МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з «Основи програмування»

(назва дисципліни)

на тему: «Розміщення»

Студентки 1 курсу, групи ІП-14

Радзівіло Валерії Артемівни

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник Головченко М.М.

Старший викладач кафедри ІПІ

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ- 2022 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-14 Семестр 2

**ЗАВДАННЯ**

на курсову роботу студента

##### Радзівіло Валерії Артемівни

(прізвище, ім’я, по батькові)

1. Тема роботи

2. Строк здачі студентом закінченої роботи

3. Вихідні дані до роботи

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

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

6. Дата видачі завдання

****

# АНОТАЦІЯ

Пояснювальна записка до курсової: 101 сторінка, 11 рисунків, 19 таблиць, 2 посилання.

Об’єкт дослідження: гра «Розміщення».

Мета роботи: розробити програмне забезпечення, яке дозволить грати в гру «Розміщення», користуючись графічним інтерфейсом.

Опановано розробку програмного забезпечення з використанням ООП. Досліджені правила гри. Описано алгоритми роботи програмного забезпечення. Проведені необхідні тестування для перевірки коректності роботи гри. Створено інструкцію та детальне описання роботи методів, використаних у програмі.

Виконана програмна реалізація гри «Розміщення».

Зміст

[АНОТАЦІЯ 4](#_Toc105948968)

[ВСТУП 6](#_Toc105948969)

[1 Постановка задачі 7](#_Toc105948970)

[2 Теоретичні відомості 8](#_Toc105948971)

[3 Опис алгоритмів 10](#_Toc105948972)

[1.1 Загальний алгоритм 10](#_Toc105948973)

[1.2 Алгоритми методів 12](#_Toc105948974)

[4 Опис програмного забезпечення 18](#_Toc105948975)

[4.1 Діаграма класів програмного забезпечення 18](#_Toc105948976)

[4.2 Опис методів частин програмного забезпечення 19](#_Toc105948978)

[4.2.1 Стандартні методи 19](#_Toc105948979)

[4.2.2 Користувацькі методи 23](#_Toc105948980)

[5 Тестування програмного забезпечення 30](#_Toc105948981)

[5.1 План тестування 30](#_Toc105948982)

[5.2 Приклади тестування 31](#_Toc105948983)

[6 Інструкція користувача 45](#_Toc105948984)

[6.1 Робота з програмою 45](#_Toc105948985)

[6.2 Вхідні та вихідні дані 50](#_Toc105948986)

[6.3 Системні вимоги 51](#_Toc105948987)

[Висновки 52](#_Toc105948988)

[Перелік посилань 53](#_Toc105948989)

[Додаток А Технічне завдання 54](#_Toc105948990)

[Додаток Б Тексти програмного коду 57](#_Toc105948991)

[1 MAIN.PY 58](#_Toc105948992)

[2 REGULAR\_GAME.PY 59](#_Toc105948993)

[3 ANSWERBUTTON.PY 65](#_Toc105948994)

[4 BUTTONTOPLAY.PY 67](#_Toc105948995)

[5 CHECKBUTTON.PY 69](#_Toc105948996)

[6 COLORAMOUNTS.PY 70](#_Toc105948997)

[7 CREATETASK.PY 71](#_Toc105948998)

[8 EXITBUTTON.PY 75](#_Toc105948999)

[9 FUNCS\_BUTTONS.PY 77](#_Toc105949000)

[10 FUNCS\_INTERFACE.PY 83](#_Toc105949001)

[11 FUNCS\_LOGIC.PY 88](#_Toc105949002)

[12 RULESBUTTON.PY 94](#_Toc105949003)

[13 SOLVEBUTTON.PY 96](#_Toc105949004)

[14 WINDOWMAKER.PY 97](#_Toc105949005)

[15 LevelButton.py 99](#_Toc105949006)

# ВСТУП

Курсова робота є закріпленням, поглибленням та узагальненням базових теоретичних знань з дисципліни «Об'єктно-орієнтоване програмування», а також їх практичним застосуванням до комплексного вирішення конкретного завдання.

Під час виконання курсової роботи я отримаю:

* вміння збирати і аналізувати відповідні матеріали про об’єкт дослідження, використовуючи сучасні джерела інформації, включаючи Інтернет-ресурси;
* спроможність проводити необхідні обґрунтування для розробки програмно-го забезпечення різного призначення;
* практичний досвід об'єктно-орієнтованого аналізу і проектування складних програмних додатків;
* практичні навички програмування на мовах високого рівня з використанням методології об’єктно-орієнтованого програмування;
* навички реалізації графічного інтерфейсу користувача;
* закріплення навичок самостійної роботи.

Отже, ця курсова робота дає мені можливість засвоїти методологію об’єктно-орієнтованого програмування та закріпити навички, отримані протягом навчання, необхідні для подальшого розвитку у сфері.

# 1 Постановка задачі

Розробити програмне забезпечення, яке дозволяє зіграти у гру «Розміщення».

Це гра з полем 15х15 для одного учасника. Зліва, справа, знизу і зверху вказані числа. Числа зліва і зверху показують, який найбільш довгий блок чорних клітин має бути присутнім в цьому рядку або стовпці. Аналогічно - числа праворуч і знизу вказують, який найбільш довгий блок з жовтих клітин має бути присутнім в даному стовпці або рядку. Таких блоків може бути кілька. Щоб перемогти треба правильно заповнити поле, не перевищуючи ні одне задане значення.

Інтерфейс програми має включати в себе поле для гри 15х15 клітинок, де натисканням кнопки миші один раз – клітинка зафарбовується чорним кольором, двічі - жовтим. Також має бути можливість змінити колір клітинки в будь-який момент. Гра має завершуватись після натискання гравцем спеціальної кнопки, після цього запускається перевірка введених даних, якщо помилок немає – має видатись повідомлення з виграшом, якщо є – з програшом. Має бути можливість обрати рівень гри – для початківців, для досвідчених та для професіоналів. Також має бути передбачена правильна відповідь, яку можна побачити натиснувши спеціальну кнопку. Також інтерфейс повинен правила гри, можливість переграти та вийти з гри. Має бути можливість створити свою гру, якщо ввести малюнок – відповідь.

Для виконання задачі необхідно реалізувати програму використовуючи об’єктно-орієнтоване програмування. Необхідно реалізувати класи, що забезпечать створення об’єктів, працюючи з якими, полегшується робота, адже немає необхідності повторення схожих або однакових частин коду. Таким чином, враховуючи специфіку гри, необхідно розробити класи для самої гри, чисел, що вказують на розмірність найдовшого блоку, клітинок, що змінюють свій колір, кнопки для перевірки вводу, для створення персонального завдання та для виводу відповіді.

# 2 Теоретичні відомості

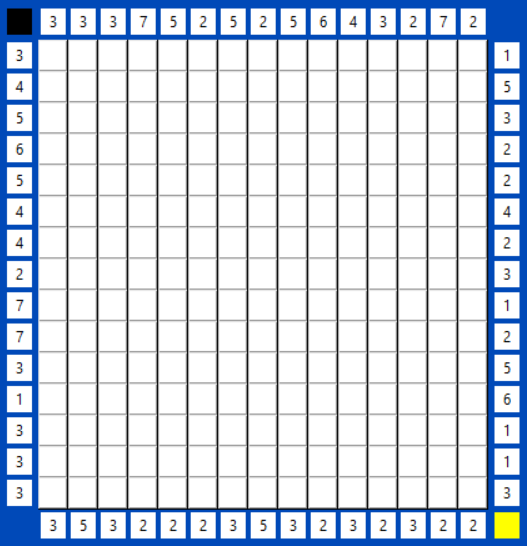
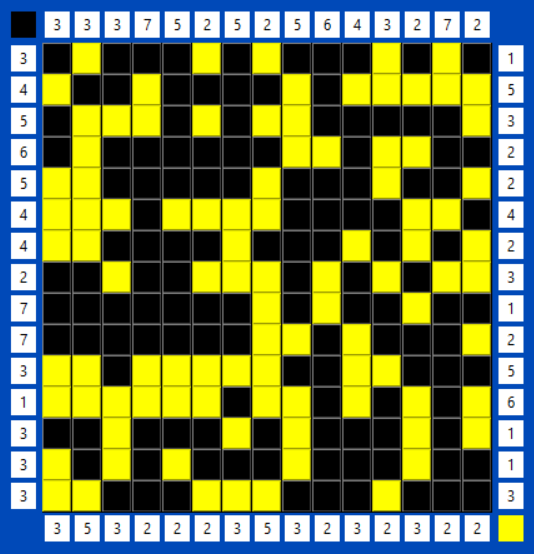
Розміщення – логічна гра на полі 15х15 клітинок на одного гравця. Зліва, справа, знизу і зверху вказані числа. Числа зліва і зверху показують, який найбільш довгий блок чорних клітин присутній в цьому рядку або стовпці. Аналогічно - числа праворуч і знизу вказують, який найбільш довгий блок з жовтих клітин присутній в даному стовпці або рядку. Таких блоків може бути кілька. Приклад задачі (рис 2.1.1), приклад правильного розв’язку рис (2.1.2).

Рисунок 2.1. 1 - Приклад завдання

Рисунок 2.1. 2 – Приклад правильного розв'язку

Якщо гравець не перевищив максимальну кількість об’єктів у блоці – він виграє, якщо ні – програє.

Сама програма має першочергово виставити «жовті клітинки», а на їх базисі вже порахувати максимальну довжину блоків. Ще програма має визначати скільки разів натиснув гравець (1 раз – чорний колір, 2 – жовтий). Має бути реалізований шлях перевірки введеної гравцем інформації.

Мають бути реалізовані також підказки, автоматичне розв’язання задачі та можливість обрання рівню складності. Користувач має мати можливість створити персональне завдання.

