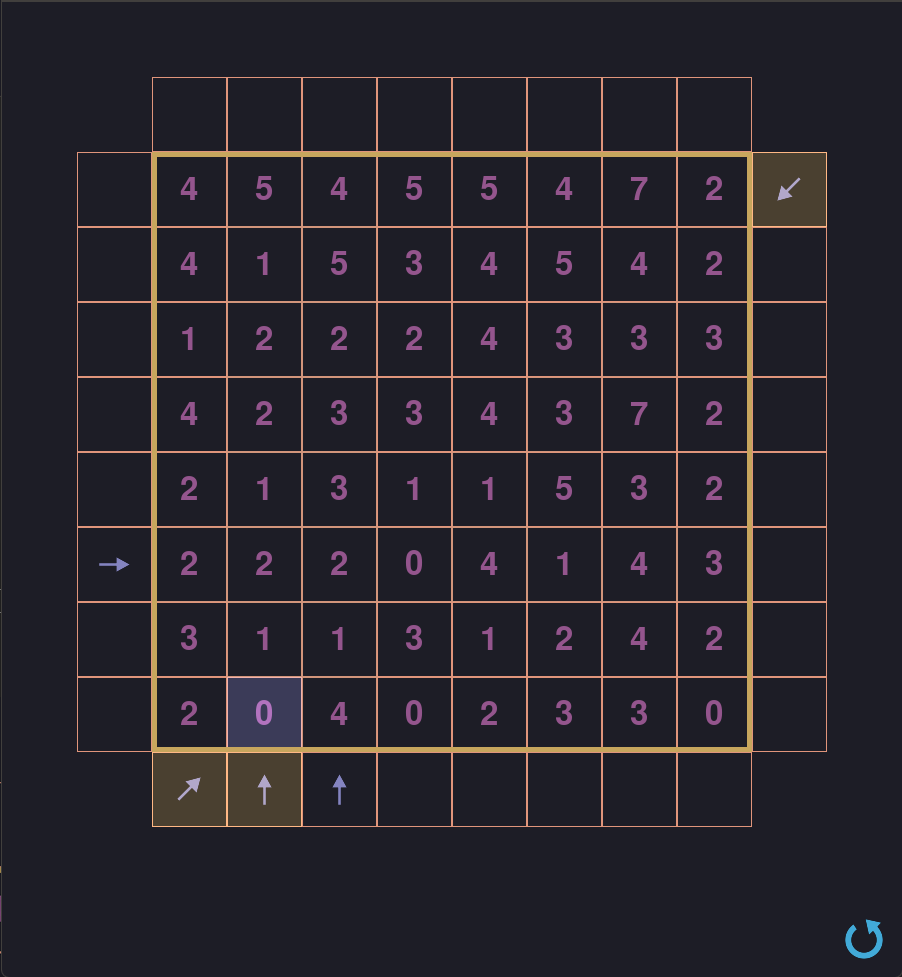


Рисунок 5.10 - після проведення тесту «Виділення чисел»

Таблиця 5.5 ‑ Приклад роботи програми під час розташування стрілки до пустої клітинки

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість розташування стрілок |
| Початковий стан програми | Відкрите вікно з програми з часткового заповненим або пустим ігровим полем |
| Вхідні дані | Виділена пуста клітинка, де розташувуються стрілки |
| Схема проведення тесту | ЛКМ на одну з іконок стрілки у нижньому кутку вікна програми |
| Очікуваний результат | Виділена заповнена клітинка стрілки з додатковою кнопкою видалення стрілки та виділенням відповідних клітинок з числами |
| Стан програми після проведення випробувань | Виділена заповнена клітинка стрілки з додатковою кнопкою видалення стрілки та виділенням відповідних клітинок з числами |

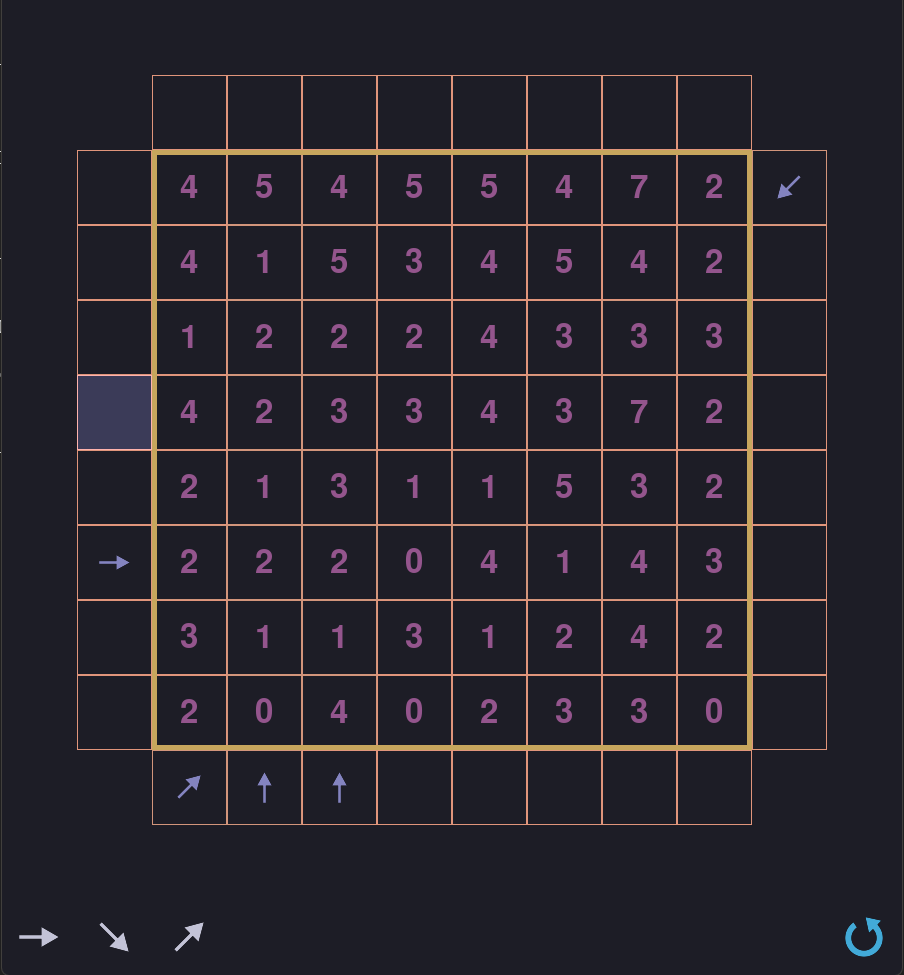


Рисунок 5.11 - до проведення тесту «Розставлення стрілки до пустої клітинки»

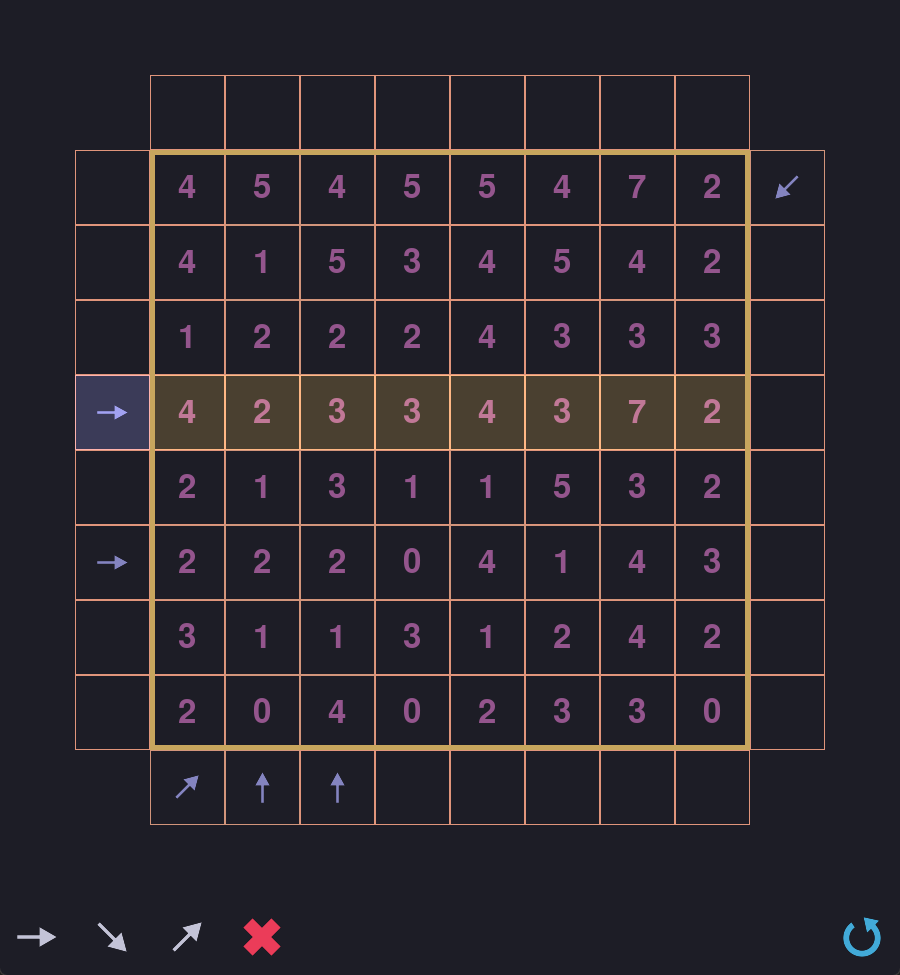


Рисунок 5.12 - після проведення тесту «Розставлення стрілки до пустої клітинки»

Таблиця 5.6 ‑ Приклад роботи програми під час розташування стрілки до заповненої клітинки

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість розташування стрілок |
| Початковий стан програми | Відкрите вікно з програми з часткового заповненим або пустим ігровим полем |
| Вхідні дані | Виділена заповнена клітинка, де розташувуються стрілки |
| Схема проведення тесту | ЛКМ на одну з іконок стрілки у нижньому кутку вікна програми, що відрізняється за напрямком |
| Очікуваний результат | Виділена заповнена клітинка стрілки, з зміненим напрямком, з додатковою кнопкою видалення стрілки та виділенням відповідних клітинок з числами |
| Стан програми після проведення випробувань | Виділена заповнена клітинка стрілки, з зміненим напрямком, з додатковою кнопкою видалення стрілки та виділенням відповідних клітинок з числами |

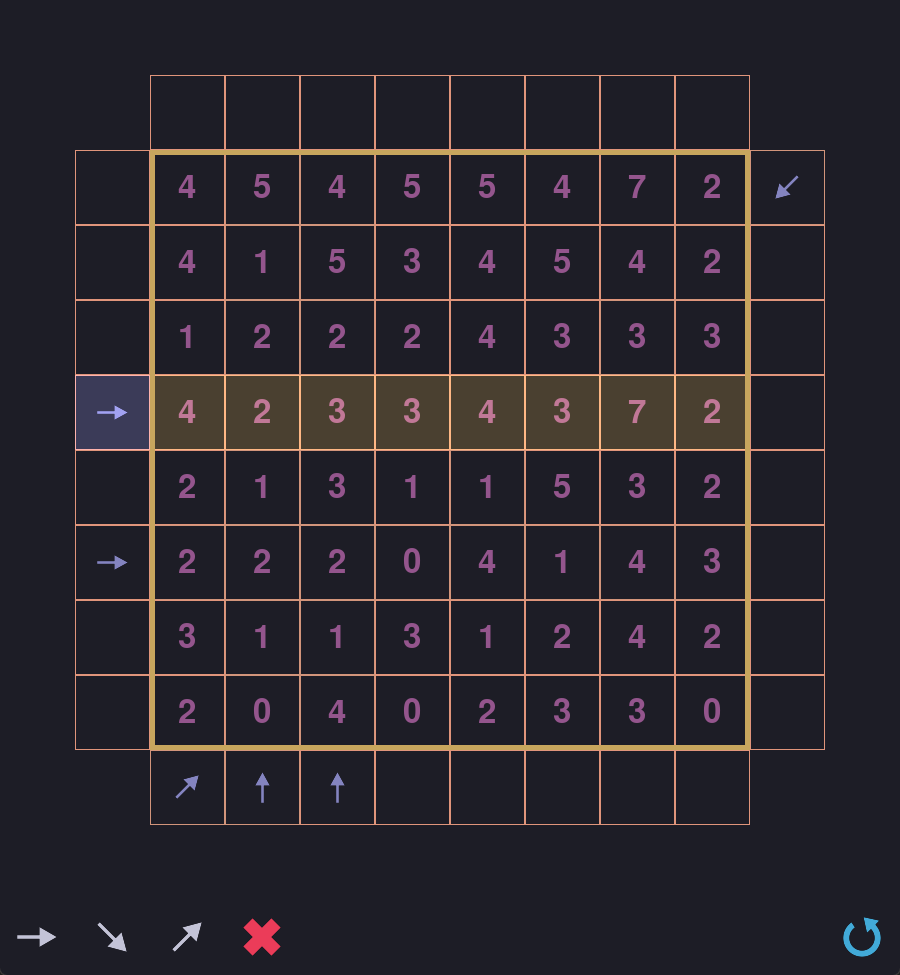


Рисунок 5.13 - до проведення тесту «Зміна напрямку стрілки»

