МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

автоматизованих систем обробки інформації та управління

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліна «Основи програмування»

(назва дисципліни)

на тему: «Гра у 15»

Студента 1 курсу, групи ІП-01

Зараніка Богдана Юрійовича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник ст.в. Головченко М. М.

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  | К.т.н.доц. Муха. І.П |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  | Ст.в. Головченко М.М. |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ- 2021 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Основи програмування

Напрям "Програмна інженерія"

Курс 1 Група ІП-01 Семестр 2

**ЗАВДАННЯ**

на курсову роботу студента

Зараніка Богдана Юрійовича

(прізвище, ім’я, по батькові)

1. Тема роботи реалізація програмного забезпечення, що моделює гру-головоломку

«Гра у 15». Реалізація алгоритму видачі підказки щодо наступного ходу.

2. Строк здачі студентом закінченої роботи 02 червня 2021 року

3. Вихідні дані до роботи: початкова комбінація гри, логін користувача для подальшого

збереження інформації щодо результату завершеної гри у спеціально відведений файл.

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

Розрахунково-пояснювальна записка вносить ясність у суть головоломки «Гра у 15»,

її правила, деталі реалізації алгоритмів пошуку підказки, перевірки комбінації на розв’язність,

підрахунку евристики, особливості архітектури розробленого ПЗ(діаграма класів та опис використаних у роботі методів класів), проводить детальний аналіз тестування роботи а також містить інструкцію користувача, що пояснює, як користуватися розробленим програмим забезпеченням.

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

6. Дата видачі завдання 03.03.2021

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи | 03.03.2021 |  |
| 2. | Підготовка ТЗ | 11.03.2021 |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи | 23.03.2021 |  |
| 4. | Розробка сценарію роботи програми | 05.04.2021 |  |
| 5. | Узгодження сценарію роботи програми з керівником | 05.04.2021 |  |
| 6. | Розробка (вибір) алгоритму рішення задачі | 05.04.2021 |  |
| 7. | Узгодження алгоритму з керівником | 05.04.2021 |  |
| 8. | Узгодження з керівником інтерфейсу користувача | 05.04.2021 |  |
| 9. | Розробка програмного забезпечення | 12.04.2021 |  |
| 10. | Налагодження розрахункової частини програми | 24.04.2021 |  |
| 11. | Розробка та налагодження інтерфейсної частини програми | 11.05.2021 |  |
| 12. | Узгодження з керівником набору тестів для контрольного прикладу | 20.05.2021 |  |
| 13. | Тестування програми | 23.05.2021 |  |
| 14. | Підготовка пояснювальної записки | 31.05.2021 |  |
| 15. | Здача курсової роботи на перевірку | 02.06.2021 |  |
| 16. | Захист курсової роботи | 04.06.2021 |  |

Студент

(підпис)

Керівник ст.в.Головченко М. М.

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

“02” червня 2021 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 88 сторінок, 12 рисунків, 19 таблиць, 3 посилання.

Об’єкт дослідження: головоломка «Гра у 15»

Мета роботи: дослідження генерації та розв’язання головоломки «Гра у 15»

Виконана програмна реалізація алгоритмів генерації та реалізація головоломки «Гра у 15», розроблений зручний графічний інтерфейс для взаємодії з алгоритмами.

У розділі 2 детально розписано суть гри, її правила.

У розділі 3 – записано алгоритми, що використовуються у даній роботі.

У розділі 4 зображено діаграму класів та розписано архітектуру програмного забезпечення.

У 5-у розділі проведено план тестування та саме тестування, подане у вигляді таблиць тест кейсів.

6-й розділ присвячено інструкції користувача. У ній детально описано весь хід роботи програми у дрібних деталях, розказано, що відбувається після кожного можливого кроку.

ГОЛОВОЛОМКА «ГРА У 15», МОДЕЛЮВАННЯ ГОВОЛОМКИ «ГРА У 15»

зміст

[Вступ 6](#_Toc73547866)

[1 Постановка задачі 8](#_Toc73547867)

[2 Теоретичні відомості 9](#_Toc73547868)

[2.1 Суть гри-головоломки «Гра у 15» 9](#_Toc73547869)

[3 Опис алгоритмів 14](#_Toc73547870)

[3.1 Загальний алгоритм 14](#_Toc73547871)

[3.2 Алгоритм перевірки на зібраність комбінації (Field::isAssembled) 15](#_Toc73547872)

[3.3 Алгоритм підрахування евристики (Field::getHeuristic) 15](#_Toc73547873)

[3.4 Алгоритм встановлення можливості вирішення комбінації (Field::canBeAssembled) 16](#_Toc73547874)

[3.5 Алгоритм змінення матриці гри(хід)(Field::changeMap) 17](#_Toc73547875)

[3.6 Алгоритм генерації комбінації (Game::generateCombination) 17](#_Toc73547876)

[3.7 Алгоритм видачі підказки (Game::solve) 18](#_Toc73547877)

[4 Опис програмного забезпечення 21](#_Toc73547878)

[4.1 Діаграма класів програмного забезпечення 21](#_Toc73547879)

[4.2 Опис методів частин програмного забезпечення 22](#_Toc73547880)

[4.2.1 Стандартні методи 22](#_Toc73547881)

[4.2.2 Користувацькі методи 30](#_Toc73547882)

[5 План тестування 37](#_Toc73547883)

[5.1 План тестування 37](#_Toc73547884)

[5.2 Приклади тестування 38](#_Toc73547885)

[6 Інструкція користувача 46](#_Toc73547886)

[6.1 Робота з програмою 46](#_Toc73547887)

[6.2 Формат вхідних та вихідних даних 51](#_Toc73547888)

[6.3 Системні вимоги 51](#_Toc73547889)

[Висновки 53](#_Toc73547890)

[Перелік посилань 54](#_Toc73547891)

[ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ 55](#_Toc73547892)

[Додаток Б Тексти програмного коду 58](#_Toc73547893)

Вступ

Коли ми їдемо у транспорті, чекаємо у черзі, або знаходимось будь-де, де в нас багато вільного часу, хочеться мати щось, чим заняти цей вільний час. Тут у пригоді можуть стати різні міні-ігри, головоломки, цікаві задачі. Про одну з таких піде мова далі.

Тема моєї курсової роботи – реалізація гри-головоломки під назвою «Гра у 15». Дана гра з’явилася наприкінці у 1880-х роках у Сполучених Штатах Америки. Гра являє собою квадратну коробочку, у якій уміщаються 15 квадратиків із числами від 1 до 15. Один квадратик залишається порожнім. Потрібно із заплутаної комбінації скласти таку, яка представлена на Таблиці 1.1:

Зібрана комбінація

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 |  |

Попри те, що даній головоломці налічується більше 100 років, вона є й досі популярною серед поціновувачів розваг такого жанру. Дана гра є головоломкою, її досить важко вирішити для довільної комбінації. Винахідником головоломки був Ной Палмер Чепмен, поштмейстер з [Канастоти](https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%B0_(%D0%9D%D1%8C%D1%8E-%D0%99%D0%BE%D1%80%D0%BA)), який ще в [1874 році](https://uk.wikipedia.org/wiki/1874) показував друзям головоломку, що складалася з шістнадцяти пронумерованих квадратиків, які треба було скласти в ряди по чотири штуки так, щоб сума чисел в кожному ряду була рівна 34.

Моїм завдання було змоделювати дану гру. А саме: користувач повинен мати змогу авторизуватися для збереження статистики та грати у «Гру у 15», клікаючи на будь-які квадратики. Квадратики, які межують із порожнім місцем можуть рухатись, тож при кліку на них повинен відбуватися хід, а от при натисканні на квадратик, що не має суміжного з ним порожнього місця при кліку рухатись не має – це є природнім. Взагалі, моєю головною метою є зробити гру максимально схожою на реальну, щоб захопити та зацікавити нею користувача. Але також головною деталлю моєї програми стала можливість видавати підказку щодо наступного ходу, що є дуже зручним для користувача. Також я вважав за потрібне виводити велику кількість повідомлень у спливаючих вікнах для інформації користувача. Таким чином користувач чітко розумітиме, що і як у програмі працює.

Програму буде реалізовано із метою зробити її максимально зручною для використання гравцем. На усіх кнопках було вирішено написати великі за розміром підказки, для того, щоб користувач мав змогу легко зрозуміти, що вони роблять.

# Постановка задачі

Розробити програмне забезпечення, що буде надавати користувачу можливість розв’язати або отримати підказку щодо наступного кроку для головоломки «Гра у 15» розмірності 4х4.

Вхідними даними для даної роботи є початкова комбінація гри – матриця 4х4, що заповнена числа 1-16, кожне з яких зустрічається один раз. Матриця вводиться з файлу або генерується програмно. При введенні комбінації з файлу програмна система повинна перевіряти комбінацію на розв’язність(і у разі подання нерозв’язної комбінації завершує гру з виведенням відповідного повідомлення на екран). Програмно генерується завжди розв’язна комбінація.

На початку гри користувач вводить свій логін у спеціальну форму з текстовим полем. Якщо файл з таким іменем уже існує, то відкрити його, інакше перед відкриттям створити у папці програми.

Якщо користувач захотів розв’язати головоломку, то програмне забезпечення повинно обробляти введені дані та перевіряти задану комбінацію на розв’язність, а потім обробляти розв’язок користувача для заданої головоломки та видавати, за забаганкою користувача, підказку щодо наступного ходу.

Вихідними даними програми являється сукупність цілих чисел від 1 до N, що є підказками щодо наступного ходу згенерованої головоломки, які виводяться на екран. Якщо користувач хоче отримати підказку під час гри, то програмне забезпечення виведе вікно повідомлення з необхідним для рішення числом(костяшка на яку потрібно натиснути). Якщо користувач розв’язав комбінацію , то він буде повідомлений про це, а його результат буде занесений до таблиці результатів(файлу з його логіном) з датою, коли комбінацію було вирішено. Якщо користувач передчасно завершив гру, то у файл з його результатами буде виведено відповідний запис з датою та часом завершення.

# Теоретичні відомості

## Суть гри-головоломки «Гра у 15»

«П'ятнашки» представляють собою квадратну коробочку, в комплект якої входить 15 однакових пластикових фішок із нанесеними цифрами (в коробочці наявне одне незаповнене квадратне поле для проведення маневру). Мета логічної «гри 15» – пересувати по коробочці нумеруванні фішки так, щоб цифри на них були побудовані в порядку зростання від 1-го до 15-ти (останній квадратик залишається порожнім).

Якщо квадрат, що містить число , з’являється перед числами, меншими ніж , то назвемо це інверсією порядку . Покладемо

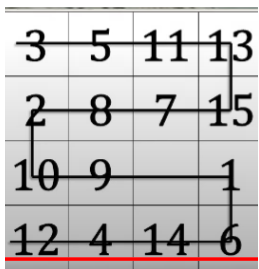
|  |  |
| --- | --- |
|  | (2.1) |

Головне твердження - якщо, то комбінація нерозв’язна, де – номер рядка, де знаходиться порожня клітинка. Інакше комбінація буде розв’язною.

Доведення даного твердження:

Розглянемо частковий випадок, коли .

Випишемо елементи матриці(без порожнього) за таким законом (Рисунок 3.1):



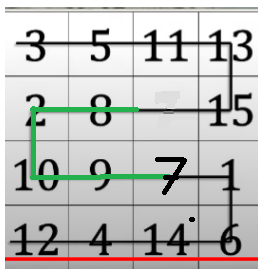
* + - * 1. Заповнення «змійкою»

Тобто: 3, 5, 11, 13, 15, 7, 8, 2, 10, 9, 1, 12, 4, 14, 6

Очевидно, що у зібраному варіанті кількість інверсій рівна нулю (парному числу).

Тоді перемістимо порожню клітинку горизонтально. У такому разі сама перестановка не зміниться. Тож не зміниться і кількість інверсій у ній.

Тепер перемістимо порожню клітинку вертикально(Рисунок 3.2)



* + - * 1. Переміщення порожньої клітинки вертикально

Роздивимось числа, що виділені зеленою лінією.

У початковому варіанті було 7, 8, 2, 10, 9.

Стало 8, 2, 10, 9, 7.

Позначимо їх – для загальності випадку.

Тоді після «вертикального кроку» дана перестановка виглядатиме так:

Якщо кількість перестановок з була 1, то після такого переміщення минула перестановка зникне, натомість з’являться ще 3. Таким чином загальна кількість перестановок у фрагменті збільшиться на 2 – парне число.

Розглядаючи випадки кількості початкових перестановок від 2 до 4 та також 0 переконаємось, що кожного разу кількість перестановок змінюється на парну кількість.

Також варто помітити, що кількість клітинок, перетнутих зеленою лінією завжди буде непарною(це насправді очевидно), тож наше доведення цілком строге, адже легко екстраполювати його на випадок з іншою кількістю перетнутих зеленою лінією квадратиків.

У цьому частковому випадку, , маємо: комбінація є розв’язною коли:

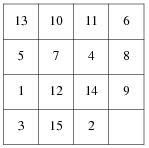
– парне, тому – парне (адже тільки в цьому разі ми можемо привести кількість інверсій до 0, кожного кроку змінюючи їх на парну кількість).

Якщо , то доведення аналогічне. Тому вважатимемо очевидним твердження, що при – парне є розв’язок, а інакше – ні.

Таким чином рівно половина комбінацій(тих, чия сума - непарна) є нерозв’язною. Натомість усі комбінації, яку маю парну таку суму є розв’язними.

