

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 ПІ-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

Напрямок "ПЗ"

Курс 1 Група ПП-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	02.02.22	
2.	Підготовка ТЗ	08.02.22	
3.	Пошук та вивчення літератури з питань курсової роботи	15.02.22	
4.	Розробка сценарію роботи програми	29.02.22	
6.	Узгодження сценарію роботи програми з керівником	05.03.22	
5.	Розробка (вибір) алгоритму рішення задачі	19.03.22	
6.	Узгодження алгоритму з керівником	14.04.22	
7.	Узгодження з керівником інтерфейсу користувача	01.05.22	
8.	Розробка програмного забезпечення	13.05.22	
9.	Налагодження розрахункової частини програми	20.05.22	
10.	Розробка та налагодження інтерфейсної частини програми	27.05.22	
11.	Узгодження з керівником набору тестів для контрольного прикладу	01.06.22	
12.	Тестування програми	04.06.22	
13.	Підготовка пояснювальної записки	11.06.22	
14.	Здача курсової роботи на перевірку	12.06.22	
15.	Захист курсової роботи	15.06.22	

Студент _____
(підпис)

Керівник _____
(підпис)

" ____ " _____ 20__ р.

Муха І. П.
(прізвище, ім'я, по батькові)

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 100 сторінок, 42 рисунки, 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-орієнтованого аналізу та архітектури проекту.

Виконана програмна реалізація гри "Стрілки".

ЗМІСТ

ВСТУП.....	6
1 ПОСТАНОВКА ЗАДАЧІ	7
2 ТЕОРЕТИЧНІ ВІДОМОСТІ	8
2.1 Основні правила гри.....	8
2.2 Інтерфейс гравця.....	8
2.3 Задача гравця.....	8
2.4 Діаграма прецедентів	9
3 ОПИС АЛГОРИТМІВ	10
3.1 Генерація стрілок.....	10
3.2 Знаходження можливих напрямків стрілки.....	10
3.3 Знаходження позиції стрілки.....	11
3.4 Знаходження чисел до яких напрямлена стрілка	11
3.5 Знаходження стрілок, що напрямлені до заданого числа.....	12
3.6 Перевірка заповнення стрілок на правильність.....	12
3.7 Генерація чисел.....	12
4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	13
4.1 Діаграма класів програмного забезпечення.....	13
4.2 Опис методів частин програмного забезпечення	16
4.2.1 Стандартні методи.....	16
4.2.2 Користувачькі методи.....	17
5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	22
5.1 План тестування.....	22
5.2 Приклади тестування.....	23
6 ІНСТРУКЦІЯ КОРИСТУВАЧА	53
6.1 Відкриття програми.....	53
6.2 Складові ігрового поля.....	53
6.3 Наповнення ігрового поля	54
6.4 Завершення гри	57
6.5 Завдання гравця	61
6.6 Системні вимоги	61

ВИСНОВКИ.....	63
ПЕРЕЛІК ПОСИЛАНЬ	64
ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ	65
ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ	68
arrows_game.py	69
collide_rect_evaluator.py	75
core.py	76
colors.py	80
grid_position.py	80
screen.py	81
settings.py	82
states.py	82
time_control.py	83
add_arrows_button.py	83
button.py	84
delete_arrow_button.py	85
end_session_button.py	86
gen_new_board_button.py	86
arrow_grid_square.py	87
grid_square.py	88
number_grid_square.py	89
correct_message.py	90
message.py	91
start_message.py	91
wrong_message.py	92
arrow.py	92
board.py	93
number.py	99

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 ІТ-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».

1 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 направлених 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 завершити гру.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Основні правила гри

Ігрове поле, розміром 8 x 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).

2.2 Інтерфейс гравця

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

2.3 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.

2.4 Діаграма прецендентів

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

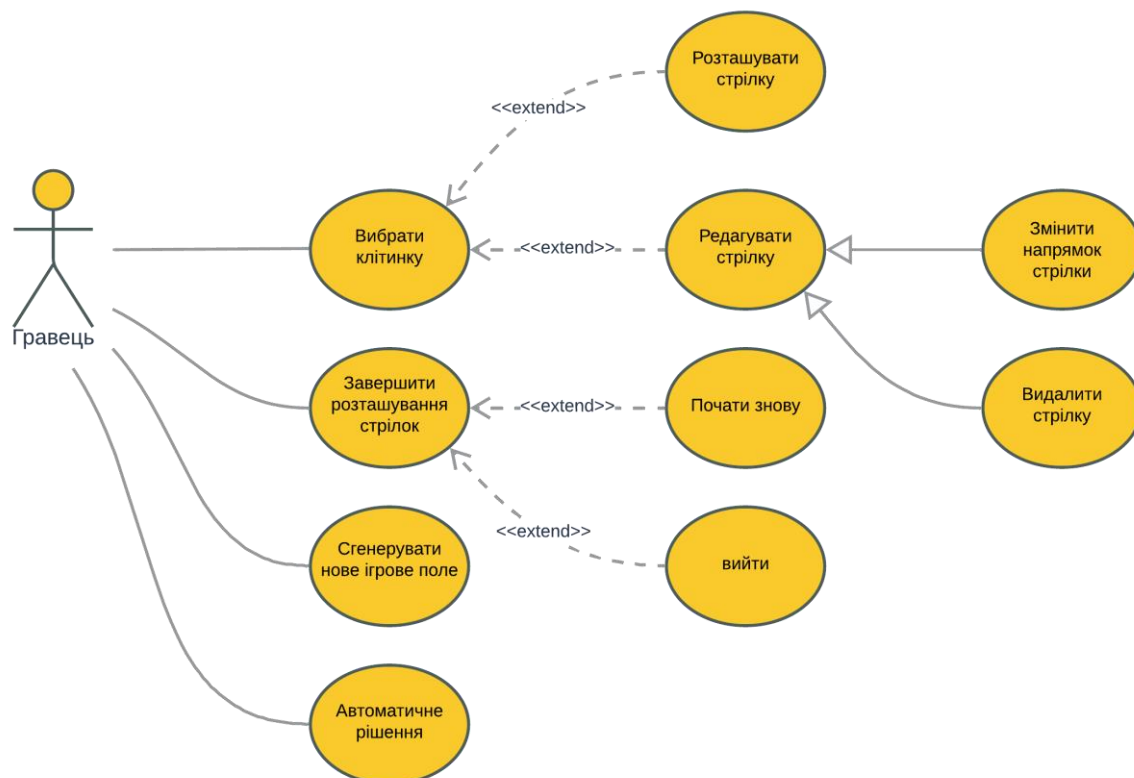


Рисунок 2.1 – Діаграма прецендентів

3 ОПИС АЛГОРИТМІВ

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

Таблиця 3.1 – Основні змінні та їхні призначення

Змінна	Призначення
<i>arrow_directions</i>	Всі напрямки стрілок
<i>arrow_sets</i>	Словник з ключами як напрямки розташування стрілок та можливих напрямків стрілки для цього розташування
<i>forbidden_directions</i>	Неможливі напрямки стрілок у кутах сітки
<i>arrows</i>	Розташування та напрямки всіх стрілок
<i>numbers</i>	Розташування та значення числ сітки
<i>position</i>	Розташування відносно ігрового поля
<i>arrow</i>	Напрямок стрілки
<i>grid_square</i>	Вектор розташування відносно ігрового поля

3.1 Генерація стрілок

1. `arrows = {}`
2. `for arrow_set in arrow_sets:`
 - 2.1. `grid_count = grid count along current set`
 - 2.2. `for i from 0 to grid_count:`
 - 2.2.1. `possible_directions = get_possible_directions(arrow_set, i)`
 - 2.2.2. `choice = random among possible_directions`
 - 2.2.3. `add choice to arrows[arrow_set]`

3.2 Знаходження можливих напрямків стрілки

1. `grid_count = grid count along current set`
2. `possible_directions = arrow_sets[arrow_set]`

3. if arrow_num == 0:
 - 3.1. remove forbidden_direction[arrow_set][0] from possible_directions
4. else if arrow_num == grid_count - 1:
 - 4.1. remove forbidden_direction[arrow_set][1] from possible_directions

3.3 Знаходження позиції стрілки

1. if arrows_set_direction == (0, -1):
 - 1.1. position = (arrow_num, -1)
2. else if arrows_set_direction == (1, 0):
 - 2.1. position = grid_count.x, arrow_num
3. else if arrows_set_direction == (-1, 0):
 - 3.1. position = -1, arrow_num
4. else if arrows_set_direction == (0, 1):
 - 4.1. position = arrow_num, grid_count.y
5. return position

3.4 Знаходження чисел до яких напрямлена стрілка

1. if arrow is empty:
 - return
2. grid_squares = []
3. grid_square = position
4. while True:
 - 4.1. grid_square += arrow
 - 4.2. if grid_square not inside gaming board:
 - 4.2.1. break
 - 4.3. add grid_square to grid_squares
5. return grid_squares

3.5 Знаходження стрілок, що напрямлені до заданого числа

1. result = []
2. for arrow_set_direction and arrow_set in arrows:
 - 2.1. for arrow_num and arrow in arrow_set:
 - 2.1.1. position = get_position(arrow_set_direction, arrow_num)
 - 2.1.2. if arrow not empty:
 - 2.1.2.1. if grid_square is in get_span(position, arrow):
 - 2.1.2.1.1. arrow_set_direction and arrow_num add to result
3. return result

3.6 Перевірка заповнення стрілок на правильність

1. wrong_numbers = []
2. for number in numbers:
 - 2.1. if number != count_pointings(number.position)
 - 2.1.1. add number to wrong_numbers
3. return wrong_numbers

3.7 Генерація чисел

1. arrows = gen_arrows()
2. for col from 0 to grid_count.x:
 - 2.1. for row from 0 to grid_count.y:
 - 2.1.1. add length of count_pointings(col, row) to numbers

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 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.

– **AutoSolveButton**

Клас кнопки для автоматичного вирішення задачі.

– **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.

- **CollideRectEvaluator**

Клас для обчислення прямокутників зіткнення для повідомлень.