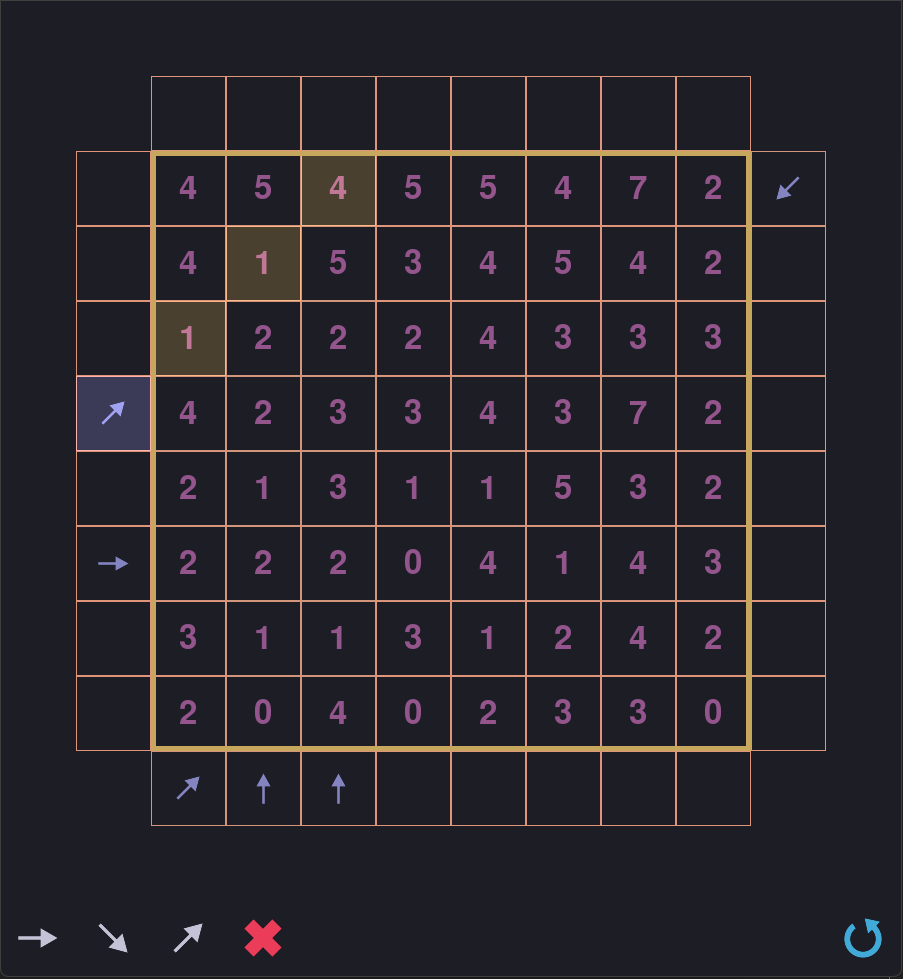


Рисунок 5.14 - після проведення тесту «Зміна напрямку стрілки»

Таблиця 5.7 ‑ Приклад роботи програми під час видалення стрілки

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість видаляти стрілки |
| Початковий стан програми | Відкрите вікно з програми з часткового заповненим полем |
| Вхідні дані | Виділена заповнена клітинка, де розташувуються стрілки |
| Схема проведення тесту | ЛКМ на одну з іконок у вигляді «х» у нижньому кутку вікна програми |
| Очікуваний результат | Виділена пуста клітинка стрілки, без додаткових кнопок у нижньому кутку |
| Стан програми після проведення випробувань | Виділена пуста клітинка стрілки, без додаткових кнопок у нижньому кутку |

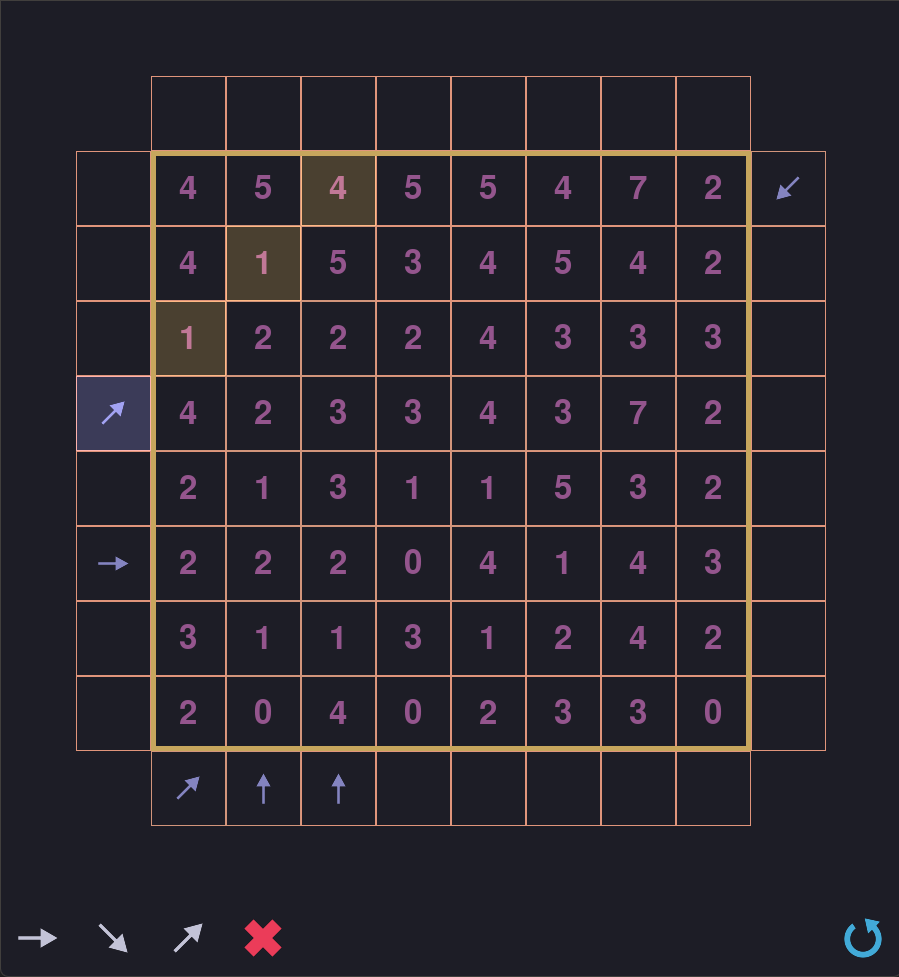


Рисунок 5.15 - до проведення тесту «Видалення стрілки»

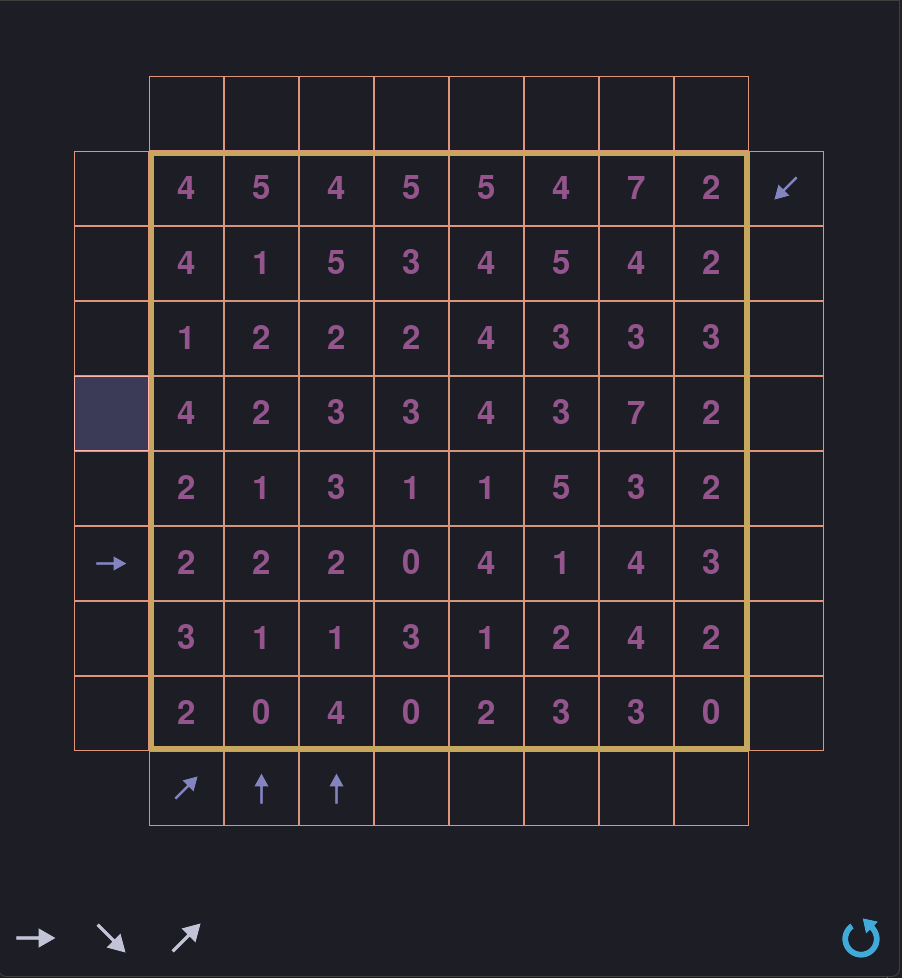


Рисунок 5.16 - після проведення тесту «Видалення стрілки»

Таблиця 5.8 ‑ Приклад роботи програми під час генерації нового поля

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість генерувати нове ігрове поле |
| Початковий стан програми | Відкрите вікно з програми з часткового заповненим полем |
| Вхідні дані | - |
| Схема проведення тесту | ЛКМ на кнопку генерації нового поля у нижньому кутку вікна програми |
| Очікуваний результат | Пусте ігрове поле |
| Стан програми після проведення випробувань | Пусте ігрове поле без виділення та жодного додаткового функціоналу |



Рисунок 5.17 - до проведення тесту «Кнопка генерації нового поля з частково заповненим полем»

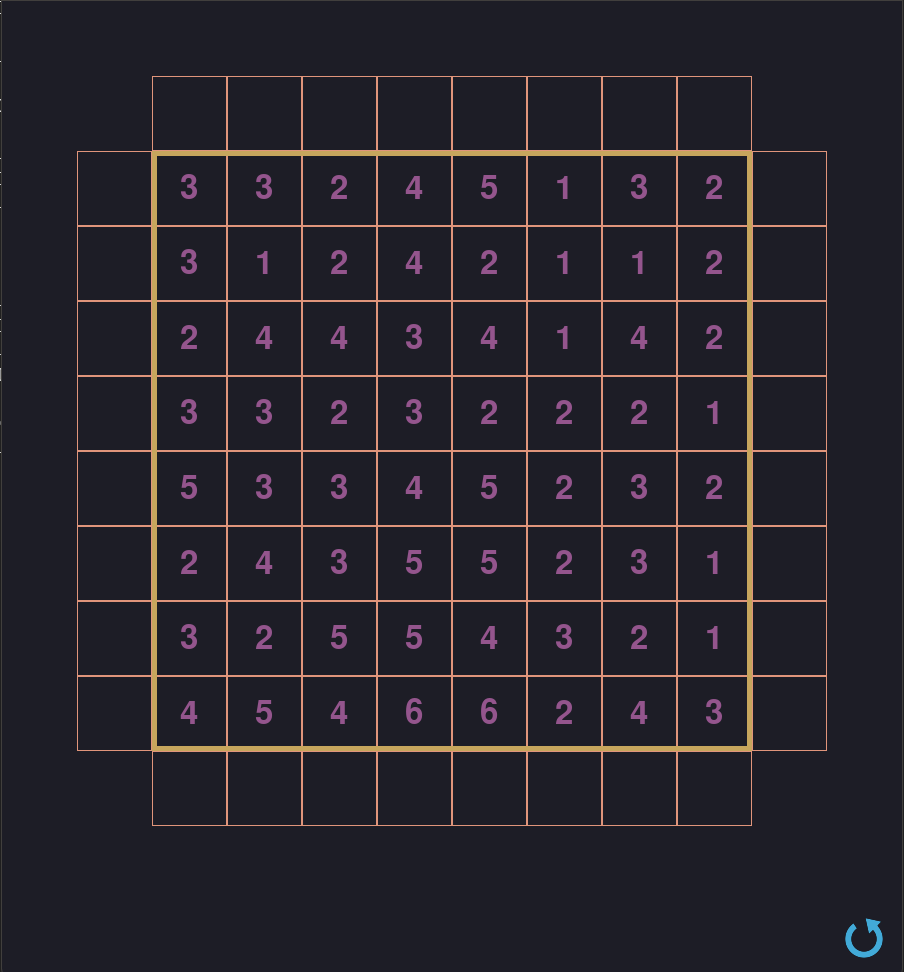


Рисунок 5.18 - після проведення тесту «Кнопка генерації нового поля з частково заповненим полем»

Таблиця 5.9 ‑ Приклад роботи програми під час перевірки неправильних результатів гри

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність перевірки результатів |
| Початковий стан програми | Відкрите вікно з програми з заповненим полем |
| Вхідні дані | Хоча б одне число у клітинці не співпадаю чисельно з кількістю стрілок, що напрямлені до цієї клітинки |
| Схема проведення тесту | ЛКМ на кнопку перевірки результатів |
| Очікуваний результат | Повідомлення про неправильний результат гри |
| Стан програми після проведення випробувань | Повідомлення про неправильний результат гри |



Рисунок 5.19 - до проведення тесту «Кнопка перевірки результатів з неправильним розташуванням стрілок»



Рисунок 5.20 - після проведення тесту «Кнопка перевірки результатів з неправильним розташуванням стрілок»

Таблиця 5.10 ‑ Приклад роботи програми під час перевірки правильних результатів гри

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість коректно перевіряти результати гри |
| Початковий стан програми | Відкрите вікно з програми з повністю заповненим полем |
| Вхідні дані | Всі клітинки з числами співпадають чисельно з кількістю стрілок, що напрямлені до цих клітинок |
| Схема проведення тесту | ЛКМ на кнопку перевірки результатів |
| Очікуваний результат | Повідомлення про правильний результат гри |
| Стан програми після проведення випробувань | Повідомлення про правильний результат гри |

