

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ**

**ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КУРСОВОЇ РОБОТИ З ПРОЄКТУВАННЯ АЛГОРИТМІВ
ТА ПРОГРАМУВАННЯ**

на тему:

«Розробка ігрової програми “Змійка” з графічним інтерфейсом мовою C++»

Виконала студентка 2 курсу
групи КН-22

Гнатюк Анна Михайлівна

Засвідчую, що курсовій роботі немає
запозичень з праць інших авторів
без відповідних посилань

(підпис студентки)

Керівник курсової роботи:

Асистент Паткін Євген Дмитрович

(підпис керівника роботи)

Оцінка за курсову роботу: _____

Члени комісії:

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ЗАВДАННЯ

на курсову роботу з проєктування алгоритмів та програмування
студентці Гнатюк Анні Михайлівні

1. Тема роботи

«Розробка ігрової програми “Змійка” з графічним інтерфейсом мовою C++»

2. Термін здачі закінченого проєкту: 26 травня 2023 р.

3. Вхідні дані до проєкту: немає

4. Зміст роботи:

Розділ 1. Аналіз класичних алгоритмів для гри змійка.

Розділ 2. Проєктування програми змійка з використанням алгоритму A*.

Розділ 3. Реалізація програмного застосунку «Змійка» з використанням бібліотеки SFML.

5. Перелік презентаційного матеріалу:

- 1) Мета та завдання курсової роботи (1-2 слайди).
 - 2) Огляд графічних ігор та основні принципи їх розробки (2-3 слайди).
 - 3) Характеристика задачі гри «Змійка» (1-2 слайди).
 - 4) Вибір технологій та інструментів для розробки гри (2-3 слайди).
 - 5) Опис алгоритму A-star для реалізації самостійного шляху змійки (3-4 слайди).
 - 6) Проєктування графічного інтерфейсу (2-3 слайди).
 - 7) Реалізація програми на мові C++ з використанням SFML (3-4 слайди).
 - 8) Огляд тестів та результатів їх виконання (1-2 слайди).
 - 9) Висновки та перспективи подальшого розвитку гри «Змійка» (1-2 слайди).
6. Дата видачі завдання 22 лютого 2023 р.

Графік виконання курсової роботи

№	Назва етапу	Терміни	Примітки / відмітка про виконання
1	Вибір теми та керівника курсової роботи	1 - 14 лютого	Заява на виконання курсової роботи, що підписана студентом керівником
2	Обговорення з керівником постановки завдання та змісту пояснювальної записки до курсової роботи	14–28 лютого	Заповнений бланк завдання на курсову роботу, що підписаний студентом та керівником роботи
3	Аналіз постановки задачі, формалізація задачі, вибір методів та засобів реалізації поставленої задачі, аналіз літературних джерел	1–12 березня	Сформований матеріал до розділу 1 пояснювальної записки курсової роботи, оформлення списку джерел
4	Перше узгодження з керівником	13–19 березня	
5	Розробка алгоритму, вибір структур даних, проєктування програмного інтерфейсу з користувачем	13 березня – 1 квітня	Сформований матеріал до розділу 2 пояснювальної
6	Друге узгодження з керівником КОНТРОЛЬНА ПЕРЕВІРКА	1–15 квітня	
7	Розробка та тестування програмного продукту.	1–23 квітня	Готовий програмний продукт
8	Третє узгодження з керівником. Демонстрація базового варіанту програмного продукту.	17–23 квітня	
9	Доопрацювання програмного продукту, всебічне заключне тестування, розробка тестових прикладів та керівництва користувача.	17–27 квітня	Сформований матеріал до розділу 3 пояснювальної записки, підготовлений демонстраційний приклад роботи програми
10	Остаточне оформлення пояснювальної записки	27–30 квітня	Готова пояснювальна записка та програмний застосунок
11	Подання роботи керівнику, перевірка роботи на плагіат, підготовка презентаційного матеріалу	1–14 травня	Готова пояснювальна записка та програмний застосунок, результати перевірки на плагіат
12	Подання роботи до захисту	15–26 травня	Готова пояснювальна записка, програмний застосунок, презентація для захисту курсової роботи.

Керівник роботи

Паткін Євген Дмитрович

Завдання прийняла
до виконання

Гнатюк Анна Михайлівна

ЗМІСТ

ВСТУП	3
РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД КЛАСИЧНИХ АЛГОРИТМІВ ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ Й ОПИС ГРИ «ЗМІЙКА» З АЛГОРИТМОМ A-star	4
1.1. Виникнення й особливості вдосконалення гри «Змійка»	4
1.2. Використання алгоритмів оптимального пошуку	5
1.3. Особливості застосування алгоритму A-star.....	7
<i>Висновки до Розділу 1</i>	10
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ	11
2.1. Постановка задачі.....	11
2.2. Опис методу й алгоритму розв’язку задачі	13
2.3. Обґрунтування вибору структур даних.....	15
<i>Висновки до Розділу 2</i>	17
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ	18
3.1. Опис головних структур даних і змінних програми.....	18
3.2. Опис головних функцій програми.....	22
3.3. Опис структури програми (програмних модулів).....	24
3.4. Опис програмного інтерфейсу з користувачем.....	25
3.5. Результати перевірки працездатності програмного продукту.....	28
<i>Висновки до Розділу 3</i>	28
ВИСНОВКИ	29
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	30
ДОДАТКИ	32

ВСТУП

Мета роботи: розробити ігрову програму «Змійка» з використанням мови програмування C++, бібліотеки SFML, та реалізацією алгоритму A* для пошуку найкоротшого шляху.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- дослідити літературні джерела, що описують алгоритми пошуку шляху;
- реалізувати алгоритм A* для розрахунку шляху між змійкою та їжею;
- написати програмний код для реалізації руху змійки;
- розробити графічний інтерфейс користувача з використанням бібліотеки SFML.
- перевірити правильність роботи програми та провести її тестування.

Стислий опис структури роботи за розділами:

У першому розділі проведено аналітичний опис гри «Змійка» та проаналізовано класичні алгоритми пошуку найкоротшого шляху. Було обрано найоптимальніший для гри алгоритм – A* – та описано принцип його роботи.

У другому розділі детально описано постановку задачі, охарактеризовано й представлено блок-схеми до розв’язання алгоритму A* й загальної логіки роботи програми. Також описано основні структури даних, що використовуються в програмі, та обґрунтовано їх вибір.

У третьому розділі описано та створено UML-діаграми класів, структур і перелічень, що створені в програмі, й детально охарактеризовано головні функції програми. Також описано структуру програми, програмний інтерфейс з користувачем і перевірено працездатність програмного продукту.

Опис використаних засобів розробки:

Для виконання курсової роботи була використана мова програмування C++.

Для реалізації графічного інтерфейсу була використана бібліотека SFML.

Розробка програмного продукту велась у Microsoft Visual Studio.

Практичне значення одержаних результатів:

Результати роботи можуть бути використані для створення застосунків із графічним інтерфейсом та пошуком найкоротшого шляху.

РОЗДІЛ 1

АНАЛІТИЧНИЙ ОГЛЯД КЛАСИЧНИХ АЛГОРИТМІВ ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ Й ОПИС ГРИ «ЗМІЙКА» З АЛГОРИТМОМ A-star

1.1. Виникнення й особливості вдосконалення гри «Змійка»

Придумана наприкінці 1970-х в епоху масштабного поширення ігрових автоматів, «Змійка» набула популярності і всесвітньої слави тільки в 1997-х, коли компанія Nokia випустила телефон з улаштованою в нього грою. Концепт її максимально простий: користувач керує змією (на двовимірній площині), вона збирає фрукти і відповідно збільшується в розмірах. Якщо ж вона зіткнеться зі стінкою чи сама із собою (хвостом, тілом) – програш. Якщо розмір змії збільшиться настільки, що на полі (площині) більше не залишиться вільного місця – користувач виграв [1].

На той час чимало ігор утратили популярність і були витіснені з ігрового середовища (за незначним виключенням) новими іграми [2]. Однак цікавість до «Змії» не зникла через те, що почали з'являтися нові можливості, правила, варіації. Усе – щоб користувачу було цікаво грати.

Але зараз, коли штучний інтелект може зробити все, що може забажати людина, постає проблема: чи справді потрібен користувач, який керуватиме змією?

Завдяки алгоритмам пошуку найкоротшого шляху, користувачу не обов'язково керувати персонажами [3]. Якщо в оригінальній «Змії» таке може видаватися трохи дивним (адже основна мета, сутність гри – керувати змією), то в пізніших її модифікаціях та інших іграх – алгоритми пошуку оптимального шляху є надзвичайно корисними. Як от для створення «NPC» – неігрових персонажів або ботів – програм-роботів [4, с. 290]. Вони функціонують на рівні користувача, але керуються комп'ютером для імітації противників чи союзників, із якими гравець може змагатися в режимі «мультиплеєр».

1.2. Використання алгоритмів оптимального пошуку

До найвідоміших і найбільш практикованих алгоритмів пошуку оптимального шляху належать:

BFS (Breadth-First Search) – алгоритм пошуку в ширину. Він розпочинається із заданої вершини й поступово розширюється на всі доступні з неї вершини за допомогою пошуку в ширину.

BFS забезпечує знаходження найкоротшого шляху в незваженому графі, оскільки він розглядає вершини в порядку їх відстані від початкової вершини. Проте у графі, де ребра мають різну вагу, BFS може не забезпечити найкоротший шлях, тому що він не враховує вагу ребер.

В ігрових програмах, де розмір графу є значно меншим, BFS може бути ефективним алгоритмом пошуку шляху [5, с. 99].

DFS (Depth-First Search) – алгоритм пошуку в глибину. Він працює на принципі обходу всіх вершин графа, причому спочатку проходить вглиб до кінцевої точки, а потім повертається назад до попередньої вершини та продовжує пошук в іншій гілці.

Алгоритм починає свою роботу з вибору початкової вершини. Потім алгоритм рекурсивно відвідує всі сусідні вершини поточної вершини, які ще не були відвідані. Коли всі сусідні вершини відвідані, алгоритм повертається до попередньої вершини та продовжує пошук в іншій гілці. Під час роботи алгоритму необхідно відстежувати відвідані вершини та запобігати зацикленню [6, с. 131-135].

DFS є досить простим алгоритмом і має використовуватись у випадках, коли потрібно відшукати один з можливих шляхів в графі. Він також зручний у випадках, коли граф має велику глибину, але невелику кількість гілок. Однак у випадках, коли граф має багато гілок і довгий шлях, DFS може стати недоцільним з точки зору швидкодії та вимогливості до пам'яті [5, с. 101]

Dijkstra – алгоритм Дейкстри, це алгоритм пошуку найкоротшого шляху в графі з не від'ємними вагами ребер. Основна ідея алгоритму полягає в тому, щоб

просуватися від однієї вершини до іншої, шукати найкоротший шлях між ними та оновлювати вагу всіх суміжних вершин, які можуть бути досягнуті через поточну вершину. Алгоритм Дейкстри знаходить найкоротший шлях до кожної вершини з вихідної вершини. Алгоритм закінчує роботу, коли всі вершини графа будуть опрацьовані, а ваги всіх ребер будуть оновлені з врахуванням найкоротшого шляху до кожної вершини.

Один з недоліків алгоритму Дейкстри полягає в тому, що він не працює з графами з від'ємними вагами ребер, оскільки може зациклитися у такому графі та не дати правильного результату [7].

Greedy – «жадібний» алгоритм. Він вирішує проблеми за допомогою миттєвого вибору оптимального рішення на кожному кроці. Алгоритм має властивість локальної оптимальності: вибирається найкращий варіант на кожному кроці, не звертаючи уваги на майбутні кроки, що може призводити до недосяжності глобальної оптимальності.

Наприклад, при розміщенні елементів на навантаженому графі, жадібний алгоритм може вибрати елемент з найменшою вагою і повторювати цей процес, додавши нові елементи до графу з кожним кроком. Однак такий алгоритм може не завжди дати глобально оптимальний результат, особливо якщо елементи взаємодіють між собою, і не завжди можна прогнозувати наслідки кожного кроку [8, с. 92].