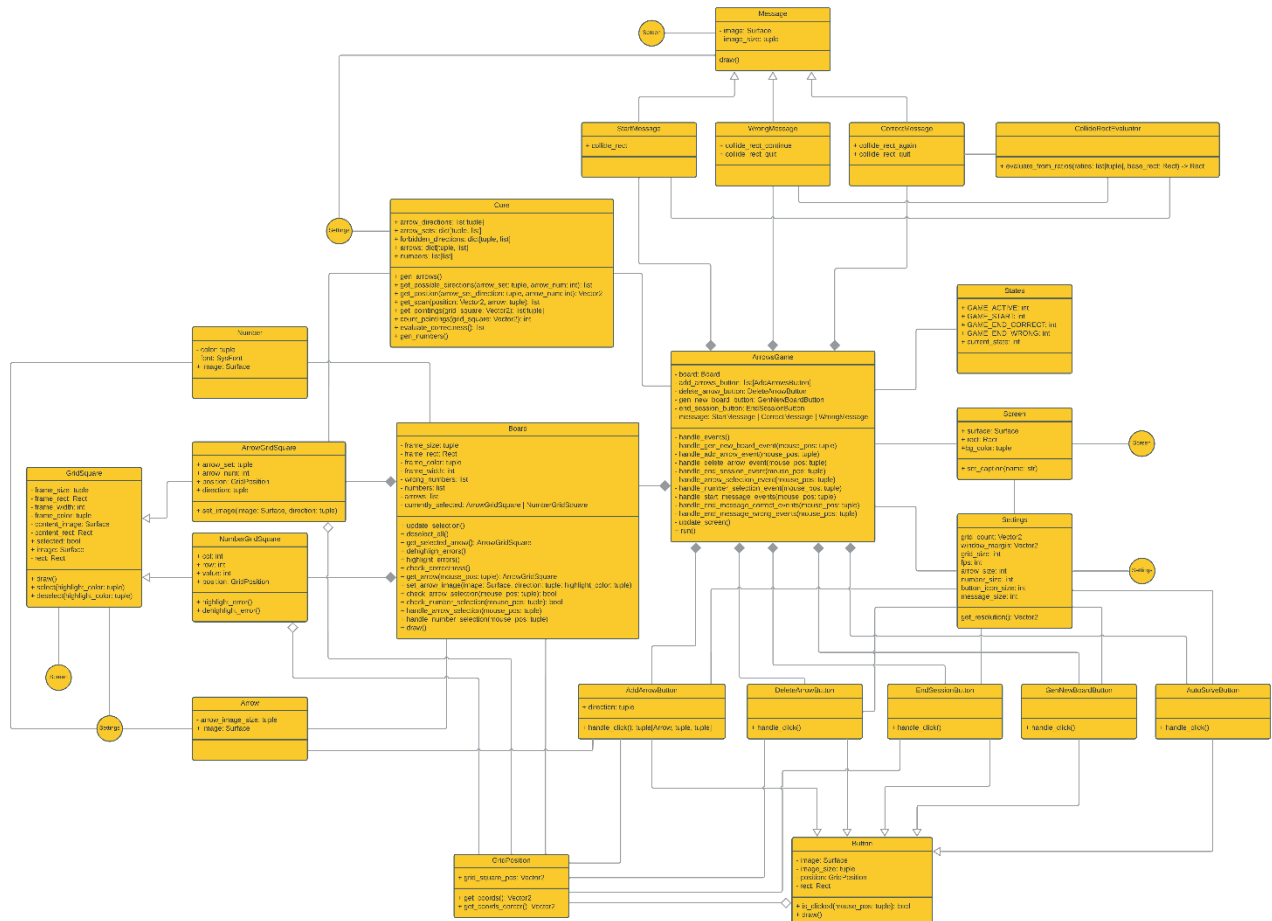


Рисунок 4.1 – Діаграма класів

4.2 Опис методів частин програмного забезпечення

4.2.1 Стандартні методи

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

Таблиця 4.1 – Стандартні методи

1 № п/п	2 Назва класу	3 Назва функції	4 Призначення функції	5 Опис вхідних параметрів	6 Опис вихідних параметрів	7 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 Constructor	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 dimintions 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

4.2.2 Користувацькі методи

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

Таблиця 4.2 – Користувацькі методи

№ п/п	Назва класу	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 Evaluator	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 - direction 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
67	ArrowsGame	handle_auto_solve_event	Automatically solve the puzzle	-	-	arrows_game
68	Board	handle_auto_solve	Automatically solve the puzzle	-	-	utils.core

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 План тестування

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

- а) Тестування початкового екрану
 - 1) Натискання кнопки «старт»
- б) Тестування виділення чисел та стрілок
 - 1) Виділення стрілок з пустою клітинкою
 - 2) Виділення стрілок з заповненою клітинкою
 - 3) Виділення чисел
- в) Тестування розставлення стрілок
 - 1) Розставлення стрілки в пусту клітинку
 - 2) Зміна напрямку стрілки
 - 3) Видалення стрілки
- г) 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
- д) Тестування повідомлень у кінці гри
 - 1) Повернення до гри
 - 2) Вихід з гри
- е) Тестування кнопки автоматичного рішення
 - 1) Перевірка правильності рішення

5.2 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

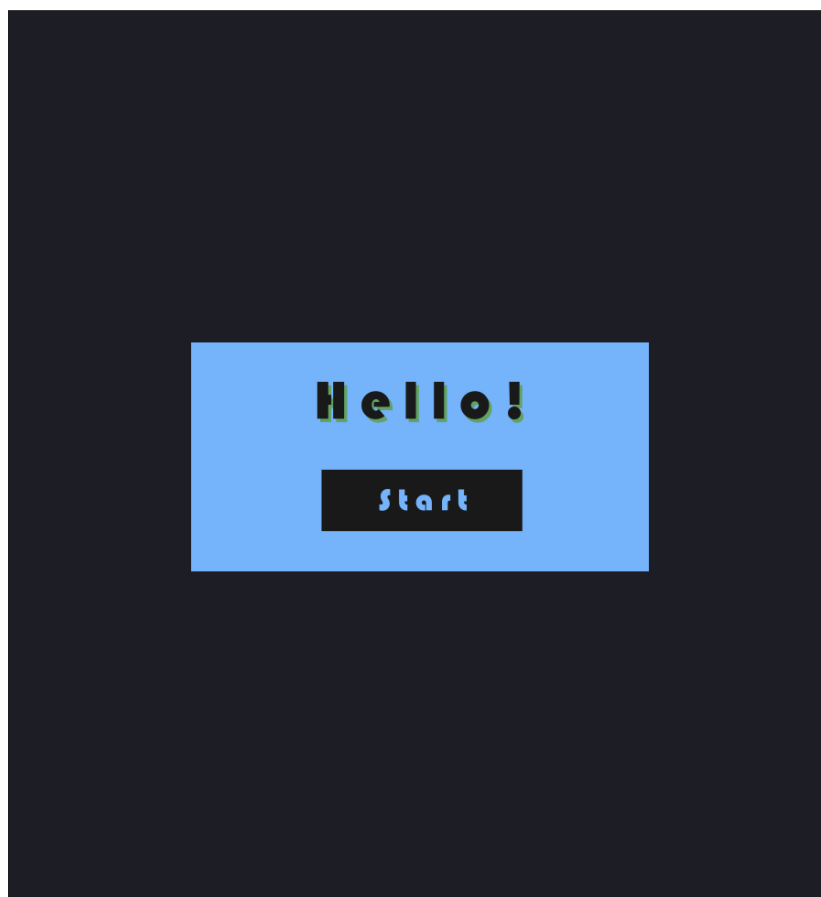
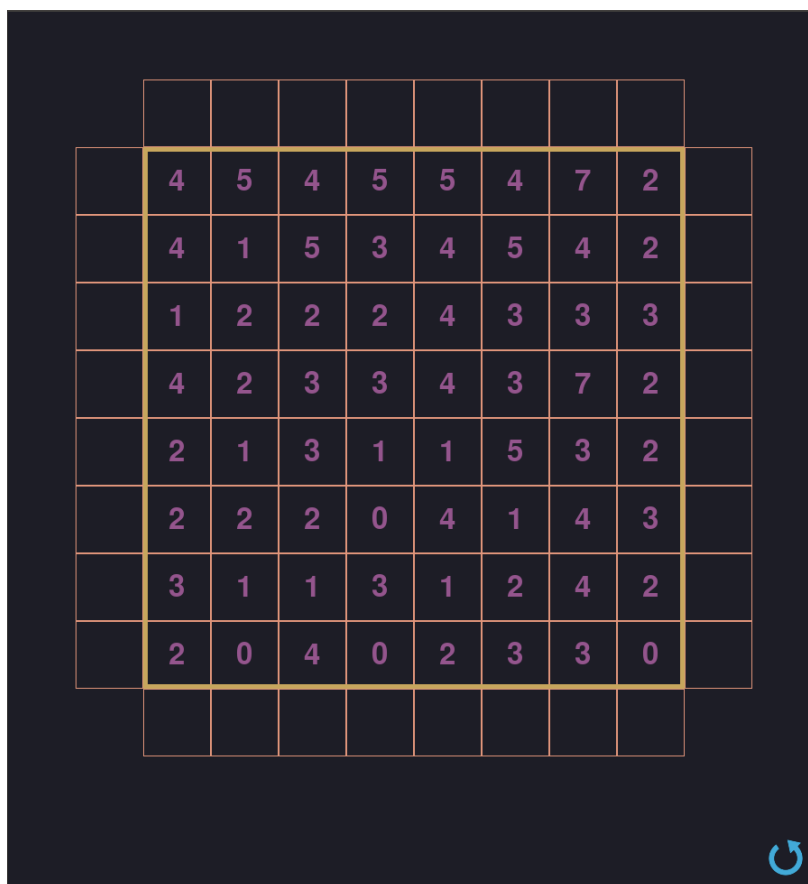


Рисунок 5.1 - до проведення тесту «Натискання кнопки «старт»»



	4	5	4	5	5	4	7	2	
	4	1	5	3	4	5	4	2	
	1	2	2	2	4	3	3	3	
	4	2	3	3	4	3	7	2	
	2	1	3	1	1	5	3	2	
	2	2	2	0	4	1	4	3	
	3	1	1	3	1	2	4	2	
	2	0	4	0	2	3	3	0	

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

	4	5	4	5	5	4	7	2	
	4	1	5	3	4	5	4	2	
	1	2	2	2	4	3	3	3	
	4	2	3	3	4	3	7	2	
	2	1	3	1	1	5	3	2	
	2	2	2	0	4	1	4	3	
	3	1	1	3	1	2	4	2	
	2	0	4	0	2	3	3	0	




Рисунок 1.3 - до проведення тесту «Виділення стрілок з пустою клітинкою»

	4	5	4	5	5	4	7	2	
	4	1	5	3	4	5	4	2	
	1	2	2	2	4	3	3	3	
	4	2	3	3	4	3	7	2	
	2	1	3	1	1	5	3	2	
	2	2	2	0	4	1	4	3	
	3	1	1	3	1	2	4	2	
	2	0	4	0	2	3	3	0	

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

	4	5	4	5	5	4	7	2
	4	1	5	3	4	5	4	2
	1	2	2	2	4	3	3	3
	4	2	3	3	4	3	7	2
	2	1	3	1	1	5	3	2
	2	2	2	0	4	1	4	3
	3	1	1	3	1	2	4	2
	2	0	4	0	2	3	3	0

↑ ↗
↻

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 заповненням 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