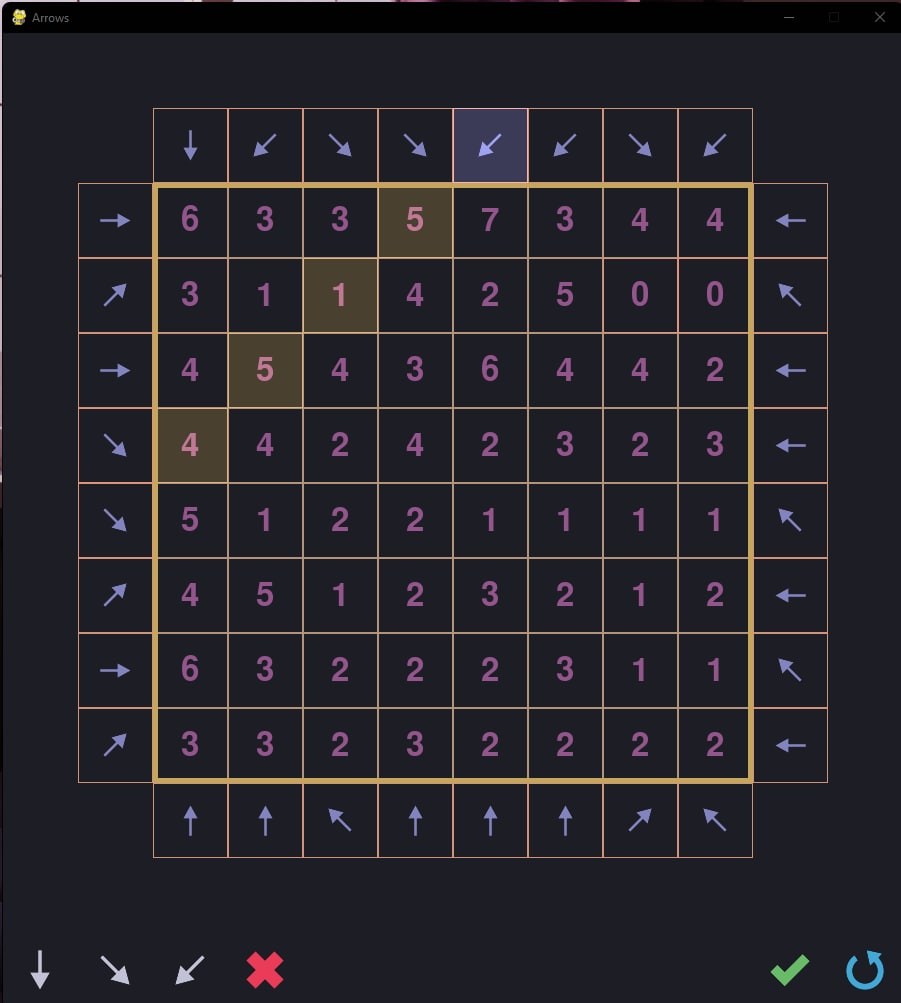


Рисунок 5.21 - до проведення тесту «Кнопка перевірки результатів з правильним розташуванням стрілок»

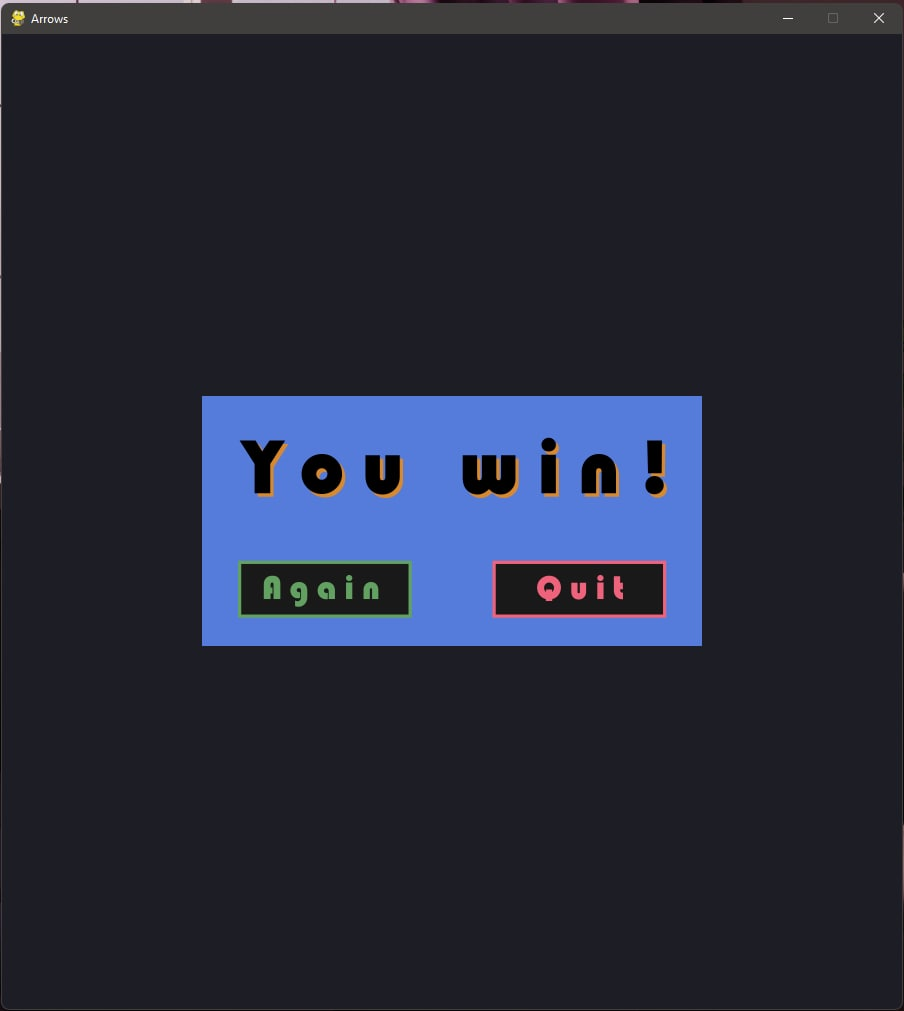


Рисунок 5.22 - після проведення тесту «Кнопка перевірки результатів з правильним розташуванням стрілок»

Таблиця 5.11 ‑ Приклад роботи програми у кінці гри, при натисканні кнопки повернення до гри

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість продовжувати гру, після перевірки результатів |
| Початковий стан програми | Відкрите вікно з повідомленям про результат гри |
| Вхідні дані | - |
| Схема проведення тесту | Натискання на кнопку «continue» або «again» |
| Очікуваний результат | У випадку натискання кнопки «continue» - незмінене поле з виділеням клітинок з числами, що не співпадають. У випадку натискання кнопки «again» - нове пусте поле |
| Стан програми після проведення випробувань | У випадку натискання кнопки «continue» - незмінене поле з виділеням клітинок з числами, що не співпадають. У випадку натискання кнопки «again» - нове пусте поле |



Рисунок 5.23 - до проведення тесту «Повернення до гри» (неправильний результат гри)

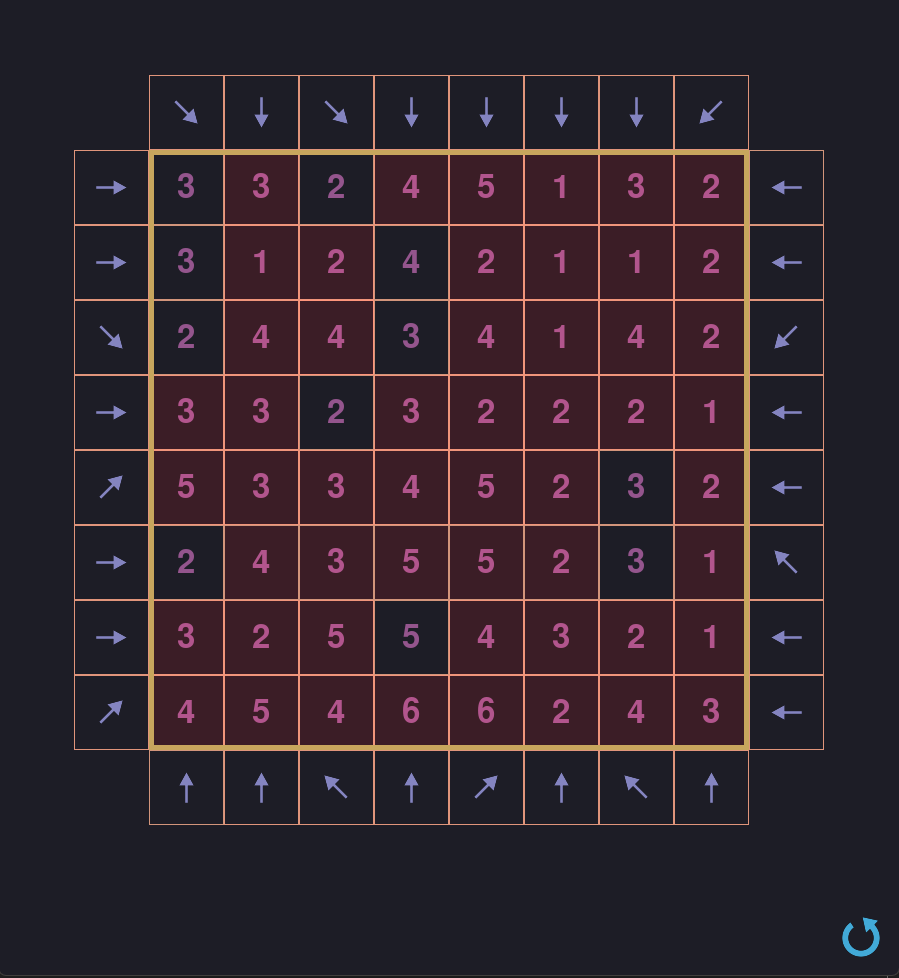


Рисунок 5.24 - після проведення тесту «Повернення до гри» (неправильний результат гри)

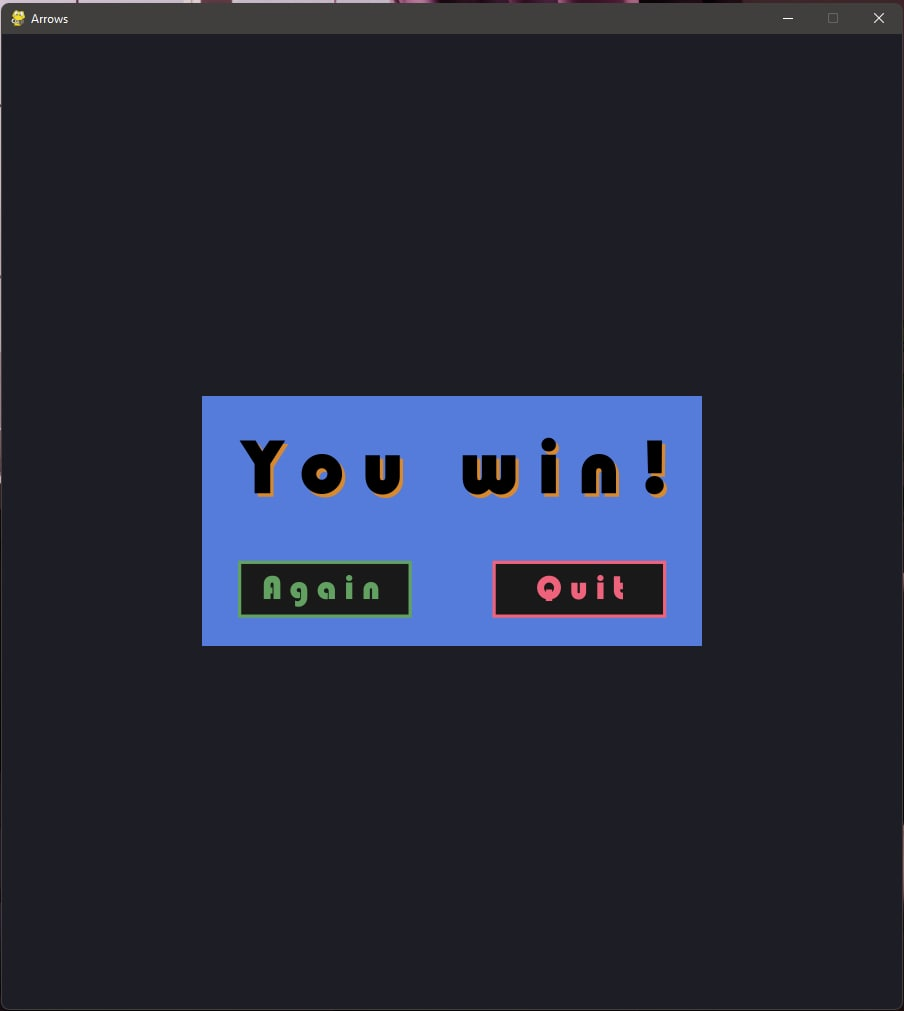


Рисунок 5.25 - до проведення тесту «Повернення до гри» (правильний результат гри)

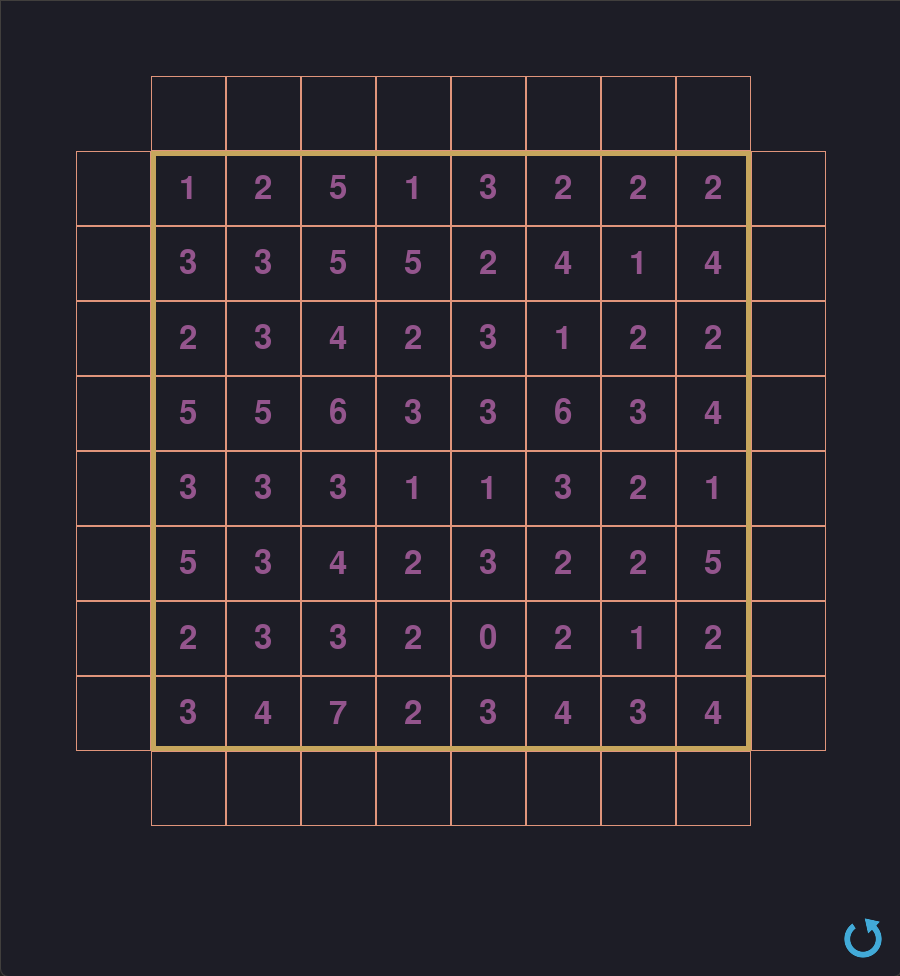


Рисунок 5.26 - після проведення тесту «Повернення до гри» (правильний результат гри)