A* (A star) – A-star алгоритм є модифікованою версією алгоритму Дейкстри та «жадібного» алгоритму [7; 8, с. 93]. Він використовує як відстань від початкового вузла, так і евристичну відстань до цілі. Евристика дає змогу відсікати вузли, які не ведуть до цілі, що дає змогу швидше знайти найкоротший шлях. Евристика (від давньогр. εὐρίσκω – *відшукую, відкриваю*) – сукупність логічних прийомів, методів і правил, що полегшують і спрощують рішення пізнавальних, конструктивних, практичних завдань, а також пошуку істини [9, с.185].

Однією з головних переваг алгоритму A* є те, що він ефективно використовує інформацію про відстань до кінцевої точки, тим самим зменшуючи

кількість перевірок вершин та забезпечуючи швидше знаходження найкоротшого шляху в графі. Він також може бути використаний для пошуку шляху в реальному часі. Гра «Змійка» вимагає знаходження найкоротшого шляху від змійки до фрукта, а потім до наступного фрукта в реальному часі, адже фрукти генеруються на ігровому полі у випадковому місці, тому алгоритм A-star є найбільш оптимальним за таких умов.

1.3. Особливості застосування алгоритму A-star

Для кращого розуміння того, як працює алгоритм, проаналізуємо випадок на практиці в грі «Змійка» з використанням рекомендацій [10]. Кожен крок до клітинки рахується як 10. Так як змійка не може ходити діагонально, кожному новому вузлові відкриваються лише вузли що зліва, справа, вгорі або знизу від нього. Кожна клітинка – це окремий вузол. Початковим вузлом є голова змійки, кінцевим вузлом є «яблуко». У лівому верхньому кутку клітинки позначається $g(n)$, у правому верхньому кутку $h(n)$, і по центру $f(n)$. Маємо формулу:

$$f(n) = g(n) + h(n) \quad (1.1)$$

де: $g(n)$ – ціна шляху від початкового вузла до цієї клітинки;

$h(n)$ – ціна шляху від цієї клітинки до кінцевого вузла;

$f(n)$ – сума цін цих шляхів.

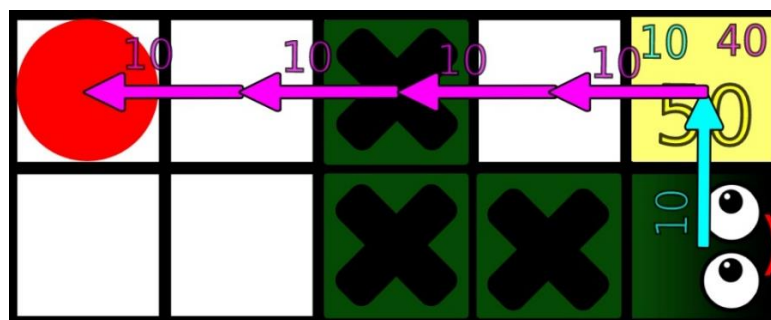


Рисунок 1.1 – Опис розрахунку цінових коефіцієнтів

Так, у кожного вузла є три значення, за якими й знаходиться найкоротший шлях.

Спершу для змійки доступно три вузли, що справа, знизу й над нею. Наступний вузол, який буде досліджено, обирається за принципом «найменша сума шляхів», тобто найменше значення $f(n)$. Після дослідження нового вузла стають доступні інші вузли, і серед доступних ще не досліджених вузлів буде обиратися новий вузол, доки не знайдеться шлях до кінцевого вузла. Тіло змійки на даному прикладі – це перешкода, тому цей вузол не рахується і його не можна дослідити.

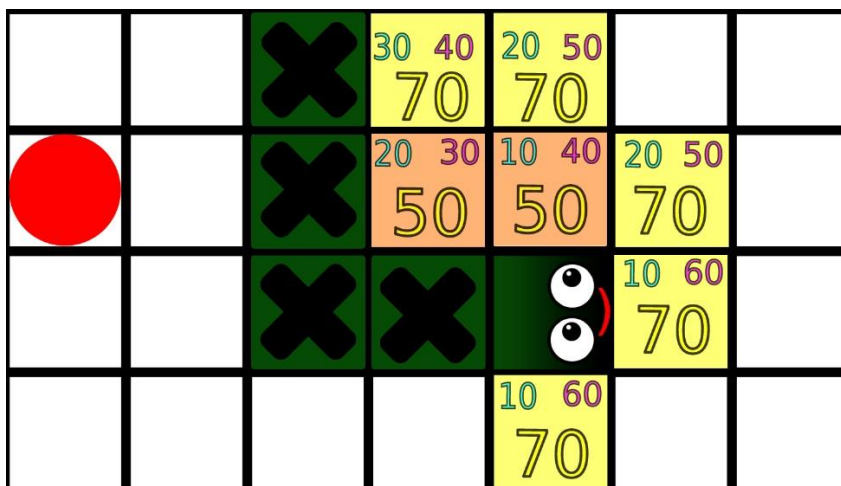


Рисунок 1.2 – Дослідження нових вузлів

Після дослідження багатьох вузлів можемо перекоонатися, що алгоритм знайшов шлях, не досліджуючи всі оточуючі його вузли. Саме цим алгоритм А-стар є кращим за інші класичні алгоритми – йому потрібно менше часу на знаходження найкоротшого шляху (див. рис. 1.1–1.3).

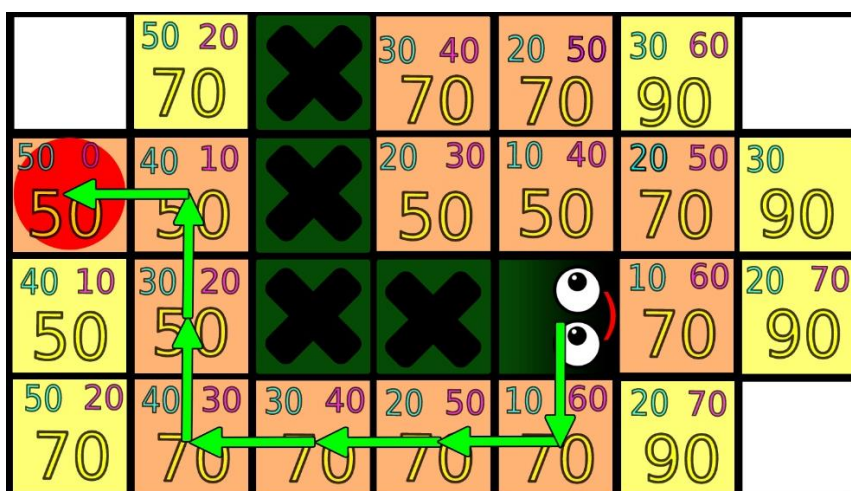


Рисунок 1.3 – Знайдений шлях

Реалізація цього алгоритму в грі «Змійка» дає змогу побачити, що він не є ідеальним. Рано чи пізно, коли змійка значно збільшиться в розмірі, вона заведе себе в пастку. Наприклад, якщо вона поверне голову так, що опиниться в замкнутому колі і яблуко буде поза колом. Алгоритм А-стар зупиниться, адже для нього шляху більше не існуватиме. Щоб запобігти таким ситуаціям, можна застосовувати інші алгоритми разом з А-стар. Проте для наглядності роботи алгоритму у курсовій роботі буде реалізовано тільки А-стар.

Фото покрокового пояснення роботи алгоритму наведено в додатку Б.

Переваги застосування алгоритму A*:

Ефективність: алгоритм A* здатен знаходити шляхи швидше, ніж багато інших алгоритмів, завдяки своїй евристичній оцінці відстані до кінцевої точки.

Оптимальність: за певних умов алгоритм гарантує знаходження найкоротшого шляху.

Варіативність: алгоритм може бути змінений для використання з різними евристичними функціями.

Завдяки ефективності алгоритму, його можна використовувати в багатьох сферах: планування маршрутів для потягів, авто та літаків, рух роботів в робототехніці, прокладання маршруту на онлайн картах, і звичайно ж в комп'ютерних іграх [11, с. 64].

Недоліки застосування алгоритму A*:

Пам'ять: алгоритм може вимагати значну кількість пам'яті для збереження відкритих та закритих списків вершин.

Алгоритм пошуку A* належить до евристичних алгоритмів пошуку [11, с. 65]. Таким чином, точність результатів залежить від якості евристичної функції, яку використовує алгоритм.

Ідея програмного проєкту:

Для наглядного представлення роботи алгоритму потрібно створити додаток із графічним інтерфейсом. У ньому користувач може обрати самостійне керування змійкою (звичайна гра) або ж стежити, як змійка сама знаходить шлях до фруктів, минаючи перешкоди.

Функціональні вимоги:

- Можливість самотійно керувати змійкою.
- Можливість споглядати за тим, як змійка сама знаходить шлях до цілі.
- Можливість ставити гру на паузу.
- Можливість виходити з гри у будь-який момент.
- Можливість бачити свій рахунок (успішність гри).
- Можливість регулювати швидкість змійки.

Нефункціональні вимоги:

- Наявність в інтерфейсі пунктів для вибору самотійної гри чи споглядання алгоритму
- Інтуїтивно зрозумілий та зручний у використанні інтерфейс

Висновки до Розділу 1

Опрацьовано інформаційні джерела, які допомогли сформулювати детальніший опис задачі, яка виконуватиметься під час роботи. Цей опис умістив узагальнений аналіз алгоритмів для розв'язку задачі, аналітичний опис актуальності обраного алгоритму для розв'язання поставленої задачі, його переваги та недоліки, функціональні та нефункціональні вимоги до програмного продукту. Також було сформульовано ідею програмного проєкту, зокрема, для наглядного представлення роботи алгоритму необхідно створити додаток із графічним інтерфейсом, у якому користувач може обрати самотійне керування змійкою (звичайна гра) або ж стежити, як змійка сама знаходить шлях до фруктів, минаючи перешкоди.

Аналіз можливостей алгоритму довів, що він не є досконалим, і в подальшому його можна покращити. Водночас доведено, алгоритм А-стар є кращим за інші класичні алгоритми – йому потрібно менше часу на знаходження найкоротшого шляху.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1. Постановка задачі

Потрібно розробити програмний продукт з графічним інтерфейсом, для реалізації гри «Змійка» та представлення нею реалізації алгоритму А-стар.

При запуску додатку виводиться графічне вікно з переліком можливостей для користувача.

Користувач може обрати:

- Play (грати) – вибравши цей пункт меню, відкриється нове графічне вікно, де користувач зможе грати в «Змійку». Керування рухом змійки робиться за допомогою клавіш на клавіатурі (стрілок вниз,вгору, вліво, вправо).
- A-star (A*) – вибравши цей пункт меню, відкриється графічне вікно для реалізації алгоритму А-стар. У вікні змійка сама знаходить шлях до яблука, тобто знаходить оптимальний шлях до цілі.
- Options (налаштування) – вибравши цей пункт меню, відкриється нове графічне вікно, де користувач зможе змінити налаштування швидкості руху змійки для режимів Play та A-star.
- About (про гру) – вибравши цей пункт меню, відкриється нове графічне вікно, де користувач зможе побачити деяку інформацію про гру.
- Exit (вихід) – вибравши цей пункт меню, додаток завершить свою роботу, і графічне вікно закриється.

За вибору пунктів Play або A-star у графічному вікні створюється ігрове поле розміром 29*53 клітинки. Кожна клітинка має розмір 25*25 пікселів. Поле оточене бар'єрами – стінками, в які змійка не повинна врзатися. Зверху, над полем, виводиться поточний рахунок (успішність), тобто кількість яблук, які на даний момент з'їла змійка.

Якщо змійка врізається в стіну або сама в себе, відкривається нове графічне вікно з повідомленням Game Over (кінець гри), рахунком і пунктами вибору подальших дій.

Користувач може обрати:

- Again (заново) – вікно про програш закриється, і гра почнеться заново.
- Exit (вихід) – вікно про програш, і вікно з грою закриється, і користувач повернеться до головного меню.

Якщо під час гри користувач натискає на клавіатурі клавішу Escape, відкривається нове графічне вікно з повідомленням Pause (Пауза), рахунком і пунктами вибору подальших дій.