Кількість різних комбінацій рівна .

Наприклад:



* + - * 1. Приклад нерозв’язної комбінації

Для даної комбінації масив інверсій має вигляд:

 , , – непарне. Отже дана комбінація є нерозв’язною.

За один крок можна зсунути лише одну клітинку(класична версія гри). І зробити це можна лише у випадку, коли вона межує з порожньою.

Алгоритм розв’язку базується на наступній ідеї. Опишемо нашу задачу у термінах теорії графів. Вершиною будемо називати будь-який стан гри (поле), а ребра – перехід (довжиною в один хід) від одного стану до іншого. Таким чином для вирішення задачі можна застосувати такі алгоритми пошуку шляху у графі як А\*, IDA\*, bfs, dfs та інші. Таким чином ми встановлюємо стартовою вершину початкового стану та намагаємося дістатися кінцевого положення(термінальної вершини) за допомогою будь-якого з даних алгоритмів. Невеликою проблемою даної задачі є те, що через неймовірну кількість більшість комбінацій, що мають знаходяться «на відстані» 20 та більше кроків від термінальної вершини зазвичай неможливо вирішити за прийнятний час будь-яким з даних алгоритмів, тому вирішенням даної проблеми стане використання модифікації одного з перелічених вище алгоритмів, а саме лімітованого бінарного пошуку. Модифікація полягає в наступному: за допомогою алгоритму пошуку в ширину (найшвидшого з перелічених, який можна використовувати в модифікації) знаходити усі кінцеві положення на відстані від термінальної вершини не більше наперед заданого числа. У ході алгоритму будуватиметься дерево усіх розв’язків. Проходячись по усіх листових вершинах даного дерева, знайдемо той листок, який евристично знаходиться найближче до термінальної вершини, а на зворотньому обході з’ясуємо, весь шлях, який за властивість пошуку в ширину буде найкоротшим, до даної листової вершини. Знаючи весь шлях, зрозуміємо, який крок має бути наступним.

Евристик до даної задачі вигадано багато, але кожна окремо не дає гарної диференціації вершин, оскільки кожна з них добре враховує лише певні комбінації, а інші диференціює за відстанню до термінальної вершини слабо. Тому я прийняв рішення рахувати евристику як суму найбільш відомих евристик до даної задачі, щоб максимально точно знаходити найкращий хід:

* евристика «не на своєму місці» - просто рахуємо, скільки чисел не стоять на своїх місцях.
* евристика «лінійний конфлікт» - якщо два числа в одному рядку або в одному стовпці утворюють інверсію, то додамо до суми усіх евристик значення 2 (емпіричним чином визначене значення)
* евристика «кутові фішки» - розглянемо правий верхній кут. Нехай числа «3» та «8» стоять на своїх позиціях, а замість «4» - будь-яке інше число. Тоді додамо до суми усіх евристик 2, оскільки як мінімум 2 рази число «3» або «8» доведеться пересунути(відвести у сторону, просунути 4, та встановити на початкове місце).

# Опис алгоритмів

Перелік всіх основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
|  | Унікальний логін користувача |
|  | Ознака розв’язності комбінації, зчитаної з файлу |
|  | Перелічувальний тип, ознака походження комбінації(генерація/з файлу) |
|  | Матриця гри |
|  | Листок, що є кінцем потрібного шляху |
|  | Проміжне значення last у пошуку |
|  | Множина вершин, які не будуть розглядатися при пошуку листа з мінімальною евристикою |
|  | Масив «векторів» «зсуву» порожньої клітинки в матриці гри |

## Загальний алгоритм

1. ПОЧАТОК
2. Зчитати ім’я користувача:
   1. ЯКЩО ім’я порожнє, вивести відповідне повідомлення та запросити його заново.
3. Запросити вибрати спосіб отримання комбінації(з файлу або згенерувати).
4. ЯКЩО вибрано спосіб отримання комбінації «з файлу», ТО зчитати з файлу input.txt комбінацію, ІНАКШЕ – згенерувати її.
5. ЯКЩО комбінація розв’язна, ТО запустити гру. ІНАКШЕ видати повідомлення про помилку та перейти до пункту 7.
6. Гра запущена:
   1. ЯКЩО гравець натискає на костяшку, з якої можливий хід, ТО відбувається хід
      1. ІНАКШЕ ЯКЩО натиснуто кнопку підказки, то програма видає підказку щодо наступного ходу,
      2. ІНАКШЕ ЯКЩО натиснуто кнопку виходу з гри, ТО
         1. ЯКЩО комбінація зібрана, вивести повідомлення про перемогу. ІНАКШЕ вивести повідомлення про програш.
7. КІНЕЦЬ

## Алгоритм перевірки на зібраність комбінації (Field::isAssembled)

1. ПОЧАТОК
2. Створити зібрану матрицю 6х6 елементів типу int:
   1. Заповнити її значеннями -1.
   2. ЦИКЛ i проходу від 2-ї строки до передостанньої
   3. ЦИКЛ j проходу від 2-го стовпця до передостаннього
   4. ЦИКЛ i проходу від 2-ї строки до передостанньої
   5. ЦИКЛ j проходу від 2-го стовпця до передостаннього
      1. ЯКЩО елемент з індексами i, j не дорівнює відповідному елементу у вхідній матриці, ТО повернути FALSE та перейти до кроку 3.
   6. За замовчуванням повернути TRUE.
3. КІНЕЦЬ

## Алгоритм підрахування евристики (Field::getHeuristic)

1. ПОЧАТОК
2. ЦИКЛ i проходу від 2-ї строки до передостанньої
3. ЦИКЛ j проходу від 2-го стовпця до передостаннього
4. ЦИКЛ i проходу від 2-ї строки до передостанньої
5. ЦИКЛ j проходу від 2-го стовпця до передостаннього
7. ЯКЩО
   1. ЯКЩО , ТО
8. ЯКЩО
   1. ЯКЩО , ТО
9. ЯКЩО
   1. ЯКЩО , ТО
10. КІНЕЦЬ

## Алгоритм встановлення можливості вирішення комбінації (Field::canBeAssembled)

1. ПОЧАТОК
2. ЦИКЛ i проходу від 2-ї строки до передостанньої
3. ЦИКЛ j проходу від 2-го стовпця до передостаннього
   1. ЯКЩО , ТО
4. ЦИКЛ i проходу від 2-ї строки до передостанньої
5. ЦИКЛ j проходу від 2-го стовпця до передостаннього
   1. ЯКЩО , ТО
6. ЦИКЛ i проходу від 1-го до передостаннього елемента snake
7. ЦИКЛ j проходу від i+1-го до останнього елемента snake
8. Повернути
9. КІНЕЦЬ

## Алгоритм змінення матриці гри(хід)(Field::changeMap)

1. ПОЧАТОК
2. ЯКЩО , ТО
   1. обміняти місцями та
   2. Повернути TRUE
   3. Перейти на крок 7
3. ЯКЩО , ТО
   1. обміняти місцями та
   2. Повернути TRUE
   3. Перейти на крок 7
4. ЯКЩО , ТО
   1. обміняти місцями та
   2. Повернути TRUE
   3. Перейти на крок 7
5. ЯКЩО , ТО
   1. обміняти місцями та
   2. Повернути TRUE
   3. Перейти на крок 7
6. Повернути FALSE
7. КІНЕЦЬ

## Алгоритм генерації комбінації (Game::generateCombination)

1. ПОЧАТОК
2. ЦИКЛ i проходу від 2-ї строки до передостанньої
3. ЦИКЛ j проходу від 2-го стовпця до передостаннього
4. ЦИКЛ 20 разів:
   1. РОБИТИ
   2. ПОКИ

* 1. Обміняти місцями та

1. КІНЕЦЬ

## Алгоритм видачі підказки (Game::solve)

1. ПОЧАТОК
2. ЯКЩО гравець не зробив кроку з минулого запиту підказки та вже давав запити на підказку, ТО
   1. Повернути останню видану підказку
   2. Перейти до кроку 4
3. ІНАКШЕ:
   1. Покладемо максимальну глибину спуску(MAXDEPTH) рівною 17
   2. Масив можливих зсувів порожньої клітинки
   3. Для стартової вершини відстань, яку ми до неї пройшли покладемо рівною 0
   4. Для стартової вершини евристику покладемо рівною евристиці від f
   5. Предок стартової вершини покладемо рівним NULL
   6. Множина пройдених вершин
   7. Додамо стартову вершину до черги
   8. Додамо до масиву усіх вершин початкову
   9. ПОКИ черга не порожня:
      1. Запишемо у перший елемент черги.
      2. ЯКЩО не належить множині та відстань від початкової вершини до не більша за MAXDEPTH:
         1. ЯКЩО є зібраною ТО:
            1. Вийти з циклу
         2. ІНАКШЕ:
            1. Додамо до множини вершину
            2. ЦИКЛ для від 1 до 4:
            3. ЯКЩО крок при зсуві порожньої клітинки на вектор можливий ТО та при такому кроці комбінація не присутня в :

Створимо нову вершину дерева із комбінацією, що відрізняється від нинішньої зсувом порожньої клітинки на вектор , як нащадок нинішньої комбінації

Додамо створену вершину до черги

Додамо створену вершину до масиву

* + 1. Видалимо елемент з голови черги
  1. Створимо множину .
  2. Створимо вершину
  3. ПОКИ :
     1. Створимо вершину
     2. Встановимо рівною ∞
     3. ЦИКЛ по всіх елементах масиву :
        1. ЯКЩО нинішня вершина є листовим та не належить множині , ТО:
           1. ЯКЩО менше евристики нинішньої вершини, ТО:

Встановимо рівним евристиці нинішньої вершини

У запишемо нинішню вершину

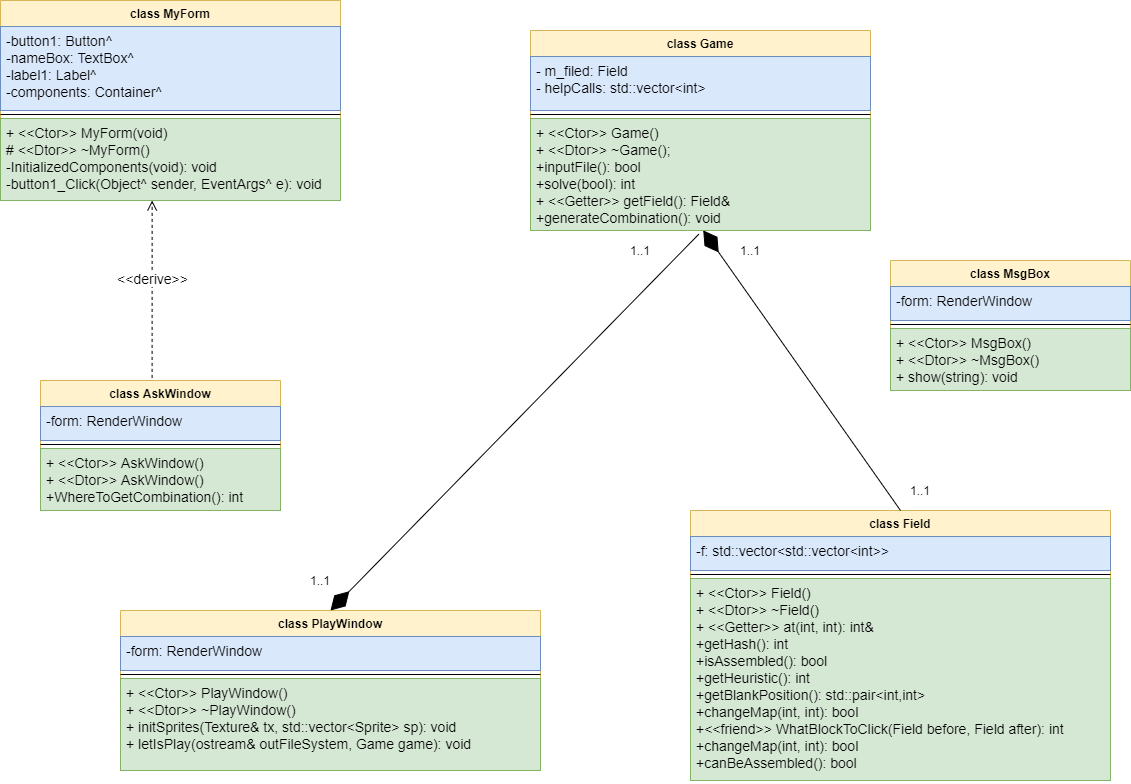
* + 1. Збережемо вершину у змінній
    2. ПОКИ не є коренем дерева ТА батько не є коренем дерева ТА батько батька не є коренем дерева, ТО:
       1. Встановимо рівним його батьку.
    3. ЯКЩО користувач ще не запитував підказку АБО крок від батька до не є таким, який видавався на минулому запиті, ТО:
    4. ІНАКШЕ:
       1. Додати вершину до
  1. ПОВЕРНУТИ крок від батька до

1. КІНЕЦЬ

# Опис програмного забезпечення

## Діаграма класів програмного забезпечення

Діаграма класів програмного забезпечення зображена на рис. 5.1:



* + - * 1. Діаграма класів

У моїй програмі наявно 6 класів, 4 з них – це класи вікон графічного інтрефейсу та вікон повідомлень. Запуск програми починається з класу MyForm, який зчитує логін гравця. Після його роботи починає працювати клас AskWindow, головна задача якого – встановити яким чином гравець хоче ввести в програму вхідні дані(з файлу input.txt або згенерувати). Даний клас знаходиться у відношення залежності(а саме <<derive>>) з класом MyForm, оскільки останній передає через змінну логін користувача, який необхідний для подальшої роботи програми) після його зчитування. Клас MsgBox – аналог вбудованого класу MessageBox, який у більш гарному візуальному форматі виводить підказки в процесі гри та повідомлення про з результатом гри.