# 3 Опис алгоритмів

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

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| replay | Масив для перезапуску гри |
| create | Масив для перевіри чи було створене поле вручну |
| amount | Рівень обраної гри |
| game\_rules | Правила гри |
| game\_name | Назва гри |
| game\_color | Колір тла |
| main\_window | Основне вікно |
| all\_coords | Користувацький ввод |
| yel\_dots | Згенерований масив координат жовтих точок |
| bl\_horiz,yel\_horiz, bl\_vert, yel\_vert | Максимальні довжини чорних(жовтих) горизонтальних(вертикальних) блоків |
| right\_answer | Правильна відповідь |

## Загальний алгоритм

1. ПОЧАТОК
2. Створити поле, розпочати гру.
3. Створити надпис, кнопки Правила, Вийти, Перевірити, Вирішити, Відповідь, Створити завдання.
4. ЯКЩО натиснута кнопка Правила:
   1. Вивести правила
5. ЯКЩО натиснута кнопка з рівнем:
   1. Створити нове завдання відповідно рівню
6. ІНАКШЕ:
   1. Рівень - випадковий
7. Поки користувач не натиснув на кнопку «не перегравати» або «вийти»:
   1. Створити 2 бокси різних кольорів
   2. Створити правильну відповідь – місцезнаходження жовтих клітин
   3. Відштовхуючись від правильної відповіді – задати найбільшу довжину для горизонтальних блоків
   4. Відштовхуючись від правильної відповіді – задати найбільшу довжину для горизонтальних блоків
   5. Створити поле кнопок для гри
   6. ЯКЩО користувач натиснув на кнопку перевірки:
      1. Перевірити введену користувачем інформацію
      2. ЯКЩО введена користувачем інформація відповідає всім обмеженням:
         1. Вивести повідомлення про перемогу
         2. Вивести вибір перегравати гру чи ні:
            1. ЯКЩО вибрано переграти:

Розпочати гру заново

* + - * 1. ІНАКШЕ:

КІНЕЦЬ

* + 1. ІНАКШЕ:
       1. Вивести повідомлення про поразку
       2. Вивести вибір перегравати гру чи ні:
          1. ЯКЩО вибрано переграти:

Розпочати гру заново

* + - * 1. ІНАКШЕ:

КІНЕЦЬ

* 1. ЯКЩО користувач натиснув на кнопку розв’язку:
     1. Вивести правильну відповідь
     2. Вивести повідомлення про перемогу
        1. Вивести вибір перегравати гру чи ні:
           1. ЯКЩО вибрано переграти:

Розпочати гру заново

* + - * 1. ІНАКШЕ:

КІНЕЦЬ

* 1. ЯКЩО користувач натиснув на кнопку відповіді:
     1. Вивести правильну відповідь
  2. ЯКЩО користувач натиснув кнопку створити задачу:
     1. Видати вікно для створення задачі
     2. Зчитати введення користувача
     3. ЯКЩО усі поля заповнені:
        1. Створити задачу з заданими координатами
     4. ІНАКШЕ:
        1. Вивести вікно помилки

1. КІНЕЦЬ

## Алгоритми методів

* + 1. Алгоритм методів класу ButtonToPlay

1. ПОЧАТОК
2. Задати колір фону, текст, ширину, висоту кнопки.
3. ЯКЩО користувач натиснув 1 раз:
   1. Змінити колір фону на чорний
   2. Додати до координат місцезнаходження цієї кнопки та її колір
   3. ПОКИ така сама координата але жовтого кольору присутня у введеній користувачем інформації:
      1. Видалити цю координату зі списку координат
4. ЯКЩО користувач натиснув 2 рази:
   1. Змінити колір фону на жовтий
   2. Додати до координат місцезнаходження цієї кнопки та її колір
   3. ПОКИ така сама координата але чорного кольору присутня у введеній користувачем інформації:
      1. Видалити цю координату зі списку координат
5. КІНЕЦЬ
   * 1. Алгоритм generate\_yel\_dots
        + 1. ПОЧАТОК
          2. Створити пустий масив yel\_dots
          3. ПОВТОРИТИ amount раз:
   1. i = randint(0,14)
   2. j = randint(0,14)
   3. Додати до масиву yel\_dots i та j
6. ПОВТОРИТИ 15 разів:
   1. Додати до масиву yel\_dots (i,15)
   2. Додати до масиву yel\_dots (15, i)
7. Видалити дублікати з масиву yel\_dots
8. Відсортувати масив yel\_dots
9. Повернути масив yel\_dots
10. КІНЕЦЬ

3.2.3 Алгоритм check\_us\_input:

1. ПОЧАТОК
2. check\_if right = []
3. Порахувати кількість введених координат
4. ЯКЩО ця кількість != 225:
   1. ВИДАТИ помилку
   2. ВИДАТИ вікно програшу
   3. Додати 1 до check\_if\_right
5. ІНАКШЕ:
   1. ПОВТОРИТИ для жовтого та чорного кольорів:
      1. j = 0
      2. ПОКИ j<15:
         1. Створити масив list\_of\_lengths = []
         2. length = 0
         3. i = 0
         4. ПОКИ i<15:
            1. ЯКЩО (j,i,колір) є в списку координат:

length+=1

ЯКЩО (j,i+1,колір) не є в списку координат:

Додати length до list\_of\_lengths

i+=1

length = 0

i+=1

* + - 1. ЯКЩО list\_of\_lengths пустий:
         1. Додати 0 до list\_of\_lengths
      2. ЯКЩО макс довжина кольору не є в list\_of lengths АБО існує значення у list\_of\_length, що перевищує його:
         1. ВИДАТИ помилку
         2. ВИДАТИ вікно програшу
         3. Додати 1 до check\_if\_right
    1. j+=1
  1. ПОВТОРИТИ для жовтого та чорного кольорів:
     1. j = 0
     2. ПОКИ j<15:
        1. Створити масив list\_of\_lengths = []
        2. length = 0
        3. i = 0
        4. ПОКИ i<15:
           1. ЯКЩО (i,j,колір) є в списку координат:

length+=1

ЯКЩО (i+1,j,колір) не є в списку координат:

Додати length до list\_of\_lengths

i+=1

length = 0

i+=1

* + - 1. ЯКЩО list\_of\_lengths пустий:
         1. Додати 0 до list\_of\_lengths
      2. ЯКЩО макс довжина кольору не є в list\_of lengths АБО існує значення у list\_of\_length, що перевищує його:
         1. ВИДАТИ помилку
         2. ВИДАТИ вікно програшу
         3. Додати 1 до check\_if\_right
    1. j+=1
  1. ЯКЩО check\_if\_right пустий:
     1. Видати вікно виграшу
  2. ІНАКШЕ:
     1. Видати вікно програшу

3.2.4 Алгоритм методів класу CreateTask

1. ПОЧАТОК
2. Створити кнопку
3. ЯКЩО користувач натиснув на кнопку:
   1. Створити нове вікно
   2. Очистити попередні введення користувача
   3. Закрити головне вікно
   4. Створити поле з кнопок
   5. Створити кнопку DoneTask
   6. ЯКЩО користувач натиснув на DoneTask:
      1. Перевірити правильність введення
      2. ЯКЩО інформація була введена правильно:
         1. Додати до масиву create 1
         2. Закрити останнє вікно
         3. Запустити гру заново вже з заданими координатами
      3. ІНАКШЕ:
         1. Вивести помилку
         2. Очистити введення
         3. Створити звичайну гру

# Опис програмного забезпечення

## 4.1 Діаграма класів програмного забезпечення

Рисунок 4. 1 - Діаграма класів

Створено класи: WindowMaker – створення вікна гри, назви і правил гри; ColorAmounts – текст з максимальними значеннями; SolveButton – кнопка правильного розв’язку, CheckButton – кнопка перевірки користувацького вводу; ButtonToPlay – кнопки для гри, що змінюють свій колір при натисканні; AnswerButton – кнопка для виведення підказок для користувача; ExitButton – кнопка завершення гри; RulesButton – кнопка виводу правил; CreateTask – кнопка створення персонального завдання; DoneTask – затвердження завдання, LevelButton – обрання рівня гри. Діаграма цих класів зображена на Рисунку 4.1.

SolveButton, CheckButton, ButtonToPlay, RulesButton, CreateTask, DoneTask – наслідують Button. AnswerButton, ExitButton, LevelButton наслідують CheckButton. TK містить WindowMaker. Між усіма іншими класами та модулем TK існують зв’язки агрегації. Усі класи можуть існувати окремо.



## Опис методів частин програмного забезпечення

### 4.2.1 Стандартні методи

У таблиці 4.1 показано методи графічної бібліотеки tkinter[1][2], що були використані при написанні програми.

Таблиця 4.1 – Стандартніметоди

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | tkinter | init() | Ініціалізувати модуль tkinter | - | - | AnswerButton.py, WindowMaker.py, ExitButton.py, ButtonToPlay.py, CheckButton.py, ColorAmounts.py, CreateTask.py, RulesButton.py, SolveButton.py |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 2 | tkinter.Tk | init() | Створити основне вікно для гри | Відсутні | об’єкт tkinter.Tk | WindowMaker.py |
| 3 | tkinter.Tk | title(text) | Задати заголовок для гри | text - заголовок | - | WindowMaker.py |
| 4 | tkinter.Tk | geometry(size) | Задати розмір основного вікна для гри | size - розмір | - | WindowMaker.py |
| 5 | tkinter.Tk | resizable (height, width) | Задати можливість зміни розміру основного вікна для гри | height – висота, width - широта | - | WindowMaker.py |
| 6 | tkinter.Tk | config(options) | Задати додаткові характеристики для основного вікна для гри | options - форматування вікна | - | WindowMaker.py |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 7 | tkinter.  messagebox | messagebox.askquestion(title, message\_to\_be\_displayed) | Вивести діалогове вікно з 2 варіантами відповіді(“Yes”/ “No”) | title – заголовок,  message\_to\_be\_displayed - повідомлення | об’єкт tkinter.messagebox | funcs\_interface.py, funcs\_buttons.py, AnswerButton.py,  ExitButton.py |
| 8 | tkinter.Label | init(master, options) | Вивести віджет з текстом або картинкою | master – вікно гри,  options - форматування віджета | об’єкт tkinter.Label | WindowMaker.py, RulesButton.py, ColorAmounts.py. |
| 9 | tkinter.Label | place(x,y) | Розташувати віджет з текстом або картинкою | x – розташування по х, у – розташування по у | - | WindowMaker.py, RulesButton.py, ColorAmounts.py. |
| 10 | tkinter.Button | init(master, options\_ | Вивести кнопку | master – вікно гри,  options - форматування кнопки | об’єкт tkinter.Button | CheckButton.py, AnswerButton.py, SolveButton.py, ExitButton.py, RulesButton.py, ButtonToPlay.py, CreateTask.py |

Продовження таблиці 4.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 11 | tkinter.Button | place(x,y) | Розташувати кнопку | x – розташування по х, у – розташування по у | - | CheckButton.py, AnswerButton.py,SolveButton.py, ExitButton.py, RulesButton.py, ButtonToPlay.py, CreateTask.py |
| 12 | tkinter.Button | bind(event, function) | Розташувати кнопку | event – подія,  function - функція | Виконання функції в залежності від події | ButtonToPlay.py |
| 13 | tkinter.Text | init(master, options) | Створити текстове поле | master – вікно гри, options - форматування тексту | об’єкт tkinter.Text | WindowMaker.py, RulesButton.py |
| 14 | tkinter.Text | insert(index, string) | Додати текст до текстового поля | index - індекс, string - текст | - | WindowMaker.py, RulesButton.py |
| 15 | tkinter.Text | place(x,y) | Розташувати текст | x – розташування по х, у – розташування по у | - | WindowMaker.py, RulesButton.py |

### 4.2.2 Користувацькі методи

В таблиці 4.2 показано методи, створені мною для більш зручної робити з програмою.