Користувач може обрати:

- Resume (продовжити) – вікно про паузу закриється, і гра продовжиться.
- Exit (Вихід) – вікно про паузу, і вікно з грою закриється, користувач повернеться до головного меню.

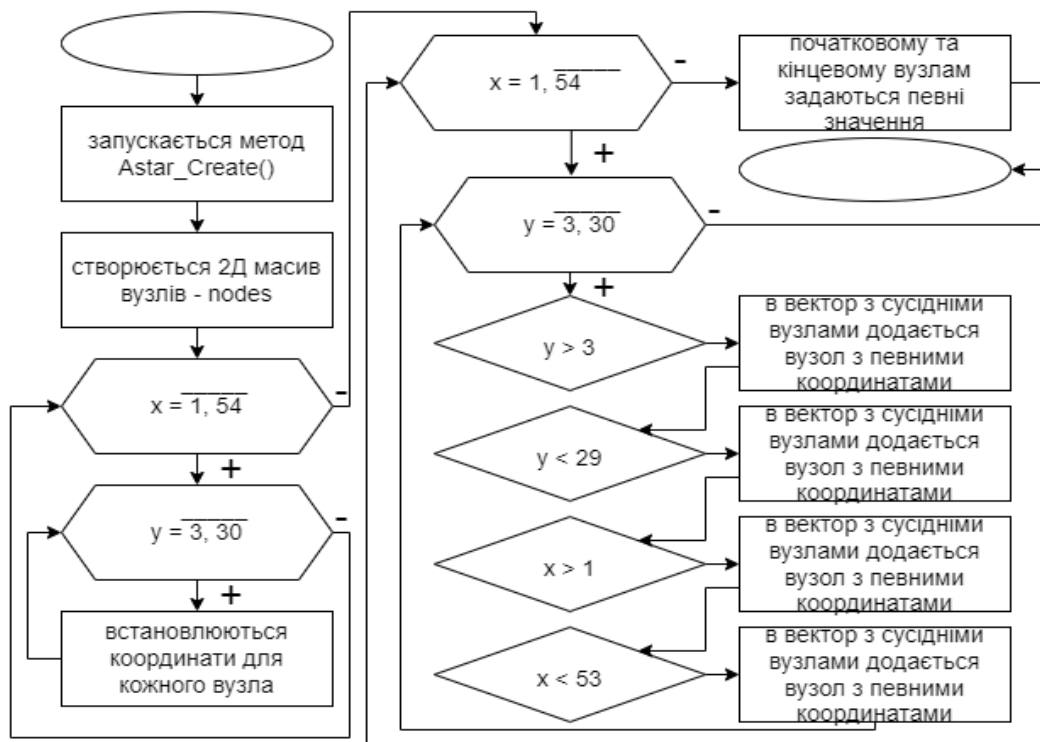


Рисунок 2.2 – Блок-схема методу Astar_Create()

На рис. 2.2 зображено метод класу Astar_class – Astar_Create(). Клас Astar_class створений для реалізації алгоритму A*. Викликається після створення об'єкту класу, для створення на заданому полі зв'язку між вузлами.

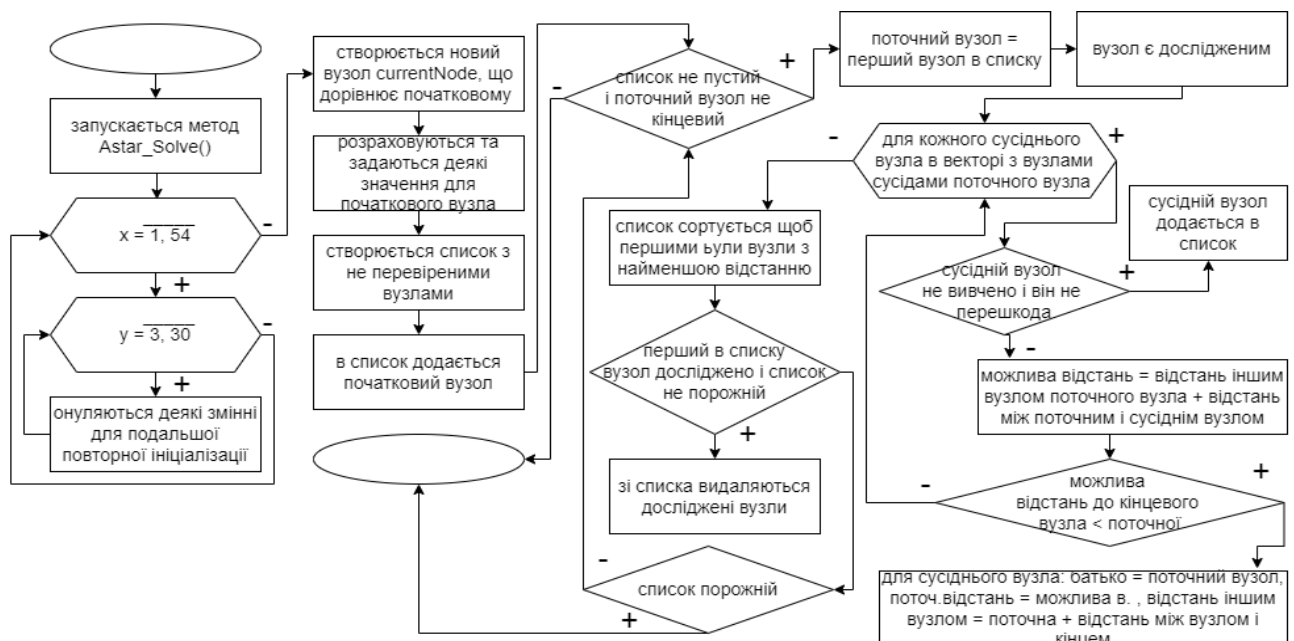


Рисунок 2.3 – Блок-схема методу Astar_Solve()

На рис. 2.3 зображено метод класу Astar_class - Astar_Solve(). Astar_Solve() реалізовує пошук оптимального шляху. Використовується у методі Astar_Update().



Рисунок 2.4 – Блок-схема методу Astar_Update()

На рис. 2.4 зображено метод класу Astar_class – Astar_Update(). Astar_Update() ініціалізує як початковий вузол голову змійки, як кінцевий вузол яблуко, як перешкоди тіло змійки без голови і хвоста і викликає метод Astar_Solve() для пошуку оптимального шляху між головою змійки і яблуком. Потім залежно від того, які координати у вузла, який метод Astar_Solve() установив як нащадок початкового вузла (тобто який вузол має стати головою), метод Astar_Solve() повертає число (напрямок голови), що пізніше використовується для встановлення наступного кроку змійки.

Детальніший опис класів, методів та функцій буде наведено в третьому розділі.

2.3. Обґрунтування вибору структур даних

У програмі використовуються enum, struct, vector, list та class.

- Enum – обрано для зручної організації назв пунктів меню та навігації змійки в подальшому використанні в switch (). Зберігається в файлі SnakeData.h з якого потім використовується в функції main().

- Struct – обрано для створення: формату тексту – TextFormat з полями int size_font – розмір шрифту, Color menu_text_color – колір тексту, float bord –

розмір обводки, `Color border_color` – колір обводки; координат змійки (Snake) з полями `int x`, `int y`; координат яблука (Food) з полями `int x`, `int y`; вузлів (`sNode`) з полями `bool isObstacle` – чи є вузол перешкодою, `bool isLearned` – чи вивчено вузол, `float distanceNow` – відстань до кінця від цього вузла, `float distanceAnother` – відстань до кінця якщо взяти інший вузол, `int x` – координата вузла `x`, `int y` – координата вузла `y`, `vector<sNode*> vector_Neighbours` – для зв'язку з сусідніми вузлами, `sNode* parent` – посилання на батька вузла. Зберігається в файлі `SnakeData.h` з якого потім використовується в функції `main()`, в файлі `Astar_nodes.h` в класі `Astar_class` для роботи в вузлами, пізніше використовується в функціях класу `Astar_Solve()`, `Astar_Create()` і `Astar_Update()`, в файлі `main.cpp` для подальшого використання в функції `GameStart()`.

- `Vector` – обрано для створення та подальшої обробки назв пунктів меню та зв'язку між вузлами в реалізації алгоритму A*. Зберігається в функції `main.cpp` для створення назв пунктів меню, `Opt_Buttons` для створення назв пунктів меню, `Settings_Game()` для створення назв пунктів меню, класі `Astar_class` в структурі `sNode` для створення зберігання вузлів та класі `SMenu` для використання назв пунктів меню.

- `List` – обрано для створення списку та зручної роботи з ще не вивченими вузлами для реалізації алгоритму A*. Зберігається в класі `Astar_class` в функції `Astar_Solve()`.

- `Class` – обрано для структуризації даних та роботи з ними. В програмі створено два класи: `SMenu` та `Astar_class`. `SMenu` створено в файлі `SnakeMenu.h`, проініціалізовано в файлі `SnakeMenu.cpp` та використано в файлі `main.cpp` в функції `GameStart()`. `SMenu` створено для налаштувань відображення пунктів меню: вирівнювання, налаштувань тексту. `Astar_class` створено в файлі `Astar_nodes.h`, проініціалізовано в файлі `Astar_nodes.cpp` та використано в файлі `main.cpp` в функції `GameStart()`. `Astar_class` створено для реалізації алгоритму A*: створення вузлів, зв'язку між ними, знаходження оптимального шляху та повернення потрібного напрямку змійки.

Більш детальний опис структур даних буде наведено в третьому розділі.

Висновки до Розділу 2

Було детально описано постановку задачі, описано методи та алгоритми використані в програмі. Також було представлено блок-схеми основної логіки програми для кращого розуміння роботи додатку в цілому, та методів для розв'язання алгоритму A^* . Також обґрунтовано вибір структур даних, та описано їх основні функції та їх використання. У третьому розділі буде детально описано структури даних що використовуються в програмі.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Опис головних структур даних і змінних програми

SMenu
- menu_X: float - menu_Y: float - menu_Step: int - max_menu: int - size_font: int - mainMenuSelected: int - font: Font - mainMenu: vector<Text> - menu_text_color: Color - chose_text_color: Color - border_color: Color - mywindow: RenderWindow&
- setInitOptText(text: Text&, str: const String&, xpos: float, ypos: float): const + SMenu(window: RenderWindow&, menux: float, menuy: float, sizeFont: int, step: int, name: vector<String&>) + draw() + MoveUp_Left() + MoveDown_Right() + setColorTextMenu(menColor: Color, ChoColor: Color, BordColor: Color) + AlignMenu(posx: int) + getSelectedMenuNumber(): const int

Рисунок 3.1 – UML-діаграма класу SMenu

Згідно з рис. 3.1, подаємо такий опис класу:

Клас:

- SMenu – клас для налаштувань виводу пунктів меню.

Поля класу:

- menu_X – координату пунктів меню по x, дійсне число;
- menu_Y – координати пунктів меню по y, дійсне число;
- menu_Step – відстань між пунктами меню, ціле число;
- max_menu – кількість пунктів меню, ціле число;
- size_font – розмір шрифту, ціле число;
- mainMenuSelected – номер вибраного пункту меню, ціле число;
- font – шрифт тексту меню, шрифт (sf::Font);
- mainMenu – назви меню, динамічний масив;
- menu_text_color – колір тексту пунктів меню, колір (sf::Color);

- chose_text_color – колір тексту вибраного пункту меню, колір (sf::Color);
- border_color – колір обведення тексту пунктів меню, колір (sf::Color);
- mywindow – графічне вікно, посилання на sf::RenderWindow.

Методи класу:

- SMenu(sf::RenderWindow& window, float menux, float menuy, int sizeFont, int step, std::vector<sf::String>& name) – конструктор з параметрами для початкової ініціалізації полів;
- void setInitOptText() – метод для налаштування тексту меню;
- void draw() – метод для відмальовки пунктів меню;
- void MoveUp_Left() – метод для переміщення обраного пункту меню вгору або вліво;
- void MoveDown_Right() – метод для переміщення обраного пункту меню вниз або вправо;
- void setColorTextMenu() – метод для налаштувань кольору тексту пунктів меню;
- void AlignMenu() – метод для вирівнювання пунктів меню;
- int getSelectedMenuNumber() – метод що повертає номер обраного пункту меню;

Клас створено у файлі SnakeMenu.h, методи та конструктор проініціалізовано у файлі SnakeMenu.cpp. Клас використовується в файлі main.cpp в функціях main(), Settings_Game(), Opt_Buttons.