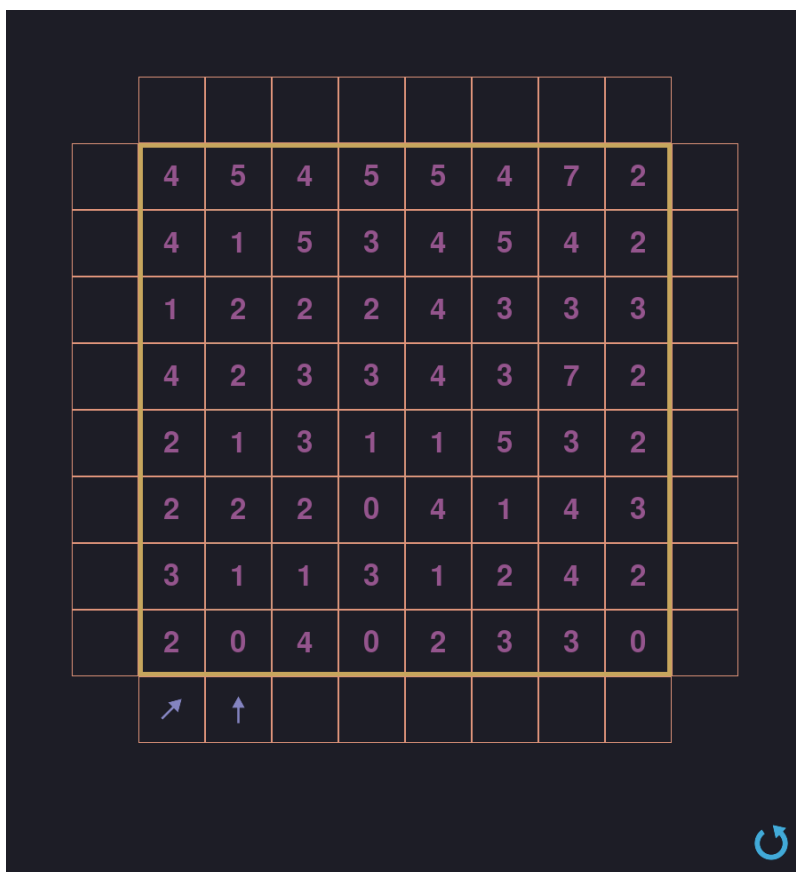


Рисунок 5.6 - до проведення тесту «Виділення стрілок з заповненою клітинкою»

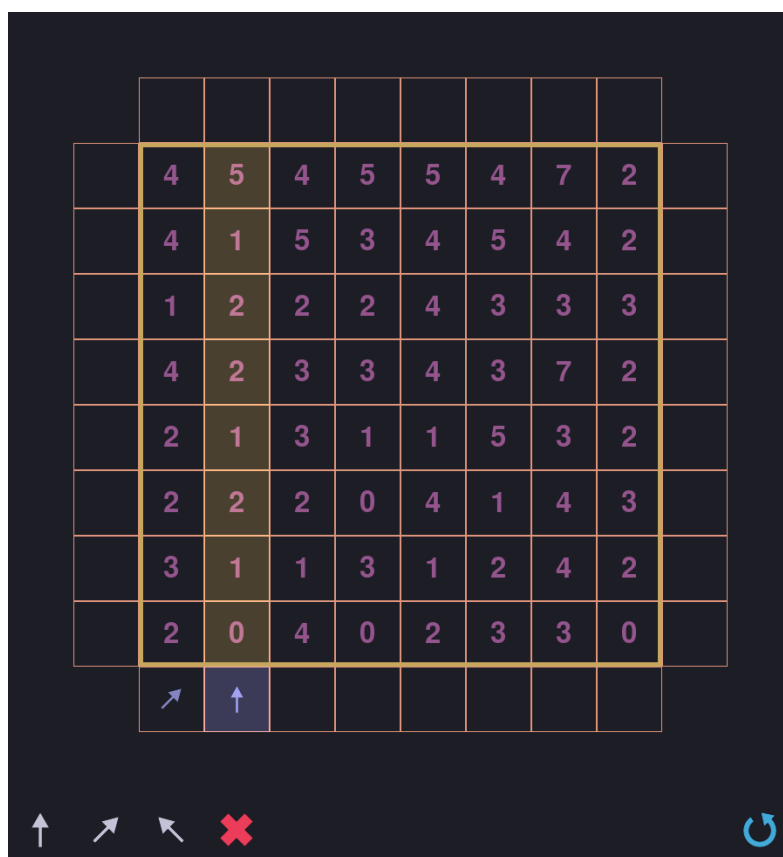


Рисунок 5.7 - після проведення тесту «Виділення стрілок з заповненою клітинкою» (виділення не діагональної стрілки)

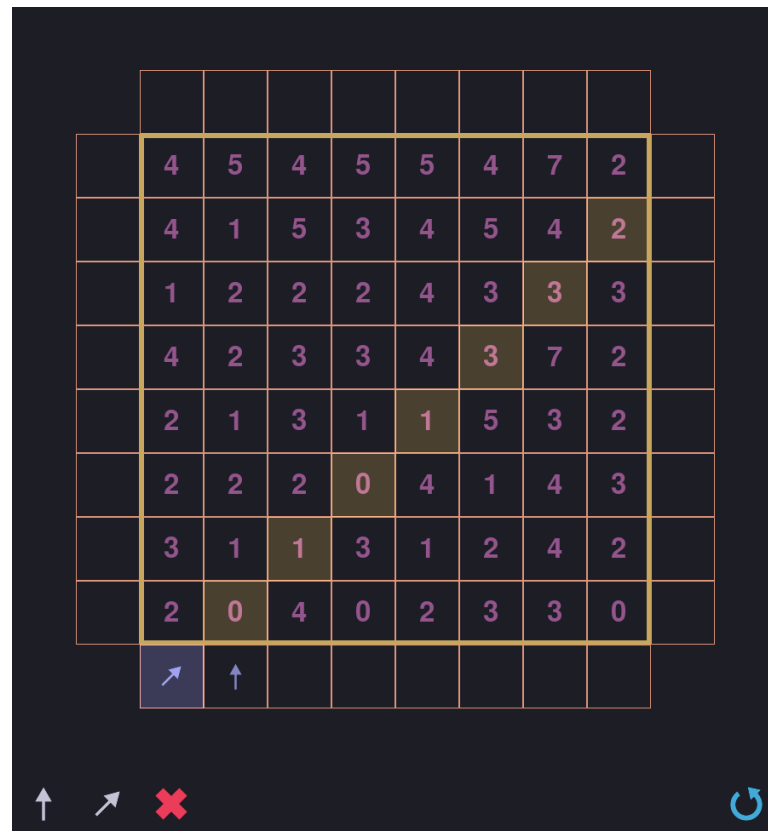


Рисунок 5.8 - після проведення тесту «Виділення стрілок з заповненою клітинкою» (виділення діагональної стрілки)

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 заповненням 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

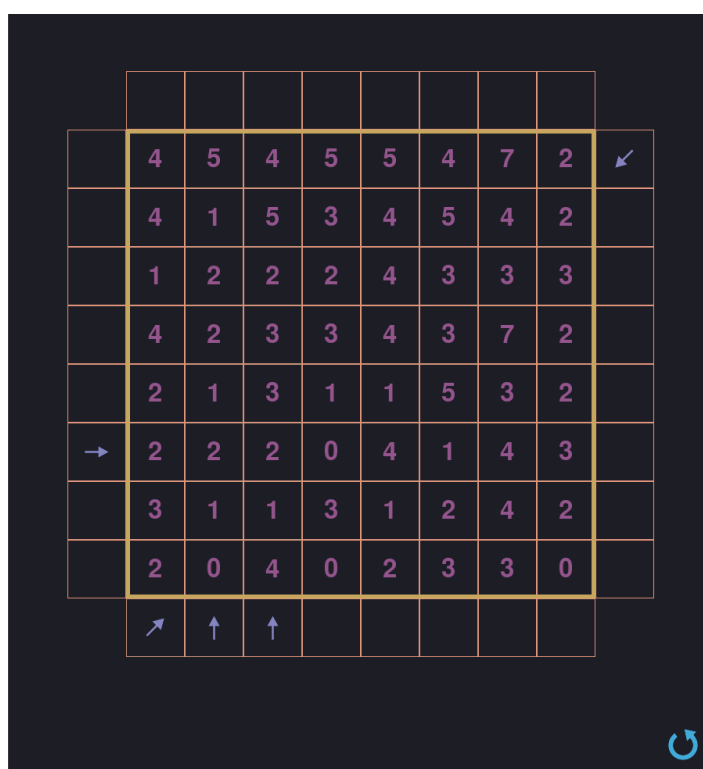


Рисунок 5.9 - до проведення тесту «Виділення чисел»

	4	5	4	5	5	4	7	2	↖
	4	1	5	3	4	5	4	2	
	1	2	2	2	4	3	3	3	
	4	2	3	3	4	3	7	2	
	2	1	3	1	1	5	3	2	
→	2	2	2	0	4	1	4	3	
	3	1	1	3	1	2	4	2	
	2	0	4	0	2	3	3	0	
↗	↑	↑							



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 заповнення або пустим ігровим полем
Вхідні дані	Виділена пуста клітинка, де розташовуються 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

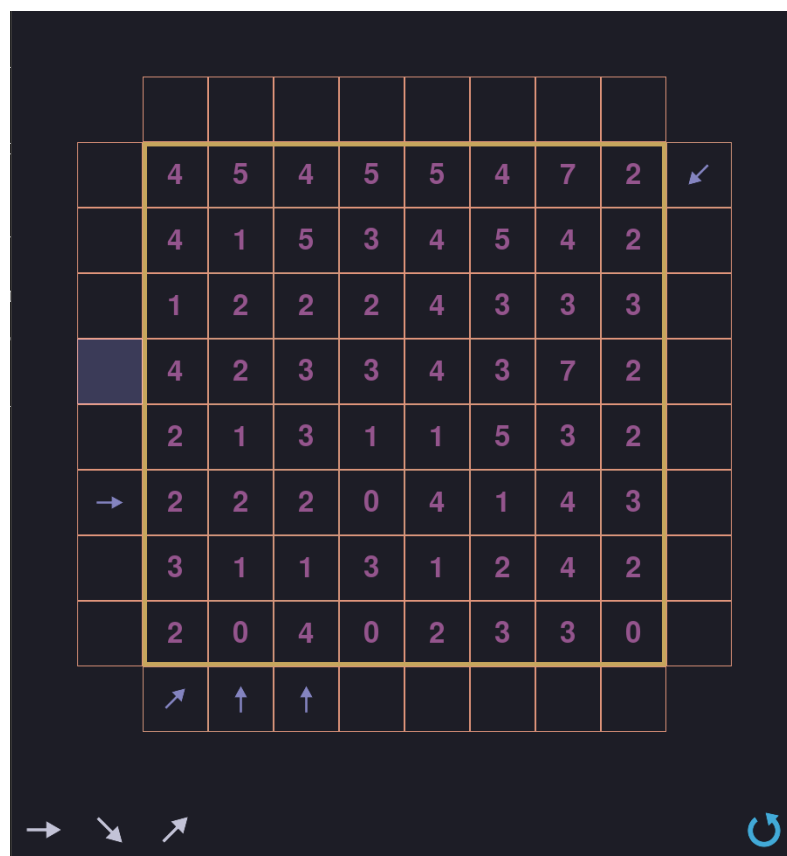


Рисунок 5.11 - до проведення тесту «Розставлення стрілки в пусту клітинку»