Таблиця 4.2 – Користувацькі методи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | WindowMaker | \_\_init\_\_(self,game\_name=" ", game\_rules=" ", bg\_color=" ", replay=[]) | Створити вікно гри | self – саме вікно  game\_name – назва гри  game\_rules – правила гри bg\_color – колір фону replay – масив для визначення чи буде гра починатись заново | об’єкт tkinter.Tk | WindowMaker.py |
| 2 | WindowMaker | make\_the\_window(self) | Створити назву програми, встановити розміри вікна, поставити колір фону | self – саме вікно | - | WindowMaker.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 3 | WindowMaker | make\_label(self) | Створити заголовок з назвою програми | self – саме вікно | - | WindowMaker.py |
| 4 | RulesButton | make\_rules(  self, new\_window) | Створити текст правил | self – сама кнопка,  new\_window – нове вікно | - | RulesButton.py |
| 5 | RulesButton | make\_new\_window(self) | Створити вікно для правил | self – сама кнопка | - | RulesButton.py |
| 6 | ColorAmounts | init(self) | Створити максимальні значення блоків | self – лейбл з текстом | об’єкт tkinter.Label | ColorAmounts.py |
| 7 | ButtonToPlay | init(self, i, j, all\_coords) | Створити кнопку для гри | self – сама кнопка,  i, j – координати кнопки по х та у відповідно, all\_coords – введення користувача | об’єкт tkinter.Button | ButtonToPlay.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 8 | ButtonToPlay | pressed\_once(event, but, i, j, all\_coords) | Змінити колір та додати до масиву нові координати при натисканні 1 раз | event – подія,  but – кнопка, i,j – координати, all\_coords – введення користувача | - | ButtonToPlay.py |
| 9 | ButtonToPlay | pressed\_twice(event, but, i, j, all\_coords) | Змінити колір та додати до масиву нові координати при натисканні 2 рази | event – подія,  but – кнопка, i,j – координати, all\_coords – введення користувача | - | ButtonToPlay.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 10 | CheckButton | init(self, all\_coords,bl\_vert, yel\_vert, bl\_horiz, yel\_horiz, window) | Створити кнопку для перевірки користувацького вводу. | self – сама кнопка; all\_coords – користувацький ввід; bl\_vert, yel\_vert, bl\_horiz, yel\_horiz, - максимальна довжина блоків по-вертикалі(по-горизонталі) чорного(жовтого) кольорів; window – вікно гри | об’єкт tkinter.Button | CheckButton.py |
| 11 | SolveButton | init(self, all\_coords, right\_coords,window) | Кнопка для виведення правильного рішення задачі | self – сама кнопка, all\_coords – користувацький ввід, right\_coords – правильна відповідь,window – вікно гри | об’єкт tkinter.Button | SolveButton.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 12 | SolveButton | create\_right\_button\_field(self) | Вивести правильне рішення задачі | self – сама кнопка | - | SolveButton.py |
| 13 | AnswerButton | init(self, yel\_coord) | Кнопка для виведення підказки до задачі | self – сама кнопка, yel\_coord – сгенеровані програмою відповіді. | об’єкт tkinter.Button | AnswerButton.py |
| 14 | AnswerButton | show\_answer(self) | Вивести підказки до задачі | self – сама кнопка | - | AnswerButton.py |
| 15 | RulesButton | init(self, game\_rules, bg\_color) | Вивести правила до гри | self – сама кнопка, game\_rules – правила гри, bg\_color – колір фону | об’єкт tkinter.Button | RulesButton.py |
| 16 | CreateTask | init(self, old\_window, bg\_color, all\_coord, create, main\_window) | Створити кнопку для створення персонального завдання | self – сама кнопка, old\_window – вікно запуску, bg\_color – колір фону, all\_coord – введення користувачем, create – створення поля, main\_window – головне вікно | об’єкт tkinter.Button | CreateTask.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 17 | CreateTask | create\_field(self) | Створити поле з кнопок | self – сама кнопка | - | CreateTask.py |
| 18 | DoneTask | init (self, old\_window, main\_window, all\_coord, create) | Створити кнопку для завершення вводу персонального завдання | self – сама кнопка, old\_window – вікно для вводу задачі, main\_window – головне вікно, all\_coord, create | - | CreateTask.py |
| 19 | LevelButton | init(self, level\_name, amount\_dots, amounts, plus\_pos, window) | Створити кнопку для обрання рівня | self – сама кнопка, level\_name – назва рівню, amount\_dots – кількість жовтих боксів, amounts – масив для створення гри певної складності, plus\_pos – положення по у, window – головне вікно | об’єкт tkinter.Button | LevelButton.py |

Продовження таблиці 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 20 | LevelButton | pressed(self) | Послідовність дій якщо кнопка натиснута | self – сама кнопка | - | LevelButton.py |
| 21 | LevelButton | create\_level\_title(self) | Створити заголовок «Level» | self – сама кнопка | - | LevelButton.py |

# 5 Тестування програмного забезпечення

## 5.1 План тестування

Треба протестувати програмне забезпечення на коректну роботу при різних даних та перевірити весь основний функціонал, для цього складемо план тестування програми.

1. Тестування кнопок поля для гри:
   1. Тестування зміни кольору при натисканні(1 раз та 2 рази)
   2. Тестування багатьох змін кольору однієї кнопки під час гри.
   3. Тестування правильного додавання координат натиснутих кнопок до масиву користувацького вводу.
2. Тестування кнопок опцій
   1. Тестування кнопки автоматичного розв’язку
   2. Тестування кнопки підказки
   3. Тестування кнопки перевірки
3. Тестування вводу
   1. Тестування неповного вводу користувача
   2. Тестування повністю неправильного вводу
   3. Тестування повністю правильного вводу
4. Тестування запуску ігри заново
   1. Після перемоги
   2. Після поразки
5. Тестування завершення гри
   1. Після перевірки
   2. З головного вікна
6. Тестування створення персональної гри
   1. Правильне створення
   2. Неправильне створення
7. Тестування кнопок рівня

Тестування ( таблиці 5.1-5.15)

## 5.2 Приклади тестування

Тестування кнопок поля для гри:

Таблиця 5.1 – Тестування зміни кольору при натисканні( 1 раз та 2 рази)

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість зміни кольору при натисканні |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути на будь-яку кнопку 1, а поті 2 рази |
| Очікуваний результат | Зміна кольору спочатку на чорний, а потім на жовтий |
| Стан програми після проведення випробувань | Поточна кнопка правильно змінила колір |

Таблиця 5.2 – Тестування багатьох змін кольору однієї кнопки під час гри.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість неоднократної зміни кольору при натисканні під час гри |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Правильно заповнити поле, переобрати колір однієї кнопки на інший, а потім знов на правильний |
| Очікуваний результат | Видача вікна перемоги |
| Стан програми після проведення випробувань | Вікно перемоги |

Таблиця 5.3 – Тестування правильного додавання координат натиснутих кнопок до масиву користувацького вводу.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність додавання позиції кнопки до масиву користувацького вводу |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Будь-яке заповнення поля |
| Очікуваний результат | Відповідне додавання координат кнопки до масиву |
| Стан програми після проведення випробувань | Усі данні в масиві є правильними |

Тестування кнопок опцій:

Таблиця 5.4 – Тестування кнопки автоматичного розв’язку.

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи кнопки автоматичного розв’язку |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути на кнопку розв’язку |
| Очікуваний результат | Правильне зафарбування поля |
| Стан програми після проведення випробувань | Поле зафарбовано правильно |

Таблиця 5.5 – Тестування кнопки підказки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи кнопки підказки |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути на кнопку підказки |
| Очікуваний результат | Видача текстової підказки |
| Стан програми після проведення випробувань | Видача повідомлення з текстовою підказкою |

Таблиця 5.6 – Тестування кнопки перевірки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи кнопки перевірки |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути на кнопку підказки |
| Очікуваний результат | Перевірка вводу користувача |
| Стан програми після проведення випробувань | Видача відповідного до вводу повідомлення |

Тестування вводу:

Таблиця 5.7 – Тестування неповного вводу користувача

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи програми при неповному вводу користувача |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути не на всі кнопки, а потім натиснути на кнопку перевірки |
| Очікуваний результат | Відповідна реакція на неповне введення інформації користувачем |
| Стан програми після проведення випробувань | Видача вікна помилки |

Таблиця 5.8 – Тестування повністю неправильного вводу

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи програми при повністю неправильному введені |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Ввести повністю неправильну відповідь, а потім натиснути на кнопку перевірки |
| Очікуваний результат | Відповідна реакція на неправильне введення інформації користувачем |
| Стан програми після проведення випробувань | Видача вікна помилки |

Таблиця 5.9 – Тестування повністю правильного вводу

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи програми при повністю неправильному введені |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Ввести повністю правильну відповідь, а потім натиснути на кнопку перевірки |
| Очікуваний результат | Відповідна реакція на правильну введення інформації користувачем |
| Стан програми після проведення випробувань | Видача вікна перемоги |

Тестування запуску ігри заново:

Таблиця 5.10 – Тестування запуску ігри заново після перемоги

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи перезапуску після перемоги у грі |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Ввести повністю правильну відповідь, а потім натиснути на кнопку перевірки, отримати вікно про виграш, натиснути «так» |
| Очікуваний результат | Нова гра |
| Стан програми після проведення випробувань | Запуск нової гри |

Таблиця 5.11 – Тестування запуску ігри заново після поразки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи перезапуску після поразки у грі |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Ввести повністю неправильну відповідь, а потім натиснути на кнопку перевірки, отримати вікно про поразку, натиснути «так» |
| Очікуваний результат | Нова гра |
| Стан програми після проведення випробувань | Запуск нової гри |

Тестування завершення гри:

Таблиця 5.12 – Тестування завершення гри після перевірки

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи при завершенні |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Ввести повністю будь-яку відповідь, а потім натиснути на кнопку перевірки, отримати вікно про поразку(перемогу), натиснути «ні» |
| Очікуваний результат | Закриття гри |
| Стан програми після проведення випробувань | Закриття програми |

Таблиця 5.13 – Тестування завершення гри з головного вікна

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи при завершенні |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Ввести повністю будь-яку відповідь(або не вводити нічого) і натиснути на кнопку виходу |
| Очікуваний результат | Закриття гри |
| Стан програми після проведення випробувань | Закриття програми |

Тестування створення персонального завдання

Таблиця 5.14 - Тестування правильного створення персонального завдання

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи при правильному створенні персонального завдання |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути на кнопку CreateTask, заповнити поле повністю, натиснути на Done |
| Очікуваний результат | Створення персонального завдання |
| Стан програми після проведення випробувань | Відриття головного вікна зі створеним персональним завданням |

Таблиця 5.15 - Тестування не правильного створення персонального завдання

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи при не правильному створенні персонального завдання |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути на кнопку CreateTask, заповнити поле не повністю, натиснути на Done |
| Очікуваний результат | Вікно помилки |
| Стан програми після проведення випробувань | Вікно помилки |

Тестування вибору рівня

Таблиця 5.16 - Тестування вибору рівня

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність роботи при виборі рівня |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | - |
| Схема проведення тесту | Натиснути на кнопку Easy або Medium або Hard |
| Очікуваний результат | Видача завдання відповідно до рівня |
| Стан програми після проведення випробувань | Видача завдання відповідно до рівня |