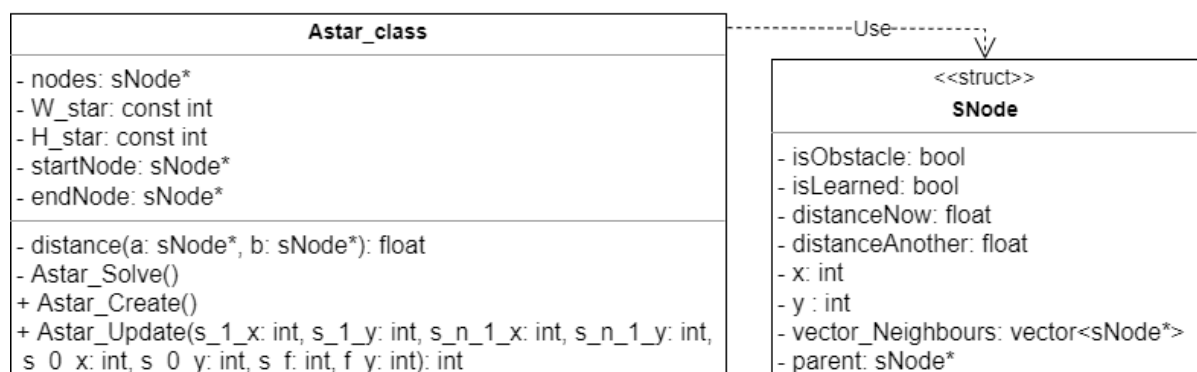


Рисунок 3.2 – UML-діаграма класу Astar_class і структури SNode

Згідно з рис. 3.2, подаємо такий опис класу:

Клас:

- Astar_class – клас для розв'язку алгоритму A*.

Поля класу:

- nodes – вузол в графі, посилання на об'єкт структури sNode*;
- W_star – ширина поля, цілочисельна константа, має значення 54;
- H_star – висота поля, цілочисельна константа, має значення 30;
- startNode – початковий вузол, голова змійки, посилання на об'єкт структури sNode*;
- endNode – кінцевий вузол, яблуко, посилання на об'єкт структури sNode*.

Методи класу:

- float distance(sNode* a, sNode* b) – метод для розрахунку відстані між двома вузлами по діагоналі за теоремою Піфагора;
- void Astar_Solve() – метод для пошуку найкоротшого шляху;
- void Astar_Create() – метод для створення зв'язку між вузлами;
- int Astar_Update(int s_1_x, int s_1_y, int s_n_1_x, int s_n_1_y, int s_0_x, int s_0_y, int f_x, int f_y) – метод для встановлення кінцевого вузла, початкового вузла та перешкод, повертає напрямок голови змійки для наступного кроку.

Клас створено у файлі Astar_nodes.h, методи та конструктор проініціалізовано в файлі Astar_nodes.cpp. Клас використовується у файлі main.cpp в функції GameStart().

Структура:

- sNode – вузол.

Поля структури:

- isObstacle – чи є вузол перешкодою, булева змінна;
- isLearned – чи вивчено вже цей вузол, булева змінна;
- distanceNow – відстань до кінцевого вузла від цього вузла, дійсне число;
- distanceAnother – відстань до кінцевого вузла від іншого вузла, дійсне число;
- x – координата вузла x, ціле число;
- y – координата вузла y, ціле число;

- `vector_Neighbours` – вектор, що складається з посилань на об’єкти структури `sNode`, що містить усі вузли в графі;
- `parent` – батько цього вузла, посилання на об’єкт структури.

Структуру створено в файлі `Astar_nodes.h` всередині класу `Astar_class`. Структура використовується в функціях класу `Astar_class` `distance()`, `Astar_Solve()`, `Astar_Create()`, `Astar_Update()`.

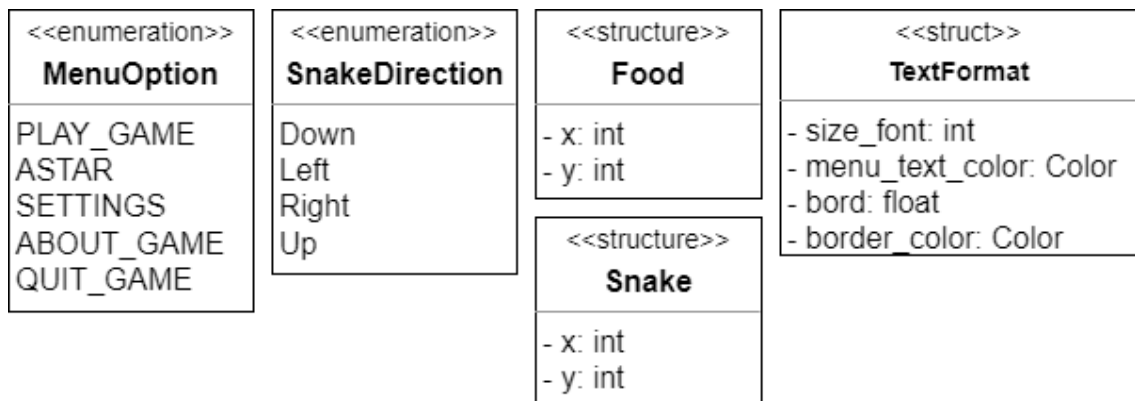


Рисунок 3.3 – UML-діаграма структур `Food`, `Snake`, `TextFormat` та перелічень `MenuOption`, `SnakeDirection`

Згідно з рис. 3.1, подаємо такий опис структур і перелічень:

Перелічення:

- `MenuOption` – для зручного використання вибору пунктів меню.

Поля структури:

- `PLAY_GAME` – для вибору режиму звичайної гри;
- `ASTAR` – для вибору режиму гри з алгоритмом A^* ;
- `SETTINGS` – для вибору пункту з налаштуваннями;
- `ABOUT_GAME` – для вибору пункту з відомостями про гру;
- `QUIT_GAME` – для виходу з гри.

Перелічення використовується в файлі `main.cpp` в функції `main()`.

Перелічення:

- `SnakeDirection` – для зручного використання обробки напрямку змійки.

Поля структури:

- `Down` – для напрямку наступного кроку змійки вниз;
- `Left` – для напрямку наступного кроку змійки вліво;

- Right – для напрямку наступного кроку змійки вправо;
- Up – для напрямку наступного кроку змійки вгору.

Перелічення використовується в файлі `main.cpp` у функції `SnakeLogic()`.

Структура:

- Food – для зручного використання координат яблука.

Поля структури:

- x – координати яблука по x, ціле число;
- y – координати яблука по y, ціле число.

Структура використовується у файлі `main.cpp` в функціях `SnakeLogic()` та `GameStart()`.

Структура:

- Snake – для зручного використання координат змійки.

Поля структури:

- x – координати змійки по x, ціле число;
- y – координати змійки по y, ціле число.

Структура використовується в файлі `main.cpp` в функціях `SnakeLogic()` та `GameStart()`.

Структура:

- TextFormat – для налаштувань тексту.

Поля структури:

- size_font – розмір шрифту тексту, ціле число;
- menu_text_color – колір тексту, колір (`sf::Color`);
- bord – розмір обведення тексту, дійсне число;
- border_color – колір обведення тексту, колір (`sf::Color`).

Структура використовується у файлі `main.cpp` в функціях `Settings_Game()`, `Opt_Buttons()`, `InitText()` та `GameStart()`.

3.2. Опис головних функцій програми

Функції що використовуються в класах описано в попередньому пункті.

Функції, що містяться у файлі `main.cpp`:

- `int main()` – головна функція програми, що виконується насамперед. У ній створюється і відмальовується вікно головного меню, використовується об'єкт та методи класу `SMenu`, в залежності від того, який пункт меню обере користувач, в функції оброблюється ввід з клавіатури і викликаються інші функції з використанням оператора `switch()` і переліку `enum MenuOption`.

- `void SnakeLogic()` – функція, що відповідає за логіку коректної генерації координат яблука, встановлення координат змійки, зміни координат змійки в залежності від напрямку її голови. Також ця функція змінює рахунок якщо змійка з'їла яблуко, і обробляє випадки коли змійка врізалась в стінку або в себе.

- `void InitText(Text& mtext, float xpos, float ypos, const String& str, TextFormat Ftext)` – функція для налаштувань тексту. Задає потрібному текст розмір шрифту, координати `x`, `y`, ініціалізує текст, колір тексту, колір обведення та його розмір.

- `void GameStart(int x)` – функція що реалізовує ігровий процес для звичайного режиму гри та режиму Астар. Приймає значення `x` – номер завдання. Якщо `x = 1` – рух змійки буде визначати користувач, якщо `x = 2` – рух змійки буде визначати алгоритм `A*`. В функції обробляється ввід з клавіатури, і створюється та відмальовується ігрове поле, змійка та яблуко. В функції використовується структура `Snake`, `Fruit`, `TextFormat`, викликаються функції `SnakeLogic()` та `Opt_Buttons()`. В ній створюється і відмальовується вікно паузи та програшу, використовується об'єкт та методи класу `SMenu`. Також якщо `x = 2` створюється об'єкт класу `Astar_class`.

- `int speed_converter(float delay)` – функція обробляє поточну швидкість змійки й повертає її в зручному форматі.

- `void Settings_Game()` – функція для налаштувань швидкості змійки в режимі звичайної гри та режиму Астар. У ній створюється і відмальовується вікно налаштувань, використовується об'єкт та методи класу `SMenu`, використовуються функції `speed_converter()` і `InitText()` і структура `TextFormat`. В функції обробляється ввід з клавіатури і залежно від нього змінюється швидкість змійки в режимах.

- void About_Game() – функція створює й відмальовує вікно з відомостями про гру. У функції обробляється ввід з клавіатури.
- void Opt_Buttons(const String& windowname, const String& buttonOne, const String& buttonTwo, int num) – функція відповідає за створення та відмальовку вікон паузи і програшу. В ній використовується об'єкт та методи класу SMenu, використовується функція InitText() і структура TextFormat. У функції оброблюється ввід з клавіатури і в залежності від нього виходить до головного меню, починає гру заново або продовжує гру.
- void GameProcess(int x) – функція відповідає за старт ігрового процесу й поновлення потрібних для нової гри значень. З неї запускається функція GameStart().

3.3. Опис структури програми (програмних модулів)

При запуску програми запускається функція main(). У ній створюється та відмальовується графічне вікно з головним меню гри. Обробляється ввід з клавіатури.

- Якщо користувач обирає пункт меню Play – запускається режим звичайної гри де користувач керує змійкою.
- Якщо ж користувач обирає пункт A-star – рух змійки визначається алгоритмом A*/

Під час гри користувач бачить поточний рахунок – скільки яблук з'їла змійка. Якщо під час гри користувач натисне на клавіатурі клавішу Escape – відкриється вікно з паузою, де відобразиться його рахунок і вибір: продовжити гру чи вийти. Якщо користувач обере продовжити гру – вікно з паузою закриється й гра продовжиться. Якщо ж обере «вийти з гри» – вікно з паузою й вікно з грою закриються, і користувач повернеться до головного меню. Якщо користувач програє – відкриється вікно програшу з рахунком і вибором: грати заново чи вийти. Якщо користувач обере грати заново – вікно з програшем закриється і гра почнеться заново. Якщо ж обере вийти з гри – вікно з програшем і вікно з грою закриються, і користувач повернеться до головного меню.

- Якщо користувач обирає пункт Settings – відкривається вікно з налаштуваннями, де користувач може змінити швидкість змійки для обраного ним режиму, або повернутись до головного меню.
- Якщо користувач обирає пункт About – відкривається вікно з інформацією про гру. Користувач може повернутись до головного меню.
- Якщо користувач обирає пункт Exit – вікно з головним меню закривається й робота програми припиняється.

3.4. Опис програмного інтерфейсу з користувачем

При запуску програми відкривається вікно головного меню.

