МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

автоматизованих систем обробки інформації та управління

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: Розв’язання систем лінійних рівнянь

Студента 1 курсу, групи ІП-81

Касьяненка М.В.

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник: ас. Головченко М.М

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

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

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

Київ- 2019 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

## Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Основи програмування

Напрям "Програмна інженерія"

Курс 1 Група ІП-81 Семестр 2

### **ЗАВДАННЯ**

#### на курсову роботу студента

##### Касьяненка Максима Володимировича

(прізвище, ім’я, по батькові)

1. Тема роботи Розв’язання систем лінійних алгебраїчних рівнянь.

Розробити програму, яка дозволяє розв’язувати системи лінійних алгебраїчних рівнянь

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

КАЛЕНДАРНИЙ ПЛАН

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

# Студент

(підпис)

Керівник Муха І. П.

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

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

# Анотація

Пояснювальна записка до курсової роботи: 56 сторінок, 12 рисунків, 8 таблиць, 5 посилань.

Об’єкт дослідження: ефективні методи розв’язання систем лінійних рівнянь.

Мета роботи: дослідження методів розв’язання систем лінійних рівнянь, а саме методу Гаусса, методу Жордана-Гаусса, Методу Обертання та створення програмного забезпечення для реалізації цих методів.

Дана курсова робота включає в себе: опис методів, застосування методу до конкретного завдання, код програми вирішення перерахованих вище методів на мові програмування С++, а також описання детального процесу розв’язання кожного з них.

Вступ

Життя будь-якої людини завжди неупорядковане та непередбачуване. Через це, люди завжди намагаються спростити своє існування. Одним із способів досягнення цієї мети є винайдення корисних для неї пристроїв. У наш час велику роль відіграють електронні обчислювальні машини, оскільки вони здатні дуже зручно спростити життя людини.

Електронні обчислювальні машини керують багатьма сферами життя людини. До того-ж люди постійнно пряцюють над їх вдосконаленням. Таким чином іх можливості постійно зростають. Постійно з’являються нові машини, деталі так само як і нові алгоритми. Нові алгоритми створюють з метою використовувати комп’ютерні ресурси більш ефективно. Також завдякиновим алгоритмам все більше видів задач можна виконати за допомогою комп’ютера.

Одним зі способів використання комп’ютера є розв’язок математичних задач, а саме розв’язок систем лінійних алгебраїчних рівнянь. Така можливість є дуже зручною, оскільки налагоджений алгоритм не допускає помилок при розв’язанні, що е властивим людям.

Саме тому у даній курсовій роботі мною було гозглянуто і проаналізовано алгоритми розв’язання систем лінійних рівнянь. Окрім цього було створено програму з графічним інтерфейсом, яка дозволяє розв’язіват системи, знаходити їх корені.

1. Постановка задачі

* Розробити прграму, завдякии якій користувач може отримувати розв’язок системи лінійних алгебраїчних рівнянь, введеної в прграму у текстовому вигляді.
* Реалізувати можливість вибору алгоритму за яким систем буде розв’язяна.
* Реалізувати два текстових поля, одне з яких необхідне для введеня користувачем системи рівнянь, а інше- для виведення на екран резуль тату роботи програми.
* Реалізувати можливість зберегти розв’язок системи в файл.
* Реалізувати можливість користувачу самостійно вводити ім’я файлу у який буде виконано збереження результатів роботи програми.
* Розробити три алгоритми для розв’вязку системи лінійних рівнянь.

1. Аналіз предметної області

2.1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Система лінійних алгебраїчних рівнянь (СЛАР) в [лінійній алгебрі](https://uk.wikipedia.org/wiki/%D0%9B%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0) має вигляд:

|  |  |
| --- | --- |
| C:\Users\123User\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\85BD49F1.tmp | (2.1) |

де , i =1,2,…,m; j =1,2,…,n – відомі коефіцієнти; , i =1,2…,m – праві частини (чи вільні члени) також відомі; – невідомі, які слід визначити. Розв’язком системи (2.1) називається набір чисел, які при підстановці в кожне рівняння перетворюють його на тотожність. Система називається сумісною, якщо вона має хоча б один розв’язок. Система називається несумісною, якщо вона розв’язків не має.