# 6 Інструкція користувача

## 6.1 Робота з програмою

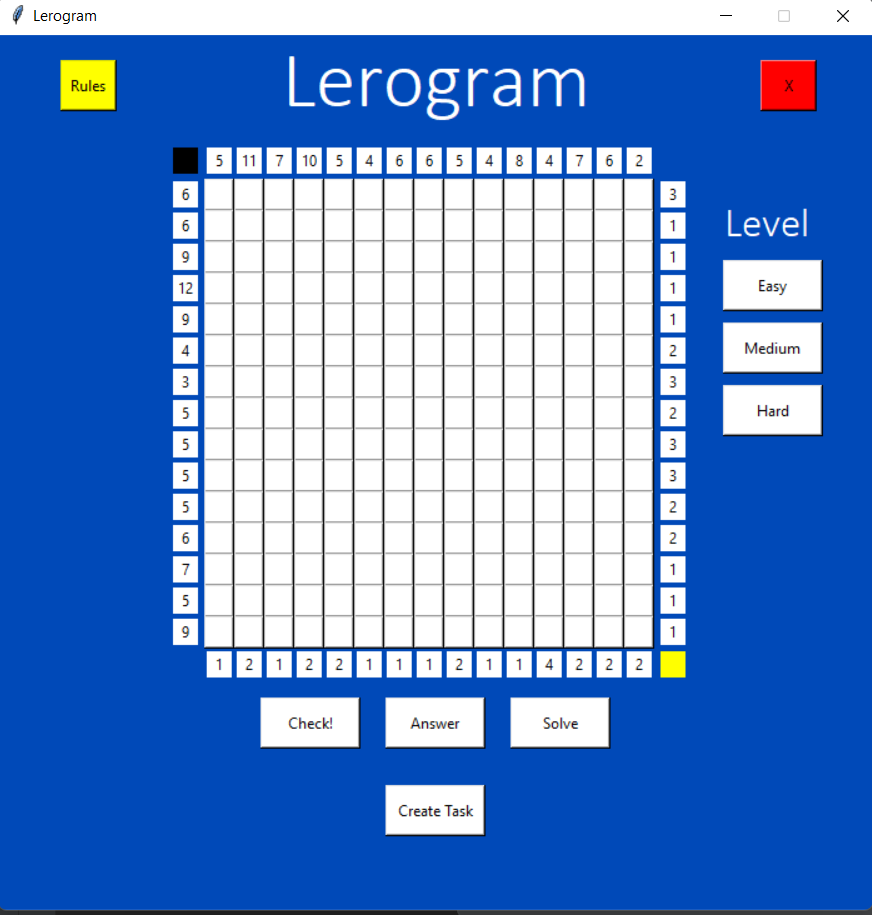
Після запуску виконавчого файлу з розширенням \*.exe, відкривається головне вікно програми (Рисунок 6.1).

Рисунок 6.1 – Головне вікно програми

Вікно гри має розмір 700x700, який не може бути зміненим. Почати грати можна з самого моменту запуску. Посередині поле для гри, знизу і зверху кнопки різного призначення. Щоб вийти треба натиснути на кнопку з хрестиком червоного кольору справа зверху.

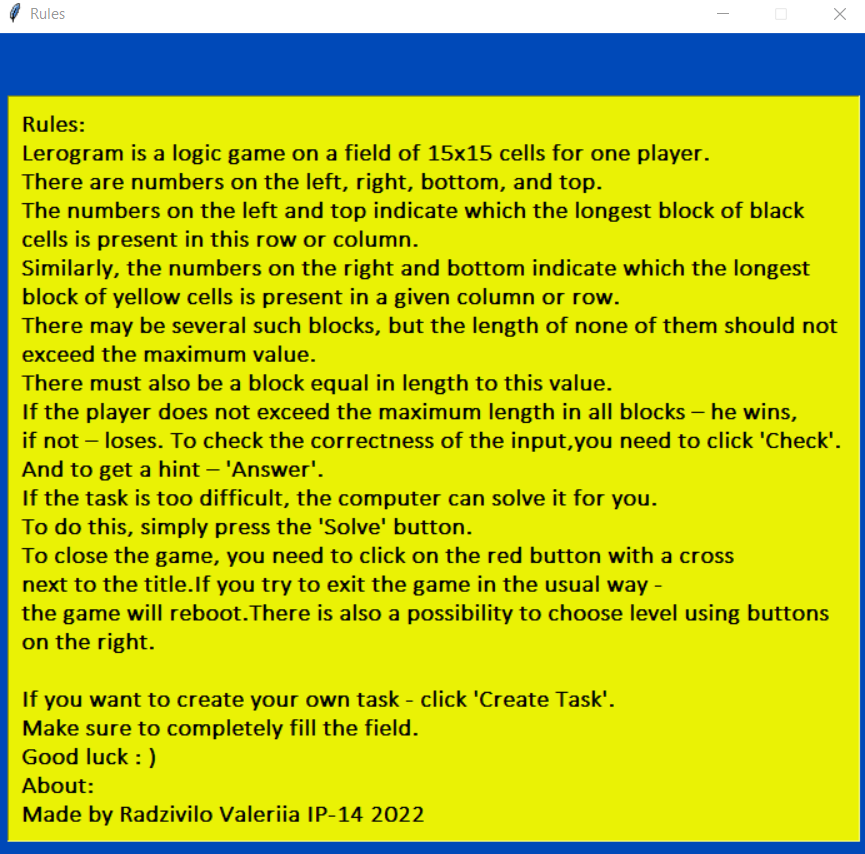
 Якщо натиснути жовту кнопку Rules у лівому верхньому кутку з’являться правила гри( Рис. 6.2). Це вікно можна закрити натиснувши на хрестик.

Рисунок 6.2 – Правила гри

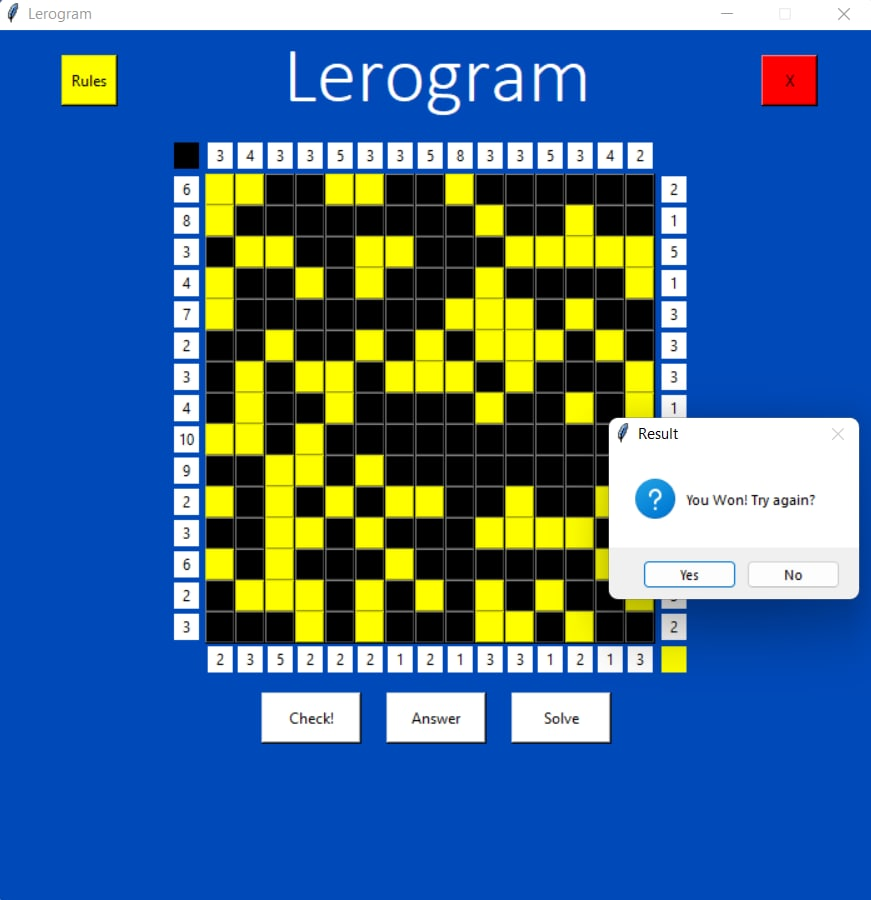
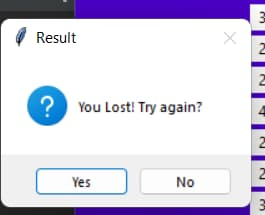
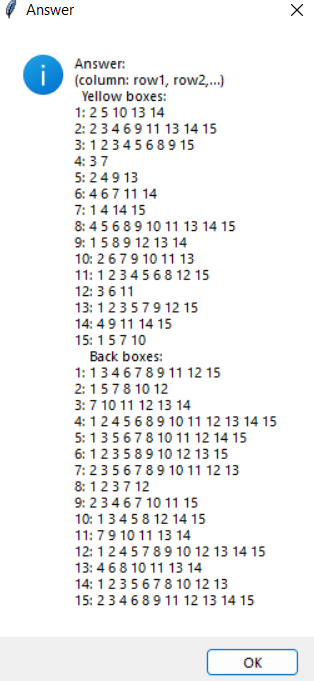
Якщо натиснути на кнопку “Check!” в залежності від правильності рішення задачі може бути 2 вікна( перемога – Рис 6.3, поразка – Рис 6.4)

Рисунок 6.3 – Перемога Рисунок 6.4 - Поразка

При натисканні кнопки Answer видається підказка до розв’язку задачі(Рис 6.5).

 Рисунок 6.5 – Підказка рішення

Якщо натиснути кнопку Solve – задача вирішується автоматично за користувача(Рис 6.6).

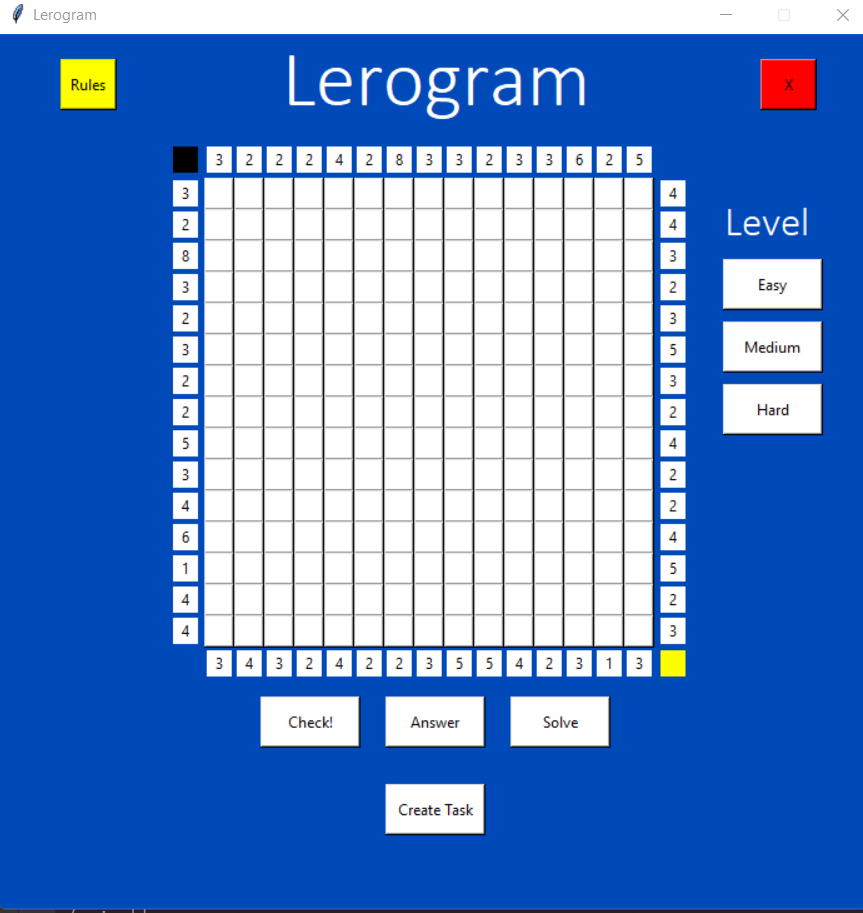
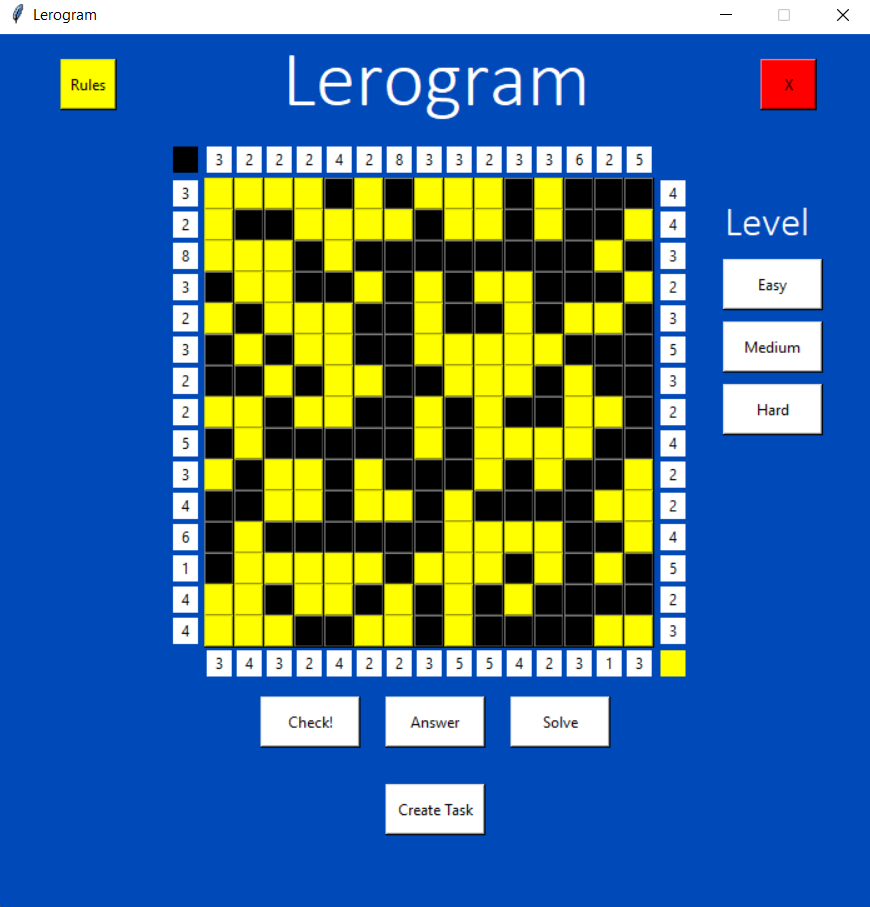
 