Якщо користувач натискає на клавіатурі клавішу «стрілка вгору» – вибір пункту меню переміщається вгору, якщо ж натискає «стрілка вниз» – вибір переміщається вниз. Наприклад, якщо зараз обрано пункт Play (виділений синім кольором) і користувач натисне «стрілку вниз» – вибір переміститься на A-star. Якщо користувач натисне клавішу Enter – перейде в нове вікно залежно від того, який пункт меню був виділений.



Рисунок 3.4 – Головне меню програми

Якщо обрано пункт About, відкривається вікно з описом програми. Якщо натиснуто Escape – вікно закриється й він повернеться до головного меню.

Якщо обрано пункт A-star – відкриється вікно з грою, де шлях змійки визначатиметься алгоритмом A*.

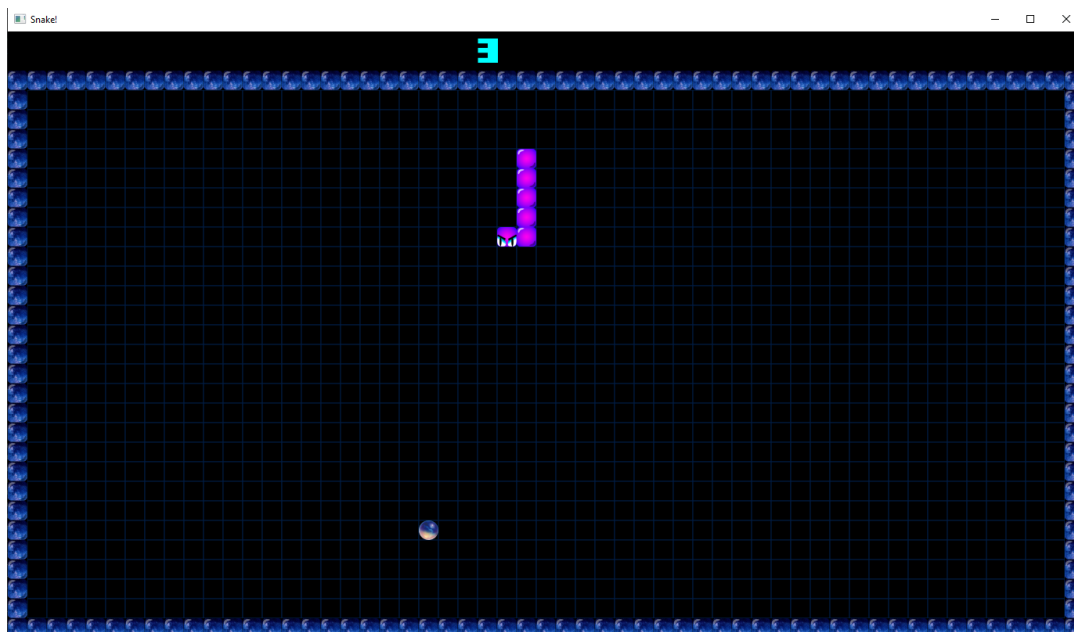


Рисунок 3.7 – Вікно A-star

Якщо обрано пункт Play – відкриється вікно з грою, де шлях змійки задає користувач. Клавiша «стрілка вгору» – змійка рухається вгору, «стрілка вниз» – змійка рухається вниз, «стрілка вліво» – змійка рухається вправо, «стрілка вправо» – стрілка рухається вправо. Вікно Play виглядає так само, як вікно A-star, що наведено на рис. 3.7.

Якщо під час гри в режимі Play або A-star користувач натисне Escape – відкриється вікно паузи й відобразиться рахунок. Якщо користувач натисне «стрілка вліво» – вибір пункту меню переміщається вліво, якщо натискає «стрілка вправо» – вибір переміщається вправо. Наприклад, якщо зараз обрано пункт Resume (виділений синім кольором) і користувач натисне «стрілку вправо» – вибір переміститься на Exit. Якщо натисне Enter і обрано Resume – гра продовжиться. Якщо ж обрано Exit – вікно з грою й вікно з паузою закриються, і користувач повернеться до головного меню.



Рисунок 3.8 – Вікно паузи

Якщо під час гри в режимі Play або A-star змійка зіткнеться зі стіною чи власним тілом – відкриється вікно програшу і відобразиться рахунок. Якщо користувач натисне «стрілка вліво» – вибір пункту меню переміщається вліво, якщо натискає «стрілка вправо» – вибір переміщається вправо. Наприклад, якщо зараз обрано пункт Again (виділений синім кольором) і користувач натисне «стрілку вправо» – вибір переміститься на Exit. Якщо користувач натисне Enter і обрано Again – гра почнеться заново. Якщо ж обрано Exit – вікно з грою й вікно з паузою закриються, і користувач повернеться до головного меню.



Рисунок 3.9 – Вікно програшу

3.5. Результати перевірки працездатності програмного продукту

Для перевірки роботи програми представлено рахунок змійки в режимі A*

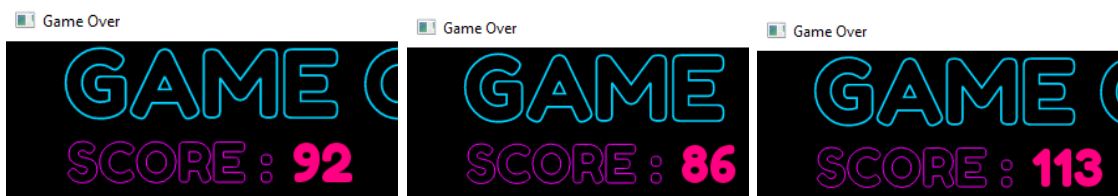


Рисунок 3.10 – Результати роботи змійки в режимі A*

Програмний код застосунку подано в Додатку А.

Висновки до Розділу 3

Описано реалізацію програмного продукту. Функціонал зручний у використанні й інтуїтивно простий. Надалі можна додати вибір вигляду змійки і поля, створити бонуси за досягнення певного розміру, реалізувати історію рахунків та збереження гри. З аналізу роботи програми видно, що алгоритм A-стар не є досконалим, особливо використовуючи його для даної гри. У подальшому можна його вдосконалити або поєднати з іншими алгоритмами.

ВИСНОВКИ

У результаті роботи на ігровою програмою «Змійка» було визначено можливості графічної бібліотеки SFML, освоєно літературні джерела з описом алгоритмів пошуку оптимального шляху та детально охарактеризовано алгоритм A*. Також на практиці визначено принцип створення десктопних додатків та ігрових програм, освоєно, як користувач може взаємодіяти з програмою.

У роботі вдалося реалізувати десктопний додаток з графічним інтерфейсом, завдяки графічній бібліотеці SFML. У додатку реалізовано алгоритм A* та показано, як він працює на прикладі гри «Змійка».

Великою перевагою розробки є те, що користувач бачить зручний та красивий інтерфейс, яким цікаво користуватись. Можна змінювати налаштування, що дає змогу кожному грати з комфортною для нього швидкістю. Також реалізація алгоритму є оптимальною в витраті ресурсів пам'яті, а реалізація алгоритму є простою, що сприяє в ньому легко розібратись і в подальшому вдосконалити.

Недоліками такого представлення алгоритму є те, що поєднання принципів алгоритму та гри не дозволяє змійці виграти, однак це водночас є наглядним прикладом того, що алгоритм A* – не ідеальний, і залежно від сфери його використання може потребувати вдосконалень.

Створену програму зі зручним інтерфейсом можна використовувати в подальшому застосуванні для реалізації нових ігор чи алгоритмів упродовж навчального процесу.

Надалі програму можна вдосконалити, додавши режим мультиплеєра, можливість обирати дизайн змійки, яблука та тла. Також додати можливість відкривати налаштування коли триває гра. Ще можна вдосконалити алгоритм A* та створити алгоритм для обробки випадків, коли змійка загнала себе в глухий кут, або використовувати його з іншими алгоритмами для знаходження оптимального шляху, що допоможе оптимізувати ефективність роботи програми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гра «Змійка»: історія популярності та еволюції. *Сайт міста Шепетівка*: офіційний веб-сайт. 31. 03. 2022. URL: <https://cutt.ly/g71OUkB>.
2. Верстюк І. Згадаймо комп'ютерні ігри початку 90-х: моя десятка геймерських хітів. *Спільнота*: медіа-сайт 21.01.2023. URL: <https://cutt.ly/i71OCjt>.
3. Сеніва К. Р. Способи керування поведінкою неігрового персонажу в RPG-грі. *Науковий огляд*. 2021. Т. 2. № 74. URL: <https://cutt.ly/S71O1dp>.
4. Зубко А. В., Майданюк В. П. Конфігурований бот для вирішення та моделювання ігрових ситуацій в стратегіях реального часу. *Інформаційні технології і автоматизація–2020* : зб. доп. XIII Міжнар. наук.-практ. конф. (Одеса, 22–23 жовт. 2020 р.) / Одес. нац. акад. харч. технологій, Інститут комп'ютерних систем і технологій «Індустрія 4.0» ім. П. М. Платонова. Одеса, 2020. С. 290–291.
5. Крєневич А. П. Алгоритми і структури даних. Київ : ВПЦ «Київський університет», 2021. 200 с.
6. Мелешко Є. В., Якименко М. С., Поліщук Л. І. Алгоритми та структури даних. Кривницький : ФОП Лисенко В. Ф., 2019. 156 с.
7. Elgabry O. Path Finding Algorithms BFS, DFS(Recursive & Iterative), Dijkstra, Greedy, & A* Algorithms. *Medium*: інформаційний веб-сайт. 11.10.2016. <https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40>.
8. Russel S. J., Norvig P. Artificial Intelligence: A Modern Approach. Third Edition. Upper Saddle River [USA] : Pearson Education Inc., 2010. 1151 p.
9. Морозов С. М., Шкарапута Л. М. Словник іншомовних слів. Київ : Наукова думка, 2000. 683 с.
10. Patel A. Amit's Thoughts on Pathfinding : Introduction to A* : *Stanford Theory* : офіційний вебсайт Стенфордського університету, 2023. URL : <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>.
11. Петришин С., Решетник В. Дослідження та порівняльний аналіз алгоритмів знаходження оптимального шляху. *ВНТУ: Інституційний*

репозитарій: Інтелектуальні інформаційні технології. 2018. С. 64–66. URL:
<https://ir.lib.vntu.edu.ua/handle/123456789/22476>.

ДОДАТКИ

Додаток А. Програмний код застосунку

Файл main.cpp:

```
#include <SFML/Graphics.hpp>
```

```
#include <string>
```

```
#include <vector>
```

```
#include <list>
```

```
#include <time.h>
```

```
#include <cmath>
```

```
#include "SnakeMenu.h"
```

```
#include "Astar_nodes.h"
```

```
#include "SnakeData.h"
```

```
using namespace sf;
```

```
int dir = 2;    // поворот
```

```
int num = 3;    // довжина змійки
```

```
int score = 0;
```

```
float game_delay = 0.1; // затримка
```

```
float astar_delay = 0.1; // затримка
```

```
int speed_converter(float delay)
```

```
{
```

```
    return std::round((delay - 0.1) / (- 0.01));
```

```
}
```

```
bool game_state = true;
bool border_touch = false;
bool exit_snake = false;
bool play_again = false;
bool playing = false;
bool pause_exit = false;

// довжина змійки, структура
struct Snake
{
    int x, y; // координати
}s[1705];    // загальна макс. кількість

// планета, структура
struct Food
{
    int x, y; // координати
} f;

// відповідає за гру
void SnakeLogic()
{

    // малюється змійка
    for (int i = num; i > 0; i--)
    {
        s[i].x = s[i - 1].x;
        s[i].y = s[i - 1].y;
    }
```

```

// рух змійки
switch (dir)
{
case SnakeDirection::Down:
    s[0].y += 1; // вниз
    break;
case SnakeDirection::Left:
    s[0].x -= 1; // вліво
    break;
case SnakeDirection::Right:
    s[0].x += 1; // вправо
    break;
case SnakeDirection::Up:
    s[0].y -= 1; // вгору
    break;
}

// врізалась в стінку - програш
if (s[0].x == W - 1 || s[0].x == 0 || s[0].y == H - 1 || s[0].y == 2) {
    game_state = false;
    border_touch = true;
}

// їсть планету
if ((s[0].x == f.x) && (s[0].y == f.y)) {
    num++; // виросла
    score++; // збільшився рахунок
}