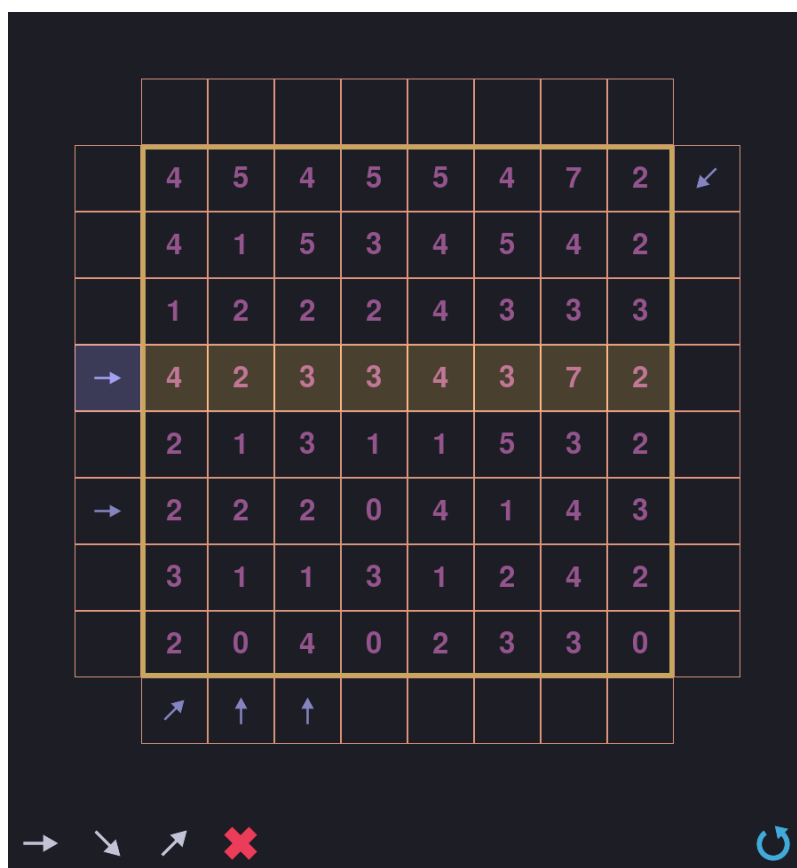


Рисунок 5.12 - після проведення тесту «Розставлення стрілки в пусту клітинку»

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 заповненням або пустим ігровим полем
Вхідні дані	Виділена заповнена клітинка, де розташовуються 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

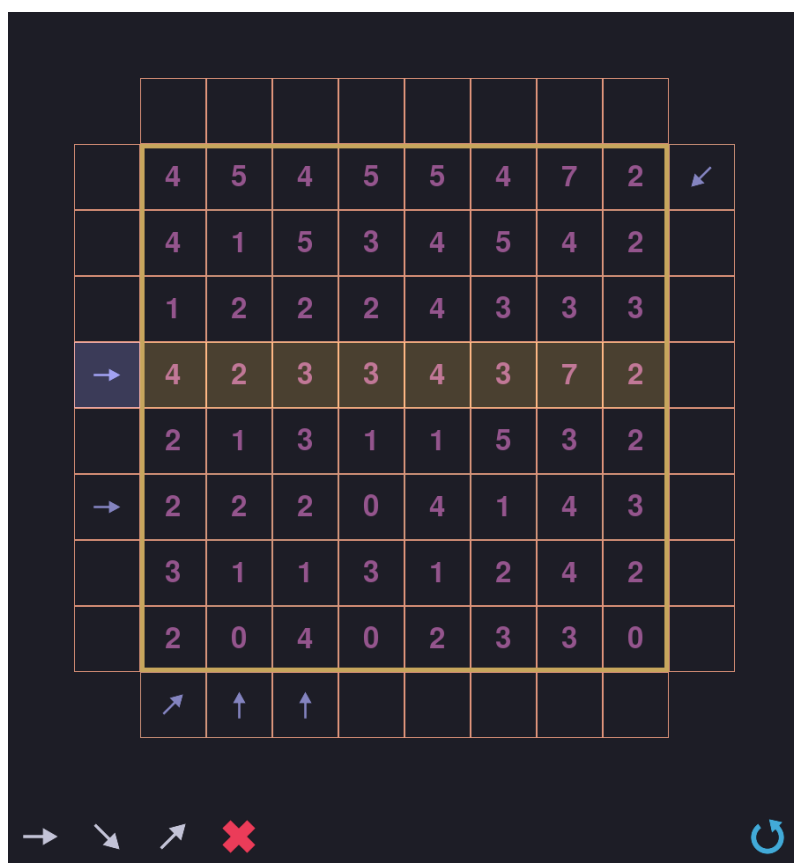


Рисунок 2.13 - до проведення тесту «Зміна напрямку стрілки»

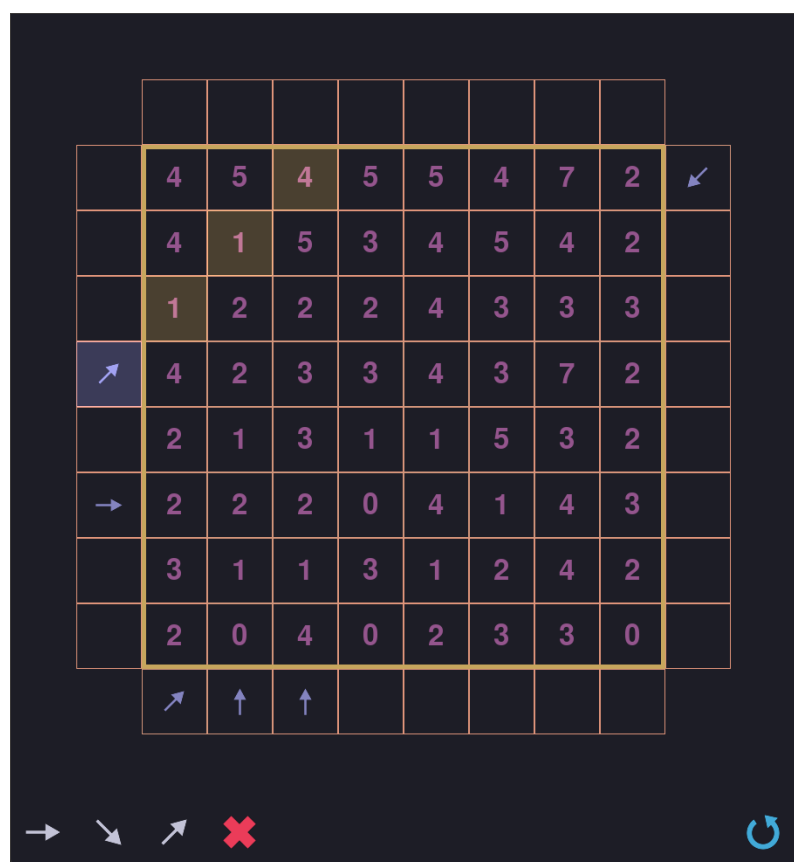


Рисунок 5.14 - після проведення тесту «Зміна напрямку стрілки»

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 розташовуються 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

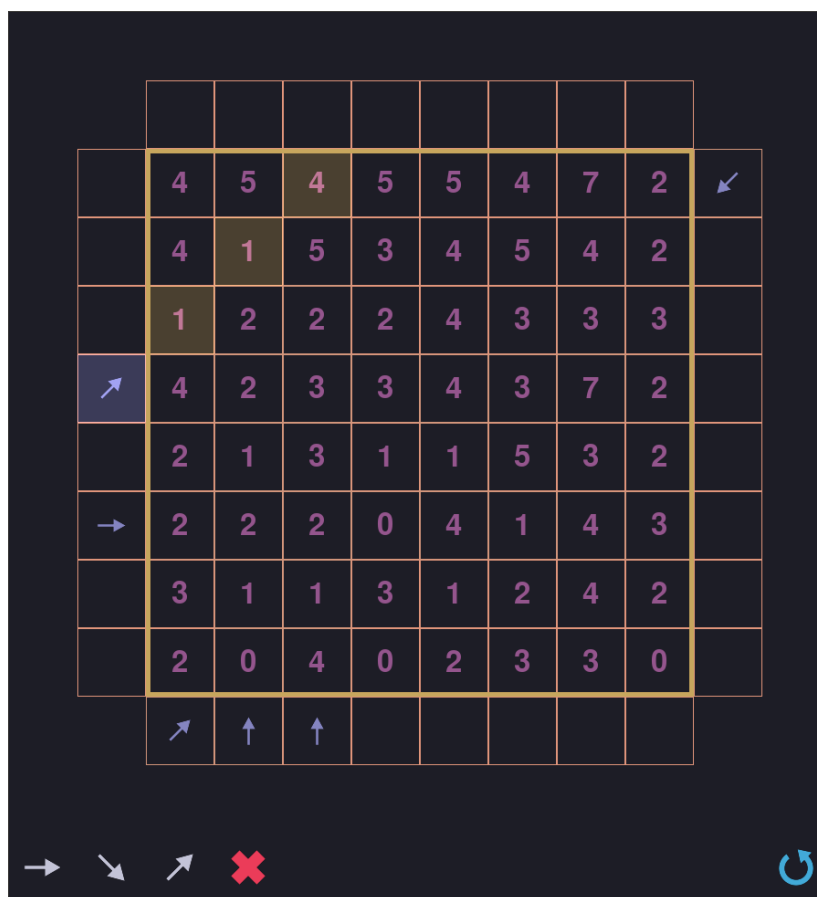


Рисунок 5.15 - до проведення тесту «Видалення стрілки»

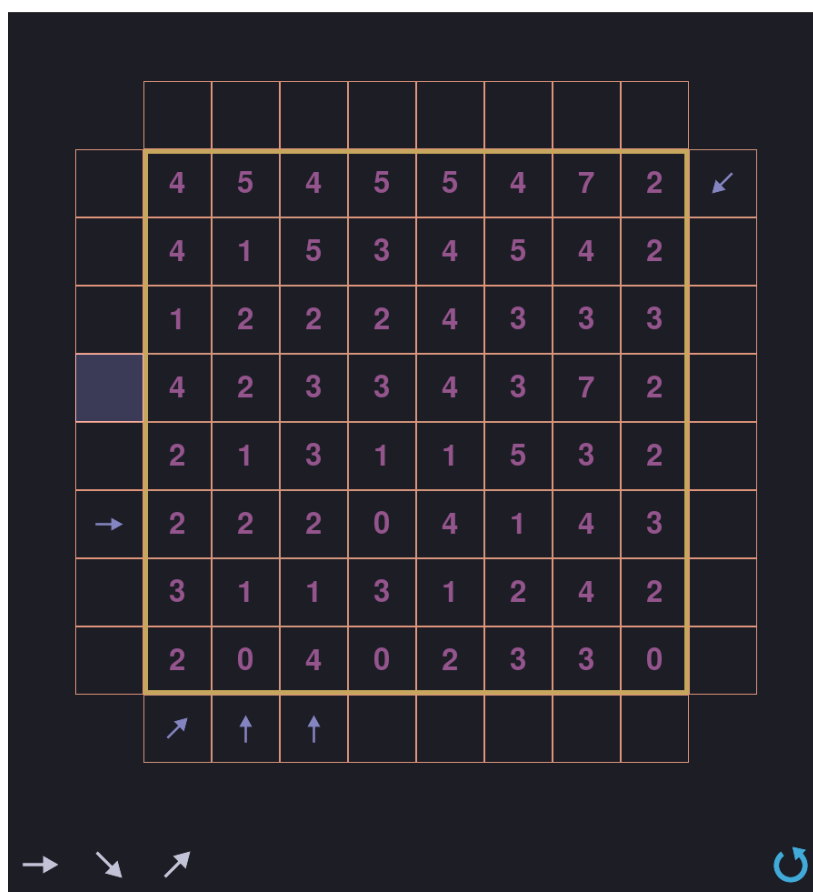


Рисунок 5.16 - після проведення тесту «Видалення стрілки»