Рисунок 6.6 – Автоматичне рішення задачі

При натисканні на кнопки рівнів (Easy, Medium, Hard) відображаються завдання відповідно до обраного варіанту.

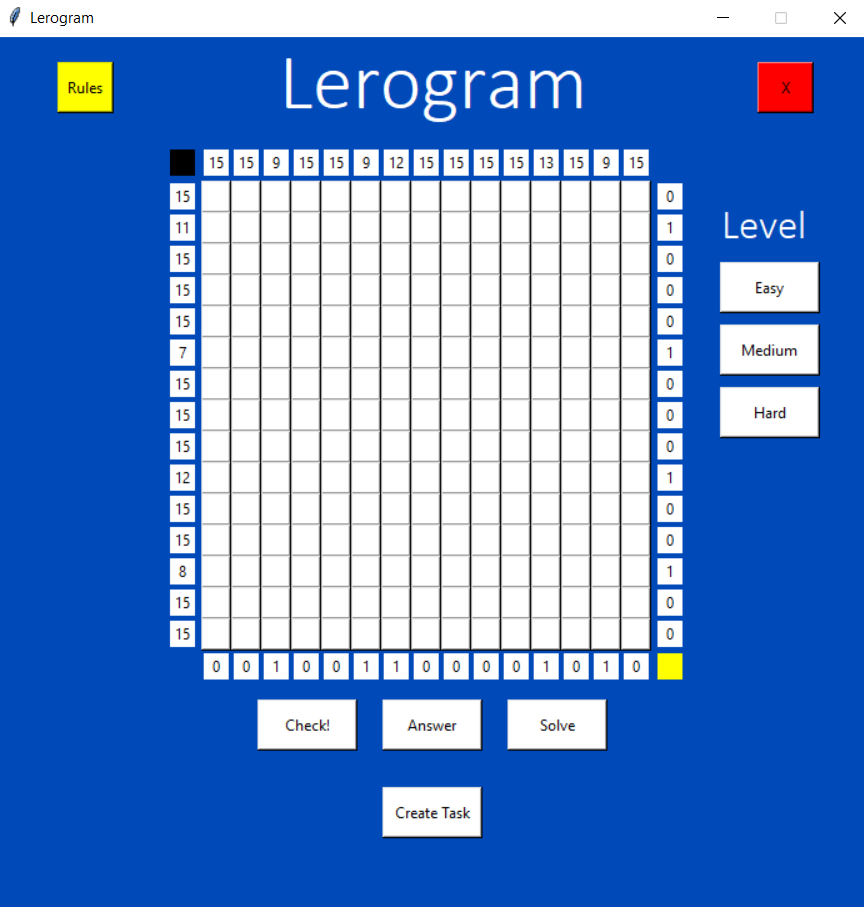
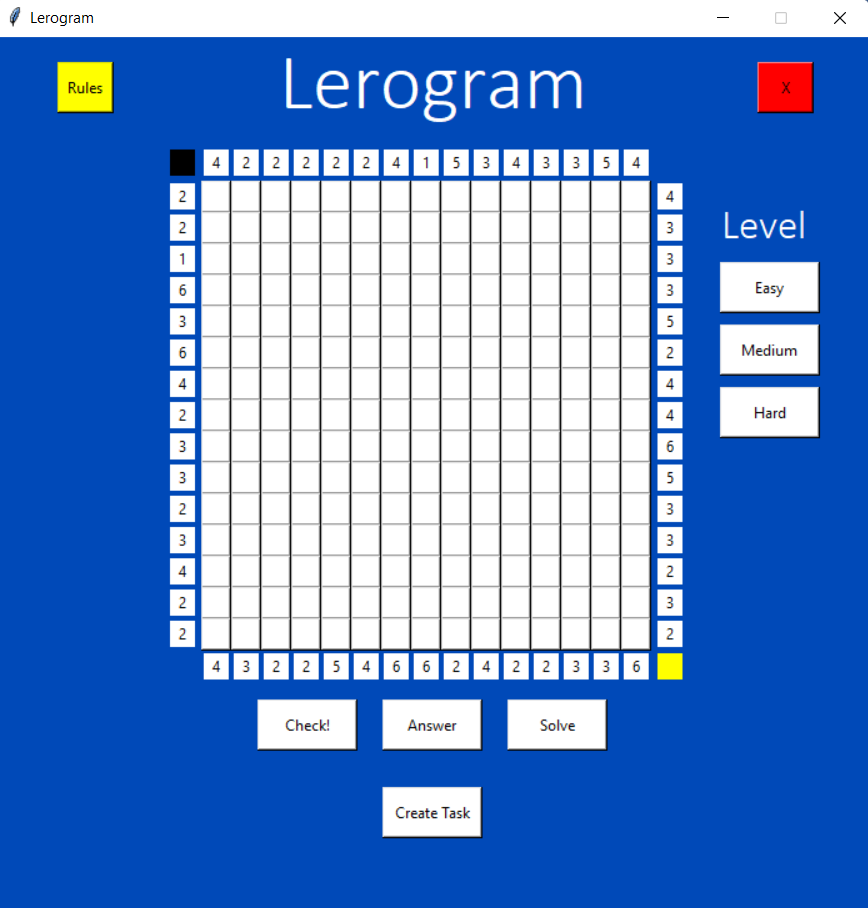
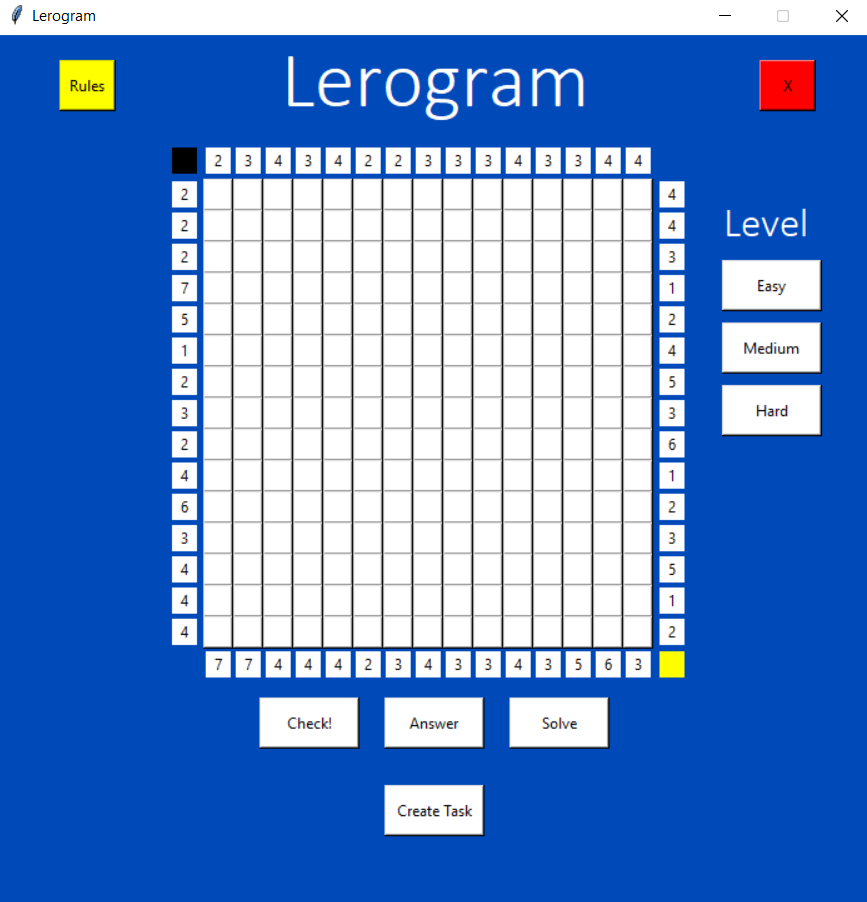
  

Рисунок 6.7 – Рівні завдань(Легкий, Середній, Важкий)

Якщо натиснути на кнопку CreateTask – можна створити завдання власноруч. Треба повністю заповнити поле і натиснути кнопку Done. Відриється головне вікно з новою, зробленою користувачем грою.