```

```
// міняються координати планети
```

```
bool goodapple = false;

while (goodapple!= true)
{
    bool flag = false;

    f.x = rand() % (W - 4) + 3;
    f.y = rand() % (H - 4) + 3;
    for (int i = 0; i < num; i++)
    {
        if (f.x == s[i].x && f.y == s[i].y)
        {
            goodapple = false;
            flag = true;
            break;
        }
    }
    if (!flag)
        goodapple = true;

}

// програв
for (int i = 1; i < num; i++) // для всіх плиток змійки
{
    // врізалась в себе
```

```

        if ((s[0].x == s[i].x) && (s[0].y == s[i].y))
        {
            game_state = false;
        }
    }
}

// Функція для налаштування тексту
void InitText(Text& mtext, float xpos, float ypos, const String& str, TextFormat Ftext)
{
    mtext.setCharacterSize(Ftext.size_font);
    mtext.setPosition(xpos, ypos);
    mtext.setString(str);
    mtext.setFillColor(Ftext.menu_text_color);
    mtext.setOutlineThickness(Ftext.bord);
    mtext.setOutlineColor(Ftext.border_color);
}

// оголошую функції
void GameStart(int x);
void About_Game();
void Settings_Game();
void Opt_Buttons(const String& windowname, const String& buttonOne, const String&
buttonTwo, int num);
void GameProcess(int x);

int main()

```

```

{
    // Створюю вікно для основного меню
    RenderWindow window(VideoMode(width_menu, height_menu), L"SuperSnake",
        Style::Default);

    // Назви пунктів
    std::vector<String> name_menu{ L"Play", L"A-star", L"Options", L>About",
        L"Exit"};

    // Об'єкт меню
    SMenu mymenu(window, 100, 180, 80, 120, name_menu);

    // Налаштування кольору тексту і вирівнювання
    mymenu.setColorTextMenu(Color::Black, Color(53, 53, 255), Color::Cyan);
    mymenu.AlignMenu(1);

    // Фон
    RectangleShape background(Vector2f(width_menu, height_menu));

    Texture texture_window;

    texture_window.loadFromFile("C://Users//Legion//Desktop//дурастіка//photos//fonmen
u.JPG");

    background.setTexture(&texture_window);

    // Шрифт
    Font font;
    font.loadFromFile("C://Users//Legion//Desktop//дурастіка//fonts//cool_font.ttf");

    // початок

```

Clock clock;

```

while (window.isOpen())
{
    Event event;
    while (window.pollEvent(event))
    {
        if (event.type == Event::KeyPressed)
        {
            // обробка клавіатури
            if (event.key.code == Keyboard::Up) { mymenu.MoveUp_Left(); }    //
вгору
            if (event.key.code == Keyboard::Down) { mymenu.MoveDown_Right(); } //
вниз
            if (event.key.code == Keyboard::Enter)                        // ентер
            {
                MenuOption selectedOption =
static_cast<MenuOption>(mymenu.getSelectedMenuNumber());

                // Перехід на обраний пункт меню
                switch (selectedOption)
                {
                    case PLAY_GAME:
                        playing = true;
                        GameProcess(1);
                        break;
                    case ASTAR:
                        playing = true;
                        GameProcess(2);

```



```

        break;
    case SETTINGS:
        Settings_Game();
        break;
    case ABOUT_GAME:
        About_Game();
        break;
    case QUIT_GAME:
        window.close();
        break;
    default:
        break;
    }
}
}

// Відмалювати все
window.clear();
window.draw(background);
mymenu.draw();
window.display();
}
return 0;
}

// Грати
void GameStart(int x)
{

```

```
srand(time(0));
```

```
RenderWindow game_window(VideoMode(tile_size * W, tile_size * H), "Snake!");
```

```
// Шрифт для рахунку
```

```
Font score_font;
```

```
score_font.loadFromFile("C://Users//Legion//Desktop//дупастіка//fonts//cyber_font.ttf"
);
```

```
// рахунок (зверху)
```

```
Text Score;
```

```
Score.setFont(score_font);
```

```
TextFormat Stext;
```

```
Stext.size_font = 100;
```

```
Stext.menu_text_color = Color::Cyan;
```

```
Stext.bord = 3;
```

```
// плитка
```

```
Texture t;
```

```
t.loadFromFile("C://Users//Legion//Desktop//дупастіка//photos//blank.png");
```

```
Sprite tiles(t);
```

```
// змійка
```

```
Texture sn;
```

```
sn.loadFromFile("C://Users//Legion//Desktop//дупастіка//photos//snake_cool.png");
```

```
Sprite snake(sn);
```

```
// планетка
```

```

Texture pl;
pl.loadFromFile("C://Users//Legion//Desktop//дурастіка//photos//apple.png");
Sprite planet(pl);

// стіна
Texture br;

br.loadFromFile("C://Users//Legion//Desktop//дурастіка//photos//brick_galaxy.png");
Sprite brick(br);

f.x = 10; // координата x планети
f.y = 10; // координата y планети

// Початкові координати для змійки
s[0].x = W / 2;
s[0].y = H / 2;

s[1].x = W / 2 - 1;
s[1].y = H / 2;

s[2].x = W / 2 - 2;
s[2].y = H / 2;

Clock clock;    // час
float timer = 0; // через який час змійка рухається

Astar_class snake_astar;
snake_astar.Astar_Create();

```

```

while (game_window.isOpen())
{
    // ігровий час
    float time = clock.getElapsedTime().asSeconds();
    clock.restart();
    timer += time;

    Event event;
    while (game_window.pollEvent(event))
    {
        if (event.type == Event::Closed) game_window.close();
        if (event.type == Event::KeyPressed)
        {
            if (event.key.code == Keyboard::Escape) // пауза
            {
                Opt_Buttons(L"Pause", L"Resume", L"Exit", 1);
                if (exit_snake)
                {
                    pause_exit = true;
                    game_window.close();

                }
            }
        }
        if (x == 1) // якщо x = 1 граю сама
        {
            if (event.key.code == Keyboard::Left) dir = 1;
            if (event.key.code == Keyboard::Right) dir = 2;
            if (event.key.code == Keyboard::Up) dir = 3;

```

```

        if (event.key.code == Keyboard::Down) dir = 0;
    }
}
}

if (x == 2)                                // якщо x = 2 викликаю алгоритм
    dir = snake_astar.Astar_Update(s[1].x, s[1].y, s[num - 1].x, s[num - 1].y, s[0].x,
s[0].y, f.x, f.y);

if (!pause_exit) {
    // робота гри
    if(x == 1)
        if (timer > game_delay && game_state) {
            timer = 0;    // час
            SnakeLogic(); // логіка гри (генерація планети, змійки і т д)
        }
    if(x == 2)
        if (timer > astar_delay && game_state) {
            timer = 0;    // час
            SnakeLogic(); // логіка гри (генерація планети, змійки і т д)
        }

    game_window.clear();

    // відмалювання поля
    for (int i = 0; i < W; i++)
    {
        for (int j = 2; j < H; j++)
        {
            if (i == 0 || i == W - 1 || j == 2 || j == H - 1)

```

```

    {
        brick.setPosition(i * tile_size, j * tile_size); // розташування стінки
        game_window.draw(brick);                          // малюю стінку
    }
else
{
    tiles.setPosition(i * tile_size, j * tile_size); // розташування поля
    game_window.draw(tiles);                          // малюю поле
}
}
}

// малюю змійку
for (int i = 0; i < num; i++) {
    if (border_touch && i == 0)
    {
        s[0].x -= tile_size;
        s[0].y -= tile_size;
        num++;
    }
    if (i != 0) // тіло
        snake.setTextureRect(IntRect(0, 0, tile_size, tile_size));
    if (!border_touch && i == 0) // голова
        snake.setTextureRect(IntRect(dir * tile_size, tile_size, tile_size, tile_size));

    // зміна голови, якщо програш
    if (!game_state && i == 1)
        snake.setTextureRect(IntRect(dir * tile_size, tile_size * 2, tile_size,
tile_size));

```

```

// розташування (x, y) кожної плитки змійки
snake.setPosition(s[i].x * tile_size, s[i].y * tile_size);

// для плиток
game_window.draw(snake);

}

planet.setPosition(f.x * tile_size, f.y * tile_size); // розташування яблука
game_window.draw(planet); //малюю яблуко

InitText(Score, 600, -60, std::to_string(score), Stext);
game_window.draw(Score);

game_window.display();
// програш - вивід Game Over
if (!game_state)
{
    Opt_Buttons(L"Game Over", L"Again", L"Exit", 2);

    game_window.close();

}
}
}
}

```

```

// Налаштування
void Settings_Game()
{
    RenderWindow Settings_Window(VideoMode(width_menu, height_menu),
    L"Settings", Style::Default);

    // Для гри і для астар
    std::vector<String> snake_sett{ L" Game mode", L" A* mode"};

    // Об'єкт меню
    SMenu mySettMenu(Settings_Window, 125, 260, 80, 225, snake_sett);

    // Колір і вирівнювання
    mySettMenu.setColorTextMenu(Color::Black, Color(53, 53, 255), Color::Cyan);
    mySettMenu.AlignMenu(1);

    Font font;
    font.loadFromFile("C://Users//Legion//Desktop//дурастіка//fonts//cool_font.ttf");

    // поточна швидкість змійки для гри
    Text Speed_game;
    Speed_game.setFont(font);

    // поточна швидкість змійки для алгоритма
    Text Speed_astar;
    Speed_astar.setFont(font);

    TextFormat Stext;
    Stext.size_font = 30;

```



```

Stext.menu_text_color = Color(23, 23, 147);

Stext.bord = 3;

// Фон

RectangleShape background_sett(Vector2f(width_menu, height_menu));

Texture texture_sett;

texture_sett.loadFromFile("C://Users//Legion//Desktop//дупастіка//photos//settings.JPG");

background_sett.setTexture(&texture_sett);

Clock clock;

while (Settings_Window.isOpen())
{
    Event event;
    while (Settings_Window.pollEvent(event))
    {

        if (event.type == Event::KeyPressed)
        {
            if (event.key.code == Keyboard::Up) { mySettMenu.MoveUp_Left(); }    //
вгору
            if (event.key.code == Keyboard::Down) { mySettMenu.MoveDown_Right(); }
} // вниз
            if (event.key.code == Keyboard::Escape) Settings_Window.close();
            if (event.key.code == Keyboard::Left)                // вліво -
зменшити
        {

```

```

        if (mySettMenu.getSelectedMenuNumber() == 0) game_delay += 0.01;

        if (mySettMenu.getSelectedMenuNumber() == 1) astar_delay += 0.01;
    }
    if (event.key.code == Keyboard::Right)                // вправо -
збільшити
    {
        if (mySettMenu.getSelectedMenuNumber() == 0) game_delay -= 0.01;

        if (mySettMenu.getSelectedMenuNumber() == 1) astar_delay -= 0.01;

    }
}

InitText(Speed_game, 350, 363, std::to_string(speed_converter(game_delay)),
Stext);

InitText(Speed_astar, 350, 587, std::to_string(speed_converter(astar_delay)),
Stext);

// Малюю все
Settings_Window.clear();
Settings_Window.draw(background_sett);
mySettMenu.draw();
Settings_Window.draw(Speed_game);
Settings_Window.draw(Speed_astar);
Settings_Window.display();
}
}

```

```

// Инициализация
void About_Game()
{
   RenderWindow About_Window(VideoMode(width_menu, height_menu), L"Игра", Style::Default);

    RectangleShape background_ab(Vector2f(width_menu, height_menu));
    Texture texture_ab;

    texture_ab.loadFromFile("C://Users//Legion//Desktop//дипломатика//photos//about.JPG");
    background_ab.setTexture(&texture_ab);

    while (About_Window.isOpen())
    {
        Event event_play;
        while (About_Window.pollEvent(event_play))
        {
            if (event_play.type == Event::Closed) About_Window.close();
            if (event_play.type == Event::KeyPressed)
            {
                if (event_play.key.code == Keyboard::Escape) About_Window.close();
            }
        }
        About_Window.clear();
        About_Window.draw(background_ab);
        About_Window.display();
    }
}