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

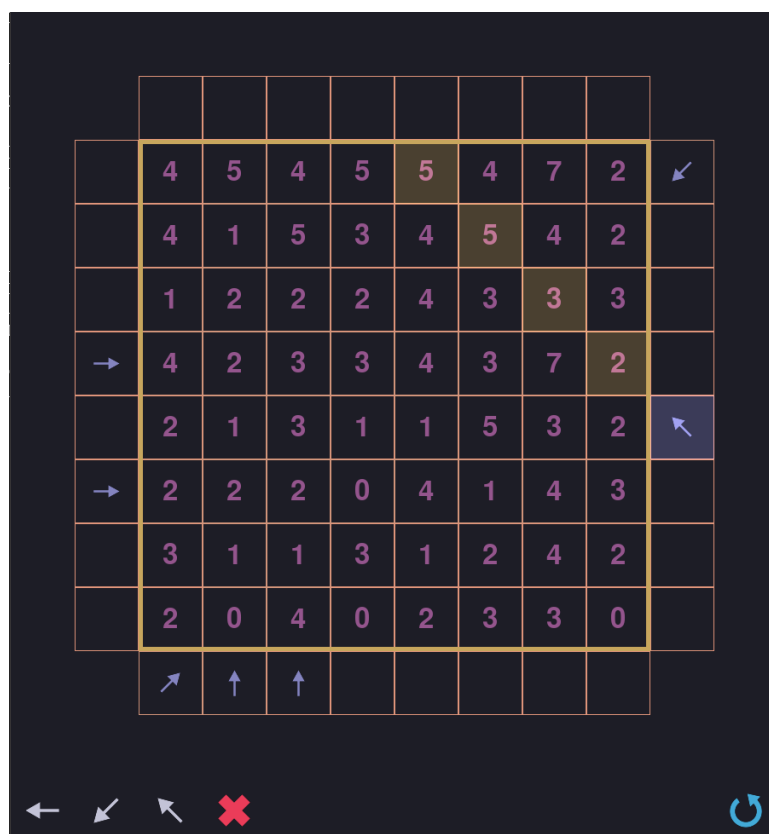


Рисунок 5.17 - до проведення тесту «Кнопка генерації нового поля з частково заповненим полем»

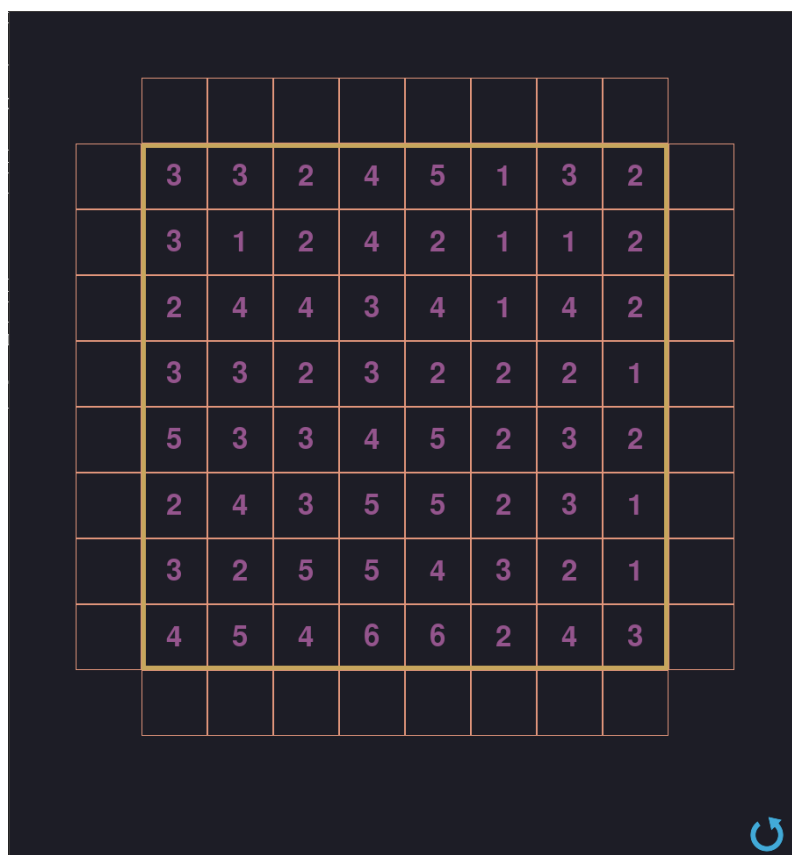


Рисунок 5.18 - після проведення тесту «Кнопка генерації нового поля з частково заповненим полем»

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 співпадає 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

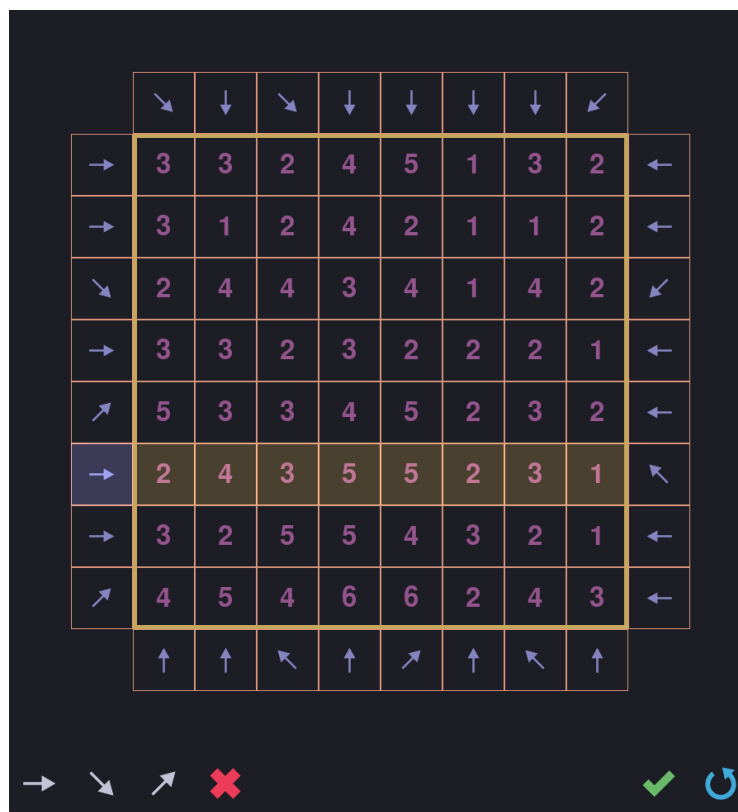


Рисунок 5.19 - до проведення тесту «Кнопка перевірки результатів з неправильним розташуванням стрілок»

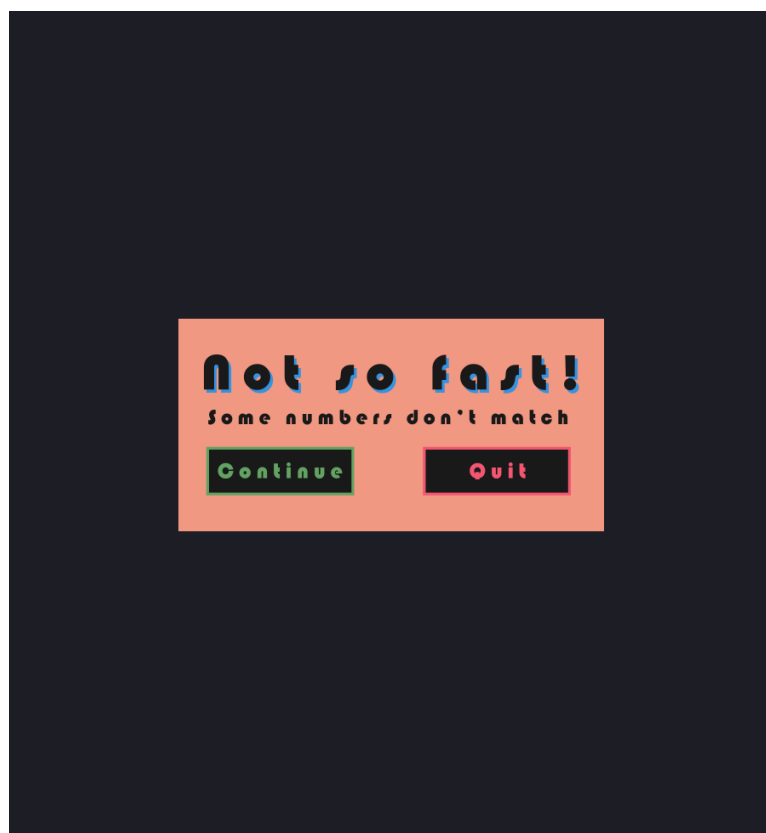


Рисунок 5.20 - після проведення тесту «Кнопка перевірки результатів з неправильним розташуванням стрілок»

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 направлені 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

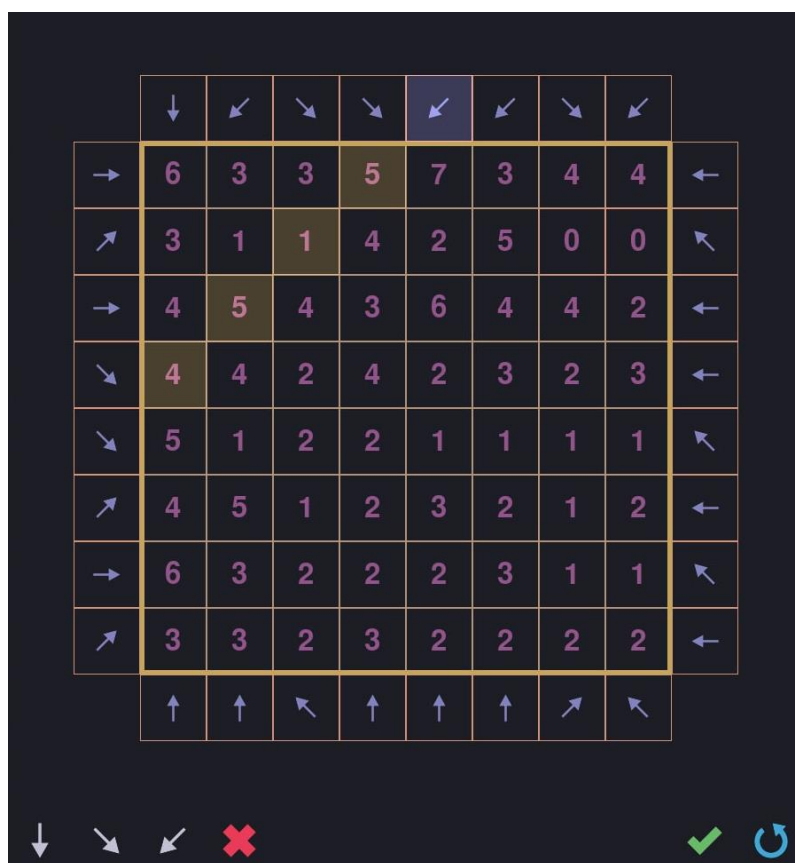


Рисунок 5.21 - до проведення тесту «Кнопка перевірки результатів з правильним розташуванням стрілок»



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 повідомленням про результат гри
Вхідні дані	-
Схема проведення тесту	Натискання на кнопку «continue» або «again»
Очікуваний результат	У випадку натискання кнопки «continue» - незмінене поле з виділенням 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 виділенням 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)