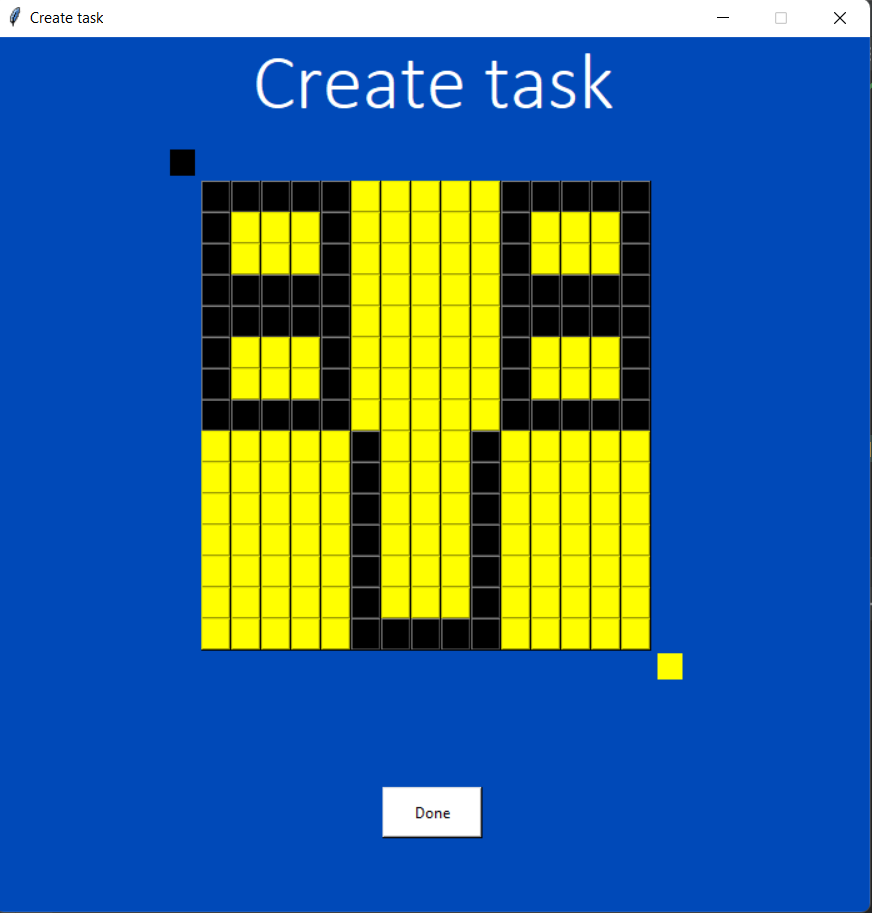
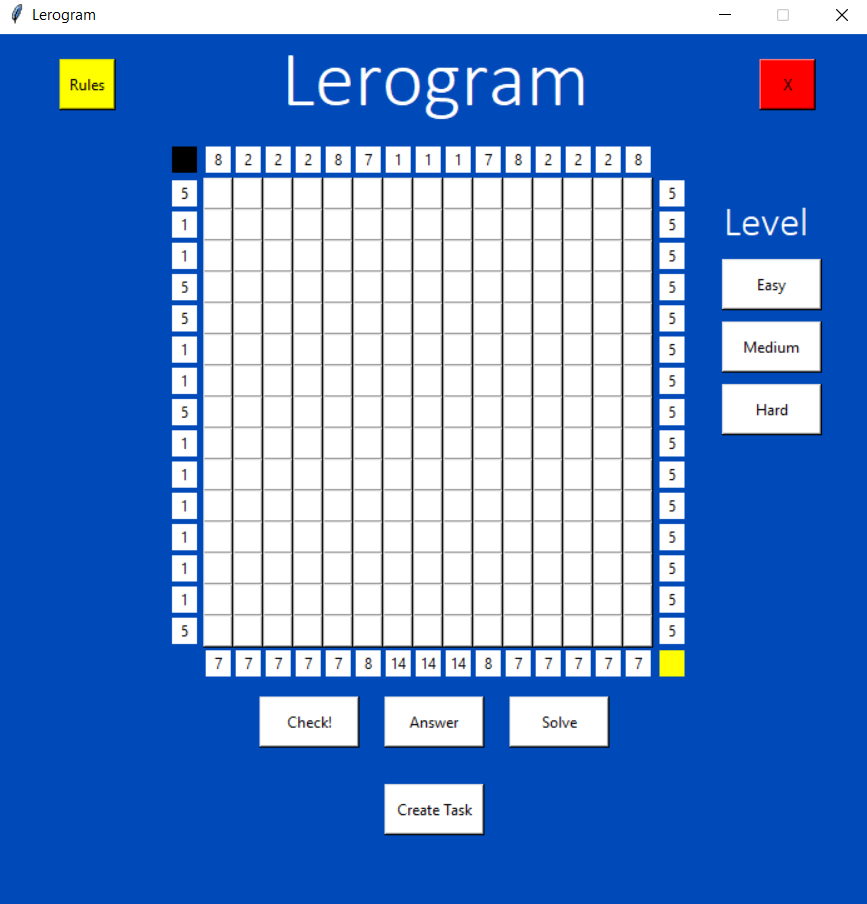
 

Рисунок 6.8 – Створення персонального завдання

Інструкція до гри:

Розміщення – логічна гра на полі 15х15 клітинок на одного гравця. Зліва, справа, знизу і зверху вказані числа. Числа зліва і зверху показують, який найбільш довгий блок чорних клітин присутній в цьому рядку або стовпці. Аналогічно - числа праворуч і знизу вказують, який найбільш довгий блок з жовтих клітин присутній в даному стовпці або рядку. Таких блоків може бути кілька, але довжина жодного з них не має бути більша за максимальне значення. Також має бути присутній блок рівний по дожині до цього значення.

Якщо гравець не перевищив максимальну довжину у всіх блоках – він виграє, якщо ні – програє.

Щоб перевірити правильність вводу треба натиснути Check. А щоб отримати підказку – Answer. Якщо задача є надто складною – її може вирішити комп’ютер за вас. Для цього достатньо натиснути кнопку Solve.

Для обрання рівня є кнопки справа Easy. Medium, Hard. Натиснувши на них запускається завдання певного рівня. За замовчуванням рівень – випадковий.

Натиснувши кнопку Create Task можна створити персональну гру, повністю заповнивши поле чорними та жовтими квадратами.

Для того, щоб закрити гру необхідно натиснути на червону кнопку з хрестиком поруч біля назви. При спробі вийти з гри звичайним шляхом – гра перезавантажиться.

## 6.2 Вхідні та вихідні дані

Вхідні дані отримуються через мишу – натискання на кнопки.

А вихідні данні виводяться на дисплей та у консоль – виводяться усі ключові дії та введені данні.

## 6.3 Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

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

Висновки

Під час виконання цієї роботи були застосовані знання з програмування та об’єктно-орієнтовного програмування, яким я навчилась за рік. Також був створений інтерфейс, що є більш складним у написанні але легшим у користуванні, ніж ті, що я робила раніше.

Була застосована графічна бібліотека tkinter, що дозволяє створювати якісний та простий у користуванні інтерфейс зі зручним додаванням власних функцій та методів. ООП дозволило розділити програму на частини і працювати з цими частинами окремо та інтегруючи їх одна в одну. Такий код набагато легше покращувати та доповнювати новим функціоналом, також він стає більш читабельним.

Гра працює при будь-якому введені, швидко, чітко та правильно. Були наведені теоретичні відомості, описано алгоритм, таблиці користувацьких і стандартних методів, діаграма класів, результати тестувань та інструкція користувача. Тому, виконання даного завдання можна назвати успішним.

Навики отримані в ході виконання даного завдання, так як самостійний пошук, ознайомлення та використання набутих навичок, знадобляться, як у подальшому розвитку у обраній мною сфері, так і у звичайному житті.

Перелік посилань

* 1. <https://docs.python.org/3/library/tkinter.html> - офіційна документація tkinter – 20.05.2022
  2. <https://web.archive.org/web/20190524140835/https://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html> - довідник по tkinter – 10.04.2022

Додаток А Технічне завдання

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник Головченко М. М.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 р.

Виконавець:

Студентка Радзівіло Валерія Артемівна

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: Гра «Розміщення»

з дисципліни:

«Основи програмування»

Київ 2022

* 1. *Мета*: Метою курсової роботи є розробка програмного забезпечення для гри «Розміщення»
  2. *Дата початку роботи*: «12» квітня 2022 р.
  3. *Дата закінчення роботи*: «12» червня 2022 р.
  4. *Вимоги до програмного забезпечення*.

1. Функціональні вимоги:

* Можливість генерації випадкового завдання.
* Можливість генерації розв’язуваної задачі.
* Можливість розв’язання задачі вручну.
* Можливість надання підказок.
* Можливість розв’язання задачі автоматично.
* Можливість перевірки відповіді користувача.
* Можливість генерації задачі вручну.
* < вимоги до функціональних характеристик>

1. Нефункціональні вимоги:

* Можливість використовувати програму на пк Windows 8 та вище.
* Можливість програмі мати головне вікно з опціями.
* Ігрове поле має мати 225 кнопок(поле 15х15) для самої гри, що можуть змінювати свій колір.
* Можливість закінчення гри без перевірки вводу.
* <*вимоги до надійності, складу і параметрів технічних засобів тощо*>
* Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

* 1. *Стадії та етапи розробки*:

1. Об'єктно-орієнтований аналіз предметної області задачі (до 01.03.2022 р.)
2. Об'єктно-орієнтоване проектування архітектури програмної системи (до 03.03.2022р.)
3. Розробка програмного забезпечення (до 08.05.2022р.)
4. Тестування розробленої програми (до 25.05.2022р.)
5. Розробка пояснювальної записки (до 01.06.2022 р.).
6. Захист курсової роботи (до 12.06.2022 р.).
   1. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

# Додаток Б Тексти програмного коду

*студента групи ІП-14 І курсу*

*Радзівіло В.А.*

(Вид носія даних)

*Електронний носій*

(Найменування програми (документа))

*Тексти програмного коду гри «Розміщення»*

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

*40 арк, 23.4 MB*

# MAIN.PY

import regular\_game

# array to record replays

replay = ['y']

# array to record created task

create = []

# create an array for all presssed buttons and their color

all\_coords = []

# level

amounts = []

# while user wants to replay

re = 0

while True:

regular\_game.regular\_game(create, replay, all\_coords, amounts)

if replay and replay.pop() == 'n':

break

print("Replay in main: ", replay)

# REGULAR\_GAME.PY

from random import randint

import funcs\_logic

import funcs\_interface

from WindowMaker import WindowMaker

from AnswerButton import AnswerButton

from ExitButton import ExitButton

from SolveButton import SolveButton

from CheckButton import CheckButton

from RulesButton import RulesButton

from CreateTask import CreateTask

from LevelButton import LevelButton

def regular\_game(create=[], replay=[], all\_coords=[], amounts=[]):

game\_rules = "Rules:\nLerogram is a logic game on a field of 15x15 cells for one player.\nThere are numbers on " \

"the left, right, bottom, and top.\nThe numbers on the left and top indicate which the longest block " \

"of black\ncells is present in this row or column.\nSimilarly, the numbers on the right and bottom " \

"indicate which the longest\nblock of yellow cells is present in a given column or row.\nThere may be " \

"several such blocks, but the length of none of them should not exceed the maximum value.\nThere " \

"must also be a block equal in length to this value.\nIf the player does not exceed the maximum " \

"length in all blocks – he wins,\nif not – loses. To check the correctness of the input,you need to " \

"click 'Check'.\nAnd to get a hint – 'Answer'.\nIf the task is too difficult, the computer can solve " \

"it " \

"for you.\nTo do this, simply press the 'Solve' button.\nTo close the game, you need to click on the " \

"red button with a cross\nnext to the title.If you try to exit the game in the usual way -\nthe game " \

"will reboot." \

"There is also a possibility to choose level using buttons on the right.\n" \

"\nIf you want to create your own task - click 'Create Task'.\nMake sure to completely fill the " \