Таблиця 5.12 ‑ Приклад роботи програми у кінці гри, при натисканні кнопки виходу з гри

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість виходу з гри після перевірки результатів |
| Початковий стан програми | Відкрите вікно з повідомленям про результат гри |
| Вхідні дані | - |
| Схема проведення тесту | Натискання на кнопку «quit» |
| Очікуваний результат | Кінець роботи програми |
| Стан програми після проведення випробувань | Закрите вікно програми |

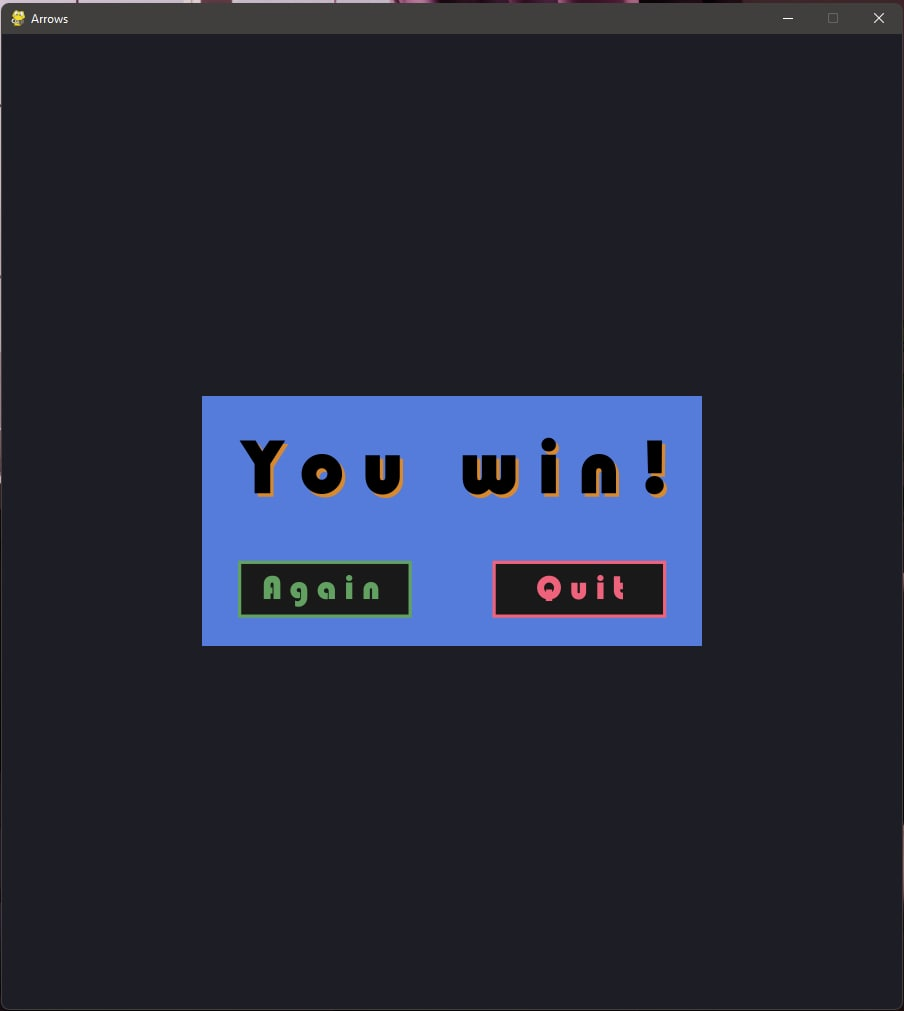


Рисунок 5.27 - до проведення тесту «Вихід з гри» (правильний результат гри)



Рисунок 5.28 - до проведення тесту «Вихід з гри» (неправильний результат гри)

Результат останнього тесту – кінець роботи програми.

ІНСТРУКЦІЯ КОРИСТУВАЧА

Відкриття програми

Після відкриття папки «src» необхідно натиснути на файл типу Python File та назвою «arrows\_game.py».

Відкривається стартове меню *(Рисунок 6.1)*, де можна натиснути кнопку Start для початку гри.



Рисунок 6.1

Складові ігрового поля

Після натискання кнопки Start з’являється ігрове поле *(Рисунок 6.2)*.

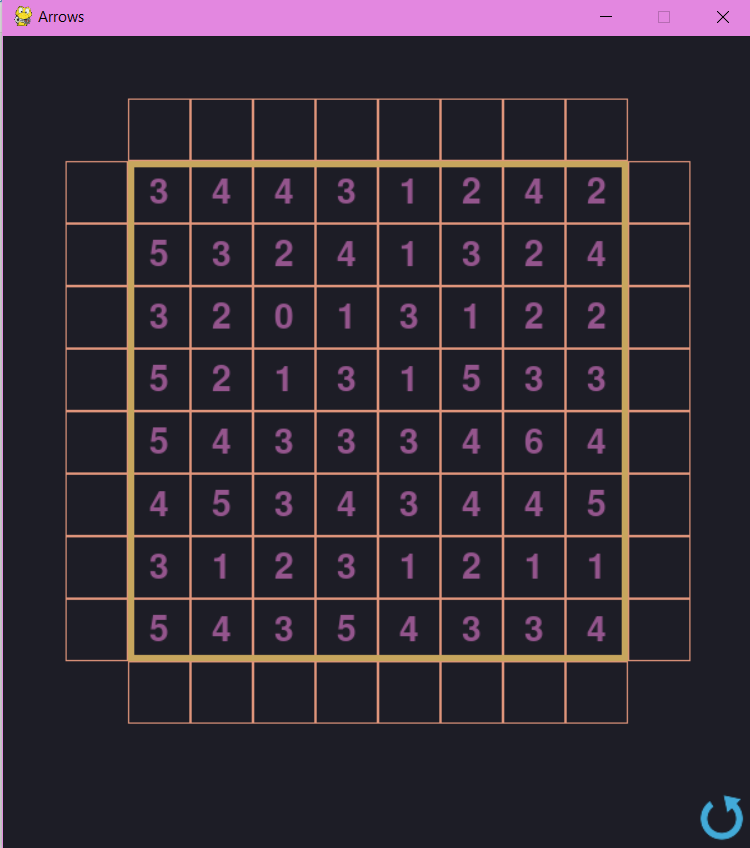


Рисунок 6.2

Ігрове поле складається з квадрату із стороною в 8 менших квадратів, рівних за розміром. Всередині квадратів вписані цифри. Мінімальна цифра, яка може бути в квадраті, - 0 (якщо немає стрілки, що направлена в цей квадрат), максимальна – 8 (за умови, якщо всі можливі стрілки направлені в бік цього квадрату).

З усіх сторін (зверху, знизу, зліва та справа) від головного квадрату є поля, в які необхідно буде вписати стрілки.

Також в нижньому правому кутку вікна є кнопка «Згенерувати нове поле»: якщо натиснути цю кнопку, буде створено нове поле із випадковим наповненням.

Наповнення ігрового поля

Наповнення квадрата – цифри, що вказують на кількість направлених на неї стрілок. Отже потрібно вписати стрілки в клітинки, що навколо квадрата. Стрілки можуть вказувати лише всередину квадрата.

Щоб поставити стрілку, потрібно натиснути на пусту клітинку – вона буде виділена після цього світлішим кольором *(Рисунок 6.3)*.

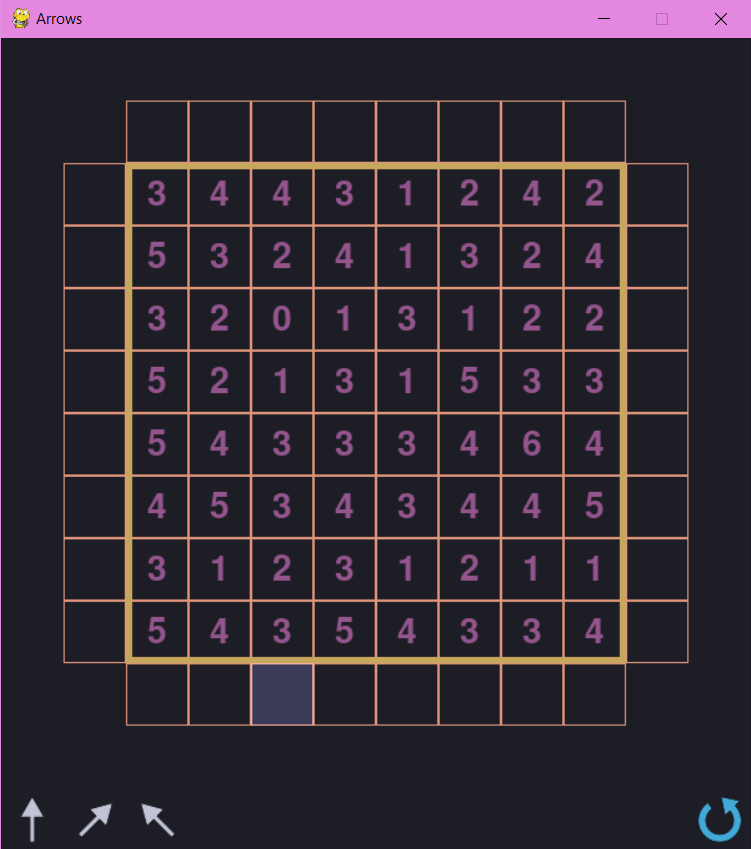


Рисунок 6.3

Після виділення клітинки з’являються опції (варіанти) стрілок, які можна поставити в цю клітинку (щоб не порушити правило щодо направлення стрілок лише всередину квадрату). Якщо клітинка знаходиться не в крайовому положенні, то для неї є три варіанти положення стрілки, як на *Рисунку 6.3*. Якщо клітинка для стрілки має крайове положення, то гравець має в розпорядженні лише 2 опції, як на *Рисунку 6.4*.

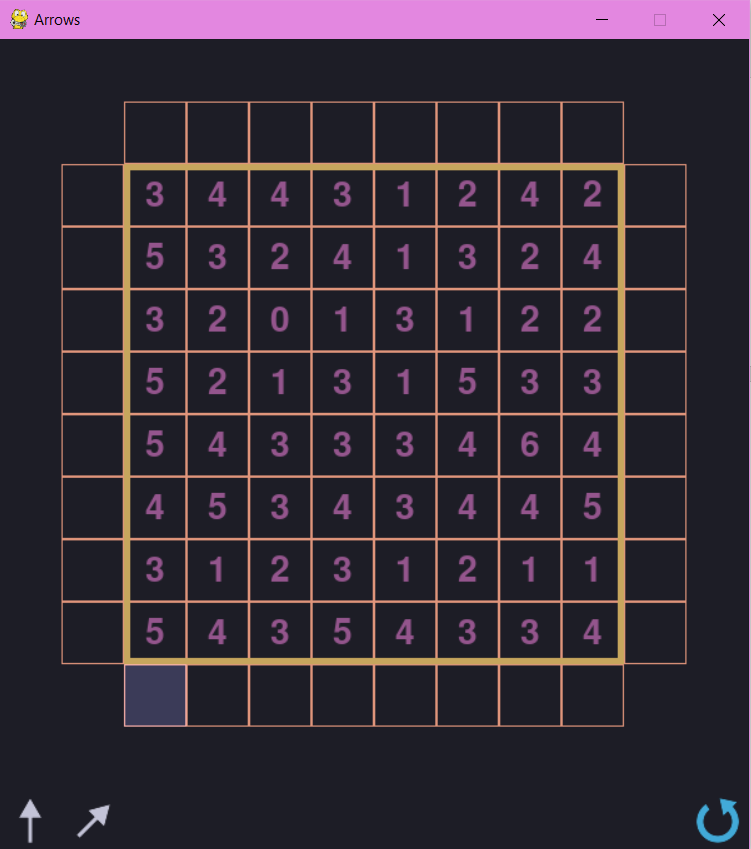


Рисунок 6.4

Щоб заповнити клітинку стрілкою, потрібно натиснути на неї, тоді обрана стрілка буде розташовуватися в клітинці, також напрямок цієї стрілки та клітинки з цифрами, через які вона проходить, будуть підсвічуватись жовтуватим кольором, а знизу справа від інших опцій стрілок з’явиться кнопка «Видалити стрілку»: якщо її натиснути, виділена стрілка буде видалена, утвориться пуста клітинка, яку ще потрібно заповнити *(Рисунок 6.5)*.

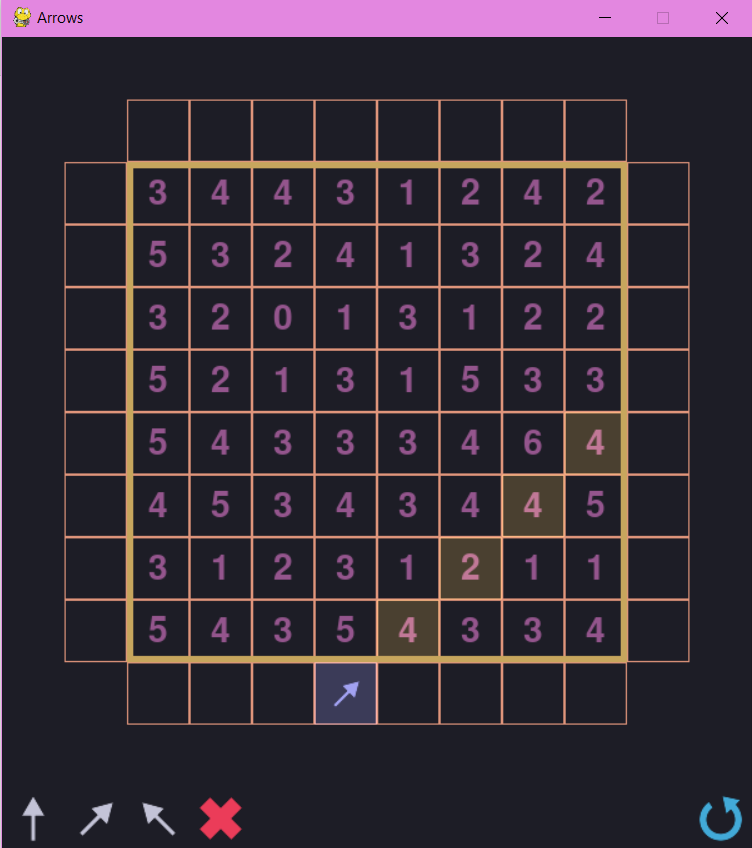


Рисунок 6.5

Кнопка «Змінити напрямок стрілки»: якщо натиснути цю кнопку, виділена стрілка змінить свій напрямок (об’єднання таких функцій як видалення старої стрілки та вписування нової). Ця кнопка є збірним поняттям інших опцій направлення стрілки. Коли вже вписана одна стрілка, натиснувши знизу зліва на інші варіанти для цієї клітинки, можна змінити напрямок стрілки в клітинці, що виділена.