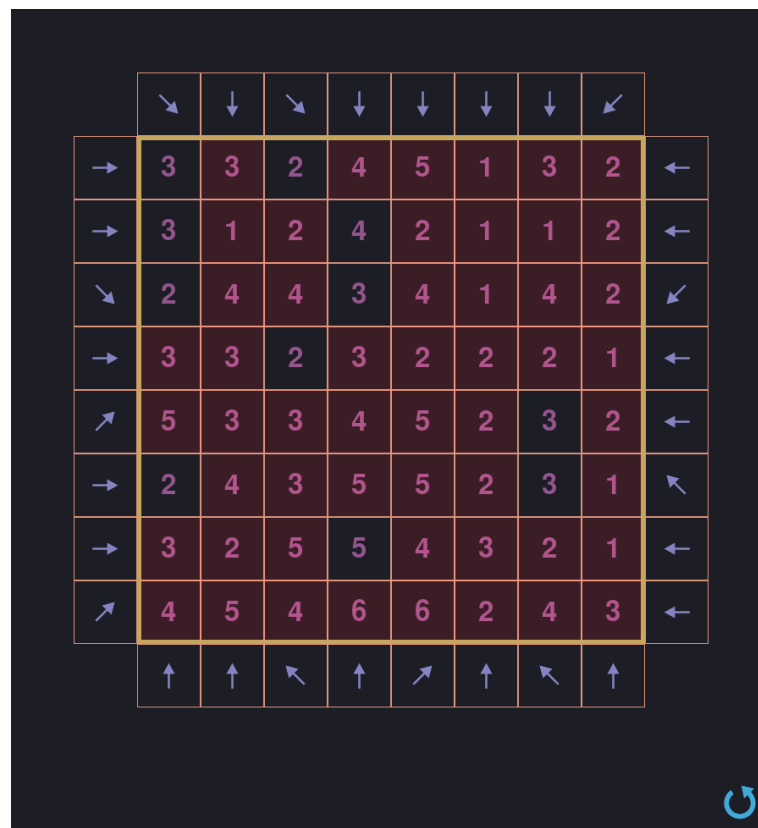


Рисунок 5.24 - після проведення тесту «Повернення до гри» (неправильний результат гри)

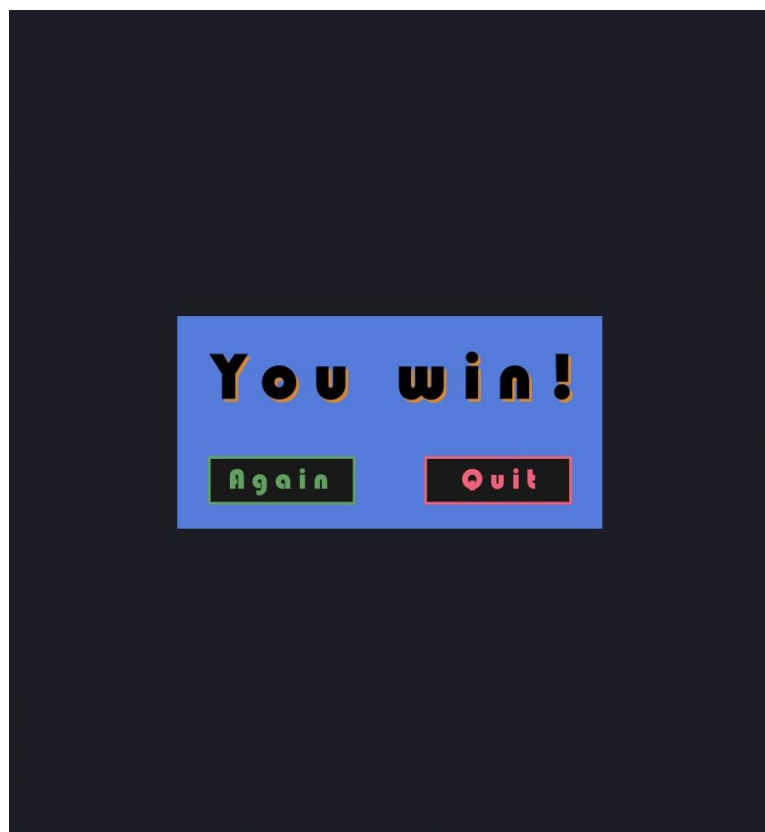


Рисунок 5.25 - до проведення тесту «Повернення до гри» (правильний результат гри)

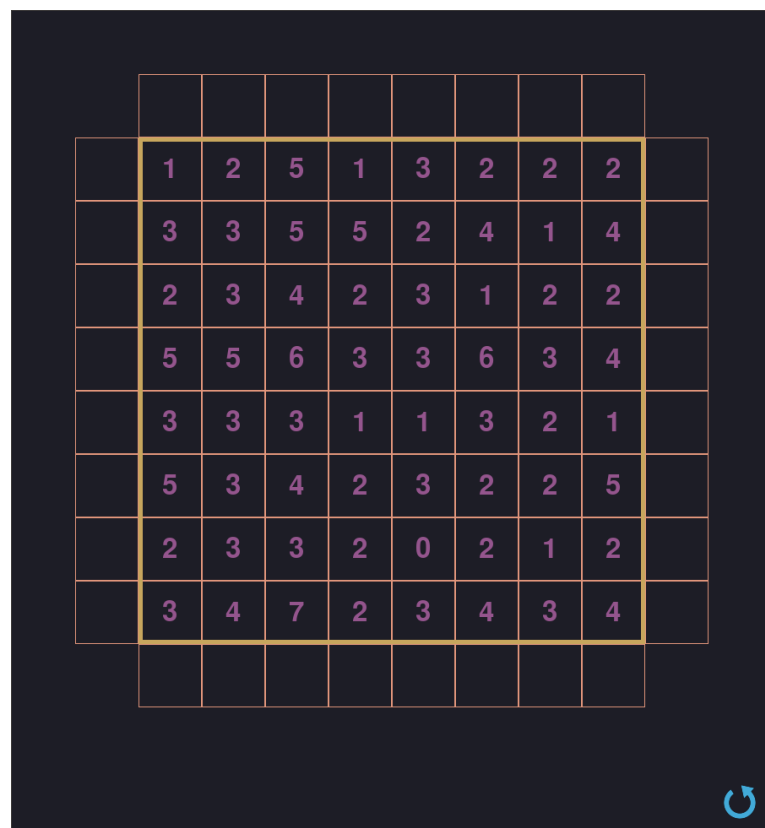


Рисунок 5.26 - після проведення тесту «Повернення до гри» (правильний результат гри)

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 повідомленням 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

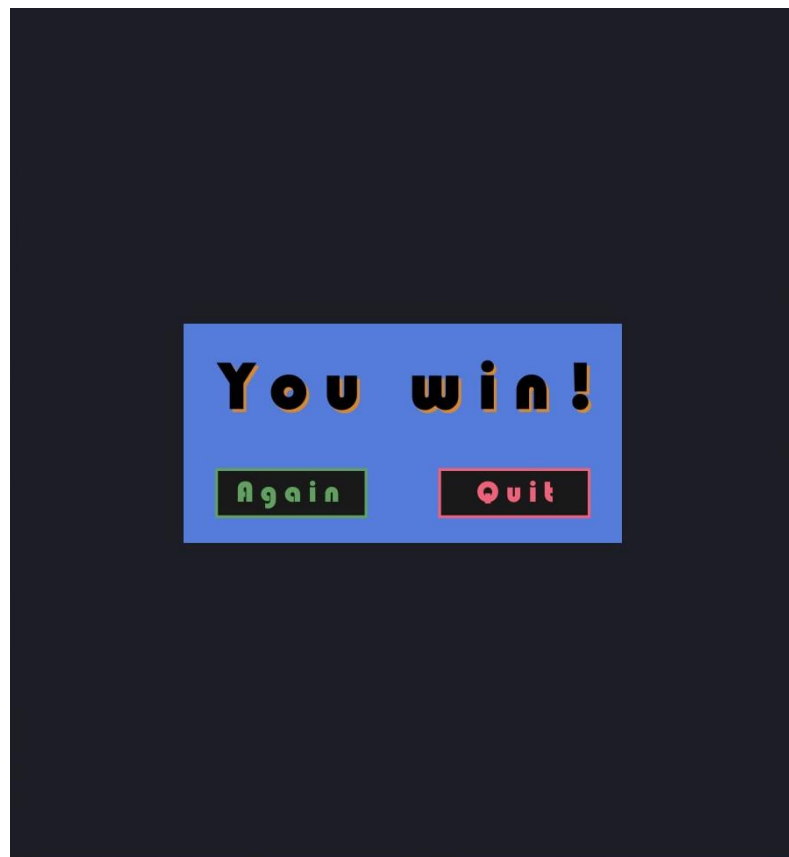


Рисунок 5.27 - до проведення тесту «Вихід з гри» (правильний результат гри)



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

Таблиця 5.13 - Приклад роботи під час натискання кнопки автоматичного вирішення

Мета тесту	Перевірити правильність автоматичного рішення
Початковий стан програми	Активна гра
Вхідні дані	-
Схема проведення тесту	Натискання на кнопку «auto»
Очікуваний результат	Заповнене поле
Стан програми після проведення випробувань	Заповнене поле з правильним розташуванням стрілок

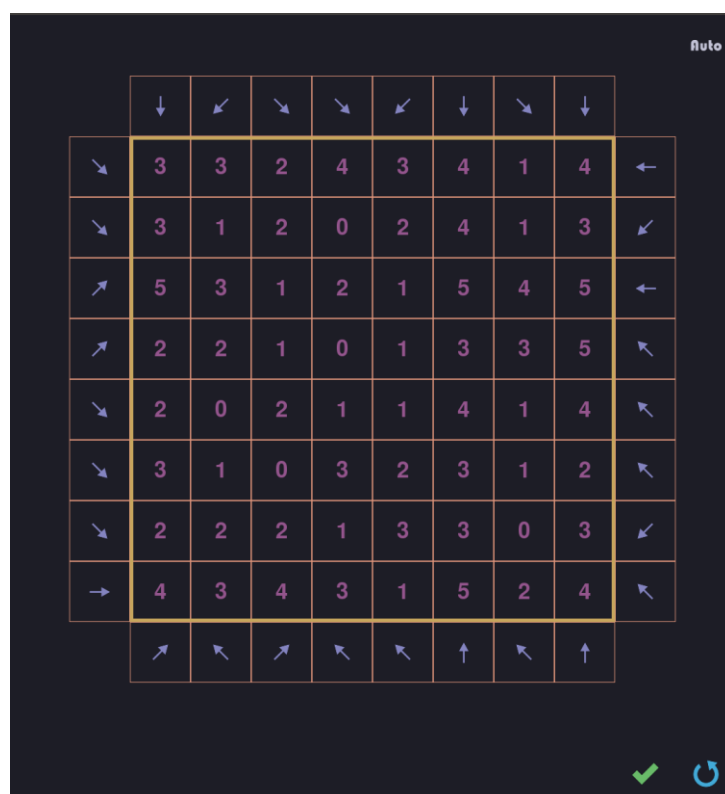


Рисунок 5.29 – ігрове поле після натискання кнопки «auto»