Головний клас – PlayWindow, з нього починається гра. Він слугує в основному для відображення інтерфейсу гри, ренедрингу зображень «костяшок» та реагування на натискання кнопок видачі підказок та виходу з гри. Він включає в себе клас Game, який слугує для зберігання поточного стану матриці гри а також для запуску алгоритму прораховування підказки. В ньому також присутні методи, зчитують комбінацію із файлу чи генерують її.

Клас Field – зручна обгортка над вектором векторів стандартної бібліотеки шаблонів, що має методи для зміни поточного стану матриці гри(по суті імплементації ходу), перевірки комбінації на розв’язність та зібраність. Знаходиться у відношення композиції з класом Game, оскільки є невід’ємною його частиною – без матриці гри гра не маю сенсу.

## Опис методів частин програмного забезпечення

### Стандартні методи

У таблиці 4.1 наведено стандартні методи, які були використані для реалізації поставленої задачі.

Стандартніметоди

| № п/п | Назва заголовного файлу | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | <msclr/marshal\_cppstd.h> | MyForm | EnableVisualStyles | Цей метод включає візуальні стилі для програми. Візуальні стилі - це кольори, шрифти і т.д. | - | - |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 2 | <msclr/marshal\_cppstd.h> | MyForm | SetCompatibleTextRenderingDefault | Встановлює значення за замовчуванням для нових елементів керування. | Флаг використання графічного інтерфейсу GDI+ | - |
| 3 | <msclr/marshal\_cppstd.h> | MyForm | Run | Запускає форму | - | - |
| 4 | <msclr/marshal\_cppstd.h> | MyForm | button1\_Click | Обробник події кліку на кнопку | Посилання на об’єкт, тип події | - |
| 5 | <msclr/marshal\_cppstd.h> | MyForm | Close | Закриття форми | - | - |
| 6 | <msclr/marshal\_cppstd.h> | MessageBox | Show | Відображення вікна повідомлення | Текст повідомлення та стилі | - |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 7 | <SFML/Graphics.hpp> | RenderWindow | Create | Створення вікна | Розмір, текст шапки, стиль | - |
| 8 | <SFML/Graphics.hpp> | RenderWindow | isOpen | Визначення, чи вікно відкрите | - | Відповідь чи вікно відкрите |
| 9 | <SFML/Graphics.hpp> | RenderWindow | close | Закрити вікно | - | - |
| 10 | <SFML/Graphics.hpp> | RenderWindow | draw | Відрисовка елементів вікна | Текстура, яку потрібно відрисувати | - |
| 11 | <SFML/Graphics.hpp> | RenderWindow | Display | Виведення на екран елементів вікна | - | - |
| 12 | <SFML/Graphics.hpp> | RenderWindow | pollEvent | Перевіряє, чи не порожня черга події | Елемент типу Event | Повертає флаг, чи не порожня черга події |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 13 | <SFML/Graphics.hpp> | RenderWindow | clear | Очищає вікно між відрисовками | Об’єкт класу Color | - |
| 14 | <SFML/Graphics.hpp> | Texture | loadFromFile | Завантаження картинки для текстури | Шлях до картинки | - |
| 15 | <SFML/Graphics.hpp> | Sprite | setTexture | Встановлює текстуру для спрайта | Текстура, яку потрібно подати у спрайт | - |
| 16 | <SFML/Graphics.hpp> | Sprite | setPosition | Встановлює положення спрайта на вікні | Координати | - |
| 17 | <SFML/Graphics.hpp> | Sprite | setTextureRect | Встановлює прямокутник для текстури, який потрібно буде виводити | Координати лівого верхнього кута та розміри прямокутника | - |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 18 | <SFML/Graphics.hpp> | Sprite | setColor | Встановлює колір, яким зафарбовується фон за спрайтом | Об’єкт класу Color(r,g,b) | - |
| 19 | <SFML/Graphics.hpp> | Mouse | getPosition | Визначає позицію мишки відносно вікна | Об’єкт класу RenderWindow  (саме вікно) | Об’єкт класу Vector2i |
| 20 | <SFML/Graphics.hpp> | Text | setFont | Встановлює стиль шрифту для тексту | Об’єкт класу Font | - |
| 21 | <SFML/Graphics.hpp> | Text | setString | Встановлює строку для виводу | Строка типу std::string | - |
| 22 | <SFML/Graphics.hpp> | Text | setCharacterSize | Встановлює розмір шрифту | Ціле число | - |
| 23 | <SFML/Graphics.hpp> | Font | loadFromFile | Завантажує шрифт із файлу | Шлях до .ttf файлу | - |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | **Назва функції** | **Призначення функції** | | **Опис вхідних параметрів** | | **Опис вихідних параметрів** | |
| 24 | <SFML/Graphics.hpp> | Color | Color (ctor) | Створює об’єкт класу Color | | 4 цілі однобайтові числа | | - | |
| 25 | <queue> | Std::queue | push | Додати елемент у хвіст черги | | Елемент, що потрібно додати | | - | |
| 26 | <queue> | Std::queue | front | Отримати перший елемент черги | | - | | Перший елемент черги | |
| 27 | <queue> | Std::queue | empty | Визначення пустоти черги | | - | | Булеве значення | |
| 28 | <set> | Std::set | count | Знайти кількість елементів у множині | | Елемент такого ж типу, як і множина | | Кількість (ціле число) | |
| 29 | <set> | Std::set | Insert | Вставити елемент у множину | | Елемент такого ж типу, як і множина | | - | |
| 30 | <vector> | Std:: vector | push\_back | | Вставити елемент у кінець вектора | | Елемент такого ж типу, як і вектор | | - |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | | **Назва функції** | **Призначення функції** | | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 31 | <vector> | Std:: vector | | Rbegin | Повертає адресу останнього елемента вектора | | - | Адреса останнього елемента вектора |
| 32 | <SFML/Graphics.hpp> | Text | | setPosition() | Встановлює позицію текста на екрані | | Координати разташування тексту на екрані | - |
| 33 | <SFML/Graphics.hpp> | Text | | setFillColor | Встановлює колір тексту | | Об’єкт класу Color | - |
| 34 | <ctime> | Не є методом класу | | Clock() | Повертає системний час | | - | Системний час у мілісекундах(ціле беззнакове число) |
| 35 | <cmath> | Не є методом класу | Abs() | | | Повертає модуль цілого числа | Ціле число | Модуль вхідного числа |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 36 | <pair> | <pair> | Make\_pair() | Компонує пару із двох змінних | Дві змінні | Пара із вхідних змінних |
| 37 | <iomanip> | Не є методом класу | Setw() | Встановлює значення ширини поля виводу | Ціле беззнакове число | Ширина поля виводу |
| 38 | <pair> | <pair> | First | Повертає перше число пари | - | Перше число пари |
| 39 | <pair> | <pair> | Second | Повертає друге число пари | - | Друге число пари |
| 40 | <algorithm> | Не є методом класу | Swap() | Обмінює значеннями дві змінні одного типу | Дві змінні | Побічний ефект – обмін значеннями двох змінних |
| 41 | <msclr/marshal\_cppstd.h> | Application | Run | Запускає форму | Приймає ім’я форми | Запускає форму |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва класу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 42 | <msclr/marshal\_cppstd.h> | MyForm | InitializeComponent | Ініціалізує початковими значеннями атрибути графічних елементів вікна | - | - |

### Користувацькі методи

У таблиці 4.2 наведено користувацькі методи, які були застосовані для реалізації задачі.

Користувацькі методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва заголовного файлу | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 1 | “AskWindow.h” | AskWindow | WhereToGetCombination | Рендеринг вікна, яке запитує користувача яким чином ввести дані в програму: зчитати із файлу або згенерувати | Немає | Ціле число: 0 – з файлу, 1-згенерувати |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва классу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 2 | “MsgBox.h” | MsgBox | show | Відображує вікно повідомлення із текстом | Строка типу std::string | Немає |
| 3 | “Game.h” | Game | inputFile | Зчитує комбінацію із файлу input.txt Та повертає, чи розв’язною є комбінація | Немає | Булеве значення |
| 4 | “Game.h” | Game | solve | Видає підказку щодо наступного ходу | Булева змінна, що відповідає за те, чи був зроблений хід від останнього запиту на підказку | Ціле число – номер блоку, на який потрібно клацнути |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва классу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 5 | “Game.h” | Game | getField | Геттер для поля m\_field | Немає | Об’єкт класу Field |
| 6 | “Game.h” | Game | generateCombination | Генерує розв’язну комбінацію | Немає | Немає |
| 7 | “PlayWindow.h” | PlayWindow | initSprites | Вирізає з загального тайлу(картинки з усіма костяшками разом) потрібні фрагменти та встановлює для них місце на екрані | Текстуру із усіма 16 костяшками(одна порожня) та масив спрайтів, властивості яких будуть змінені | Немає (побічно змінює переданий масив спрайтів) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва классу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 8 | “PlayWindow.h” | PlayWindow | letIsPlay | Головний метод, який запускає гру. Потрібен для відображення стану гри на екрані. | Файловий потік (файл статистики), об’єкт класу Game | Немає |
| 9 | “Field.h” | Field | at | Геттер для елемент матриці. Аналог звернення по індексах через квадратні дужки | Два цілих числа: номер рядка(1-4) та номер стовпця(1-4) | Елемент матриці із потрібними координатами |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва классу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 10 | “Field.h” | Field | getHash | Повертає унікальне значення комбінації (потрібно для зменшення обсягу занятої алгоритмом пошуку підказки пам’яті) | Немає | Ціле число – хеш |
| 11 | “Field.h” | Field | isAssembled | Встановлює, чи зібрана комбінація на даний момент | Немає | Булеве значення |
| 12 | “Field.h” | Field | getHeuristic | Оцінює, наскільки далеко(у кількості кроків) від зібраного стану знаходиться дана комбінація | Немає | Ціле число |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва классу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 13 | “Field.h” | Field | getBlankPosition | Повертає пару індексів, де на матриці знаходиться порожня клітинка | Немає | Пара індексів типу std::pair<int,int> - відповідно номер рядка та стовпця |
| 14 | “Field.h” | Field | changeMap | Симулює хід. Змінює стан матриці | Координати(номер рядка та стовпця) клітинки, на яку натиснув користувач. | Повертає, чи відбулося змінення стану матриці |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | **Назва заголовного файлу** | **Назва классу** | **Назва функції** | **Призначення функції** | **Опис вхідних параметрів** | **Опис вихідних параметрів** |
| 15 | “Field.h” | Field | <<friend>> WhatBlocktoClick | Встановлює, яку клітинку було натиснуто у процесі ходу | Приймає два об’єкти класу Field | Повертає номер клітинки, яку було натиснуто у процесі ходу |
| 16 | “Field.h” | Field | canBeAssembled | Визначає, чи можливо зібрати комбінацію взагалі | Немає | Булеве значення |
| 17 | “Field.h” | Field | <<friend>>  operator<< | Виводить матрицю у потік ostream | Посилання на об’єкт класу ostream та об’єкт класу Field | Посилання на об’єкт класу ostream |

# План тестування

## План тестування

1. Авторизація користувача
   1. Введення порожнього логіну
   2. Введення імені, якого ще не було
   3. Введення імені, яке вже було
2. Введення даних у систему
   1. Файл input.txt порожній
   2. Файл input.txt частково порожній(менше 16 чисел)
   3. Файл input.txt має не менше 16 чисел, але перші 16 чисел не задовольняють формату вхідних даних
   4. Файл input.txt має не менше 16 чисел, перші 16 чисел задовольняють формат вхідних даних, але комбінація нерозв’язна
   5. Файл input.txt має не менше 16 чисел, перші 16 чисел задовольняють формат вхідних даних, комбінація розв’язна
3. Процес гри
   1. При натисканні на костяшку, яка може бути здвинута, вона рухається в тому напрямку, який потрібен(він, очевидно, єдиний)
   2. При натисканні на костяшку, яка не може бути натиснута, вона не рухається, нічого не відбувається
   3. При натисканні на кнопку видачі підказки, видається підказка
   4. При натисканні на кнопку виходу з гри, програма завершує роботу із повідомленням про програш.
   5. При розв’язанні комбінації гри завершує роботу автоматично із повідомленням про виграш
   6. При поданні вже розв’язаної комбінації програма просто завершує роботу із повідомленням про виграш