Завершення гри

Після заповнення ігрового поля стрілками різного напрямку *(Рисунок 6.6)* з’являється кнопка «Завершення розташування стрілок» знизу справа орієнтовно екрану (поряд з кнопкою «Згенерувати нове поле»).



Рисунок 6.6

Потрібно натиснути кнопку «Завершення розташування стрілок», таким чином відбувається перевірка: чи правильно розташовані стрілки. У разі правильності буде напис на екрані, що свідчить про перемогу гравця *(Рисунок 6.7).* У разі неправильності буде показано інший напис *(Рисунок 6.8)* та де сталися помилки (цифри, що не відповідають розташуванню стрілок, будуть підсвічуватись червоним).



Рисунок 6.7



Рисунок 6.8

Якщо як в випадку на *Рисунку 6.7*, написано «You win!» («Ви переможець!»), то з’являється нижче від напису можливість обрати з 2 кнопок: кнопка «Again» («Ще раз») та кнопка «Quit» («Вийти»). Якщо натиснути кнопку «Again» («Ще раз»), то згенерується нове ігрове поле та можна починати його заповнювати. Якщо натиснути кнопку «Quit» («Вийти»), то гра автоматично закриється.

Якщо як в випадку на *Рисунку 6.8*, написано «Not so fast!» («Не так швидко!»), то з’являється нижче від напису можливість обрати з 2 кнопок: кнопка «Continue» («Продовжити») та кнопка «Quit» («Вийти»). Якщо натиснути кнопку «Continue» («Продовжити»), то відкриється минуле ігрове поле, де будуть підсвічені цифри, які не відповідають кількості стрілкам, що направлені до них *(Рисунок 6.9)*.

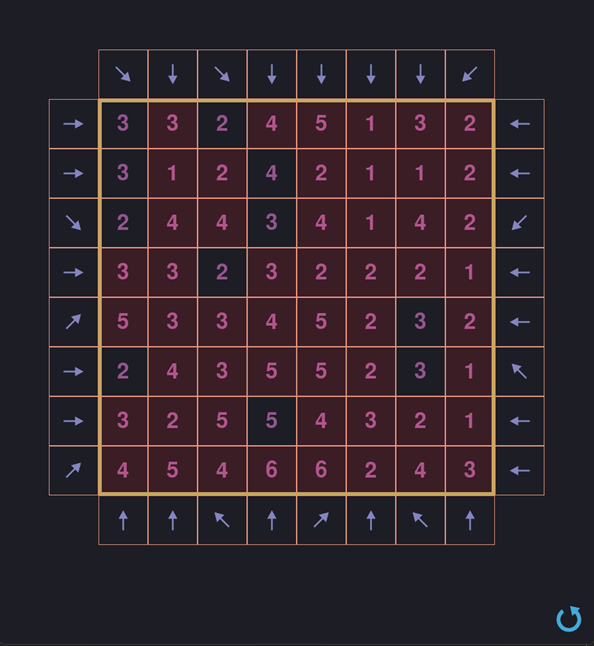


Рисунок 6.9

Якщо натиснути на цифру на ігровому полі, то буде підсвічуватись вона та стрілки, що направлені до неї *(Рисунок 6.10)*. Таким чином буде легше порахувати та зрозуміти, що неправильно було розташовано. Кнопку «Завершення розташування стрілок» можна натискати скільки завгодно разів, ліміту немає.

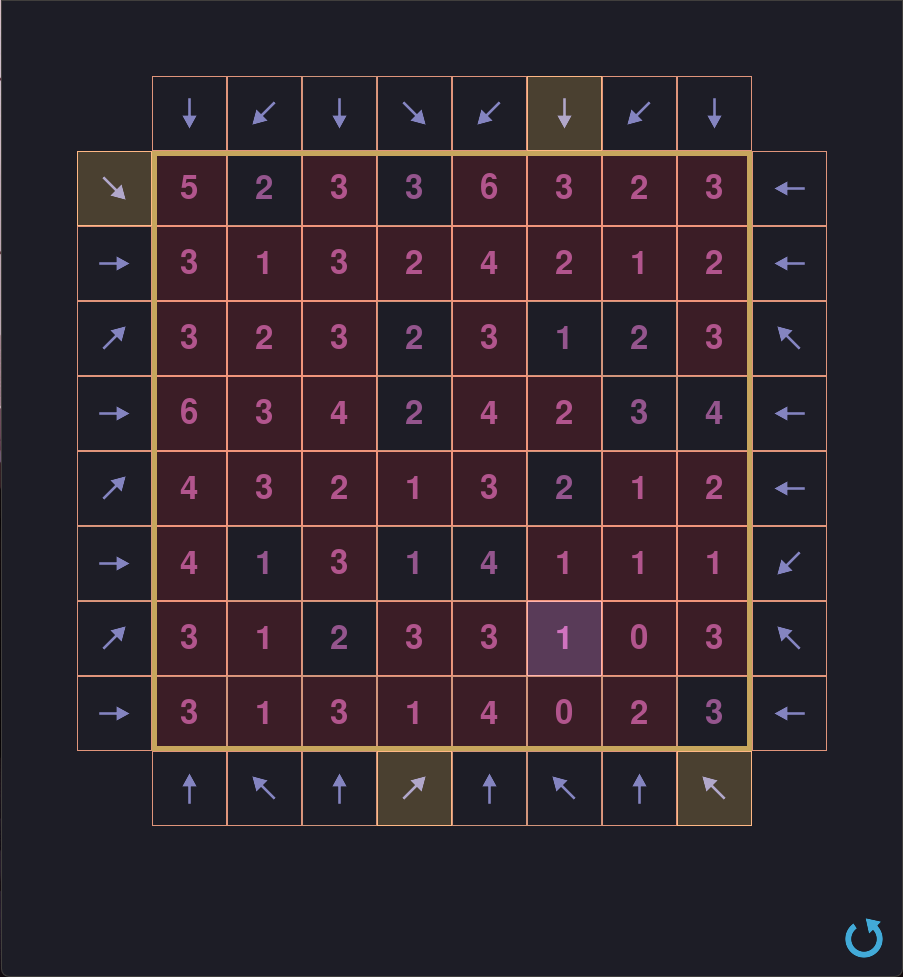


Рисунок 6.10

Завдання гравця

Розставити стрілки навколо квадрату на підставі цифр всередині. Натиснути кнопку «Завершення розташування стрілок», таким чином відбувається перевірка правильно чи ні розташовані стрілки. Завдання побачити напис «You win!» («Ви переможець!»), як на *Рисунку 6.7*.

Системні вимоги

Системні вимоги до програмного забезпечення наведені в *Таблиці 6.1*.

Таблиця 6.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 7/ Windows 8/Windows 10  (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 | 900х975  Мінімальні | 1080х1920 або краще Рекомендовані |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Інтерпретатор для Python версії 3.10 і більше [1]  Пакет для Python – Pygame версії 2.1 і більше [2] | |

ВИСНОВКИ

Основним завданням роботи була розробка програмного забезпечення шляхом використання ООП на прикладі комп’ютерної гри «Стрілки». Програма дозволяє необмежено генерувати нове поле, перевіряти правильність розташування поставлених стрілок, а також досить зручно показує, де сталася помилка, щоб гравець швидко міг це виправити.

Усі об’єкти гри були реалізовані через класи, що були описані в таблицях методів *(Таблиця 4.1, Таблиця 4.2)* та діаграмі класів *(Рисунок 4.1)*.

Під час тестування було використано різні функціональні можливості програми та порівняно графічний результат з очікуваним.

Завдання для гравця - розставити стрілки навколо квадрату на підставі цифр всередині, натиснути кнопку «Завершення розташування стрілки», таким чином відбувається перевірка правильно чи ні розташовані стрілки, у разі правильності буде напис на екрані, що свідчить про перемогу гравця *(Рисунок 6.7)*.

ПЕРЕЛІК ПОСИЛАНЬ

Таблиця 7.1 – Перелік посилань

|  |  |
| --- | --- |
| Електронні ресурси | 1. Офіційна веб сторінка Python: Завантаження останньої версії Python. URL: <https://www.python.org/downloads/> 2. Документація Pygame. URL: <https://www.pygame.org/docs/> |

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

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

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

Виконавець:

Студент Лисенко Андрій Юрійович

«22» березня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Комп’ютерна гра «Стрілки»»

з дисципліни:

«Основи програмування»

Київ 2022

* 1. *Мета*: Метою курсової роботи є розробка комп'ютерної гри "Стрілки". Гра повинна складатись з ігрового поля 8х8. В кожній клітинці розташоване число, яке вказує скільки стрілок направлено на дане поле. Гравець має розтавити стрілки зверху, знизу, зліва та справа від поля на підставі цифр.
  2. *Дата початку роботи*: «02» квітня 2022 р.
  3. *Дата закінчення роботи*: «12» червня 2022 р.
  4. *Вимоги до програмного забезпечення*.

1. Функціональні вимоги:

* Можливість розташовувати стрілки зверху, знизу, зліва та справа від поля,
* Можливість обирати напрямок стрілки,
* Можливість видаляти поставлену стрілку,
* Можливість перевірки правильності розташуваня стрілок

1. Нефункціональні вимоги:

* Програма повинна мати простий та зрозумілий інтерфейс,
* Можливість повторної генерації поля,
* Можливість продовжити грати гру після виграшу не виходячи з програми
* 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 (до16.04.2022 р.)
2. Об'єктно-орієнтоване проектування архітектури програмної системи (до 23.04.2022р.)
3. Розробка програмного забезпечення (до 21.05.2022р.)
4. Тестування розробленої програми (до 04.06.2022р.)
5. Розробка пояснювальної записки (до 11.06.2022 р.).
6. Захист курсової роботи (до 15.06.2022 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 ТЕКСТИ ПРОГРАМНОГО КОДУ

*студента групи ІП-11 2 курсу*

*Лисенка А. Ю.*

*31 арк., 83,5 KB*

*CD-RW*

*Тексти програмного коду комп’ютерної гри «Стрілки»*

arrows\_game.py

import os  
  
import pygame  
  
from assets.board import Board  
from assets.buttons.add\_arrow\_button import AddArrowButton  
from assets.buttons.delete\_arrow\_button import DeleteArrowButton  
from assets.buttons.end\_session\_button import EndSessionButton  
from assets.buttons.gen\_new\_board\_button import GenNewBoardButton  
from assets.messages.correct\_message import CorrectMessage  
from assets.messages.start\_message import StartMessage  
from assets.messages.wrong\_message import WrongMessage  
from control import colors  
from control.screen import Screen  
from control.settings import Settings  
from control.states import States  
from control.time\_control import clock  
from utils.core import Core  
  
  
class ArrowsGame:  
 *"""Main app class"""* def \_\_init\_\_(self):  
 *"""  
 Init game objects  
 """* pygame.init()  
  
 # initialize game objects  
 self.board: Board = Board()  
 self.gen\_new\_board\_button: GenNewBoardButton = GenNewBoardButton()  
 self.add\_arrows\_buttons: list = []  
 self.delete\_arrow\_button: DeleteArrowButton | None = None  
 self.end\_session\_button: EndSessionButton | None = None  
 self.message: StartMessage | WrongMessage | CorrectMessage | None = StartMessage()  
  
 # get screen surface to create window  
 Screen.set\_caption('Arrows')  
  
 def \_handle\_events(self) -> None:  
 *"""  
 Handle pygame events queue  
 """* for event in pygame.event.get():  
 # handle quit event  
 if event.type == pygame.QUIT:  
 exit(0)  
  
 # handle mouse events  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 mouse\_pos = pygame.mouse.get\_pos()  
  
 # active game events  
 if States.current\_state == States.GAME\_ACTIVE:  
 # button click events  
 self.\_handle\_gen\_new\_board\_event(mouse\_pos)  
 self.\_handle\_add\_arrow\_event(mouse\_pos)  
 self.\_handle\_delete\_arrow\_event(mouse\_pos)  
 self.\_handle\_end\_session\_event(mouse\_pos)  
 # arrow and number selection event  
 self.\_handle\_arrow\_selection\_event(mouse\_pos)  
 self.\_handle\_number\_selection\_event(mouse\_pos)  
  