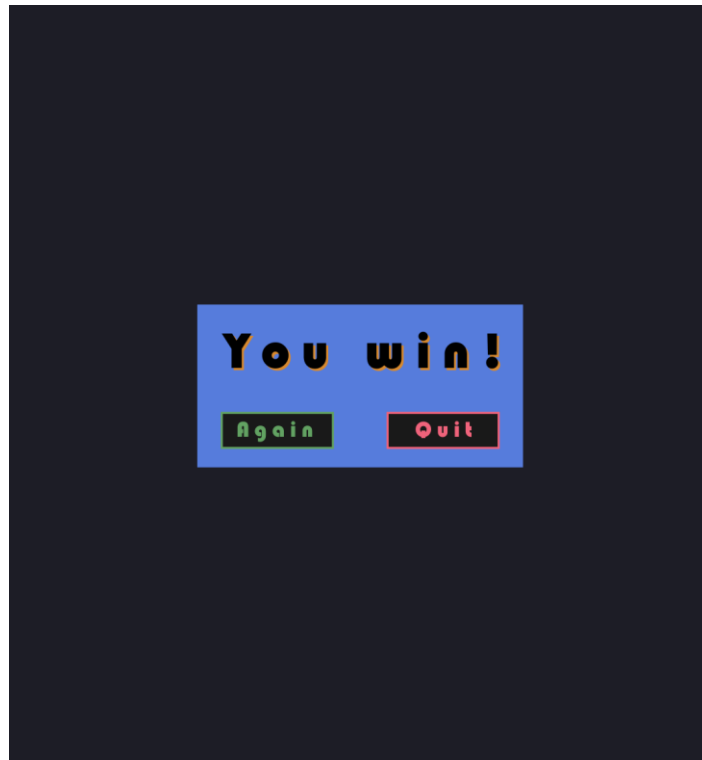


Рисунок 5.30 – Автоматичне рішення правильне

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 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 – Стартове меню

6.2 Складові ігрового поля

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

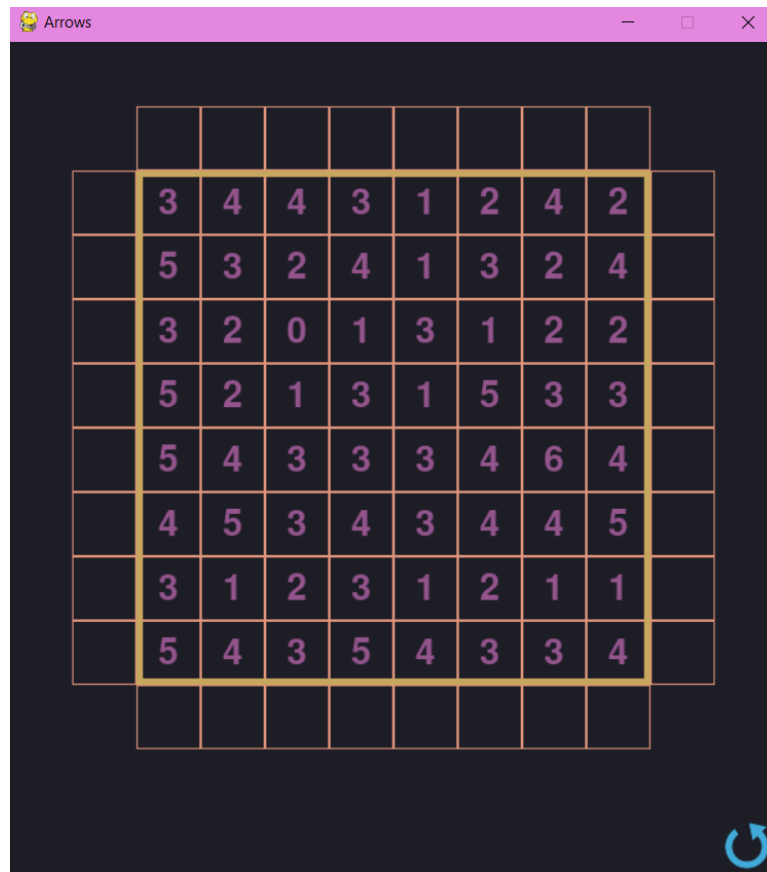


Рисунок 6.2 – Ігрове поле

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.

6.3 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*).

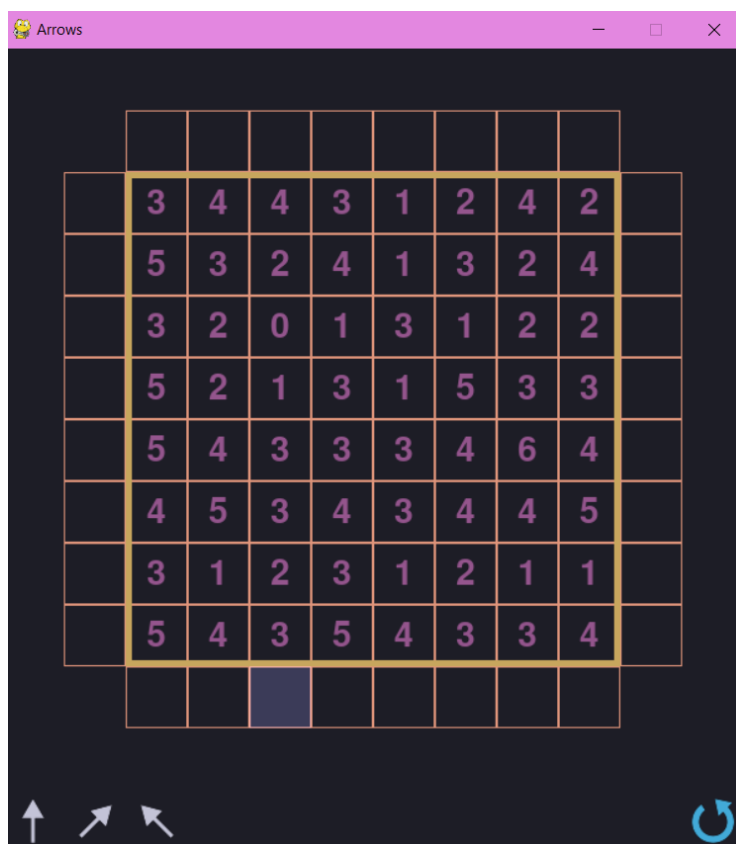


Рисунок 6.3 – Центральне положення клітинки для стрілки

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 *Рисунок 6.3*. Якщо клітинка для стрілки має крайове положення, то гравець має в розпорядженні лише 2 опції, як на *Рисунок 6.4*.

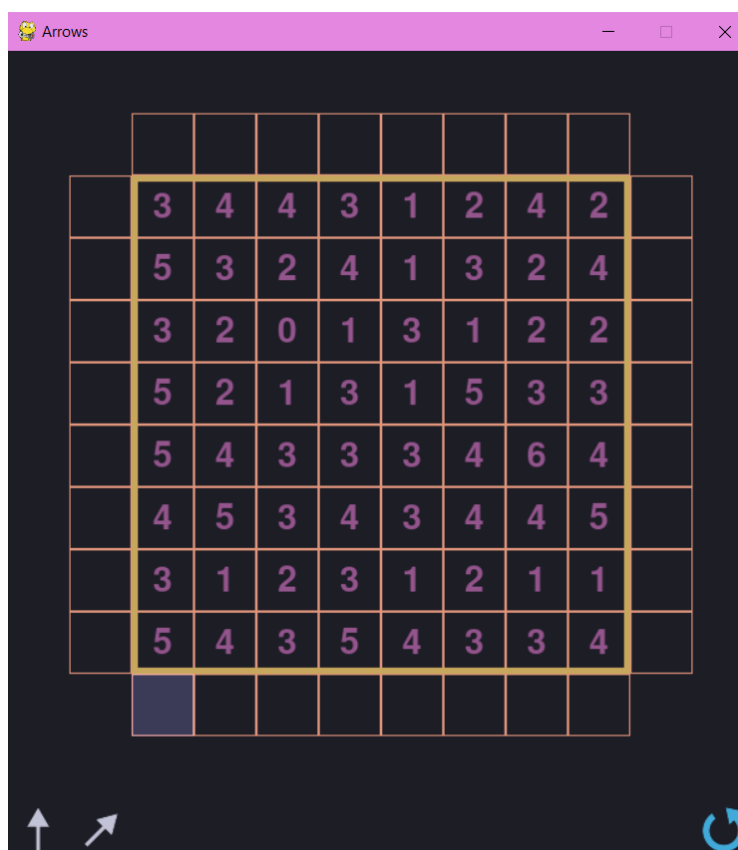


Рисунок 6.4 – Крайове положення клітинки для стрілки

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).

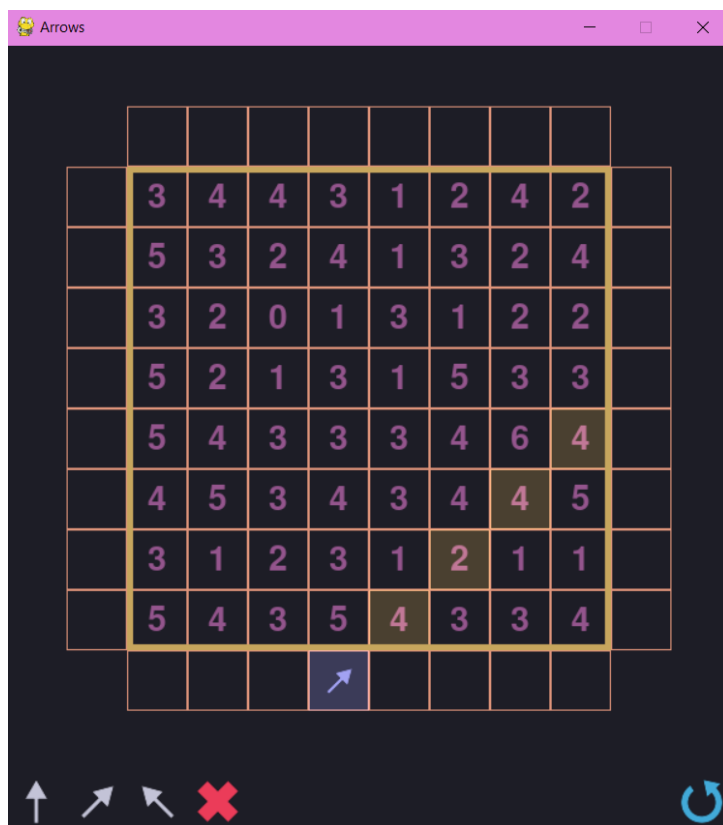


Рисунок 6.5 – Кнопка «Видалити стрілку»

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.

6.4 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»).