"field.\nGood luck : ) \n" \

"About:\nMade by Radzivilo Valeriia IP-14 2022 "

game\_name = "Lerogram" # title

bg\_color = '#0049b8' # game's background color

# create main window with the game

main\_window = WindowMaker(game\_name, bg\_color, replay)

# levels

LevelButton("Easy", randint(1, 20), amounts, 0, main\_window).create\_level\_title()

LevelButton("Medium", randint(190, 220), amounts, 50, main\_window)

LevelButton("Hard", randint(100, 200), amounts, 100, main\_window)

# button to quit game

ExitButton(main\_window)

# button to open the rules

RulesButton(game\_rules, bg\_color)

# make label bg and rules sections

funcs\_interface.make\_window\_label\_board(main\_window)

# make black and yellow boxes in the corners

funcs\_interface.make\_two\_color\_boxes()

# button for user to create personal task

CreateTask(main\_window, bg\_color, all\_coords, create, main\_window)

# if level is not chosen

if not amounts:

print("Level: Random")

amounts.append(randint(10, 220))

# if user did not choose CreateTask - the task will be random

if not create or create.pop() != 1:

print("Random:")

# generate yellow dots in amount of random range

yel\_dots = funcs\_logic.generate\_yel\_dots(amounts.pop())

else:

if len(all\_coords) == 225:

print("Created:")

yel\_dots = funcs\_logic.take\_yels(all\_coords)

print(yel\_dots)

else:

print("Random:")

# generate yellow dots in amount of random range

yel\_dots = funcs\_logic.generate\_yel\_dots(amounts.pop())

# solution to any task

funcs\_logic.print\_answer(yel\_dots)

# depending on the placement of yellow dots count maximum amounts for boxes

bl\_horiz, yel\_horiz = funcs\_logic.find\_max\_hor(yel\_dots)

bl\_vert, yel\_vert = funcs\_logic.find\_max\_vert(yel\_dots)

# create vertical and horizontal lines of numbers

funcs\_interface.make\_num\_horiz\_board(bl\_horiz, yel\_horiz)

funcs\_interface.make\_num\_vert\_board(bl\_vert, yel\_vert)

# create field with buttons

all\_coords.clear()

funcs\_interface.create\_button\_field(all\_coords)

# work with an array all\_coords

# create the right answer

right\_answer = funcs\_logic.right\_answer(yel\_dots)

# button to solve task

SolveButton(all\_coords, right\_answer, main\_window)

# button to check user input

CheckButton(all\_coords, bl\_horiz, yel\_horiz, bl\_vert, yel\_vert, main\_window)

# button to show right answer

AnswerButton(yel\_dots)

if replay and replay.pop() == 'n':

main\_window.destroy()

# if the answer is correct right\_answer window will appear

# otherwise fail\_window

main\_window.mainloop()

# ANSWERBUTTON.PY

from tkinter import messagebox

from CheckButton import CheckButton

class AnswerButton(CheckButton):

def \_\_init\_\_(self, yel\_coord = []):

self.yel\_coord = yel\_coord

super().\_\_init\_\_()

self['text'] = "Answer"

self['command'] = lambda: AnswerButton.show\_answer(self)

self.place(x=310, y=530)

def show\_answer(self):

text\_answ = ""

text\_answ += "Answer: \n"

text\_answ += "(column: row1, row2,...)\n"

text\_answ += " Yellow boxes:\n"

for i in range(15):

text\_answ += str(i + 1) + ": "

for j in range(15):

if (i, j, 'y') in self.yel\_coord and j != 15:

text\_answ += str(j + 1) + " "

text\_answ += '\n'

text\_answ += " Back boxes:\n"

for i in range(15):

text\_answ += str(i + 1) + ": "

for j in range(15):

if (i, j, 'y') not in self.yel\_coord and j != 15:

text\_answ += str(j + 1) + " "

text\_answ += '\n'

messagebox.showinfo("Answer", text\_answ)

# BUTTONTOPLAY.PY

from tkinter import Button

import funcs\_buttons

class ButtonToPlay(Button):

def \_\_init\_\_(self, i = 0, j = 0, all\_coords = []):

Button.\_\_init\_\_(self)

self['bg'] = 'white'

self['text'] = " "

self['width'] = 2

self['height'] = 1

self.bind('<Button-1>',

lambda event, but=self, i=i, j=j: ButtonToPlay.pressed\_once(event, but, i, j, all\_coords))

self.bind('<Double-1>',

lambda event, but=self, i=i, j=j: ButtonToPlay.pressed\_twice(event, but, i, j, all\_coords))

def pressed\_once(event, but, i, j, all\_coords):

but["bg"] = "black"

all\_coords.append((i, j, "b"))

while (i, j, "y") in all\_coords:

all\_coords.remove((i, j, "y"))

def pressed\_twice(event, but, i, j, all\_coords):

but["bg"] = "yellow"

all\_coords.append((i, j, "y"))

while (i, j, "b") in all\_coords:

all\_coords.remove((i, j, "b"))

# CHECKBUTTON.PY

from tkinter import Button

import funcs\_buttons

class CheckButton(Button):

def \_\_init\_\_(self, all\_coords=[], bl\_vert=[], yel\_vert=[], bl\_horiz=[], yel\_horiz=[], window=None):

Button.\_\_init\_\_(self)

self['bg'] = 'white'

self['text'] = "Check!"

self['width'] = 10

self['height'] = 2

# function that will check user input after user presses the button

self['command'] = lambda: funcs\_buttons.check\_us\_input(all\_coords, bl\_vert, yel\_vert, bl\_horiz, yel\_horiz, window)

self.place(x=210, y=530)

# COLORAMOUNTS.PY

from tkinter import Label

class ColorAmounts(Label):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Label.\_\_init\_\_(self, \*args, \*\*kwargs)

self['bg'] = 'white'

self['width'] = 2

self['height'] = 1

# CREATETASK.PY

from tkinter import Button

from WindowMaker import WindowMaker

import funcs\_interface

import regular\_game

class CreateTask(Button):

def \_\_init\_\_(self, old\_window=None, bg\_color=None, all\_coord=[], create=[], main\_window=None):

self.old\_window = old\_window

self.bg\_color = bg\_color

self.all\_coord = all\_coord

self.create = create

self.main\_window = main\_window

Button.\_\_init\_\_(self)

self['bg'] = 'white'

self['text'] = "Create Task"

self['width'] = 10

self['height'] = 2

self['command'] = lambda: CreateTask.create\_field(self)

self.place(x=310, y=600)

# method to create a new window to collect user input to create personal task

def create\_field(self):

self.all\_coord.clear()

# clear all previous input

self.old\_window.destroy()

# kill the main window

new\_window = WindowMaker("Create task", self.bg\_color)

# create new window

new\_window.make\_the\_window()

funcs\_interface.make\_two\_color\_boxes()

funcs\_interface.create\_button\_field(self.all\_coord)

new\_window.make\_label()

# add '1' to array to show that the solution was created by user

self.create.append(1)

# make button to check and finish input

DoneTask(new\_window, self.main\_window, self.all\_coord, self.create)

class DoneTask(Button):

def \_\_init\_\_(self, old\_window, main\_window, all\_coord, create):

self.old\_window = old\_window

self.main\_window = main\_window

self.all\_coord = all\_coord

self.create = create

Button.\_\_init\_\_(self)

self['bg'] = 'white'

self['text'] = "Done"

self['width'] = 10

self['height'] = 2

self['command'] = lambda: DoneTask.make\_new\_task(self)

self.place(x=310, y=600)

# check and create personal task

def make\_new\_task(self):

# delete duplicates

self.all\_coord = list(set(self.all\_coord))

self.all\_coord.sort()

print("Input coords: ", self.all\_coord)

# if the user colours all buttons - create new game from input

if len(self.all\_coord) == 15 \* 15:

self.create.append(1)

self.old\_window.destroy()

regular\_game.regular\_game(self.create, self.main\_window.replay, self.all\_coord, amounts=[])

# else show the error message

else:

funcs\_interface.wrong\_create\_message(self, "You did not fill the field.")

# EXITBUTTON.PY

from tkinter import Button, messagebox

class ExitButton(Button):

def \_\_init\_\_(self, window=None):

self.window = window

Button.\_\_init\_\_(self)

self['bg'] = 'red'

self['text'] = "X"

self['width'] = 5

self['height'] = 2

self['command'] = lambda: ExitButton.exit\_message(self)

self.place(x=610, y=20)

# message that appears after you press 'Exit'

def exit\_message(self):

msgBox = messagebox.askquestion("Exit", "Do you want to exit the game?")

if msgBox == 'yes':

# exit the game

print("Exit")

self.window.destroy()

self.window.replay.append('n')

self.window.replay.append('n')

else:

# close message

print("No exit")

# FUNCS\_BUTTONS.PY

import funcs\_interface

# method to check user input

def check\_us\_input(all\_coords, bl\_vert, yel\_vert, bl\_horiz, yel\_horiz, window):

# delete duplicates

all\_coords = list(set(all\_coords))

# sort an array for easier use

all\_coords.sort()

print("User input:", all\_coords)

check\_if\_right = []

# check if all boxes are colored

check\_all\_colored(all\_coords, check\_if\_right)

check\_verticals(all\_coords, bl\_vert, yel\_vert, check\_if\_right)

check\_horizontals(all\_coords, bl\_horiz, yel\_horiz, check\_if\_right)

if not check\_if\_right:

funcs\_interface.result\_message("You Won!", window)

else:

funcs\_interface.result\_message("You Lost!", window)

# method check if all boxes are coloured

def check\_all\_colored(all\_coords, check\_if\_right):

user\_coord\_amount = len(all\_coords)

needed\_coord\_amount = 15 \* 15

# start all checks

# if user doesn't fill the field

if user\_coord\_amount != needed\_coord\_amount:

print("error")

print("in filling the field")

check\_if\_right.append(1)

# method to check if all vertical lines are correctly filled

def check\_verticals(all\_coords, bl\_vert, yel\_vert, check\_if\_right):

# Black verticals

for j in range(15):

list\_of\_lengths = []

length = 0

for i in range(15):

if (j, i, 'b') in all\_coords:

length += 1

if (j, i + 1, 'b') not in all\_coords:

list\_of\_lengths.append(length)

i += 1

length = 0

if not list\_of\_lengths:

list\_of\_lengths.append(0)

if bl\_vert[j] not in list\_of\_lengths or all(x > bl\_vert[j] for x in list\_of\_lengths):

print("error")

print("in black verticals")

print(list\_of\_lengths)

print("Supposed max: ", bl\_vert[j])

check\_if\_right.append(1)

return 1

# Yellow verticals

for j in range(15):

list\_of\_lengths = []

length = 0

for i in range(15):

if (j, i, 'y') in all\_coords:

length += 1

if (j, i + 1, 'y') not in all\_coords:

list\_of\_lengths.append(length)

i += 1

length = 0

if not list\_of\_lengths:

list\_of\_lengths.append(0)

if yel\_vert[j] not in list\_of\_lengths or all(x > yel\_vert[j] for x in list\_of\_lengths):

print("error")

print("in yellow verticals")

print(list\_of\_lengths)

print("Supposed max: ", yel\_vert[j])

check\_if\_right.append(1)

return 1

# method to check if all horizontal lines are correctly filled

def check\_horizontals(all\_coords, bl\_horiz, yel\_horiz, check\_if\_right):