## Приклади тестування

Результати тестувань наведено у таблицях 5.1 – 5.14

Тестування роботи програми при введені порожнього імені.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення порожнього поля імені |
| Початковий стан програми | Відкрита форма авторизації |
| Вхідні дані | {порожня строка} |
| Схема проведення тесту | Введення порожнього імені, натискання на кнопку «Play» |
| Очікуваний результат | Наступна сторінка не відкриється допоки не буде введено непорожнє ім’я, буде виведено повідомлення про неможливість такого ім’я |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування роботи програми при введенні імені, якого досі не зустрічалося.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення нового ім’я |
| Початковий стан програми | Відкрита форма авторизації |
| Вхідні дані | newUser |
| Схема проведення тесту | Введення нового імені, натискання на кнопку «Play» |
| Очікуваний результат | Наступна сторінка відкриється, створиться новий файл з таким іменем |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування роботи програми при введені існуючого імені.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість «авторизації» вже існуючого користувача |
| Початковий стан програми | Відкрита форма авторизації |
| Вхідні дані | Bogdan (уже було один раз введено) |
| Схема проведення тесту | Введення нового імені, натискання на кнопку «Play» |
| Очікуваний результат | Наступна сторінка відкриється, відкриється файл з таким іменем |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму зчитування даних з порожнього файлу.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму при порожньому файлі вхідних даних |
| Початковий стан програми | Користувач авторизувався та вибрав опцію зчитати з файла вхідну комбінацію |
| Вхідні дані | {порожній файл} |
| Схема проведення тесту | Вибрати опцію «зчитати із файлу», залишивши файл порожнім |
| Очікуваний результат | Виведено помилку про неможливість знайти розв’язок |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму зчитування даних з файлу, у якому менше 16 чисел.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму при неповному файлі вхідних даних |

|  |  |
| --- | --- |
| Початковий стан програми | Користувач авторизувався та вибрав опцію зчитати з файла вхідну комбінацію |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | - | - | - | - | | - | - | - | - | | - | - | - | - | |
| Схема проведення тесту | Вибрати опцію «зчитати із файлу», залишивши файл |
| Очікуваний результат | Виведено помилку про неможливість знайти розв’язок |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму зчитування даних з файлу, у якому рівно 16 чисел, але числа не відповідають формату вхідних даних.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму при вхідних даних, що не відповідають формату |
| Початковий стан програми | Користувач авторизувався та вибрав опцію зчитати з файла вхідну комбінацію |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 11 | 12 | | 13 | 14 | 15 | 173761 | |
| Схема проведення тесту | Вибрати опцію «зчитати із файлу», залишивши файл |
| Очікуваний результат | Виведено помилку про неможливість знайти розв’язок |

|  |  |
| --- | --- |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму зчитування даних з файлу, у якому рівно 16 чисел, але комбінація нерозв’язна.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму при нерозв’язній комбінації |
| Початковий стан програми | Користувач авторизувався та вибрав опцію зчитати з файла вхідну комбінацію |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 11 | 12 | | 13 | 15 | 14 |  | |
| Схема проведення тесту | Вибрати опцію «зчитати із файлу», залишивши файл |
| Очікуваний результат | Виведено помилку про неможливість знайти розв’язок |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму зчитування даних з файлу, у якому рівно 16 чисел, і комбінація розв’язна.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму при розв’язній комбінації |
| Початковий стан програми | Користувач авторизувався та вибрав опцію зчитати з файла вхідну комбінацію |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 12 |  | | 13 | 14 | 11 | 15 | |
| Схема проведення тесту | Вибрати опцію «зчитати із файлу», залишивши файл |
| Очікуваний результат | Перехід до форми гри |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму обробки натискання на костяшки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму натисканні на костяшки, які можуть на даний момент рухатись |
| Початковий стан програми | Гру запущено |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 12 |  | | 13 | 14 | 11 | 15 | |
| Схема проведення тесту | Натиснуто костяшку 12 |
| Очікуваний результат | Костяшка під номером 12 зсунеться уліво |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму обробки натискання на костяшки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму натисканні на костяшки, які неможуть на даний момент рухатись |
| Початковий стан програми | Гру запущено |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 12 |  | | 13 | 14 | 11 | 15 | |
| Схема проведення тесту | Натиснуто костяшку 10 |
| Очікуваний результат | Костяшка під номером 10 не зсунеться нікуди |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму обробки натискання на кнопки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму натисканні на кнопку видачі підказки |
| Початковий стан програми | Гру запущено |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 12 |  | | 13 | 14 | 11 | 15 | |
| Схема проведення тесту | Натиснуто кнопку “HELP” |

|  |  |
| --- | --- |
| Очікуваний результат | Буде видано підказку щодо наступного ходу(у даному випадку «12») |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи алгоритму обробки натискання на кнопки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи алгоритму натисканні на кнопку виходу з гри |
| Початковий стан програми | Гру запущено |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 12 |  | | 13 | 14 | 11 | 15 | |
| Схема проведення тесту | Натиснуто кнопку “EXIT AND SAVE” |
| Очікуваний результат | Гра завершиться, буде виведено повідомлення про поразку, у файл буде зроблено відповідний запис |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи програми при розв’язанні комбінації гравцем

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи програми при розв’язанні головоломки |
| Початковий стан програми | Гру запущено |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 12 |  | | 13 | 14 | 11 | 15 | |
| Схема проведення тесту | Зібрати комбінацію |
| Очікуваний результат | Гра завершиться, буде виведено повідомлення про перемогу, у файл буде зроблено відповідний запис |
| Стан програми після проведення випробувань | Як і очікувано |

Тестування коректності роботи програми при введенні вже розв’язаної комбінації

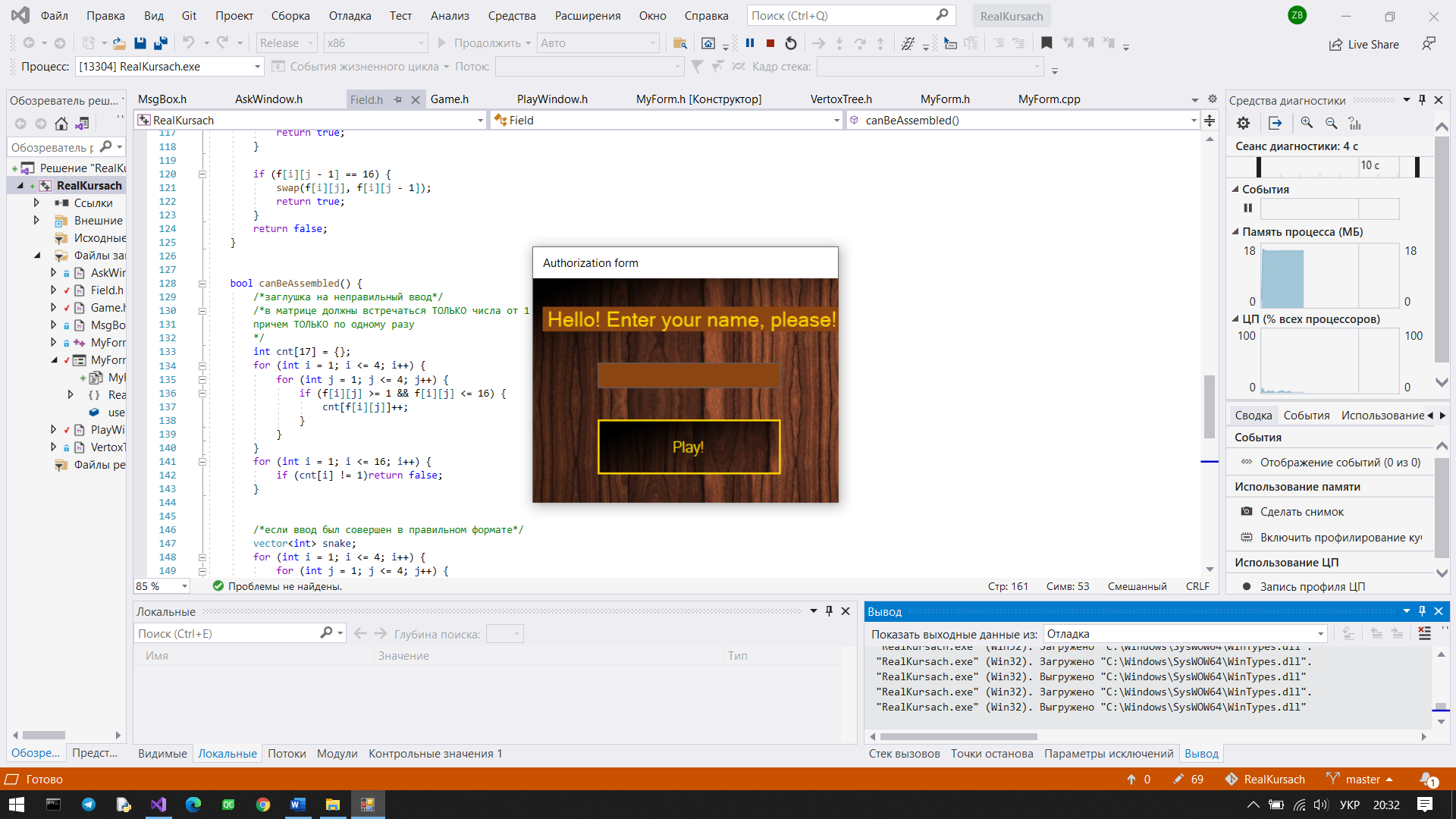
|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи програми при розв’язанні головоломки |
| Початковий стан програми | Гру запущено |
| Вхідні дані | |  |  |  |  | | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | | 9 | 10 | 11 | 12 | | 13 | 14 | 15 | 16 | |
| Схема проведення тесту | У файл input.txt ввести зібрану комбінацію |
| Очікуваний результат | Гра завершиться, буде виведено повідомлення про перемогу, у файл буде зроблено відповідний запис |
| Стан програми після проведення випробувань | Як і очікувано |

# Інструкція користувача

## Робота з програмою

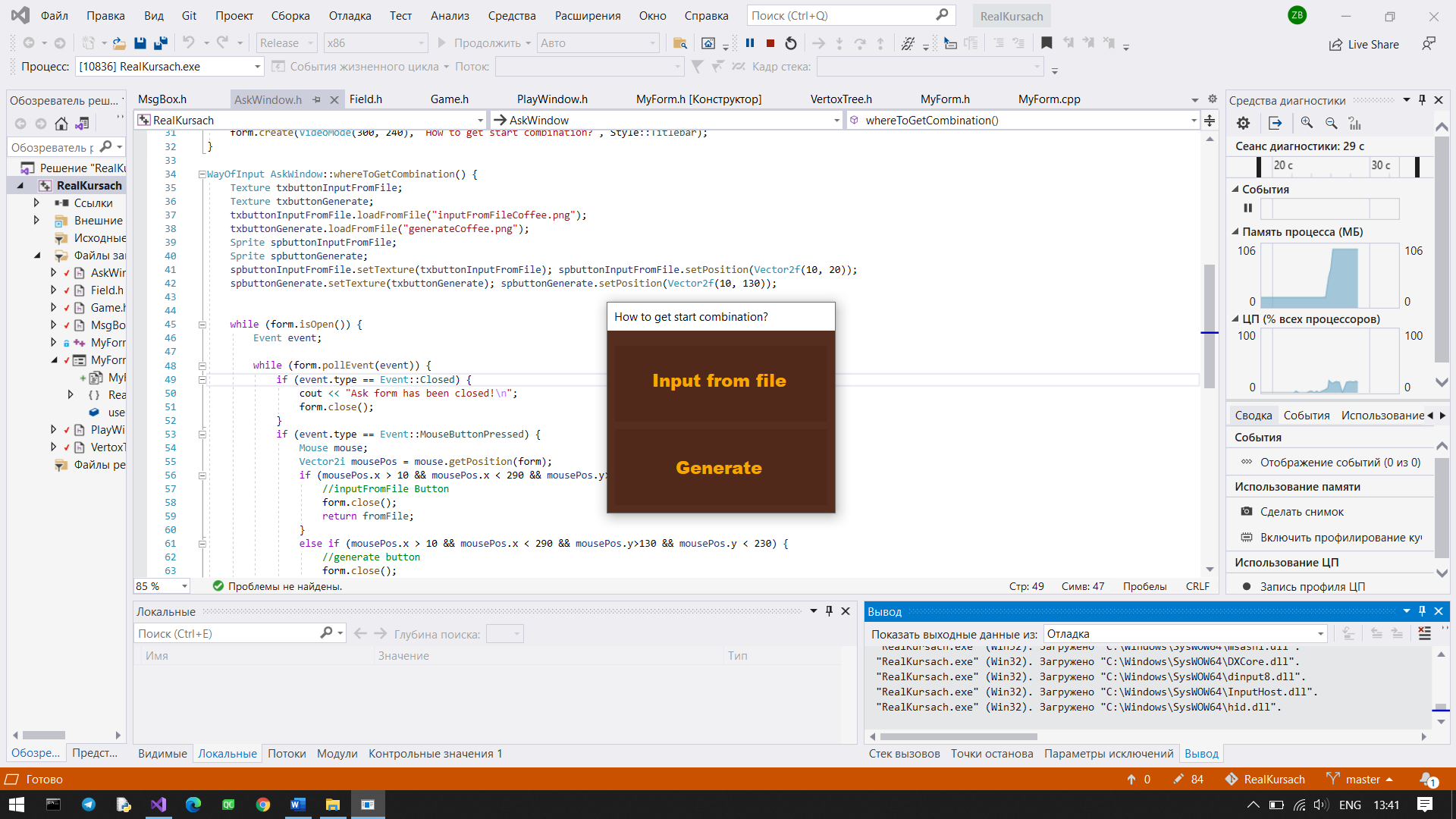
Перш за все хочу попередити користувача про те, що ні в якому разі не слід виймати \*.exe файл із папки програми. У ній присутні файли \*.dll бібліотеки SFML та зображення, які потрібні для роботи програми, тому \*.exe файл та вище згадані файли а також файл input.txt повинні знаходитись у одній директорії.

Після запуску виконуваного файлу з розширенням \*.exe, вікно авторизації (Рисунок 6.1).