 # non-active game events  
 elif States.current\_state == States.GAME\_START:  
 self.\_handle\_start\_message\_events(mouse\_pos)  
 elif States.current\_state == States.GAME\_END\_WRONG:  
 self.\_handle\_end\_message\_wrong\_events(mouse\_pos)  
 elif States.current\_state == States.GAME\_END\_CORRECT:  
 self.\_handle\_end\_message\_correct\_events(mouse\_pos)  
  
 def \_handle\_gen\_new\_board\_event(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Generate new board if gen\_new\_board\_button is clicked, clear add and delete arrow buttons  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if self.gen\_new\_board\_button.is\_clicked(mouse\_pos):  
 self.board = Board()  
 self.add\_arrows\_buttons.clear()  
 self.delete\_arrow\_button = None  
  
 def \_handle\_add\_arrow\_event(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Set arrow image and direction of select arrow grid square if add\_arrow\_button is clicked. Updates selection for  
 correct highlighting of numbers that the arrow points to. Creates delete arrow button. Creates and end session  
 button if every arrow grid square is filled and gets rid of error numbers highlighting.  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* for add\_arrow\_button in self.add\_arrows\_buttons:  
 if add\_arrow\_button.is\_clicked(mouse\_pos):  
 self.board.set\_arrow\_image(\*add\_arrow\_button.handle\_click())  
 self.board.update\_selection()  
 self.delete\_arrow\_button = DeleteArrowButton(len(self.add\_arrows\_buttons))  
  
 filled\_arrow\_squares = [arrow for arrow in self.board.arrows if arrow.direction]  
 if len(filled\_arrow\_squares) == 2 \* (Settings.grid\_count.x + Settings.grid\_count.y):  
 self.end\_session\_button = EndSessionButton()  
 if self.board.wrong\_numbers:  
 self.board.dehighlight\_errors()  
 self.board.wrong\_numbers.clear()  
  
 def \_handle\_delete\_arrow\_event(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Sets selected image and direction to None. Updates selection for correct highlighting of numbers that the arrow  
 points to. Deletes delete arrow button and end session button, gets rid of error numbers highlighting.  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if not self.delete\_arrow\_button:  
 return  
 if self.delete\_arrow\_button.is\_clicked(mouse\_pos):  
 self.board.set\_arrow\_image(None**,** None**,** colors.highlighted\_blue)  
 self.delete\_arrow\_button = None  
 self.board.update\_selection()  
 self.end\_session\_button = None  
 if self.board.wrong\_numbers:  
 self.board.dehighlight\_errors()  
 self.board.wrong\_numbers = []  
  
 def \_handle\_end\_session\_event(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Evaluates correctness of arrows and sets appropriate game state  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if not self.end\_session\_button:  
 return  
 if self.end\_session\_button.is\_clicked(mouse\_pos):  
 self.board.check\_correctness()  
 if self.board.wrong\_numbers:  
 States.current\_state = States.GAME\_END\_WRONG  
 self.message = WrongMessage()  
 else:  
 States.current\_state = States.GAME\_END\_CORRECT  
 self.message = CorrectMessage()  
  
 def \_handle\_arrow\_selection\_event(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Highlights selected arrow and numbers that it points to. Adds button for adding arrows to selected square.  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if self.board.check\_arrow\_selection(mouse\_pos):  
 self.board.handle\_arrow\_selection(mouse\_pos)  
 arrow = self.board.get\_arrow(mouse\_pos)  
 self.add\_arrows\_buttons.clear()  
 self.delete\_arrow\_button = None  
 if arrow.selected:  
 possible\_directions = Core.get\_possible\_directions(arrow.arrow\_set**,** arrow.arrow\_num)  
 for i**,** direction in enumerate(possible\_directions):  
 self.add\_arrows\_buttons.append(AddArrowButton(direction**,** i))  
 if arrow.direction:  
 self.delete\_arrow\_button = DeleteArrowButton(len(self.add\_arrows\_buttons))  
  
 def \_handle\_number\_selection\_event(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Highlights selected number and arrows that point to it. Clears all arrow manipulation buttons  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if self.board.check\_number\_selection(mouse\_pos):  
 self.board.handle\_number\_selection(mouse\_pos)  
 self.add\_arrows\_buttons.clear()  
 self.delete\_arrow\_button = None  
  
 def \_handle\_start\_message\_events(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Sets game state to active and deletes message.  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if self.message.collide\_rect.collidepoint(mouse\_pos):  
 States.current\_state = States.GAME\_ACTIVE  
 self.message = None  
  
 def \_handle\_end\_message\_correct\_events(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Handles end message button click when game is over and all numbers on grid match with number of arrows that  
 point to it.  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if self.message.collide\_rect\_again.collidepoint(mouse\_pos):  
 States.current\_state = States.GAME\_ACTIVE  
 self.board = Board()  
 self.message = None  
 self.add\_arrows\_buttons.clear()  
 self.delete\_arrow\_button = None  
 self.end\_session\_button = None  
 elif self.message.collide\_rect\_quit.collidepoint(mouse\_pos):  
 exit(0)  
  
 def \_handle\_end\_message\_wrong\_events(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Handles end message button click when game is over and at least one number on grid don't with number of arrows  
 that point to it.  
  
 :return: None  
 :param mouse\_pos: Mouse position  
 """* if self.message.collide\_rect\_continue.collidepoint(mouse\_pos):  
 self.board.highlight\_errors()  
 States.current\_state = States.GAME\_ACTIVE  
 self.message = None  
 self.board.deselect\_all()  
 self.add\_arrows\_buttons.clear()  
 self.delete\_arrow\_button = None  
 self.end\_session\_button = None  
 elif self.message.collide\_rect\_quit.collidepoint(mouse\_pos):  
 exit(0)  
  
 def \_update\_screen(self):  
 *"""  
 Render updated objects on screen and update screen  
 """* # fill background  
 Screen.surface.fill(Screen.bg\_color)  
  
 # redraw objects based on current state  
 if States.current\_state == States.GAME\_ACTIVE:  
 self.board.draw()  
 self.gen\_new\_board\_button.draw()  
 for add\_replace\_button in self.add\_arrows\_buttons:  
 add\_replace\_button.draw()  
 if self.delete\_arrow\_button:  
 self.delete\_arrow\_button.draw()  
 if self.end\_session\_button:  
 self.end\_session\_button.draw()  
 if States.current\_state in [States.GAME\_START**,** States.GAME\_END\_CORRECT**,** States.GAME\_END\_WRONG]:  
 self.message.draw()  
  
 # update screen to expose newly drawn objects  
 pygame.display.update()  
  
 def run(self):  
 *"""  
 Run main game loop  
 """* while True:  
 self.\_handle\_events()  
 self.\_update\_screen()  
 clock.tick(Settings.fps)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # set absolute path when launching from shortcuts  
 os.chdir(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
 # create and run game  
 arrows\_game = ArrowsGame()  
 arrows\_game.run()

collide\_rect\_evaluetor.py

import pygame  
  
  
class CollideRectEvaluator:  
 *"""Evaluates collision rects for message buttons"""* @classmethod  
 def evaluate\_from\_ratios(cls**,** ratios: list[tuple[float**,** float]]**,** base\_rect: pygame.Rect) -> pygame.Rect:  
 *"""  
 Get collision rect for message buttons  
  
 :return: Collision rect for message buttons  
 :param ratios: Coordinates of collide rect box divided by image size  
 :param base\_rect: Rect of image that holds collide rect  
 """* collide\_rect\_width = (ratios[1][0] - ratios[0][0]) \* base\_rect.width  
 collide\_rect\_top = base\_rect.top + ratios[0][1] \* base\_rect.height  
 collide\_rect\_left = base\_rect.left + ratios[0][0] \* base\_rect.width  
 collide\_rect\_height = (ratios[1][1] - ratios[0][1]) \* base\_rect.height  
  
 collide\_rect = pygame.Rect(  
 (collide\_rect\_left**,** collide\_rect\_top)**,** (collide\_rect\_width**,** collide\_rect\_height)  
 )  
  
 return collide\_rect

core.py

import random  
  
import pygame.sprite  
from pygame.math import Vector2  
  
from control.settings import Settings  
  
  
class Core:  
 *"""Class for various functions and variables for core game logic"""* arrow\_directions: list[tuple[int**,** int]] = [  
 (1**,** -1)**,** (1**,** 0)**,** (1**,** 1)**,** (0**,** 1)**,** (-1**,** 1)**,** (-1**,** 0)**,** (-1**,** -1)**,** (0**,** -1)**,** ]  
  
 arrow\_sets: dict[tuple[int**,** int]**,** list[tuple[int**,** int]]] = {  
 (0**,** -1): ((0**,** 1)**,** (1**,** 1)**,** (-1**,** 1))**,** (1**,** 0): ((-1**,** 0)**,** (-1**,** 1)**,** (-1**,** -1))**,** (-1**,** 0): ((1**,** 0)**,** (1**,** 1)**,** (1**,** -1))**,** (0**,** 1): ((0**,** -1)**,** (1**,** -1)**,** (-1**,** -1))  
 }  
  
 forbidden\_directions: dict[tuple[int**,** int]**,** list[tuple[int**,** int]]] = {  
 (0**,** -1): [(-1**,** 1)**,** (1**,** 1)]**,** (1**,** 0): [(-1**,** -1)**,** (-1**,** 1)]**,** (-1**,** 0): [(1**,** -1)**,** (1**,** 1)]**,** (0**,** 1): [(-1**,** -1)**,** (1**,** -1)]  
 }  
  
 arrows: dict[tuple[int**,** int]**,** list[tuple[int**,** int]]] = {}  
 numbers: list[list[int]] = []  
  
 @classmethod  
 def gen\_arrows(cls) -> None:  
 *"""  
 Generate arrows dict with keys as direction on game board and list of arrows as values  
 """* arrows = {}  
 for arrow\_set in cls.arrow\_sets:  
 arrows[arrow\_set] = []  
 grid\_count = Settings.grid\_count.x if arrow\_set in [(0**,** -1)**,** (0**,** 1)] else Settings.grid\_count.y  
 for i in range(int(grid\_count)):  
 possible\_directions = cls.get\_possible\_directions(arrow\_set**,** i)  
 choice = random.choice(possible\_directions)  
 arrows[arrow\_set].append(choice)  
  
 cls.arrows = arrows  
  
 @classmethod  
 def get\_possible\_directions(cls**,** arrow\_set: tuple[int**,** int]**,** arrow\_num: int) -> list[tuple[int**,** int]]:  
 *"""  
 Get possible arrow directions for given arrow location  
  
 :return: List of possible directions that arrow can point to  
 :param arrow\_set: Direction in which the arrow is located  
 :param arrow\_num: Sequence number of arrow on arrow set counting from up or left  
 """* grid\_count = Settings.grid\_count.x if arrow\_set in [(0**,** -1)**,** (0**,** 1)] else Settings.grid\_count.y  
 possible\_directions = list(cls.arrow\_sets[arrow\_set])  
 if arrow\_num == 0:  
 possible\_directions.remove(cls.forbidden\_directions[arrow\_set][0])  
 if arrow\_num == grid\_count - 1:  
 possible\_directions.remove(cls.forbidden\_directions[arrow\_set][1])  
 return possible\_directions  
  
 @classmethod  
 def get\_position(cls**,** arrows\_set\_direction: tuple[int**,** int]**,** arrow\_num: int) -> Vector2:  
 *"""  
 Get position relative to board of given arrow  
  
 :return: Position vector of arrow relative to game board  
 :param arrows\_set\_direction: Direction in which the arrow is located  
 :param arrow\_num: Sequence number of arrow on arrow set counting from up or left  
 """* if arrows\_set\_direction == (0**,** -1):  
 position = arrow\_num**,** -1  
 elif arrows\_set\_direction == (1**,** 0):  
 position = Settings.grid\_count.x**,** arrow\_num  
 elif arrows\_set\_direction == (-1**,** 0):  
 position = -1**,** arrow\_num  
 elif arrows\_set\_direction == (0**,** 1):  
 position = arrow\_num**,** Settings.grid\_count.y  
 else:  
 raise ValueError('arrow set direction not valid')  
  
 return Vector2(position)  
  
 @classmethod  
 def get\_span(cls**,** position: Vector2**,** arrow: tuple[int**,** int] | None = None) -> list[Vector2**,** ...]:  
 *"""  
 Get all grid squares that given arrow points to  
  
 :return: List of grid square positions that given arrow points to  
 :param position: Position vector of arrow relative to game board  
 :param arrow: Direction in which arrow points to  
 """* if not arrow:  
 return []  
  
 grid\_squares = []  
 grid\_square = position.copy()  
 while True:  
 grid\_square += Vector2(arrow)  
  
 rect = pygame.Rect((0**,** 0)**,** tuple(Settings.grid\_count))  
 if not rect.collidepoint(tuple(grid\_square)):  
 break  
  
 grid\_squares.append(grid\_square.copy())  
  
 return grid\_squares  
  
 @classmethod  
 def get\_pointings(cls**,** grid\_square: Vector2) -> list[tuple[tuple[int**,** int]**,** int]]:  
 *"""  
 Get arrows that point to specified location on board  
  
 :return: List of arrow sets and arrow numbers that point to given grid square  
 :param grid\_square: Position of grid square relative to board  
 """* result = []  
 for arrows\_set\_direction**,** arrows\_set in cls.arrows.items():  
 for arrow\_num**,** arrow in enumerate(arrows\_set):  
 position = cls.get\_position(arrows\_set\_direction**,** arrow\_num)  
 if arrow:  
 if grid\_square in cls.get\_span(position**,** arrow):  
 result.append((arrows\_set\_direction**,** arrow\_num))  
 return result  
  
 @classmethod  
 def count\_pointings(cls**,** grid\_square: Vector2) -> int:  
 *"""  
 Count number of arrows that point to specified location on board  
  
 :return: Number of arrows that point to given grid square  
 :param grid\_square: Position of grid square relative to board  
 """* return len(cls.get\_pointings(grid\_square))  
  
 @classmethod  
 def evaluate\_correctness(cls) -> list[tuple[int**,** int]]:  
 *"""  
 Get all numbers that don't match with number of arrows that point to them  
  
 :return: List of numbers, values of which don't match with number of arrows that point to them  
 """* wrong\_numbers = []  
 for col**,** numbers\_col in enumerate(cls.numbers):  
 for row**,** number in enumerate(numbers\_col):  
 if number != cls.count\_pointings(Vector2(col**,** row)):  
 wrong\_numbers.append((col**,** row))  
 return wrong\_numbers  
  