```

```

// Пауза або вікно програшу
void Opt_Buttons(const String& windowname, const String& buttonOne, const String&
buttonTwo, int num)
{
    RenderWindow OptButton(VideoMode(width_opt, height_opt), windowname,
Style::Default);

    // Далі-Заново + Вихід
    std::vector<String> snake_options{ buttonOne, buttonTwo };

    // Об'єкт меню
    SMenu myOptmenu(OptButton, 75, 120, 50, 120, snake_options);

    // Колір і вирівнювання
    myOptmenu.setColorTextMenu(Color::Black, Color(53, 53, 255), Color::Cyan);
    myOptmenu.AlignMenu(2);

    Font score_font;

    score_font.loadFromFile("C://Users//Legion//Desktop//дупастіка//fonts//cool_font.ttf");

    // рахунок
    Text Score;
    Score.setFont(score_font);

    TextFormat Stext;
    Stext.size_font = 40;
    Stext.menu_text_color = Color(255, 0, 130);

```

```
Stext.bord = 0;
```

```
// Фон паузи(1) або фон GAMEOVER(2)
```

```
RectangleShape background_options(Vector2f(width_opt, height_opt));
```

```
Texture texture_options;
```

```
if (num == 1)
```

```
texture_options.loadFromFile("C://Users//Legion//Desktop//дурастіка//photos//options  
_P.PNG");
```

```
if (num == 2)
```

```
texture_options.loadFromFile("C://Users//Legion//Desktop//дурастіка//photos//options  
_GO.PNG");
```

```
background_options.setTexture(&texture_options);
```

```
Clock clock;
```

```
while (OptButton.isOpen())
```

```
{
```

```
    Event event;
```

```
    while (OptButton.pollEvent(event))
```

```
    {
```

```
        if (event.type == Event::KeyPressed)
```

```
        {
```

```
            if (event.key.code == Keyboard::Left) { myOptmenu.MoveUp_Left(); }    //
```

```
ВЛІВО
```

```

        if (event.key.code == Keyboard::Right) { myOptmenu.MoveDown_Right();
    } // вправо

    if (event.key.code == Keyboard::Enter)                // ентер
    {
        switch (myOptmenu.getSelectedMenuNumber())
        {
            case 0: play_again = true; exit_snake = false; OptButton.close(); break;
            case 1: exit_snake = true; play_again = false; OptButton.close(); break;
            default: OptButton.close(); break;
        }
    }
}

```

```

InitText(Score, 220, 67, std::to_string(score), Stext);

```

```

// Малюю все
OptButton.clear();
OptButton.draw(background_options);
myOptmenu.draw();
OptButton.draw(Score);
OptButton.display();
}
}

```

```

// Відповідає за поновлення значень

```

```

void GameProcess(int x)

```

```

{
    while (playing)

```

```
{  
    GameStart(x);  
  
    if (play_again)  
    {  
        game_state = true;  
        border_touch = false;  
        exit_snake = false;  
        play_again = false;  
        pause_exit = false;  
        dir = 2;  
        num = 3;  
        score = 0;  
        playing = true;  
    }  
  
    if (exit_snake)  
    {  
        game_state = true;  
        border_touch = false;  
        exit_snake = false;  
        play_again = false;  
        pause_exit = false;  
        dir = 2;  
        num = 3;  
        score = 0;  
        playing = false;  
        break;
```

```

    }
}
}

```

Файл SnakeData.h:

```

#pragma once
#include <SFML/Graphics.hpp>
#include <string>

const int width_menu = 1375; // Ширина основного меню + поля гри
const int height_menu = 775; // Висота основного меню + поля гри

const int width_opt = 687; // Ширина вікна паузи
const int height_opt = 194; // Висота вікна паузи

const int W = 55; // Ширина в плитках всього вікна
const int H = 31; // Висота в плитках всього вікна
const int tile_size = 25; // Розмір кожної плитки

const int W_pole = 53; // Ширина в плитках ПОЛЯ
const int H_pole = 27; // Висота в плитках ПОЛЯ

enum MenuOption // Перелік пунктів меню
{
    PLAY_GAME,
    ASTAR,
    SETTINGS,
    ABOUT_GAME,
    QUIT_GAME
};

enum SnakeDirection // Навігація змійки
{
    Down,
    Left,
    Right,
    Up
};

struct TextFormat // Формат тексту
{

```



```

int size_font;
sf::Color menu_text_color;
float bord;
sf::Color border_color;
};

```

```

/*
*
0 1 2 3 4 5 . . . 52 53 54
1                               |
2 - - - - - - - - - - - -
3                               |
|                               |
|                               |
|                               |
28                              |
29                              |
30 - - - - - - - - - - - -

```

Ширина _ самого поля - 53 , починається [1 ; 53] або (0 ; 54)
Висота | самого поля - 27 , починається [3 ; 29] або (2 ; 30)
*/

Файл Astar_node.h:

```

#include <vector>
#include <iostream>
#include <string>
#include <algorithm>
#include <list>

using namespace std;

class Astar_class
{
private:

    struct sNode
    {
        bool isObstacle = false;           // Чи вузол перешкода
        bool isLearned = false;             // Чи вивчено вже вузол
        float distanceNow;                  // Відстань до кінця прямо зараз
    }

```

```

        float distanceAnother;                // Відстань до кінця якщо піти
іншим вузлом
        int x;                                // Координата вузла x
        int y;                                // Координата вузла y
        vector<sNode*> vector_Neighbours; // Зв'язок з сусідніми вузлами
        sNode* parent;                        // Батько вузла
    };

    sNode* nodes = nullptr;
    const int W_star = 54; // 54 (0; 54) [1; 53] [0; 54)
    const int H_star = 30; // 30 (2; 30) [3; 29] [3; 30)

    sNode* startNode = nullptr;
    sNode* endNode = nullptr;

    // розраховую відстань між вузлом а і б по діагоналі за теоремою піфагора
    float distance(sNode* a, sNode* b);

    void Astar_Solve();

public:

    void Astar_Create();

    int Astar_Update(int s_1_x, int s_1_y, int s_n_1_x, int s_n_1_y, int s_0_x, int
s_0_y, int f_x, int f_y);

};

```

Файл Astar_nodes.cpp:

```

#include "Astar_nodes.h"

void Astar_class::Astar_Create()
{
    // Створюю 2Д масив вузлів
    nodes = new sNode[W_star * H_star];

    for (int x = 1; x < W_star; x++)
        for (int y = 3; y < H_star; y++)
        {
            nodes[y * W_star + x].x = x; // Кожен вузол отримує свої x y
координати , розмір в плитках, звертатись:
            nodes[y * W_star + x].y = y; // nodes[координата_y *
Ширина_поля + координата_x].(x/y)

```

```

        nodes[y * W_star + x].isObstacle = false;
        nodes[y * W_star + x].parent = nullptr;
        nodes[y * W_star + x].isLearned = false;
    }

    // Створюю зв'язок між вузлами (GRID)
    // Для вузла за координатами додаю сусідній вузол в вектор де він буде
    сусідом.
    for (int x = 1; x < W_star; x++)
        for (int y = 3; y < H_star; y++)
        {
            if (y > 3)
                nodes[y * W_star +
x].vector_Neighbours.push_back(&nodes[(y - 1) * W_star + (x + 0)]);
            if (y < H_star - 1)
                nodes[y * W_star +
x].vector_Neighbours.push_back(&nodes[(y + 1) * W_star + (x + 0)]);
            if (x > 1)
                nodes[y * W_star +
x].vector_Neighbours.push_back(&nodes[(y + 0) * W_star + (x - 1)]);
            if (x < W_star - 1)
                nodes[y * W_star +
x].vector_Neighbours.push_back(&nodes[(y + 0) * W_star + (x + 1)]);
        }

    // Початкові рандомні координати, потім зміню
    startNode = &nodes[(H_star / 2) * W_star + 1];
    endNode = &nodes[(H_star / 2) * W_star + W_star - 2];
}

// розраховую відстань між вузлом а і б по діагоналі за теоремою піфагора
float Astar_class::distance(sNode* a, sNode* b)
{
    return sqrt((a->x - b->x) * (a->x - b->x) + (a->y - b->y) * (a->y - b->y));
}

void Astar_class::Astar_Solve()
{
    // онуляю деякі змінні щоб використати заново з нуля (залишаю перешкоди)
    for (int x = 1; x < W_star; x++)
        for (int y = 3; y < H_star; y++)
        {
            nodes[y * W_star + x].isLearned = false;
            nodes[y * W_star + x].distanceNow = INFINITY;
            nodes[y * W_star + x].distanceAnother = INFINITY;
        }
}

```

```

        nodes[y * W_star + x].parent = nullptr;
    }

    // задаю поточному вузлу дані початкового (голови змійки)
    sNode* currentNode = startNode;
    startNode->distanceAnother = 0.0f;
    startNode->distanceNow = distance(startNode, endNode);

    // початковий вузол одразу додаю в список , щоб його дослідити
    // в процесі, всі вузли будуть додаватись, щоб їх дослідити.
    std::list<sNode*> list_notTestedNodes;
    list_notTestedNodes.push_back(startNode);

    // якщо в списку є вузли, значить існують ще шляхи якими можна піти.
    // пошук припиниться якщо ми дійдемо до кінцевого вузла.

    // шукаю найкоротший шлях
    while (!list_notTestedNodes.empty() && currentNode != endNode)
    {
        currentNode = list_notTestedNodes.front();

        // досліджую вузол тільки раз
        currentNode->isLearned = true;

        // Перевіряємо кожного сусіда вузла
        for (auto neighbourNode : currentNode->vector_Neighbours)
        {
            // якщо вузол ще не досліджений, і він не перешкода, додаю в
            список
            if (!neighbourNode->isLearned && neighbourNode->isObstacle ==
            0)
                list_notTestedNodes.push_back(neighbourNode);

            // розраховую можливу найменшу відстань від вузла до нащадка
            (сусіда)
            float maybeLowerGoal = currentNode->distanceAnother +
            distance(currentNode, neighbourNode);

            if (maybeLowerGoal < neighbourNode->distanceAnother)
            {
                neighbourNode->parent = currentNode;
                neighbourNode->distanceNow = maybeLowerGoal;
                neighbourNode->distanceAnother = neighbourNode->
                >distanceNow + distance(neighbourNode, endNode);
            }
        }
    }

```

```

    }

    // сортує список щоб першими були вузли з найменшою відстанню
    list_notTestedNodes.sort([](const sNode* lhs, const sNode* rhs)
    {
        return lhs->distanceNow < rhs->distanceNow;
    }
    );

    // викидаю зі списку вже досліджені вузли
    while (!list_notTestedNodes.empty() && list_notTestedNodes.front()-
>isLearned)
        list_notTestedNodes.pop_front();

    // якщо більше нема вузлів в списку - кінець
    if (list_notTestedNodes.empty())
        break;
    }
}

int Astar_class::Astar_Update(int s_1_x, int s_1_y, int s_n_1_x, int s_n_1_y, int s_0_x,
int s_0_y, int f_x, int f_y)
{
    // початковий вузол - голова змійки
    startNode = &nodes[s_0_y * W_star + s_0_x];

    // кінцевий вузол - планетка
    endNode = &nodes[f_y * W_star + f_x];

    // перешкоди (тіло змійки)
    nodes[s_1_y * W_star + s_1_x].isObstacle = true;

    // прибираю статус перешкоди з хвостика
    nodes[s_n_1_y * W_star + s_n_1_x].isObstacle = false;

    Astar_Solve();

    if (endNode != nullptr)
    {
        sNode* p = endNode;
        sNode* prev = p;

        // проходжусь від планетки до голови
        while (p->parent != nullptr)
        {

```

```

        prev = p;
        // батько вузла - це вузол
        p = p->parent;
    }

    // визначаю напрямок голови
    if (prev->x > s_0_x)
        return 2;
    else if (prev->x < s_0_x)
        return 1;
    else if (prev->y > s_0_y)
        return 0;
    else if (prev->y < s_0_y)
        return 3;
    }
}