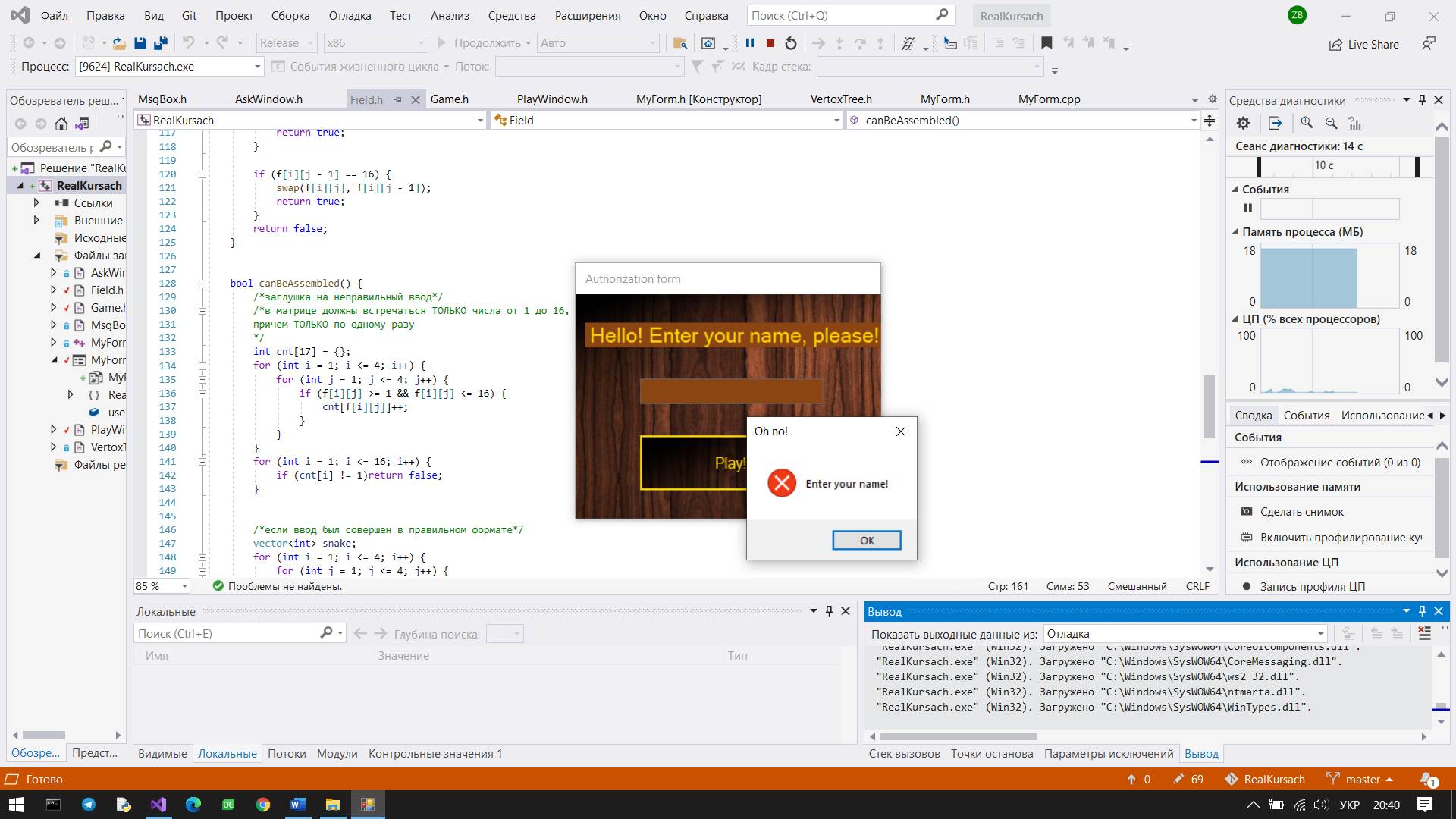
* + - * 1. Вікно авторизації

Увівши свій логін та натиснувши кнопку “Play!” ви переходите до вікна вибору способу задання вхідних даних. (Рисунок 6.2).



* + - * 1. Вибір способу задання вхідних даних

У разі, якщо логін не було введено, а кнопку “Play” було натиснуто, програма видасть повідомлення про неможливість виконання такої операції та запросить користувача ввести логін(Рисунок 6.3)

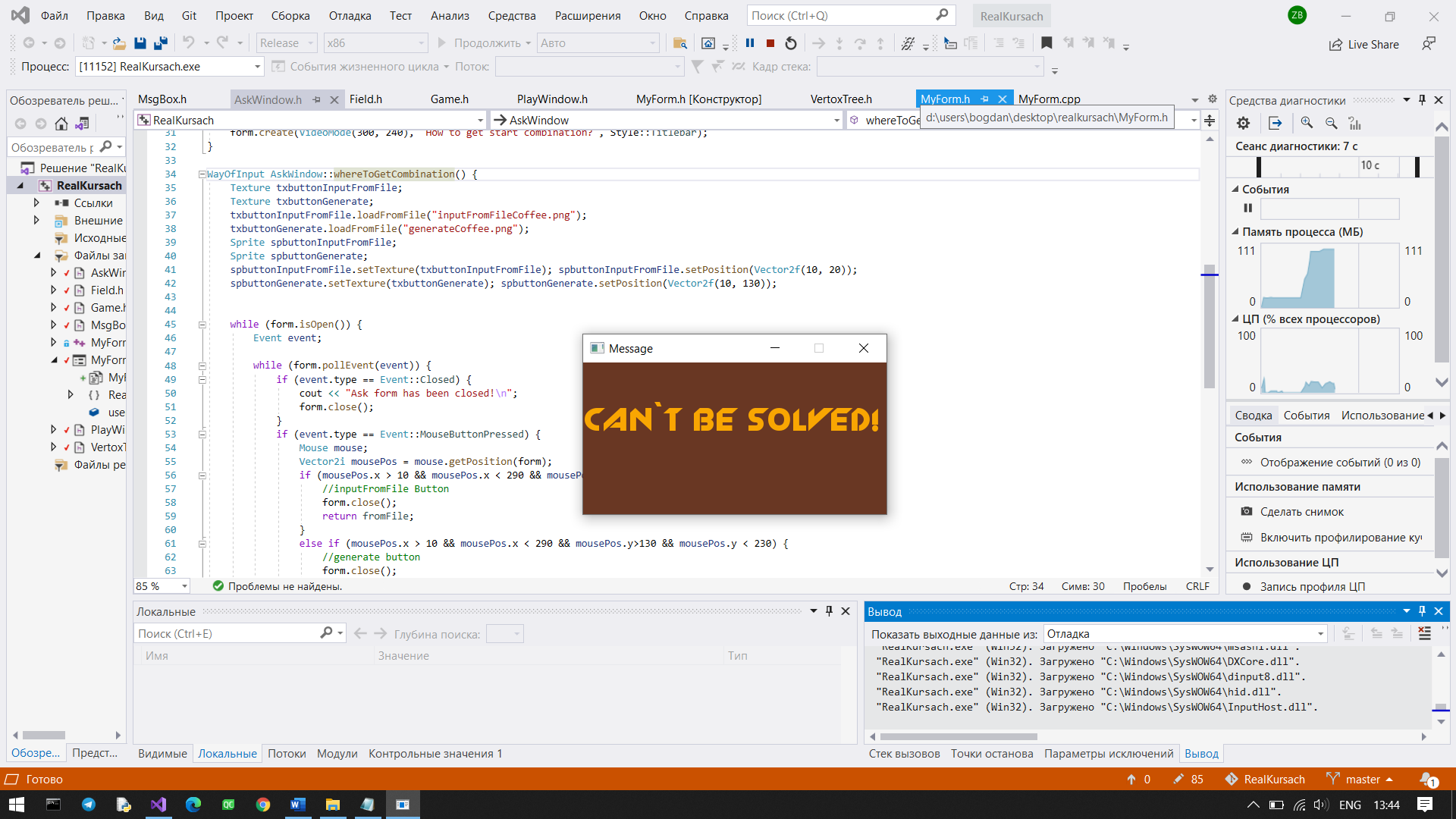


* + - * 1. Повідомлення про неможливість виконання такої операції

Коли логін було введено і файла з такою назвою створено не було, то такий файл буде створено – це файл зі статистикою користувача. У нього буде виноситись інформація про початкову комбінацію, дату та час завершення гри а також результат гри – перемога чи поразка та кількість зроблених у грі кроків.

Після цього, як було сказано вище, гра переходить до форми вибору способу задання вхідних даних.

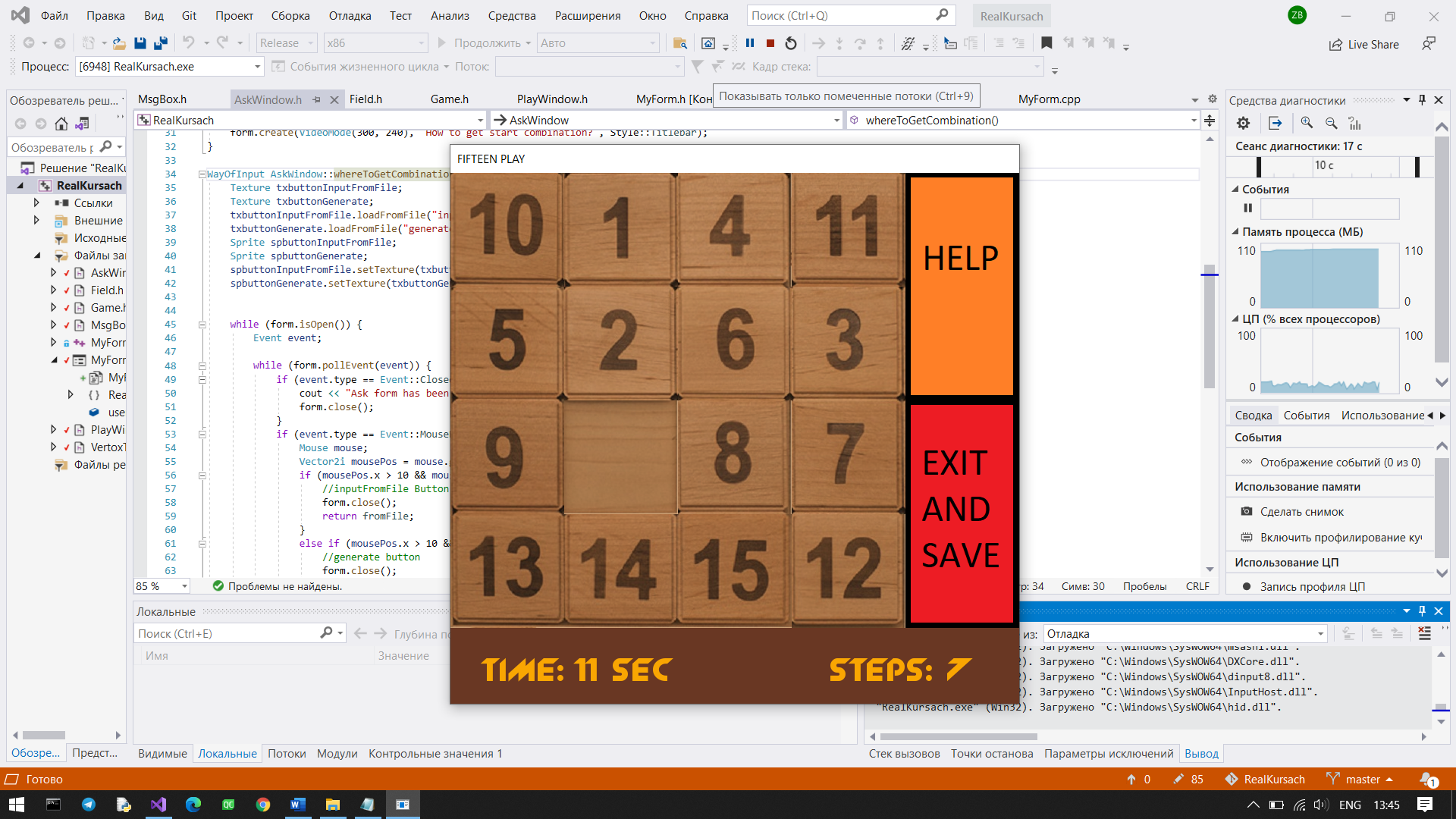
Натиснувши на кнопку із надписом «Input from file» відбувається зчитування комбінації із файлу input.txt, який потрібно заздалегідь цією комбінацією заповнити. Якщо зчитана комбінація є нерозв’язною, або було якимось чином порушено формат вхідних даних, програма видасть повідомлення про помилку та завершиться (Рисунок 6.4).



* + - * 1. Повідомлення про нерозв’язність комбінації

У разі натискання на кнопку із написом «Generate», комбінацію буде згенеровано, причому така комбінація точно буде розв’язною.