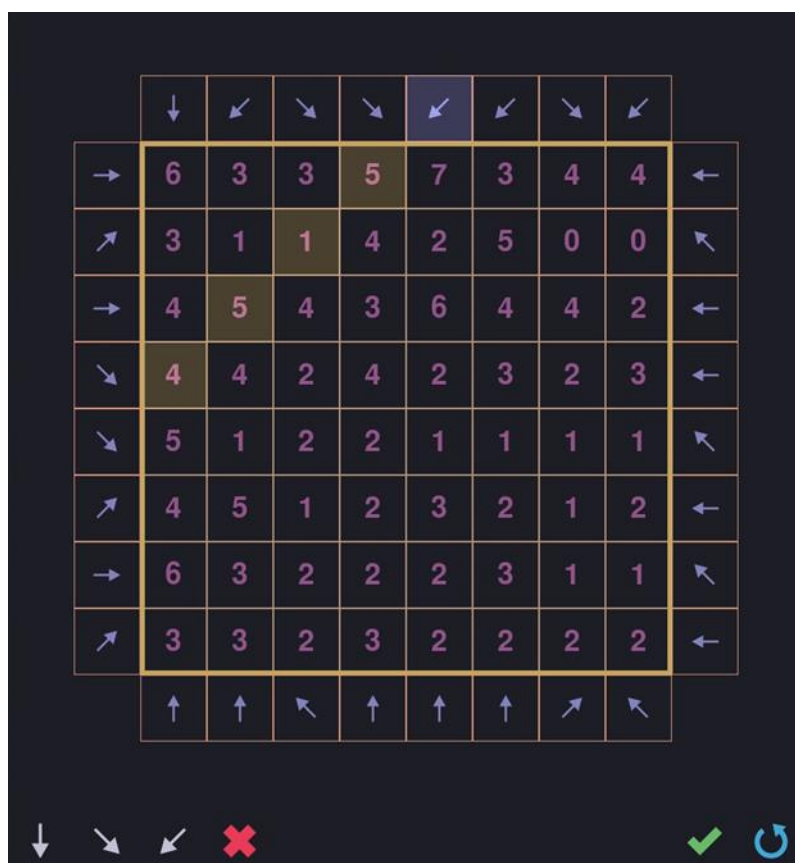


Рисунок 6.6 – Виділення стрілки та цифр, до яких вона напрямлена

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).

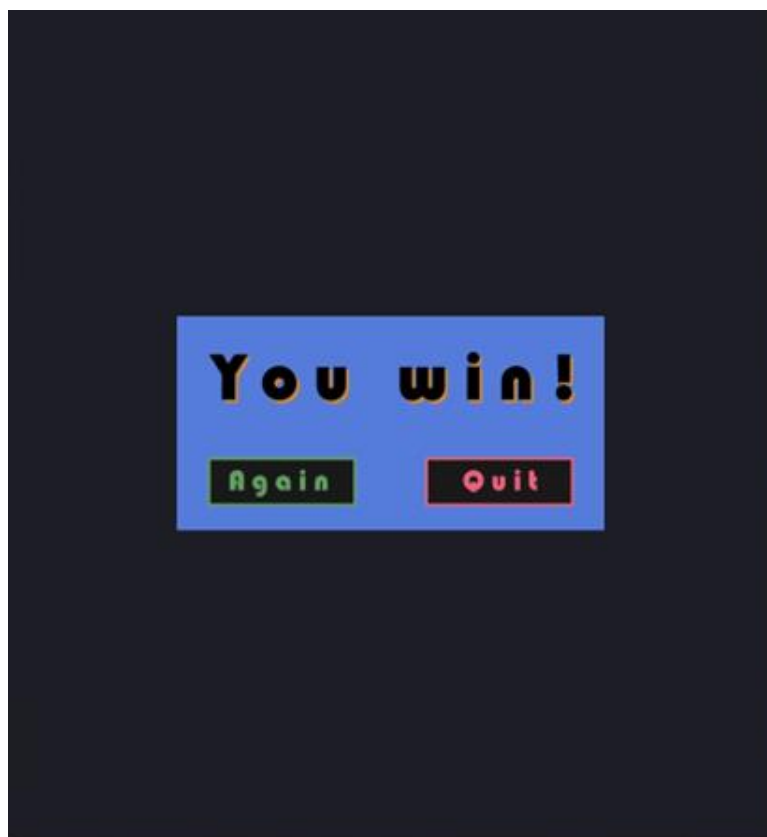


Рисунок 6.7 – Повідомлення про перемогу

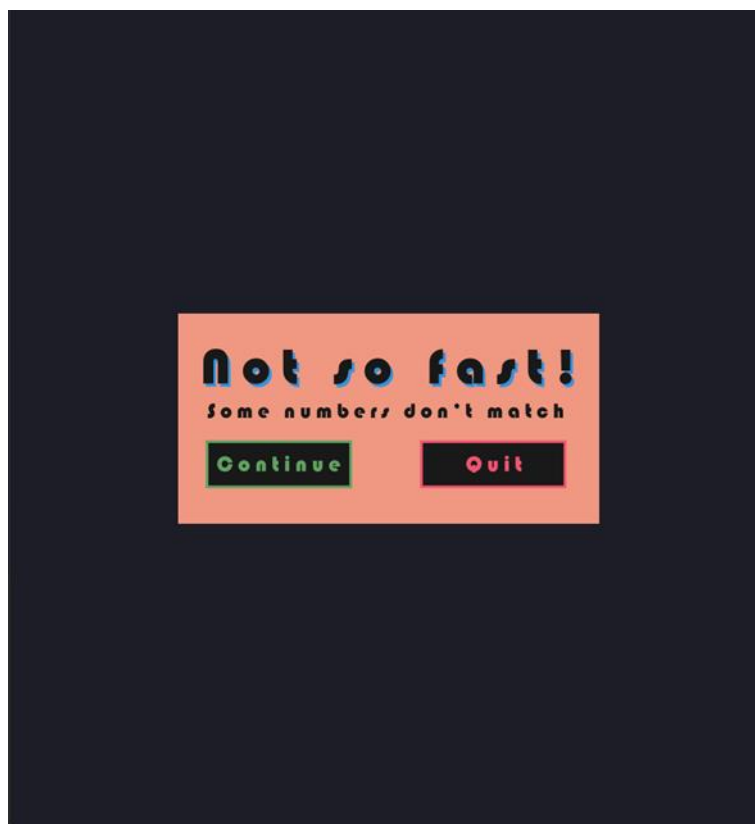


Рисунок 6.8 – Повідомлення про неправильне розташування стрілок

Якщо як в випадку на *Рисунок 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 *Рисунок 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*).

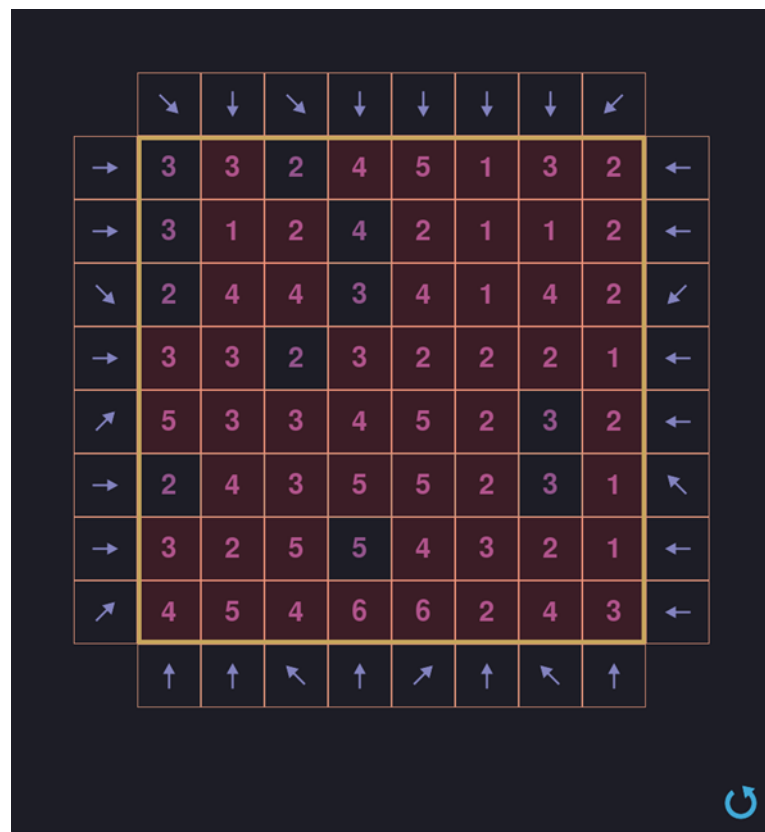


Рисунок 6.9 – Виділення неправильних клітинок з цифрами

Якщо натиснути на цифру на ігровому полі, то буде підсвічуватись вона та стрілки, що направлені до неї (*Рисунок 3.10*). Таким чином буде легше порахувати та зрозуміти, що неправильно було розташовано. Кнопку «Завершення розташування стрілок» можна натискати скільки завгодно разів, ліміту немає.

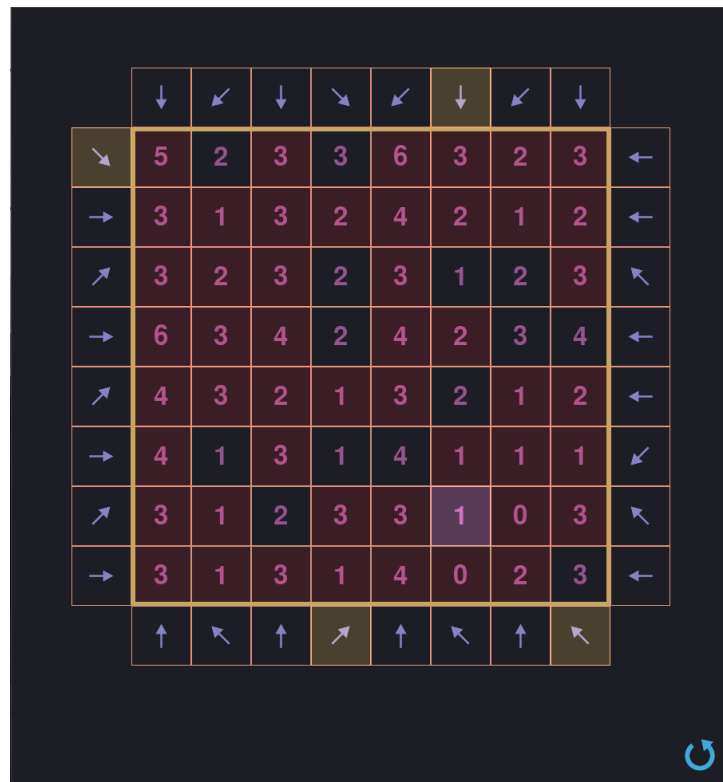


Рисунок 3.10 – Виділення цифри та стрілок, що напрямлені до неї

Також існує опція автоматичного рішення задачі. Для цього достатньо натиснути на кнопку у верхньому кутку вікна програми – «auto».

6.5 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 *Рисунок 6.7*.

6.6 Системні вимоги

Системні вимоги до програмного забезпечення наведені *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 оновленнями)	Windows 7/ Windows 8/Windows 10 (з останніми оновленнями)
Процесор	Intel® Pentium® III 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	900x975 Мінімальні	1080x1920 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 – Діаграма класів*).

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

1. Офіційна веб сторінка Python: Завантаження останньої версії Python.
URL: <https://www.python.org/downloads/> (дата звернення: 05.06.22)
2. Документація Pygame. URL: <https://www.pygame.org/docs/> (дата звернення: 05.06.22)

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»

1. *Мета:* Метою курсової роботи є розробка комп'ютерної гри "Стрілки".
Гра повинна складатись з ігрового поля 8x8. 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» лютий 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 розташування стрілок
 - 2) 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.
5. *Стадії та етапи розробки:*
 - 1) Об'єктно-орієнтований аналіз предметної області задачі (до 16.04.2022 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.).

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

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду комп'ютерної гри «Стрілки»

Електронний носій

31 арк., 83,5 KB

студента групи ІІІ-11 2 курсу

Лисенка А. Ю.

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_evaluator.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_evaluator 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_evaluator 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_evaluator 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: direction 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)
```