# Blacks horizontals

for j in range(15):

list\_of\_lengths = []

length = 0

for i in range(15):

if (i, j, 'b') in all\_coords:

length += 1

if (i + 1, j, 'b') not in all\_coords:

list\_of\_lengths.append(length)

i += 1

length = 0

if not list\_of\_lengths:

list\_of\_lengths.append(0)

if bl\_horiz[j] not in list\_of\_lengths or all(x > bl\_horiz[j] for x in list\_of\_lengths):

print("error")

print("in black horizontals")

print(list\_of\_lengths)

print("Supposed max: ", bl\_horiz[j])

check\_if\_right.append(1)

return 1

# Yellow horizontals

for j in range(15):

list\_of\_lengths = []

length = 0

for i in range(15):

if (i, j, 'y') in all\_coords:

length += 1

if (i + 1, j, 'y') not in all\_coords:

list\_of\_lengths.append(length)

i += 1

length = 0

if not list\_of\_lengths:

list\_of\_lengths.append(0)

if yel\_horiz[j] not in list\_of\_lengths or all(x > yel\_horiz[j] for x in list\_of\_lengths):

print("error")

print("in yellow horizontals")

print(list\_of\_lengths)

print("Supposed max: ", yel\_horiz[j])

check\_if\_right.append(1)

return 1

# FUNCS\_INTERFACE.PY

from tkinter import Label, messagebox

from ColorAmounts import ColorAmounts

from ButtonToPlay import ButtonToPlay

import regular\_game

from CreateTask import CreateTask

# create the main window

def make\_window\_label\_board(main\_window):

main\_window.make\_the\_window()

main\_window.make\_label()

# create 2 boxes in up-left corner and down-right

def make\_two\_color\_boxes():

# black box

bl = Label(text=' ', bg='black', width=2, height=1)

bl.place(x=140, y=90)

# yellow box

yel = Label(text=' ', bg='yellow', width=2, height=1)

yel.place(x=530, y=493)

# create 2 vertical boards of numbers ( black and yellow )

def make\_num\_vert\_board(bl\_vert, yel\_vert):

for i in range(15):

bl\_vert\_num = ColorAmounts(text=bl\_vert[i])

bl\_vert\_num.place(x=140, y=117 + i \* 25)

for i in range(15):

yel\_vert\_num = ColorAmounts(text=yel\_vert[i])

yel\_vert\_num.place(x=530, y=117 + i \* 25)

# create 2 horizontal boards of numbers ( black and yellow )

def make\_num\_horiz\_board(bl\_horiz, yel\_horiz):

for i in range(15):

bl\_horiz\_num = ColorAmounts(text=bl\_horiz[i])

bl\_horiz\_num.place(x=167 + i \* 24, y=90)

for i in range(15):

yel\_horiz\_num = ColorAmounts(text=yel\_horiz[i])

yel\_horiz\_num.place(x=167 + i \* 24, y=493)

# create field 15 x 15 with buttons

def create\_button\_field(all\_coords):

for i in range(15):

for j in range(15):

ButtonToPlay(i, j, all\_coords).place(x=165 + i \* 24, y=115 + j \* 25)

# messagebox that appears after user pressed on 'Check'

def result\_message(mess, window):

msgBox = messagebox.askquestion("Result", mess + " Try again?")

if msgBox == 'yes':

print("Replaying the game")

window.replay.clear()

window.replay.append('y')

window.destroy()

else:

print("No replay")

window.replay.clear()

window.replay.append('n')

window.destroy()

# message after 'Done' button is pressed

def wrong\_create\_message(self, mess):

msgBox = messagebox.askquestion("Result", mess + " Do you want to retry?", icon='error')

if msgBox == 'yes':

print("Trying again")

self.all\_coord.clear()

b = CreateTask(self.old\_window, bg\_color=self.main\_window.bg\_color, all\_coord=[], create=[],

main\_window=self.main\_window)

b.create\_field()

# give person a chance to change the input

else:

print("Turning back to main window")

self.old\_window.destroy()

self.main\_window.replay.append('y')

regular\_game.regular\_game(self.create, self.main\_window.replay, self.all\_coord, amounts=[]))

# FUNCS\_LOGIC.PY

from random import randint

# generate random task

def generate\_yel\_dots(amount):

yel\_dots = []

for s in range(amount):

i = randint(0, 15)

j = randint(0, 15)

yel\_dots.append((i, j, 'y'))

for i in range(15):

yel\_dots.append((i, 15, 'y'))

yel\_dots.append((15, i, 'y'))

yel\_dots = list(set(yel\_dots))

yel\_dots.sort()

return yel\_dots

# create full right answer from random yellow dots

def right\_answer(yel\_dots):

answer = yel\_dots.copy()

for i in range(15):

for j in range(15):

if (i, j, 'y') not in yel\_dots and j != 15:

answer.append((i, j, "b"))

answer.sort()

return answer

# print answer in console

def print\_answer(yel\_dots):

print("Answer(yellow dots): ")

for i in range(15):

row = []

for j in range(15):

if (i, j, 'y') in yel\_dots and j != 15:

row.append(j + 1)

if j == 14:

print(i + 1, " : ", row)

# count maximum amounts in horizontal lines

def find\_max\_hor(yel\_dots):

# Black verticals

amounts\_bl\_vert = []

for i in range(15):

result = 0

max\_bl = 0

for j in range(15):

if (i, j, 'y') not in yel\_dots:

max\_bl += 1

if (i, j + 1, 'y') in yel\_dots:

if max\_bl >= result:

result = max\_bl

max\_bl = 0

amounts\_bl\_vert.append(result)

# Yellow verticals

amounts\_yel\_ver = []

for i in range(15):

max\_yel = 0

result\_yel = 0

for j in range(15):

if (i, j, 'y') in yel\_dots:

max\_yel += 1

if (i, j + 1, 'y') not in yel\_dots or j + 1 == 15:

if max\_yel >= result\_yel:

result\_yel = max\_yel

max\_yel = 0

amounts\_yel\_ver.append(result\_yel)

return amounts\_bl\_vert, amounts\_yel\_ver

# count maximum amounts in vertical lines

def find\_max\_vert(yel\_dots):

# Black horizontals

amounts\_bl\_horiz = []

for j in range(15):

result = 0

max\_bl = 0

for i in range(15):

if (i, j, 'y') not in yel\_dots:

max\_bl += 1

if (i + 1, j, 'y') in yel\_dots:

if max\_bl >= result:

result = max\_bl

max\_bl = 0

amounts\_bl\_horiz.append(result)

# Yellow verticals

amounts\_yel\_horiz = []

for j in range(15):

max\_yel = 0

result\_yel = 0

for i in range(15):

if (i, j, 'y') in yel\_dots:

max\_yel += 1

if (i + 1, j, 'y') not in yel\_dots or i + 1 == 15:

if max\_yel >= result\_yel:

result\_yel = max\_yel

max\_yel = 0

amounts\_yel\_horiz.append(result\_yel)

return amounts\_bl\_horiz, amounts\_yel\_horiz

# take only yellow coordinates from user's input

def take\_yels(all\_coords):

all\_coords.sort()

yel\_dots = []

for i in range(16):

for j in range(16):

if (i, j, 'y') in all\_coords:

yel\_dots.append((i, j, 'y'))

if j == 15 or i == 15:

yel\_dots.append((i, j, 'y'))

return yel\_dots

# RULESBUTTON.PY

from tkinter import Button, INSERT

import tkinter as tk

from WindowMaker import WindowMaker

class RulesButton(Button):

def \_\_init\_\_(self, game\_rules, bg\_color):

self.bg\_color = bg\_color

self.game\_rules = game\_rules

Button.\_\_init\_\_(self)

self['bg'] = 'yellow'

self['text'] = "Rules"

self['width'] = 5

self['height'] = 2

self['command'] = lambda: RulesButton.make\_new\_window(self)

self.place(x=50, y=20)

# method to create text with rules

def make\_rules(self, new\_window):

rules = tk.Text(new\_window, bg='#eaf205', bd=1, font=('Calibri Light', 14, 'bold'), padx=10, pady=10, width=60,

height=25)

rules.insert(INSERT, self.game\_rules)

rules.place(x=10, y=50)

# method to create new window for the rules

def make\_new\_window(self):

new\_window = WindowMaker("Rules", self.bg\_color)

new\_window.make\_the\_window()

RulesButton.make\_rules(self, new\_window)

# SOLVEBUTTON.PY

from ButtonToPlay import ButtonToPlay

import funcs\_interface

from CheckButton import CheckButton

class SolveButton(CheckButton):

def \_\_init\_\_(self, all\_coords, right\_coords, window):

self.all\_coords = all\_coords

self.right\_coords = right\_coords

self.window = window

super().\_\_init\_\_()

self['text'] = "Solve"

self['command'] = lambda: SolveButton.create\_right\_button\_field(self)

self.place(x=410, y=530)

# create solution

def create\_right\_button\_field(self):

self.all\_coords.clear()

for i in range(15):

for j in range(15):

b = ButtonToPlay(i, j, self.all\_coords)

b.place(x=165 + i \* 24, y=115 + j \* 25)

if (i, j, 'b') in self.right\_coords:

b.config(bg="black")

self.all\_coords.append((i, j, 'b'))

else:

b.config(bg="yellow")

self.all\_coords.append((i, j, 'y'))

# WINDOWMAKER.PY

from tkinter import Tk

import tkinter as tk

class WindowMaker(Tk):

def \_\_init\_\_(self, game\_name=" ", bg\_color=" ", replay=[]):

super().\_\_init\_\_()

self.game\_title = game\_name

self.bg\_color = bg\_color

self.replay = replay

# method to create the window

def make\_the\_window(self):

# configure the root window

self.title(self.game\_title)

self.geometry('700x700')

# forbid to change window size

self.resizable(0, 0)

self.config(bg=self.bg\_color)

# method to create title

def make\_label(self):

# text "Lerogram" on top of the game

tk.Label(

text=self.game\_title,

foreground="white",

bg=self.bg\_color,

font=("Calibri Light", 48)

).place(

relx=0.5,

rely=0.05,

anchor='center'

)

# LevelButton.py

import tkinter as tk

import funcs\_logic

import regular\_game

from CheckButton import CheckButton

class LevelButton(CheckButton):

def \_\_init\_\_(self, level\_name=None, amount\_dots=0, amounts=[], plus\_pos=0, window=None):

self.level\_name = level\_name

self.plus\_pos = plus\_pos

self.amount\_dots = amount\_dots

self.amounts = amounts

self.window = window

super().\_\_init\_\_(self)

self['text'] = level\_name

# function that will check user input after user presses the button

self['command'] = lambda: LevelButton.pressed(self)

self.place(x=580, y=180 + self.plus\_pos)

def pressed(self):

funcs\_logic.generate\_yel\_dots(self.amount\_dots)

self.amounts.append(self.amount\_dots)

self.window.destroy()

print("Level: ", self.level\_name)

regular\_game.regular\_game(create=[], replay=['n'], all\_coords=[], amounts=self.amounts)

def create\_level\_title(self):

# text "Level" on right of the game

tk.Label(

text="Level",

foreground="white",

bg=self.window.bg\_color,

font=("Calibri Light", 25)

).place(

x=615,

y=150,

anchor='center'

)