 @classmethod  
 def gen\_numbers(cls) -> None:  
 *"""  
 Generate numbers matrix based on previously generated arrows  
 """* cls.gen\_arrows()  
  
 cls.numbers = [  
 [  
 cls.count\_pointings(Vector2(col**,** row))  
 for row in range(int(Settings.grid\_count.y))  
 ]  
 for col in range(int(Settings.grid\_count.x))  
 ]

colors.py

*"""  
Colors of all game objects  
"""*background = (29**,** 29**,** 38)  
board\_frame = (199**,** 166**,** 93)  
grid\_square\_frames = (224**,** 149**,** 123)  
numbers = (148**,** 85**,** 141)  
arrows = (135**,** 135**,** 197)  
  
highlighted\_blue = (30**,** 30**,** 50)  
highlighted\_yellow = (45**,** 35**,** 10)  
highlight\_red = (30**,** 0**,** 0)  
  
delete\_arrow\_button = (240**,** 62**,** 92)  
end\_session\_button = (108**,** 192**,** 108)  
arrows\_button = (200**,** 200**,** 220)  
new\_board\_button = (68**,** 174**,** 221)

grid\_position.py

from control.settings import Settings  
from pygame.math import Vector2  
  
  
class GridPosition:  
 *"""Class to manage coordinates of game objects on game board"""* def \_\_init\_\_(self**,** grid\_square\_pos: Vector2 | tuple):  
 *"""  
 :return: None  
 :param grid\_square\_pos: Column and row of grid square relative to board  
 """* self.grid\_square\_pos = Vector2(grid\_square\_pos)  
  
 def get\_coords(self) -> Vector2:  
 *"""  
 Get exact pixel position on upper right corner of the object  
  
 :return: Vector of pixel coordinates of topleft corner of grid square  
 """* return (self.grid\_square\_pos + Settings.window\_margin) \* Settings.grid\_size  
  
 def get\_coords\_center(self) -> Vector2:  
 *"""  
 Get center of specified grid square in pixels  
  
 :return: Vector of pixel coordinates of the center of grid square  
 """* return self.get\_coords() + Vector2(Settings.grid\_size**,** Settings.grid\_size) / 2

screen.py

import pygame  
from dataclasses import dataclass  
  
from control import colors  
from control.settings import Settings  
  
  
@dataclass  
class Screen:  
 *"""Class to store screen rect and surface"""* surface = pygame.display.set\_mode(Settings.get\_resolution())  
 rect = surface.get\_rect()  
 bg\_color = colors.background  
  
 @classmethod  
 def set\_caption(cls**,** name: str) -> None:  
 *"""  
 Set caption for game window  
  
 :return: None  
 :param name: Name for window caption  
 """* pygame.display.set\_caption(name)

settings.py

from dataclasses import dataclass  
  
from pygame.math import Vector2  
  
  
@dataclass(frozen=True)  
class Settings:  
 *"""Class to store settings for the app"""* # basic settings  
 grid\_count: Vector2 = Vector2(8**,** 8)  
 window\_margin: Vector2 = Vector2(2**,** 2)  
 grid\_size: int = 75  
 fps: int = 30  
  
 # arrows settings  
 arrow\_size: int = 35  
  
 # numbers size  
 number\_size: int = 48  
  
 # button settings  
 button\_icon\_size: int = 40  
  
 # message settings  
 message\_size: int = 500  
  
 @classmethod  
 def get\_resolution(cls) -> Vector2:  
 *"""  
 Get actual pixel resolution of entire screen  
  
 :return: Actual pixel resolution of entire screen  
 """* return (cls.grid\_count + 2 \* cls.window\_margin + Vector2(0**,** 1)) \* cls.grid\_size

states.py

from dataclasses import dataclass  
  
  
@dataclass  
class States:  
 *"""Class to store all and current game states"""* # all game states  
 GAME\_ACTIVE: int = 0  
 GAME\_START: int = 1  
 GAME\_END\_CORRECT: int = 2  
 GAME\_END\_WRONG: int = 3  
  
 # current game state  
 current\_state = GAME\_START

time\_control.py

*"""  
Clock to control fps of the game and other time dependent objects  
"""*import pygame  
  
clock = pygame.time.Clock()

add\_arrows\_button.py

from assets.arrow import Arrow  
from assets.buttons.button import Button  
from control import colors  
from control.grid\_position import GridPosition  
from control.settings import Settings  
  
  
class AddArrowButton(Button):  
 *"""Buttons for adding and replacing"""* def \_\_init\_\_(self**,** direction: tuple[int**,** int]**,** position: int):  
 *"""  
 :return: None  
 :param direction: Direction of arrow that it adds when button is clicked  
 """* super().\_\_init\_\_(  
 f'../assets/buttons/arrow{direction}.png'**,** GridPosition((position - 2**,** Settings.grid\_count.y + 2))**,** colors.arrows\_button**,** False**,** Settings.button\_icon\_size / 500  
 )  
 self.direction = direction  
  
 def handle\_click(self) -> tuple[Arrow**,** tuple[int**,** int]**,** tuple[int]]:  
 *"""  
 Return new arrow image of given direction to set as image attribute of selected square  
  
 :return: Needed attributes for changing the image of arrow grid square: new arrow image, its direction and color  
 that it was highlighted with  
 """* return Arrow(self.direction).image**,** self.direction**,** colors.highlighted\_blue

button.py

import pygame  
from pygame.constants import BLEND\_RGB\_MULT  
from pygame.math import Vector2  
  
from abc import abstractmethod  
  
from control import colors  
from control.grid\_position import GridPosition  
from control.screen import Screen  
from control.settings import Settings  
  
  
class Button:  
 *"""Abstract button class for user interface buttons"""* def \_\_init\_\_(self**,** image\_path: str**,** position: GridPosition**,** color: tuple[int**,** ...]**,** conform\_size: bool = True**,** scaling\_nonconformal: float = None):  
 *"""  
 :return: None  
 :param image\_path: Filepath to button image  
 :param position: Grid square position relative to board  
 :param color: Color to fill the button image with  
 :param conform\_size: Scale image according to button icon size value in settings. If False -  
 scaling\_nonconformal parameter is required  
 :param scaling\_nonconformal: Value to scale image with  
 """* # basic attributes  
 self.image = pygame.image.load(image\_path).convert\_alpha()  
 self.image\_size = self.image.get\_size()  
 self.position = position  
  
 # scale image if necessary  
 if conform\_size:  
 image\_size = Vector2(self.image.get\_size())  
 scaling\_factor = Settings.button\_icon\_size / image\_size.x  
 self.image\_size = tuple(scaling\_factor \* image\_size)  
 self.image = pygame.transform.smoothscale(self.image**,** self.image\_size)  
 if scaling\_nonconformal:  
 self.image\_size = tuple(scaling\_nonconformal \* Vector2(self.image\_size))  
 self.image = pygame.transform.smoothscale(self.image**,** self.image\_size)  
  
 # color button image  
 self.image.fill(color**,** special\_flags=BLEND\_RGB\_MULT)  
 background = pygame.Surface(self.image\_size)  
 background.fill(colors.background)  
 background.blit(self.image**,** (0**,** 0))  
 self.image = background  
  
 # set rect  
 self.rect = self.image.get\_rect()  
 self.rect.center = position.get\_coords\_center()  
  
 def is\_clicked(self**,** mouse\_pos: tuple[int**,** int]) -> bool:  
 *"""Check if button is clicked"""* if self.rect.collidepoint(mouse\_pos):  
 return True  
 return False  
  
 def draw(self) -> None:  
 *"""Draw object to given surface"""* Screen.surface.blit(self.image**,** self.rect)  
  
 @abstractmethod  
 def handle\_click(self**,** \*args**,** \*\*kwargs):  
 *"""Abstract method for action that button makes"""* return

delete\_arrow\_button.py

from assets.buttons.button import Button  
from control import colors  
from control.grid\_position import GridPosition  
from control.settings import Settings  
  
  
class DeleteArrowButton(Button):  
 *"""Buttons for deleting arrows"""* def \_\_init\_\_(self**,** position: int):  
 super().\_\_init\_\_(  
 '../assets/buttons/delete\_arrow\_button.png'**,** GridPosition((position - 2**,** Settings.grid\_count.y + 2))**,** colors.delete\_arrow\_button  
 )  
  
 def handle\_click(self):  
 *"""Handles in main game class"""* return

end\_session\_button.py

from pygame import Vector2  
  
from assets.buttons.button import Button  
from control import colors  
from control.grid\_position import GridPosition  
from control.settings import Settings  
  
  
class EndSessionButton(Button):  
 *"""Button class for ending session and checking if player won"""* def \_\_init\_\_(self):  
 super().\_\_init\_\_(  
 '../assets/buttons/end\_session\_button.png'**,** GridPosition(Settings.grid\_count + Vector2(0**,** 2))**,** colors.end\_session\_button  
 )  
  
 def handle\_click(self):  
 *"""Handles in main game class"""* return

gen\_new\_board\_button.py

from pygame.math import Vector2  
  
from assets.buttons.button import Button  
from control import colors  
from control.grid\_position import GridPosition  
from control.settings import Settings  
  
  
class GenNewBoardButton(Button):  
 *"""Button class for genereting new board"""* def \_\_init\_\_(self):  
 super().\_\_init\_\_(  
 '../assets/buttons/gen\_new\_board\_button.png'**,** GridPosition(Settings.grid\_count + Vector2(1**,** 2))**,** colors.new\_board\_button  
 )  
  
 def handle\_click(self):  
 *"""Handles in main game class"""* return

arrow\_grid\_square.py

import pygame  
  
from assets.grid\_squares.grid\_square import GridSquare  
from control.grid\_position import GridPosition  
from utils.core import Core  
  
  
class ArrowGridSquare(GridSquare):  
 *"""Subclass of grid square to hold arrows and related attributes"""* def \_\_init\_\_(self**,** content: pygame.Surface | None**,** arrow\_set: tuple[int**,** int]**,** arrow\_num: int**,** direction: tuple[int**,** int] | None = None):  
 *"""  
 :return: None  
 :param arrow\_set: Direction in which the arrow is located  
 :param arrow\_num: Sequence number of arrow on arrow set counting from up or left  
 """* super().\_\_init\_\_(content)  
 self.arrow\_set = arrow\_set  
 self.arrow\_num = arrow\_num  
 self.position = GridPosition(Core.get\_position(arrow\_set**,** arrow\_num))  
 self.rect.topleft = self.position.get\_coords()  
 self.direction = direction  
  
 def set\_image(self**,** image: pygame.Surface**,** direction: tuple[int**,** int]) -> None:  
 *"""  
 Set arrow image for arrow grid square  
  
 :return: None  
 :param image: Arrow image to fill the arrow grid square  
 :param direction: Direction that arrow points to  
 """* self.\_\_init\_\_(image**,** self.arrow\_set**,** self.arrow\_num**,** direction)

grid\_square.py

import pygame  
from pygame.constants import BLEND\_RGB\_ADD**,** BLEND\_RGB\_SUB  
from pygame.math import Vector2  
  
from control import colors  
from control.screen import Screen  
from control.settings import Settings  
  
  
class GridSquare:  
 *"""Class for grid square objects that hold arrows or numbers"""* def \_\_init\_\_(self**,** content: pygame.Surface | None):  
 *"""  
 :return: None  
 :param content: Image to fill the square with  
 """* super().\_\_init\_\_()  
 # frame that surrounds the square  
 self.frame\_size = (Settings.grid\_size**,** Settings.grid\_size)  
 self.frame\_rect = pygame.Rect((0**,** 0)**,** self.frame\_size)  
 self.frame\_width = 1  
 self.frame\_color = colors.grid\_square\_frames  
  
 # get content rect  
 if content:  
 self.content\_image = content  
 self.content\_rect = content.get\_rect()  
 self.content\_rect.center = Vector2(Settings.grid\_size**,** Settings.grid\_size) / 2  
  
 # Indicator for toggling selection and related attributes  
 self.selected = False  
  
 # get image and rect for sprite.draw() method  
 self.image = pygame.Surface(self.frame\_size)  
 self.image.fill(colors.background)  
 pygame.draw.rect(self.image**,** self.frame\_color**,** self.frame\_rect**,** self.frame\_width)  
 if content:  
 self.image.blit(self.content\_image**,** self.content\_rect)  
  
 self.rect = self.image.get\_rect()  
  
 def draw(self) -> None:  
 Screen.surface.blit(self.image**,** self.rect)  
  
 def select(self**,** highlight\_color: tuple[int]) -> None:  
 *"""  
 Add given color to image  
  
 :return: None  
 :param highlight\_color: Color to restore highlighting with  
 """* self.selected = True  
 self.image.fill(highlight\_color**,** special\_flags=BLEND\_RGB\_ADD)  
  
 def deselect(self**,** highlight\_color: tuple[int]) -> None:  
 *"""  
 Subtract given color from image  
  
 :return: None  
 :param highlight\_color: Color to restore highlighting with  
 """* self.selected = False  
 self.image.fill(highlight\_color**,** special\_flags=BLEND\_RGB\_SUB)

number\_grid\_square.py

import pygame  
from pygame.constants import BLEND\_RGB\_ADD**,** BLEND\_RGB\_SUB  
  
from assets.grid\_squares.grid\_square import GridSquare  
from control import colors  
from control.grid\_position import GridPosition  
  
  
class NumberGridSquare(GridSquare):  
 *"""Subclass of grid square to hold arrows and related attributes"""* def \_\_init\_\_(self**,** content: pygame.Surface | None**,** col: int**,** row: int**,** value: int):  
 *"""  
 :return: None  
 :param content: Image to fill the square with  
 :param col: Column of number position relative to board  
 :param row: Row of number position relative to board  
 :param value: Numeric value of number object  
 """* super().\_\_init\_\_(content)  
 self.col = col  
 self.row = row  
 self.value = value  
 self.position = GridPosition((col**,** row))  
 self.rect.topleft = self.position.get\_coords()  
  
 def highlight\_error(self) -> None:  
 *"""  
 Add red highlight color to image  
 """* self.image.fill(colors.highlight\_red**,** special\_flags=BLEND\_RGB\_ADD)  
  
 def dehighlight\_error(self) -> None:  
 *"""  
 Subtract red highlight color from image  
 """* self.image.fill(colors.highlight\_red**,** special\_flags=BLEND\_RGB\_SUB)

correct\_message.py

from assets.messages.message import Message  
from utils.collide\_rect\_evaluetor import CollideRectEvaluator  
  
  
class CorrectMessage(Message):  
 *"""Message that appears before the game starts"""* def \_\_init\_\_(self):  
 super().\_\_init\_\_('../assets/messages/message\_correct.png')  
 collide\_rect\_ratios\_again = [(144 / 2000**,** 660 / 1000)**,** (838 / 2000**,** 887 / 1000)]  
 collide\_rect\_ratios\_quit = [(1162 / 2000**,** 660 / 1000)**,** (1856 / 2000**,** 887 / 1000)]  
 self.collide\_rect\_again = CollideRectEvaluator.evaluate\_from\_ratios(collide\_rect\_ratios\_again**,** self.rect)  
 self.collide\_rect\_quit = CollideRectEvaluator.evaluate\_from\_ratios(collide\_rect\_ratios\_quit**,** self.rect)

message.py

import pygame  
from pygame.math import Vector2  
  
from control.screen import Screen  
from control.settings import Settings  
  
  
class Message:  
 *"""Base class for messages"""* def \_\_init\_\_(self**,** image\_path: str):  
 *"""  
 :return: None  
 :param image\_path: Path to message image path  
 """* # get and scale image  
 self.image = pygame.image.load(image\_path).convert\_alpha()  
 size = self.image.get\_size()  
 scaling\_factor = Settings.message\_size / size[0]  
 new\_size = tuple(scaling\_factor \* Vector2(size))  
 self.image = pygame.transform.smoothscale(self.image**,** new\_size)  
 self.image\_size = self.image.get\_size()  
  