Якщо було на натиснуто кнопку “Generate” або кнопку “Input from file” та комбінація виявилася розв’язною, дане вікно закривається, натомість відкривається вікно гри (Рисунок 6.5)

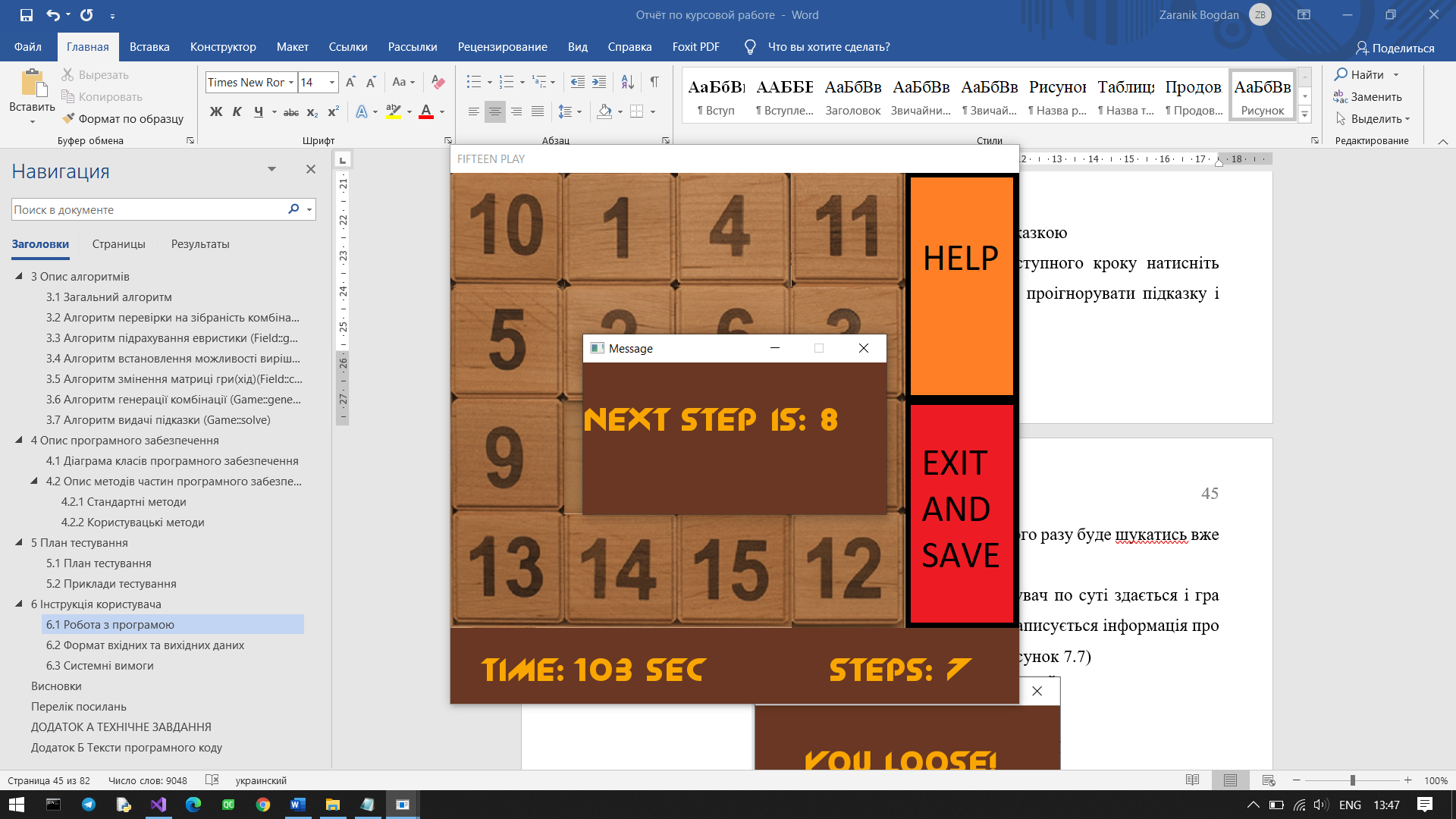


* + - * 1. Вікно гри

Детальніше ознайомимося із вікном гри. У лівій частині вікна присутні 15 квадратиків із числами від 1 до 15 - самі костяшки. У правій частині – присутні дві кнопки: “HELP” та “EXIT AND SAVE”. У лівому нижньому куті вікна відображується поточний час гри, а у правому нижньому – кількість зроблених кроків. Хочу зауважити, що будь-яку з трьох форм користувач не може закрити, натиснувши на хрестик у правому верхньому куті форми. Це зроблено навмисно, аби зробити поведінку інтерфейсу більш захищеною.

Користувач може натискати на будь-які елементи екрану. Якщо користувач натисне на костяшку, яка межує стороною із порожньою, то костяшка, на яку було натиснуто, переміститься на місце, де знаходилася порожня. Інакше – нічого не відбудеться(що є природньо).

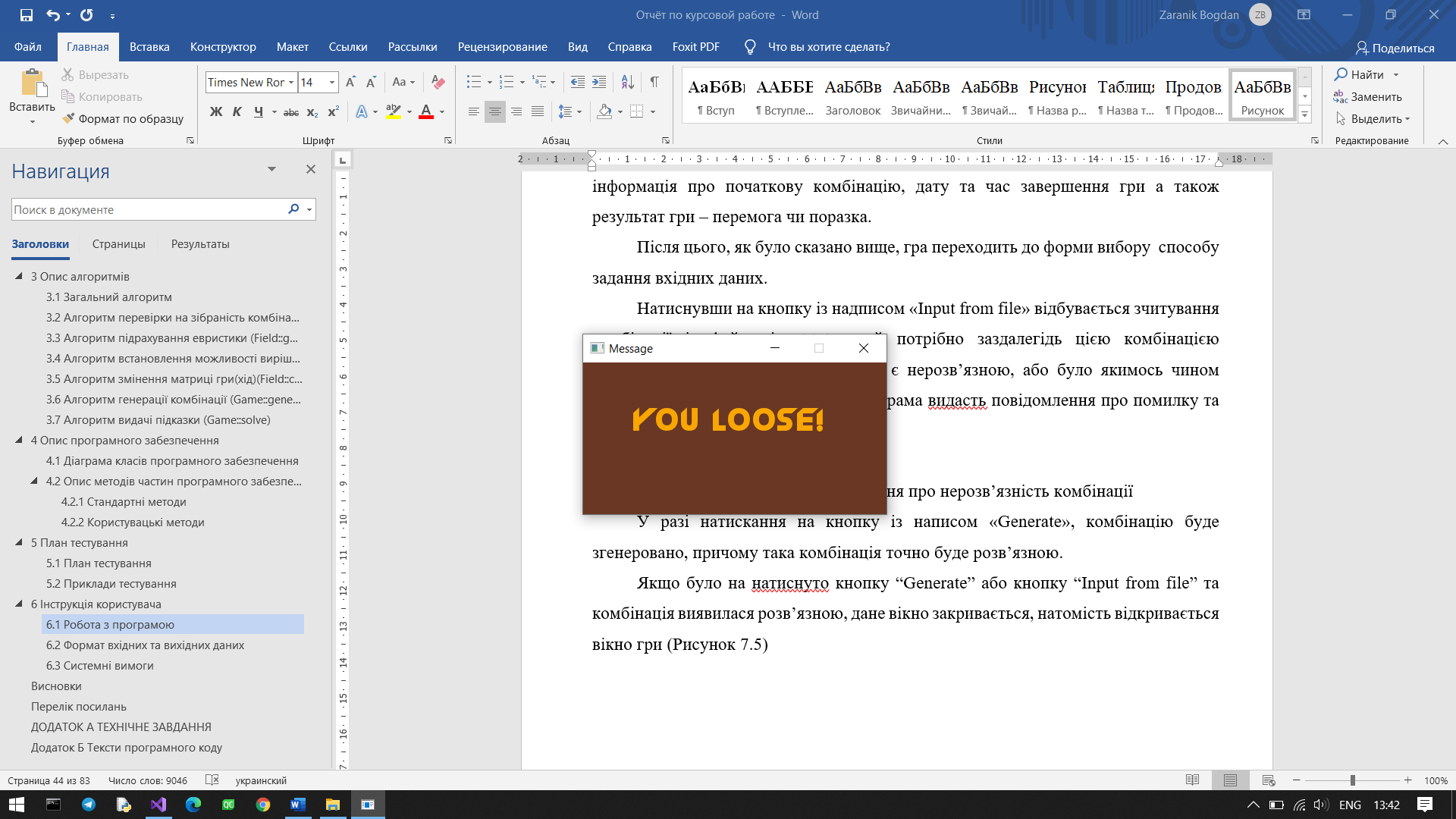
Натиснувши на кнопку “HELP”, користувач побачить повідомлення із підказкою щодо наступного кроку(Рисунок 6.6)



* + - * 1. Повідомлення із підказкою

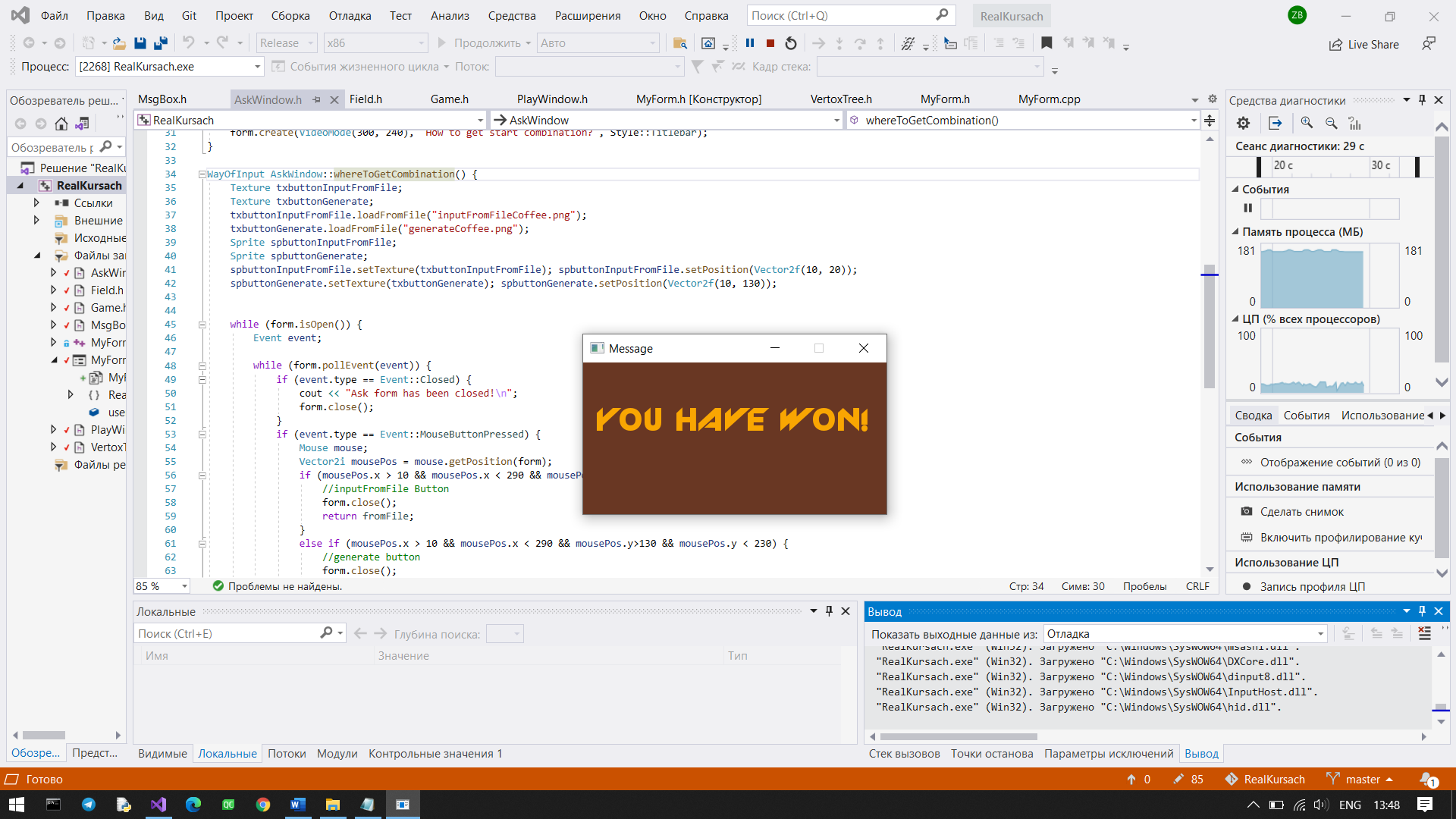
Дане повідомлення слід розуміти так: «для наступного кроку натисніть костяшку із номером 8». Користувач має повне право проігнорувати підказку і зробити хід на власний розсуд. Пошук підказки наступного разу буде шукатись вже зі зміненої комбінації у будь-якому разі.

Натиснувши кнопку “EXIT AND SAVE”, користувач по суті здається і гра завершується поразкою. У файл із іменем користувача записується інформація про гру, а на екран виводиться відповідне повідомлення (Рисунок 6.7)



* + - * 1. Повідомлення про поразку

Якщо ж користувач все-таки зібрав комбінацію, то після кроку автоматично закривається вікно гри, інформація про результат гри записується у файл з іменем користувача і видається повідомлення про перемогу(Рисунок 6.8)



* + - * 1. Повідомлення про перемогу

## Формат вхідних та вихідних даних

Користувачем на вхід програми подається логін та послідовність із 4 рядки по 4 числа чисел від 1 до 16, причому кожне число повинно зустрічатися строго по одному разу. 1-15 – відповідають костяшкам із такими ж числами, 16 – порожнє місце.

Результатом виконання програми виведена у файл, що має назву таку саму як і логін користувача, а саме: початкова комбінація(формат такий, як у вхідному файлі), дата та час закінчення гри та один із двох написів: “You have won!” у разі перемоги та “You loose!” – інакше.

## Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.9.

Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 10  (з останніми обновленнями) |
| Процесор | Intel® Core® i5-4690K  3.50 GHz або AMD FX-8300 3.3 GHz | Intel® Core® i5-10400F  2.90 GHz або AMD Ryzen 5 1600 3.2 GHz |

|  |  |  |
| --- | --- | --- |
|  | **Мінімальні** | **Рекомендовані** |
| Оперативна пам'ять | 1 GB | 2 GB |
| Відеоадаптер | NVIDIA GeForce 940MX  (або сумісний аналог) | |
| Дисплей | 1600х1024 | 1920х1080 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Microsoft .Net Framework 4.7.2 або вище | |

Висновки

Мною було реалізовано програмне забезпечення, що моделює головоломку «Гра у 15». Гра реалізована із урахуванням наявності зручного, інтуїтивно зрозумілого графічного інтерфейсу користувача. Реалізовано форму реєстрації, можливість вводу комбінації з файлу та перевірки її на розв’язність, генерація розв’язної комбінації програмно а також можливість видачі підказки щодо наступного ходу.

У пункті 2 детально викладено про суть самої гри, її правила та математичне доведення можливості або неможливості розв’язання певної наперед заданої комбінації. Наведено усі формули із додержанням стилю, описаного у нормативних документах(ГОСТ).

У пункті 3 зображено псевдокоди усіх важливих алгоритмів, що зустрічаються у програмі. А саме: алгоритму перевірки комбінації на зібраність, підрахунку евристики, встановлення можливості вирішення комбінації, генерації комбінації, видачі підказки та інших.

У пункті 4 представлено діаграму класів програмного забезпечення, опис методів, вбудованих у середовище розробки, а також написаних власноруч(користувацькі методи).

У пункті 5 представлено детальний план тестування, який охоплює усі можливі варіанти розвитку подій. Саме тестування приведено у вигляді таблиць тесткейсів. Усі складені тести програмне забезпечення витримало.

У пункті 6 представлено інструкцію користувача. Вона, як і розділ тестування, охоплює всі етапи роботи програми. Таким чином у програмі немає ситуацій, які були б незрозумілими користувачеві, що прочитав даний розділ.

Гра максимально передає усі тонкощі реальної гри, велику увагу під час написання програмного забезпечення було приділено оформленню графічного інтерфейсу, який нагадує реальні «п’ятнашки», зроблені з дерева.

Реалізована програма цілком задовольняє вимогам технічного завдання, його функціональним та нефункціональним вимогам. Алгоритм видачі підказки працює коректно. Розроблене мною програмне забезпечення є зручним цифровим симулятором головоломки «Гра у 15», який моделює всі можливості даної головоломки.

При тестуванні розробленого програмного забезпечення недоліків знайдено не було. Програма успішно пройшла усі тести, тому можна вважати коректно реалізованими усі написані алгоритми. До тестування також було залучено сторонніх людей, яким було надано інструкцію щодо користування та саму програму. Гравець залишився задоволеним, програма відпрацювала стабільно та передбачувано та отримала схвальні відгуки, в особливості за точність підказки а також за чудово оформлений графічний інтерфейс.

Перелік посилань

1. Попов В. В. Интеллектуальные системы. Алгоритм A\* и игра "Пятнашки" [Електронний ресурс] / Владимир Викторович Попов. – 2012. – Режим доступу до ресурсу: <https://www.dokwork.ru/2012/03/blog-post.html>
2. Игра Пятнашки: существование решения [Електронний ресурс]. – 2008. – Режим доступу до ресурсу: <https://e-maxx.ru/algo/15_puzzle>.
3. Стюарт Р. Интеллектуальные системы. Алгоритм A\* и игра "Пятнашки" / Р. Стюарт, Н. Питер // Искусственный интеллект. Современный подход. / Р. Стюарт, Н. Питер. – СПБ: Издательский дом "Вильямс", 1999. – С. 150–187.

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

автоматизованих систем обробки інформації та управління

Затвердив

Керівник Головченко. М. М.

«03» березня 2021 р.

Виконавець:

Студент Заранік Богдан Юрійович

«03» березня 2021р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Гра у 15»

з дисципліни:

«Основи програмування»

Київ 2021

* 1. *Мета*: Метою курсової роботи є розробка додатку, який генерує головоломку «Гра у 15»
  2. *Дата початку роботи*: «09» березня 2021 р.
  3. *Дата закінчення роботи*: «31» травня 2021 р.
  4. *Вимоги до програмного забезпечення*.

Функціональні вимоги:

* Можливість авторизації вже існуючого гравця або реєстрації нового.
* Можливість запам’ятовувати статистику кожного гравця(ім’я вводиться на початку кожного сеансу гри) у файлі, що називається його іменем у конкретній папці.
* Можливість зчитування комбінації з файлу.
* Можливість генерувати розв’язну комбінацію програмно.
* Можливість перевірки комбінації на розв’язність(у випадку нерозв’язності завершувати сенс гри).
* Можливість перевірки кінця гри.
* Можливість виходу з гри до її фактичного завершення.
* Можливість виведення поточної кількості зроблених кроків у відповідному полі.
* Можливість виведення підказки щодо наступного кроку.

Нефункціональні вимоги:

* Програма повинна генерувати комбінацію швидко(менше ніж за секунду).
* Якщо було введено невідоме ім’я користувача, то має створюватись новий файл і відкривати його, а не намагатися зчитувати неіснуючий файл, що призведе до системної помилки.
* Програма повинна відображати поточну комбінацію у зручній для користувача формі(користувацький інтерфейс).
* Програма повинна відповідати також і модульному підходу вирішення задачі.
* 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 - 95 - Розробка технічної документації.

* 1. *Стадії та етапи розробки*:

1. Об'єктно-орієнтований аналіз предметної області задачі (до 23.03.2021 р.)
2. Об'єктно-орієнтоване проектування архітектури програмної системи (до 30.03.2021р.)
3. Розробка програмного забезпечення (до 12.05.2021р.)
4. Тестування розробленої програми (до 20.05.2021р.)
5. Розробка пояснювальної записки (до 31.05.2021р.).
6. Захист курсової роботи (до 04.06.2021 р.).
   1. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

Додаток Б Тексти програмного коду

*студента групи ІП-01 І курсу*

*Зараніка Б.Ю*

(Вид носія даних)

*Жорсткий диск*

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення та генерації головоломки «Гра у 15»*

(Обсяг програми (документа), арк., Кб)

*28 арк, 200 Kb*

**/\*заголовний файл VertoxTree.h \*/**

#include <cstdlib>

#include <vector>

#include "Field.h"

using namespace std;

/\*typedefs\*/

typedef int heuristic\_t;

typedef uint64\_t hash\_t;

struct VertoxTree {

hash\_t hash;

Field m;

int currDepth;

VertoxTree\* prev;

vector<VertoxTree\*> next;

VertoxTree() {//ctor

hash = 0;

prev = NULL;

currDepth = 0;

}

};

**/\*заголовний файл PlayWindow.h \*/**

#pragma once

#include "SFML/Graphics.hpp"

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <vector>

#include <utility>

#include <fstream>

#include "Game.h"

#include "MsgBox.h"

using namespace std;

using namespace sf;

const int blockSize = 150;

class PlayWindow {

RenderWindow form;

public:

PlayWindow();//ctor

void initSprites(Texture& tx, vector<Sprite>& sp);//to make sprites from tile and "give" `em positions on scree

void letIsPlay(ostream& outFileStream, Game game);//open main game window

};

PlayWindow::PlayWindow() {

form.create(VideoMode(750, 600), "FIFTEEN PLAY", Style::Titlebar);

}

void PlayWindow::initSprites(Texture& tx, vector<Sprite>& sp) {

for (int i = 1; i <= 16; i++) {

sp[i].setTexture(tx);

sp[i].setTextureRect(IntRect((i - 1) % 4 \* blockSize, (i - 1) / 4 \* blockSize, blockSize, blockSize));

sp[i].setColor(sf::Color(207, 174, 139, 255));

}

}

void PlayWindow::letIsPlay(ostream& outFileStream, Game game) {

outFileStream << game.getField();//printing a start combination into file

Texture tx;

tx.loadFromFile("15.jpg");

vector<Sprite> block(17);

initSprites(tx, block);

Texture txbuttonHelp; txbuttonHelp.loadFromFile("help.png");

Texture txbuttonExitAndSave; txbuttonExitAndSave.loadFromFile("exitandsave.png");

Sprite spbuttonHelp, spbuttonExitAndSave;

spbuttonHelp.setTexture(txbuttonHelp); spbuttonHelp.setPosition(600, 0);

spbuttonExitAndSave.setTexture(txbuttonExitAndSave); spbuttonExitAndSave.setPosition(600, 300);

bool madeStepAfterHelp = false;//дописать

while (form.isOpen() && !game.getField().isAssembled()) {//\*\*\*\*\*

Event event;

while (form.pollEvent(event)) {

if (event.type == Event::Closed) {

cout << "Form has been closed!\n";

form.close();

}

if (event.type == Event::MouseButtonPressed) {

Mouse mouse;

Vector2i mousePos = mouse.getPosition(form);

//cout << "Mouse: " << mousePos.x << " " << mousePos.y << endl;

for (int i = 1; i <= 4; i++)

for (int j = 1; j <= 4; j++)

if ((i - 1) \* blockSize < mousePos.y && (j - 1) \* blockSize < mousePos.x && (i - 1) \* blockSize + blockSize > mousePos.y && (j - 1) \* blockSize + blockSize > mousePos.x) {

if (game.getField().changeMap(i, j)) {

madeStepAfterHelp = true;

}

break;

}

if (mousePos.x > 600 && mousePos.x < 750 && mousePos.y>0 && mousePos.y < 300) {

//Help Button

MsgBox helpWindow;

helpWindow.show("Next step is: " + to\_string(game.solve(madeStepAfterHelp)));

//cout << "Next step is: " << game.solve(madeStepAfterHelp) << "\n";

madeStepAfterHelp = false;

//throw MSGBOX with next step

}

else if (mousePos.x > 600 && mousePos.x < 750 && mousePos.y>300 && mousePos.y < 600) {

//ExitButton

form.close();

}

}//MouseButtonPressedevent

}

form.clear(sf::Color(207, 174, 139, 255));

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

block[game.getField().at(i, j)].setPosition((j - 1) \* blockSize, (i - 1) \* blockSize);//\*\*\*\*\*

form.draw(block[game.getField().at(i, j)]);//\*\*\*\*\*

}

}

form.draw(spbuttonHelp);

form.draw(spbuttonExitAndSave);

form.display();

}

form.close();

MsgBox finalMessage;

//printing to file

//and printing onto screen

//info about the game

/\*

getting current date

\*/

time\_t now = time(0);

char\* dateString = ctime(&now);

if (game.getField().isAssembled()) {

finalMessage.show("You have won!");

outFileStream << dateString;

outFileStream << "You have won!" << endl;

outFileStream << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

else {

finalMessage.show("You loose!");

outFileStream << dateString;

outFileStream << "You loose!" << endl;

outFileStream << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

}

**/\*заголовний файл MsgBox.h \*/**

#pragma once

#include "SFML/Graphics.hpp"

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <vector>

#include <utility>

#include <fstream>

#include <string>

using namespace std;

using namespace sf;

class MsgBox {

RenderWindow form;

public:

MsgBox() {//ctor

form.create(VideoMode(400, 200), "Message", Style::Close);

}

void show(string text) {//to render message window

while (form.isOpen()) {

Event event;

while (form.pollEvent(event)) {

if (event.type == Event::Closed) {

form.close();

}

}

sf::Text textOnScreen;

sf::Font font;

font.loadFromFile("myFont.ttf");

textOnScreen.setFont(font);

textOnScreen.setString(text);

textOnScreen.setCharacterSize(40);

form.clear(sf::Color::Blue);

form.draw(textOnScreen);

form.display();

}

}

};

**/\*заголовний файл AskWindow.h \*/**

#pragma once

#include "SFML/Graphics.hpp"

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <vector>

#include <utility>

#include <fstream>

#include "Game.h"

#include "MsgBox.h"

#include "PlayWindow.h"

using namespace std;

using namespace sf;

enum WayOfInput {

fromFile = 0,

generateCombination = 1

};

class AskWindow {

RenderWindow form;

public:

AskWindow();//ctor

WayOfInput whereToGetCombination();//to read combination from file or to generate it?