```

Файл SMenu.h:

```

#pragma once
#include <SFML/Graphics.hpp>

class SMenu
{
    float menu_X, menu_Y;           // Координати меню по X, Y
    int menu_Step;                  // Відстань між пунктами меню
    int max_menu;                   // Кількість пунктів меню
    int size_font;                  // Розмір шрифту
    int mainMenuSelected;           // Номер вибраного пункту меню
    sf::Font font;                  // Шрифт тексту меню
    std::vector<sf::Text> mainMenu; // Назви меню, динамічний масив

    sf::Color menu_text_color;      // Колір тексту пунктів меню
    sf::Color chose_text_color;     // Колір вибраного пункту меню
    sf::Color border_color;         // Колір обведення тексту пунктів меню

    // Налаштування тексту меню
    void setInitOptText(sf::Text& text, const sf::String& str, float xpos, float ypos)
const;

    sf::RenderWindow& mywindow;     // Посилання на граф.вікно

public:

```

```
SMenu(sf::RenderWindow& window, float menux, float menuy, int sizeFont, int
step, std::vector<sf::String>& name);
```

```
void draw();           // Малювання меню
```

```
void MoveUp_Left();    // Переміщення обраного пункту вгору-вліво
```

```
void MoveDown_Right(); // Переміщення обраного пункту вниз-вправо
```

```
// Колір пунктів меню
```

```
void setColorTextMenu(sf::Color menColor, sf::Color ChoColor, sf::Color
BordColor);
```

```
void AlignMenu(int posX); // Вирівнювання пунктів
```

```
int getSelectedMenuNumber() // Повертає номер вибраного пункту
```

```
const
```

```
{
```

```
    return mainMenuSelected;
```

```
}
```

```
};
```

Файл SMenu.cpp:

```
#include "SnakeMenu.h"
```

```
#include <vector>
```

```
void SMenu::setInitOptText(sf::Text& text, const sf::String& str, float xpos, float ypos)
```

```
const
```

```
{
```

```
    text.setFont(font);
```

```
    text.setFillColor(menu_text_color);
```

```
    text.setString(str);
```

```
    text.setCharacterSize(size_font);
```

```
    text.setPosition(xpos, ypos);
```

```
    text.setOutlineThickness(3);
```

```
    text.setOutlineColor(border_color);
```

```
}
```

```
void SMenu::AlignMenu(int MenuNum)
```

```
{
```

```
    if (MenuNum == 1)
```

```

    {
        for (int i = 0; i < max_menu; i++) {
            mainMenu[i].setPosition(mainMenu[i].getPosition().x,
mainMenu[i].getPosition().y);
        }
    }
    else
    {
        mainMenu[0].setPosition(mainMenu[0].getPosition().x,
mainMenu[0].getPosition().y);
        mainMenu[1].setPosition(mainMenu[0].getPosition().x + 350,
mainMenu[0].getPosition().y);
    }
}

```

```

SMenu::SMenu(sf::RenderWindow& window, float menux, float menuy, int sizeFont,
int step, std::vector<sf::String>& name)
    :menu_X(menux), menu_Y(menuy), menu_Step(step),
max_menu(static_cast<int>(name.size())), size_font(sizeFont), mainMenu(name.size()),
mywindow(window)
{
    font.loadFromFile("C://Users//Legion//Desktop//дѣпактїка//fonts//cool_font.ttf");

    for (int i = 0, ypos = static_cast<int>(menu_Y); i < max_menu; i++, ypos +=
menu_Step)
        setInitOptText(mainMenu[i], name[i], menu_X, static_cast<float>(ypos));
    mainMenuSelected = 0;
    mainMenu[mainMenuSelected].setFillColor(chose_text_color);
}

```

```

void SMenu::MoveUp_Left()
{
    mainMenuSelected--;

    if (mainMenuSelected >= 0) {
        mainMenu[mainMenuSelected].setFillColor(chose_text_color);
        mainMenu[mainMenuSelected + 1].setFillColor(menu_text_color);
    }
    else
    {
        mainMenu[0].setFillColor(menu_text_color);
        mainMenuSelected = max_menu - 1;
        mainMenu[mainMenuSelected].setFillColor(chose_text_color);
    }
}

```



```

void SMenu::MoveDown_Right()
{
    mainMenuSelected++;

    if (mainMenuSelected < max_menu) {
        mainMenu[mainMenuSelected - 1].setFillColor(menu_text_color);
        mainMenu[mainMenuSelected].setFillColor(chose_text_color);
    }
    else
    {
        mainMenu[max_menu - 1].setFillColor(menu_text_color);
        mainMenuSelected = 0;
        mainMenu[mainMenuSelected].setFillColor(chose_text_color);
    }
}

void SMenu::draw()
{
    for (int i = 0; i < max_menu; i++) mywindow.draw(mainMenu[i]);
}

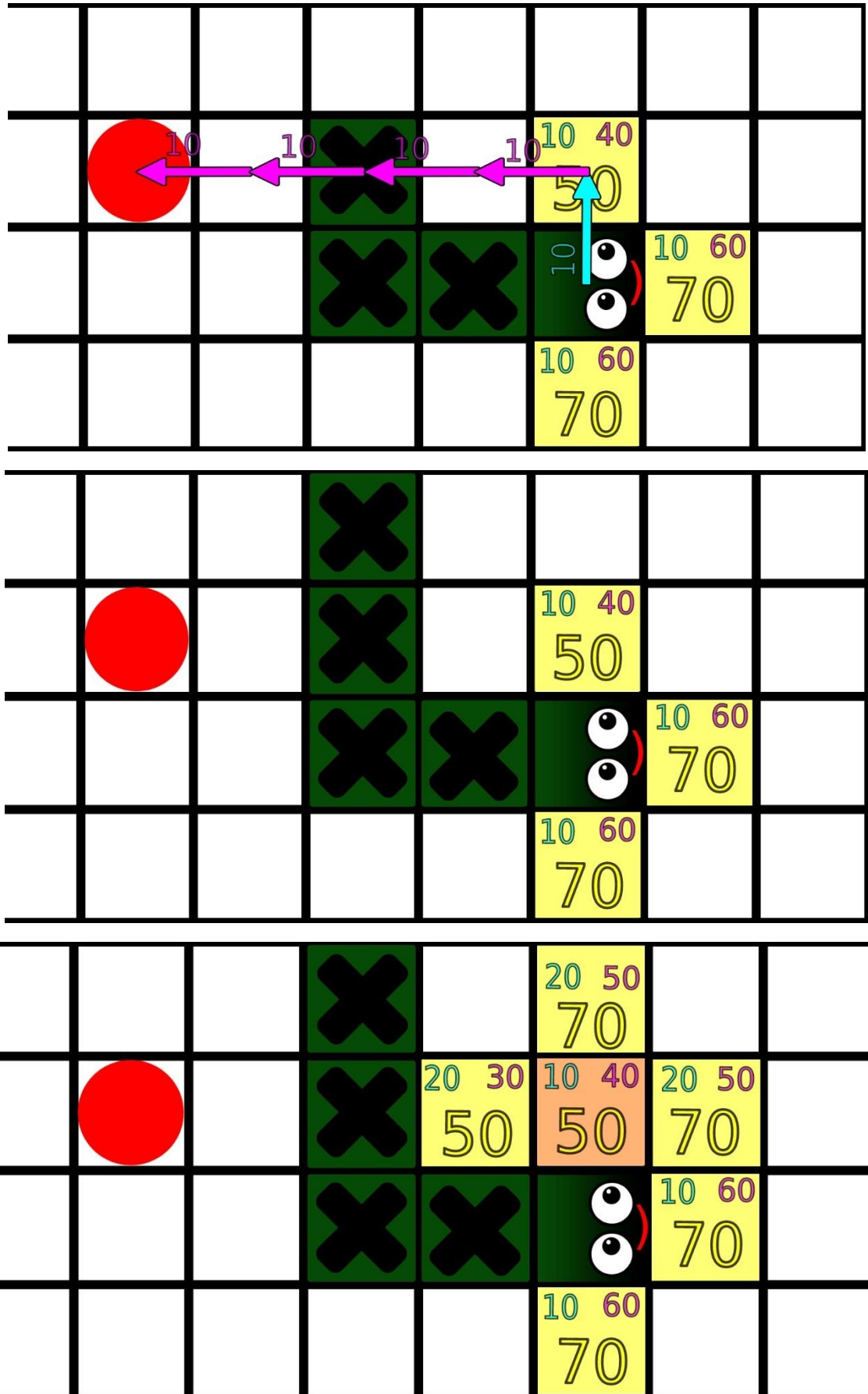
void SMenu::setColorTextMenu(sf::Color menColor, sf::Color ChoColor, sf::Color
BordColor)
{
    menu_text_color = menColor;
    chose_text_color = ChoColor;
    border_color = BordColor;







    for (int i = 0; i < max_menu; i++) {
        mainMenu[i].setFillColor(menu_text_color);
        mainMenu[i].setOutlineColor(border_color);
    }


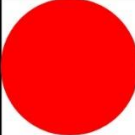




    mainMenu[mainMenuSelected].setFillColor(chose_text_color);
}







```


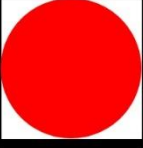




Додаток Б. Покрокове пояснення алгоритму


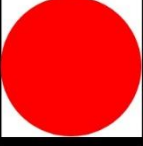







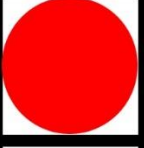




				30 40 70	20 50 70		
				20 30 50	10 40 50	20 50 70	
						10 60 70	
					10 60 70		


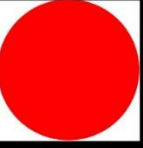




				30 40 70	20 50 70		
				20 30 50	10 40 50	20 50 70	
						10 60 70	
					10 60 70		


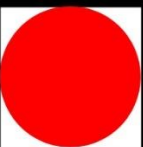




				30 40 70	20 50 70	30 60 90	
				20 30 50	10 40 50	20 50 70	
						10 60 70	
					10 60 70		


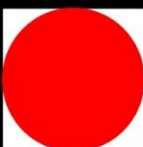




				30 40 70	20 50 70	30 60 90	
				20 30 50	10 40 50	20 50 70	30 60 90
						10 60 70	
					10 60 70		

				30 40 70	20 50 70	30 60 90	
				20 30 50	10 40 50	20 50 70	30 90
						10 60 70	20 70 90
					10 60 70	20 70 90	

				30 40 70	20 50 70	30 60 90	
				20 30 50	10 40 50	20 50 70	30 90
						10 60 70	20 70 90
				20 50 70	10 60 70	20 70 90	

				30 40 70	20 50 70	30 60 90	
				20 30 50	10 40 50	20 50 70	30 90
						10 60 70	20 70 90
				30 40 70	20 50 70	10 60 70	20 70 90

				30 40 70	20 50 70	30 60 90	
				20 30 50	10 40 50	20 50 70	30 90
						10 60 70	20 70 90
				40 30 70	30 40 70	20 50 70	10 60 70

				30 40 70	20 50 70	30 60 90	
				20 30 50	10 40 50	20 50 70	30 90
						10 60 70	20 70 90
				50 20 70	40 30 70	30 40 70	20 50 70

			×	30 40 70	20 50 70	30 60 90	
	●	40 10 50	×	20 30 50	10 40 50	20 50 70	30 90
	40 10 50	30 20 50	×	×	●●	10 60 70	20 70 90
	50 20 70	40 30 70	30 40 70	20 50 70	10 60 70	20 70 90	

		50 20 70	×	30 40 70	20 50 70	30 60 90	
	●	50 0 50	×	20 30 50	10 40 50	20 50 70	30 90
	40 10 50	30 20 50	×	×	●●	10 60 70	20 70 90
	50 20 70	40 30 70	30 40 70	20 50 70	10 60 70	20 70 90	

		50 20 70	×	30 40 70	20 50 70	30 60 90	
	●	50 0 50	×	20 30 50	10 40 50	20 50 70	30 90
	40 10 50	30 20 50	×	×	●●	10 60 70	20 70 90
	50 20 70	40 30 70	30 40 70	20 50 70	10 60 70	20 70 90	