 # get and center rect  
 self.rect = self.image.get\_rect()  
 self.rect.center = Settings.get\_resolution() // 2  
  
 def draw(self) -> None:  
 Screen.surface.blit(self.image**,** self.rect)

start\_message.py

from assets.messages.message import Message  
from utils.collide\_rect\_evaluetor import CollideRectEvaluator  
  
  
class StartMessage(Message):  
 *"""Message that appears before the game starts"""* def \_\_init\_\_(self):  
 super().\_\_init\_\_('../assets/messages/message\_start.png')  
 collide\_rect\_ratios = [(557 / 2000**,** 544 / 1000)**,** (1459 / 2000**,** 837 / 1000)]  
 self.collide\_rect = CollideRectEvaluator.evaluate\_from\_ratios(collide\_rect\_ratios**,** self.rect)

wrong\_message.py

from assets.messages.message import Message  
from utils.collide\_rect\_evaluetor import CollideRectEvaluator  
  
  
class WrongMessage(Message):  
 *"""Message that appears before the game starts"""* def \_\_init\_\_(self):  
 super().\_\_init\_\_('../assets/messages/message\_wrong.png')  
 collide\_rect\_ratios\_continue = [(131 / 2000**,** 604 / 1000)**,** (826 / 2000**,** 832 / 1000)]  
 collide\_rect\_ratios\_quit = [(1149 / 2000**,** 604 / 1000)**,** (1843 / 2000**,** 832 / 1000)]  
 self.collide\_rect\_continue = CollideRectEvaluator.evaluate\_from\_ratios(collide\_rect\_ratios\_continue**,** self.rect)  
 self.collide\_rect\_quit = CollideRectEvaluator.evaluate\_from\_ratios(collide\_rect\_ratios\_quit**,** self.rect)

arrow.py

import pygame  
from pygame.constants import BLEND\_RGB\_MULT  
from pygame.math import Vector2  
  
from control import colors  
from control.settings import Settings  
  
  
class Arrow:  
 *"""Class for rendering arrows"""* def \_\_init\_\_(self**,** direction: tuple[int**,** int]):  
 *"""  
 :return: None  
 :param direction: Direction of arrow to render appropriate image  
 """* # load image and scale  
 arrow\_image = pygame.image.load(f'../assets/arrows/{direction}.png').convert\_alpha()  
  
 image\_size\_vec = Vector2(arrow\_image.get\_size())  
 scaling\_factor = Settings.arrow\_size / image\_size\_vec.length()  
 self.arrow\_image\_size = tuple(scaling\_factor \* image\_size\_vec)  
  
 arrow\_image = pygame.transform.smoothscale(arrow\_image**,** self.arrow\_image\_size)  
  
 # fill arrow with specific color  
 arrow\_image.fill(colors.arrows**,** special\_flags=BLEND\_RGB\_MULT)  
  
 # background surface to draw arrow on  
 background = pygame.Surface(self.arrow\_image\_size)  
 background.fill(colors.background)  
  
 # blit arrow to background  
 background.blit(arrow\_image**,** (0**,** 0))  
  
 self.image = background

board.py

import pygame  
from pygame.math import Vector2  
  
from assets.grid\_squares.arrow\_grid\_square import ArrowGridSquare  
from assets.grid\_squares.number\_grid\_square import NumberGridSquare  
from assets.number import Number  
from control import colors  
from control.grid\_position import GridPosition  
from control.screen import Screen  
from control.settings import Settings  
from utils.core import Core  
  
  
class Board:  
 *"""Class for game board that holds arrows and numbers"""* def \_\_init\_\_(self):  
 # frame that surrounds grid squares that hold numbers  
 self.frame\_size = tuple(Settings.grid\_count \* Settings.grid\_size)  
 self.frame\_rect = pygame.Rect(  
 tuple(GridPosition((0**,** 0)).get\_coords())**,** self.frame\_size  
 )  
 self.frame\_color = colors.board\_frame  
 self.frame\_width = 5  
  
 # generate random values matrix and arrows dictionary  
 Core.gen\_numbers()  
 arrows = Core.arrows  
 numbers = Core.numbers  
  
 # wrong numbers list for highlighting at the end of the game  
 self.wrong\_numbers = []  
  
 # create grid squares group that hold numbers  
 self.numbers = []  
 for col**,** numbers\_col in enumerate(numbers):  
 for row**,** number in enumerate(numbers\_col):  
 self.numbers.append(NumberGridSquare(  
 Number(number).image**,** col**,** row**,** number  
 ))  
  
 # create empty grid squares that hold arrows  
 self.arrows = []  
 for arrows\_set\_direction**,** arrows\_set in arrows.items():  
 for arrow\_num**,** arrow in enumerate(arrows\_set):  
 self.arrows.append(ArrowGridSquare(  
 None**,** arrows\_set\_direction**,** arrow\_num  
 ))  
  
 # current selection indicator for managing selections  
 self.currently\_selected: ArrowGridSquare | NumberGridSquare | None = None  
  
 def update\_selection(self) -> None:  
 *"""  
 Deselect and select object for correct arrow adding and deletion  
 """* selected\_arrow = self.get\_selected\_arrow()  
 if selected\_arrow:  
 self.handle\_arrow\_selection(selected\_arrow.position.get\_coords\_center())  
 self.handle\_arrow\_selection(selected\_arrow.position.get\_coords\_center())  
  
 def deselect\_all(self) -> None:  
 *"""  
 Deselect any selection  
 """* if isinstance(self.currently\_selected**,** ArrowGridSquare):  
 self.handle\_arrow\_selection(self.currently\_selected.position.get\_coords\_center())  
 if isinstance(self.currently\_selected**,** NumberGridSquare):  
 self.handle\_number\_selection(self.currently\_selected.position.get\_coords\_center())  
  
 def get\_selected\_arrow(self) -> ArrowGridSquare:  
 *"""  
 Return selected arrow object if any were selected  
  
 :return: Arrow object that is selected if any  
 """* for arrow in self.arrows:  
 if arrow.selected:  
 return arrow  
  
 def dehighlight\_errors(self) -> None:  
 *"""  
 Get rid of highlighting on numbers that don't match  
 """* for number in self.numbers:  
 if tuple(number.position.grid\_square\_pos) in self.wrong\_numbers:  
 number.dehighlight\_error()  
  
 def highlight\_errors(self) -> None:  
 *"""  
 Highlight numbers that don't match  
 """* for number in self.numbers:  
 if tuple(number.position.grid\_square\_pos) in self.wrong\_numbers:  
 number.highlight\_error()  
  
 def check\_correctness(self) -> None:  
 *"""  
 Load current arrows and numbers to core class and evaluate correctness  
 """* for arrow in self.arrows:  
 Core.arrows[arrow.arrow\_set][arrow.arrow\_num] = arrow.direction  
 for number in self.numbers:  
 Core.numbers[number.col][number.row] = number.value  
 self.wrong\_numbers = Core.evaluate\_correctness()  
  
 def get\_arrow(self**,** pos: tuple[int**,** int]) -> ArrowGridSquare:  
 *"""  
 Get arrow by pixel position  
  
 :return: arrow that is under given position  
 :param pos: position to get arrow by  
 """* for arrow in self.arrows:  
 if arrow.rect.collidepoint(pos):  
 return arrow  
  
 def set\_arrow\_image(self**,** image: pygame.Surface | None**,** direction: tuple[int**,** int] | None**,** highlight\_color: tuple[int]) -> None:  
 *"""  
 Set arrow image for selected arrow square if any  
  
 :return: None  
 :param image: New image to set  
 :param direction: direcion of arrow on image to set direction attribute  
 :param highlight\_color: color that object was highlighted to restore it  
 """* for arrow in self.arrows:  
 if arrow.selected:  
 arrow.set\_image(image**,** direction)  
 arrow.select(highlight\_color)  
  
 def check\_arrow\_selection(self**,** mouse\_pos: tuple[int**,** int]) -> bool:  
 *"""  
 Check if arrow was selected  
  
 :return: True if arrow under current mouse position is selected  
 :param mouse\_pos: Mouse position in pixel coordinates  
 """* for arrow in self.arrows:  
 if arrow.rect.collidepoint(mouse\_pos):  
 return True  
 return False  
  
 def check\_number\_selection(self**,** mouse\_pos: tuple[int**,** int]) -> bool:  
 *"""  
 Check if number was selected  
  
 :return: True if number under current mouse position is selected  
 :param mouse\_pos: Mouse position in pixel coordinates  
 """* for number in self.numbers:  
 if number.rect.collidepoint(mouse\_pos):  
 return True  
 return False  
  
 def handle\_arrow\_selection(self**,** mouse\_pos: tuple[int**,** int]) -> None:  
 *"""  
 Select arrow and numbers it points to  
  
 :return: None  
 :param mouse\_pos: Current mouse position  
 """* # deselect previously selected numbers and its arrows  
 if isinstance(self.currently\_selected**,** NumberGridSquare):  
 pointing\_arrows = Core.get\_pointings(Vector2(self.currently\_selected.col**,** self.currently\_selected.row))  
 for arrow in self.arrows:  
 if (arrow.arrow\_set**,** arrow.arrow\_num) in pointing\_arrows:  
 arrow.deselect(colors.highlighted\_yellow)  
 self.currently\_selected.deselect(colors.highlighted\_blue)  
 self.currently\_selected = None  
  
 # deselect previous arrow and its spanned numbers  
 if isinstance(self.currently\_selected**,** ArrowGridSquare):  
 for number in self.numbers:  
 if number.selected:  
 number.deselect(colors.highlighted\_yellow)  
 self.currently\_selected.deselect(colors.highlighted\_blue)  
  
 # get currently selected arrows  
 curr\_arrow = None  
 for arrow in self.arrows:  
 if arrow.rect.collidepoint(mouse\_pos):  
 curr\_arrow = arrow  
 break  
  
 # select current arrow and spanned numbers  
 if curr\_arrow != self.currently\_selected:  
 pointed\_numbers = Core.get\_span(curr\_arrow.position.grid\_square\_pos**,** curr\_arrow.direction)  
 for number in self.numbers:  
 if (number.col**,** number.row) in pointed\_numbers:  
 number.select(colors.highlighted\_yellow)  
 curr\_arrow.select(colors.highlighted\_blue)  
 self.currently\_selected = curr\_arrow  
 else:  
 self.currently\_selected = None  
  
 def handle\_number\_selection(self**,** mouse\_pos: tuple[int**,** int]):  
 *"""  
 Select number and arrows that point to it  
  
 :return: None  
 :param mouse\_pos: Current mouse position  
 """* # deselect previously selected arrow and numbers that it points to  
 if isinstance(self.currently\_selected**,** ArrowGridSquare):  
 pointing\_numbers = Core.get\_span(self.currently\_selected.position.grid\_square\_pos**,** self.currently\_selected.direction)  
 for number in self.numbers:  
 if (number.col**,** number.row) in pointing\_numbers:  
 number.deselect(colors.highlighted\_yellow)  
 self.currently\_selected.deselect(colors.highlighted\_blue)  
 self.currently\_selected = None  
  
 # deselect previous number and all selected arrows if there is  
 if isinstance(self.currently\_selected**,** NumberGridSquare):  
 for arrow in self.arrows:  
 if arrow.selected:  
 arrow.deselect(colors.highlighted\_yellow)  
 self.currently\_selected.deselect(colors.highlighted\_blue)  
  
 # get previously selected and currently selected numbers  
 curr\_num = None  
 for number in self.numbers:  
 if number.rect.collidepoint(mouse\_pos):  
 curr\_num = number  
 break  
  
 # select current number and arrows if there is  
 for arrow in self.arrows:  
 Core.arrows[arrow.arrow\_set][arrow.arrow\_num] = arrow.direction  
 if curr\_num != self.currently\_selected:  
 pointed\_arrows = Core.get\_pointings(Vector2(curr\_num.col**,** curr\_num.row))  
 for arrow in self.arrows:  
 if (arrow.arrow\_set**,** arrow.arrow\_num) in pointed\_arrows:  
 arrow.select(colors.highlighted\_yellow)  
 curr\_num.select(colors.highlighted\_blue)  
 self.currently\_selected = curr\_num  
 else:  
 self.currently\_selected = None  
  
 def draw(self) -> None:  
 *"""  
 Draw object to given surface  
 """* # draw numbers grid squares  
 for number in self.numbers:  
 number.draw()  
  
 # draw arrows grid squares  
 for arrow in self.arrows:  
 arrow.draw()  
  
 # draw frame  
 pygame.draw.rect(Screen.surface**,** self.frame\_color**,** self.frame\_rect**,** self.frame\_width)

number.py

import pygame  
  
from control import colors  
from control.settings import Settings  
  
  
class Number:  
 *"""Class for number image to render on game board"""* def \_\_init\_\_(self**,** value: int):  
 *"""  
 :return: None  
 :param value: Numeric value of number object  
 """* # get text font and related attributes  
 self.color = colors.numbers  
 self.font = pygame.font.SysFont('BAUHS93'**,** Settings.number\_size)  
  
 # render image  
 self.image = self.font.render(str(value)**,** True**,** self.color**,** colors.background)