};

AskWindow::AskWindow() {

form.create(VideoMode(300, 240), "How to get start combination?", Style::Titlebar);

}

WayOfInput AskWindow::whereToGetCombination() {

Texture txbuttonInputFromFile;

Texture txbuttonGenerate;

txbuttonInputFromFile.loadFromFile("inputFromFile.png");

txbuttonGenerate.loadFromFile("generate.png");

Sprite spbuttonInputFromFile;

Sprite spbuttonGenerate;

spbuttonInputFromFile.setTexture(txbuttonInputFromFile); spbuttonInputFromFile.setPosition(Vector2f(10, 30));

spbuttonGenerate.setTexture(txbuttonGenerate); spbuttonGenerate.setPosition(Vector2f(10, 135));

while (form.isOpen()) {

Event event;

while (form.pollEvent(event)) {

if (event.type == Event::Closed) {

cout << "Ask form has been closed!\n";

form.close();

}

if (event.type == Event::MouseButtonPressed) {

Mouse mouse;

Vector2i mousePos = mouse.getPosition(form);

if (mousePos.x > 5 && mousePos.x < 285 && mousePos.y>30 && mousePos.y < 130) {

//inputFromFile Button

form.close();

return fromFile;

}

else if (mousePos.x > 5 && mousePos.x < 285 && mousePos.y>135 && mousePos.y < 235) {

//generate button

form.close();

return generateCombination;

}

}

}

form.clear(sf::Color(207, 174, 139, 255));

form.draw(spbuttonInputFromFile);

form.draw(spbuttonGenerate);

form.display();

}//main while

}

**/\*заголовний файл Field.h \*/**

#pragma once

#include <vector>

using namespace std;

/\*typedefs\*/

typedef int heuristic\_t;

typedef uint64\_t hash\_t;

class Field {

vector<vector<int>> f;

public:

Field();//ctor

int& at(int i, int j);//getter for element [i, j]

hash\_t getHash();//to get hash

bool isAssembled();//to get is combination assembled

heuristic\_t getHeuristic();//to get heuristic

pair<int, int> getBlankPosition();//to get where is blank position

friend int whatBlockToClick(Field beforeTheStepCondition, Field afterOneStepCondition);//difference between 2 combinations(before and after step)

bool canBeAssembled();//is combination can be assembled?

bool changeMap(int i, int j);//to make a step

friend ostream& operator<<(ostream& fileStream, Field f);//to print info into file easier

};

Field::Field() {

f = vector<vector<int>>(6);

for (int i = 0; i < 6; i++)

f[i] = vector<int>(6, -1);

}

int& Field::at(int i, int j) {

return f[i][j];

}

hash\_t Field::getHash() {

hash\_t sum = 0;

int cnt = 15;

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

sum += f[i][j] \* (1LLU << (((i - 1) \* 4 + j - 1) \* 4));

}

}

return sum;

}

bool Field::isAssembled() {

vector<vector<int>> ideal(6, vector<int>(6, -1));

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++)

ideal[i][j] = (i - 1) \* 4 + j;

}

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++)

if (ideal[i][j] != f[i][j])return false;

}

return true;

}

heuristic\_t Field::getHeuristic() {

int sum = 0;

/\*math\_distance\*/

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

sum += abs(i - (f[i][j] / 4 + 1)) + abs(j - (f[i][j] % 4 + 1));

}

}

/\*not on their positions\*/

for (int i = 1; i <= 4; i++)

for (int j = 1; j <= 4; j++)

sum += (i - 1) \* 4 + j != f[i][j];

/\*Last move\*/

sum += (f[4][4] == 15 || f[4][4] == 12) ? 2 : 0;

/\*Corner Tiles\*/

int conflictCount = 0;

//upper left corner

if (f[1][1] != 1) {

if (f[1][2] == 2 || f[2][1] == 5)conflictCount++;

}

//upper right corner

if (f[1][4] != 4) {

if (f[1][3] == 3 || f[2][3] == 8)conflictCount++;

}

//lower lefft corner

if (f[4][1] != 13) {

if (f[3][1] == 9 || f[4][2] == 14)conflictCount++;

}

sum += 2 \* conflictCount;

return sum;

}

pair<int, int> Field::getBlankPosition() {

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

if (f[i][j] == 16)return make\_pair(i, j);

}

}

}

int whatBlockToClick(Field beforeTheStepCondition, Field afterOneStepCondition) {

pair<int, int> beforeStep;

pair<int, int> afterStep;

beforeStep = beforeTheStepCondition.getBlankPosition();

afterStep = afterOneStepCondition.getBlankPosition();

pair<int, int> shiftVector = make\_pair(afterStep.first - beforeStep.first, afterStep.second - beforeStep.second);

return beforeTheStepCondition.at(beforeStep.first + shiftVector.first, beforeStep.second + shiftVector.second);

}

bool Field::changeMap(int i, int j) {

if (f[i + 1][j] == 16) {

swap(f[i][j], f[i + 1][j]);

return true;

}

if (f[i - 1][j] == 16) {

swap(f[i][j], f[i - 1][j]);

return true;

}

if (f[i][j + 1] == 16) {

swap(f[i][j], f[i][j + 1]);

return true;

}

if (f[i][j - 1] == 16) {

swap(f[i][j], f[i][j - 1]);

return true;

}

return false;

}

bool Field::canBeAssembled() {

/\*заглушка на неправильный ввод\*/

/\*в матрице должны встречаться ТОЛЬКО числа от 1 до 16,

причем ТОЛЬКО по одному разу

\*/

int cnt[17] = {};

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

if (f[i][j] >= 1 && f[i][j] <= 16) {

cnt[f[i][j]]++;

}

}

}

for (int i = 1; i <= 16; i++) {

if (cnt[i] != 1)return false;

}

/\*если ввод был совершен в правильном формате\*/

vector<int> snake;

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

if (f[i][j] != 16)snake.push\_back(f[i][j]);

}

}

int e = 0;

int N = 0;

//on which line blank is

for (int i = 1; i <= 4; i++)

for (int j = 1; j <= 4; j++)

if (f[i][j] == 16)e = i;

//how many inversions

for (int i = 0; i < snake.size() - 1; i++) {

for (int j = i + 1; j < snake.size(); j++) {

N += (int)(snake[j] < snake[i]);

}

}

return !((N + e) % 2);

}

ostream& operator<<(ostream& fileStream, Field f) {

fileStream << "Start combination: " << endl;

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

fileStream << setw(3) << (f.at(i, j) == 16 ? 0 : f.at(i, j));

}

fileStream << endl;

}

return fileStream;

}

**/\*заголовний файл MyForm.h \*/**

#pragma once

#include <string>

#include <iostream>

#include <stdio.h>

#include <msclr/marshal\_cppstd.h>

std::string userName = "none";

namespace RealKursach {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::TextBox^ nameBox;

private: System::Windows::Forms::Label^ label1;

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(MyForm::typeid));

this->button1 = (gcnew System::Windows::Forms::Button());

this->nameBox = (gcnew System::Windows::Forms::TextBox());

this->label1 = (gcnew System::Windows::Forms::Label());

this->SuspendLayout();

//

// button1

//

this->button1->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"button1.BackgroundImage")));

this->button1->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->button1->ForeColor = System::Drawing::Color::Gold;

this->button1->Location = System::Drawing::Point(91, 183);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(257, 72);

this->button1->TabIndex = 0;

this->button1->Text = L"Play!";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// nameBox

//

this->nameBox->BackColor = System::Drawing::Color::SaddleBrown;

this->nameBox->BorderStyle = System::Windows::Forms::BorderStyle::FixedSingle;

this->nameBox->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 13, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->nameBox->ForeColor = System::Drawing::Color::Gold;

this->nameBox->Location = System::Drawing::Point(91, 110);

this->nameBox->Name = L"nameBox";

this->nameBox->Size = System::Drawing::Size(257, 32);

this->nameBox->TabIndex = 1;

//

// label1

//

this->label1->AutoSize = true;

this->label1->BackColor = System::Drawing::Color::SaddleBrown;

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 16.2F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label1->ForeColor = System::Drawing::Color::Gold;

this->label1->Location = System::Drawing::Point(14, 37);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(412, 32);

this->label1->TabIndex = 2;

this->label1->Text = L"Hello! Enter your name, please!";

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"$this.BackgroundImage")));

this->ClientSize = System::Drawing::Size(444, 329);

this->ControlBox = false;

this->Controls->Add(this->label1);

this->Controls->Add(this->nameBox);

this->Controls->Add(this->button1);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::Fixed3D;

this->Name = L"MyForm";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"Authorization form";

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

std::string name = msclr::interop::marshal\_as<std::string>(nameBox->Text);

if (name.size() == 0) {

MessageBox::Show("Enter your name!", "Oh no!",

MessageBoxButtons::OK, MessageBoxIcon::Error);

}

else {

//save name and go out from

//authorization form

userName = "DataBase\\" + name;

MyForm::Close();

}

} };

}

**/\*заголовний файл Game.h \*/**

#pragma once

#pragma once

#include <vector>

#include <fstream>

#include <iostream>

#include "VertoxTree.h"

#include <queue>

#include <set>

#include <iomanip>

#include <cstdlib>

#include <ctime>

#include "Field.h"

#include "MsgBox.h"

using namespace std;

/\*constants\*/

/\*defines\*/

#define mp make\_pair

/\*typedefs\*/

typedef int heuristic\_t;//heuristic type

typedef uint64\_t hash\_t;//hash of combination type

/\*+++++++++++++++++++++++++++++\*/

class Game {

Field m\_field;

vector<int> helpCalls;

public:

bool inputFile();//for reading combination from file

int solve(bool);//to give and advice how to solve the problem

Field& getField() {//getter for filed

return m\_field;

}

void generateCombination();//for generating combination

};

void Game::generateCombination() {

srand(time(NULL));

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

m\_field.at(i, j) = 4 \* (i - 1) + j;

}

}

int ii = 4, jj = 4;

pair<int, int> shift[4] = { mp(0, 1), mp(1, 0), mp(0, -1), mp(-1, 0) };

int curr = 0, last = 0;

for (int step = 1; step <= 30; step++) {

do {

last = curr;

curr = rand() % 4;

} while (m\_field.at(ii + shift[curr].first, jj + shift[curr].second) == -1 || last == (curr+2)%4 );

swap(m\_field.at(ii, jj), m\_field.at(ii + shift[curr].first, jj + shift[curr].second));

ii += shift[curr].first;

jj += shift[curr].second;

}

}

bool Game::inputFile() {

ifstream in("input.txt");

for (int i = 1; i <= 4; i++)

for (int j = 1, x; j <= 4; j++) {

in >> x;

m\_field.at(i, j) = (x == 0) ? 16 : x;

}

return m\_field.canBeAssembled();

}

int Game::solve(bool madeStepAfterHelp) {

//srand(time(NULL));

if (madeStepAfterHelp==false && helpCalls.size()>0) {

return \*helpCalls.rbegin();

}

else {

int \_\_MAXDEPTH\_\_ = 17;//rand() % 5 + 5;

pair<int, int> shift[4] = { mp(0, 1), mp(0, -1), mp(1, 0), mp(-1, 0) };

//limited bfs

//создаём стартовую вершину

VertoxTree\* start = new VertoxTree;

start->currDepth = 0;

start->hash = m\_field.getHash();

start->m = m\_field;

start->prev = NULL;

queue<VertoxTree\*> q;

vector<VertoxTree\*> allVertoxes;

set<hash\_t> used;

q.push(start);

allVertoxes.push\_back(start);

while (!q.empty()) {

VertoxTree\* curr = q.front();

if (!used.count(curr->hash) && curr->currDepth <= \_\_MAXDEPTH\_\_) {

if (curr->m.isAssembled()) { /\*cout << "ASSEMBLED!!!\n" << curr->currDepth << "\n";\*/ break; }

else {

used.insert(curr->hash);

VertoxTree\* p;//для создания новых вершин

pair<int, int> blankPoint = curr->m.getBlankPosition();

for (int sh = 0; sh < 4; sh++) {

if (curr->m.at(shift[sh].first + blankPoint.first, shift[sh].second + blankPoint.second) != -1) {//есть куда сдвинуть пустой место

p = new VertoxTree;

p->m = curr->m;

swap(p->m.at(blankPoint.first, blankPoint.second), p->m.at(shift[sh].first + blankPoint.first, shift[sh].second + blankPoint.second));

p->hash = p->m.getHash();

if (!used.count(p->hash)) {

p->prev = curr;

p->currDepth = curr->currDepth + 1;

curr->next.push\_back(p);

//добавление в очередь

q.push(p);

allVertoxes.push\_back(p);

}

else delete p;//:)

}

}

}

}

q.pop();

}

//пробрать листья

//и выбрать среди них с минимальной эвристикой

bool gotHeuristic = false;

set<VertoxTree\*> setOfClosedVertoxes;//используется для отсечения тех вершин, которые приводят к зацикливаемуму шагу

/\*

для чего нужен setOfClosedVertoxes

Была проблема в том, что зацикливался ход, выдаваемый подсказкой.

Мы делаем следующую заглушку:

проходимся по всем вершинам

для литков, которые не в setOfClosedVertoxes находим тот,

который имеет минимальную евристику.

Если вторая вершина в пути, оканчивающийся в этом листке, равна последней

в helpCalls(зацикливание), то засунем эту вершину в setOfClosedVertoxes и по новой прогонимся.

Таким образо второй раз по этому листку мы не пройдёмся

\*/

VertoxTree\* last = NULL;

while (!gotHeuristic) {

VertoxTree\* currlast = NULL;

int minHeuristic = 1'000'000;

for (auto vertox : allVertoxes) {//поиск лиска с минимальной эвристикой, который ещё не отсечен

if (!vertox->next.size() && !setOfClosedVertoxes.count(vertox)) {

if (minHeuristic > vertox->m.getHeuristic()) {

minHeuristic = vertox->m.getHeuristic();

currlast = vertox;

}

}

}

//в currlast лежит указатель на такой листок

VertoxTree\* save = currlast;

//переходим на предпоследнюю вершину(1 шаг от начала)

while (currlast && currlast->prev && (currlast->prev)->prev) { //

currlast = currlast->prev;

}

if (!helpCalls.size() || (whatBlockToClick(currlast->m, currlast->prev->m) != helpCalls[helpCalls.size() - 1])) {

gotHeuristic = true;

last = currlast;

}

else setOfClosedVertoxes.insert(save);

}//now we have gotten help

int answer = -1;

if (last && last->prev) {

answer = whatBlockToClick(last->m, last->prev->m);

helpCalls.push\_back(answer);

}

//удаление вершин

for (auto v : allVertoxes)

delete v;

return answer;

}}