**Методи розв’язування системи лінійних рівнянь.**

**Метод Гауса**

Запишемо систему Ax=f, в розгорнутому вигляді

|  |  |
| --- | --- |
|  | (2.2) |

Метод Гауса полягає в послідовному виключенні невідомих з даної системи [1]. Припустимо, що . Якщо це не так, то переставимо на перше місце рівняння з першим ненульовим доданком. Послідовно множачи перше рівняння на та додаючи його до i-го рівняння, виключимо з усіх рівнянь окрім першого. Отримаємо систему:

|  |  |
| --- | --- |
|  | (2.3) |

Аналогічним чином із отриманої системи виключимо . Послідовно вилючаючи усі невідомі, отримаємо систему трикутного вигляду

|  |  |
| --- | --- |
|  | (2.4) |

Описана процедура називається прямим ходом методу Гауса. Зауважимо, що її виконання можливе за умови, що усі не дорівнюють нулю.

Виконуючи послідовні підстановки у останній системі, починаючи із останнього рівняння, можна отримати усі значення невідомих

|  |  |
| --- | --- |
|  | (2.5) |

Ця процедура отримала назву зворотній хід методу Гауса.

Алгоритм методу Гауса можна легко реалізувати на комп’ютері. При виконанні обчислень, як правило, не цікавлять проміжні значення матриці А. Тому чисельна реалізація методу зводиться до перетворення елементів матриці (m×(m+1)), де m+1 стовпчик, який містить праву частину системи.

**Метод обертання розв’язку лінійних систем.**

Як і у методі Гауса, мета прямого ходу перетворень у цьому методі – приведення системи до трикутного виду послідовно отримуючи нульові значення елементів під діагоналлю спочатку першого стовпчика, потім другого і т.д.

|  |  |
| --- | --- |
|  | (2.6) |

Помножимо перше рівняння вихідної системи (2.6) на *с1* ,друге *s1* та додаємо їх, отриманим рівнянням замінимо перше рівняння системи [2]. Потім перше рівняння системи помножимо на –*s1* , друге на *с1* і результатом їх додавання замінимо друге рівняння системи. Таким чином перші два рівняння системи (2.6) замінюються рівняннями

|  |  |
| --- | --- |
|  | (2.7) |

На величини *с1* і *s1* накладемо дві умови:

|  |  |
| --- | --- |
|  | (2.8) |
|  | (2.9) |

Звідси

|  |  |
| --- | --- |
| . | (2.10) |

Ці числа можна інтерпретувати як косинус та синус деякого кута .

У результаті перетворень отримаємо систему

|  |  |
| --- | --- |
|  | (2.11) |

де

|  |  |
| --- | --- |
|  | (2.12) |

Далі перше рівняння системи замінюється новим, отриманим додаванням результатів множення першого і третього рівнянь відповідно на

|  |  |
| --- | --- |
|  | (2.13) |

а третє – рівнянням, яке отримано додаванням результатів множення тих самих рівнянь відповідно на –*s1* і *с1*. Отримаємо систему

|  |  |
| --- | --- |
|  | (2.14) |

де

|  |  |
| --- | --- |
|  | (2.15) |

Виконавши перетворення *m-1* раз, отримаємо систему

|  |  |
| --- | --- |
|  | (2.16) |

Отримана система має вигляд такий самий, як після першого етапу перетворення методу Гауса. Ця система має наступну властивість: довжина будь-якого вектору-стовпчика розширеної матриці залишається такою самою, як у вихідної матриці. Отже, при виконанні перетворень не спостерігається збільшення елементів.

Далі по такому самому алгоритму, виконуються перетворення матриці

|  |  |
| --- | --- |
|  | (2.17) |

і т.д.

У результаті *m*-1 етапів прямого ходу система буде приведена до трикутного виду.

|  |  |
| --- | --- |
|  | (2.18) |

Невідомі знаходяться за допомогою зворотнього ходу методу Гауса.

2.2 ДІАГРАМА ПРЕЦЕДЕНТІВ

На рисунку 2.1схему взаємодії користувача з програмою.

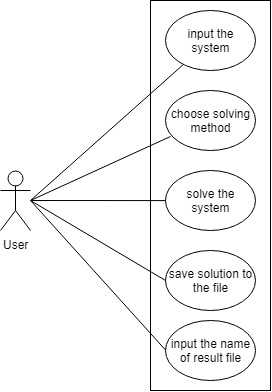


Рис 2.1 - Діаграма прецедентів.

* Актор User – це користувач програми. Він є єдиним актором у даній ситуації.
* Input the system – користувач має можливість вводити у врограму систему лінійних алгебраїчних рівнянь.
* Choose solving method – корстувач має можливість обирати метоз яким буде розв’язна введена ним система лінійних рівнянь.
* Solve the System – користувач може отримати розв’язок системи рівнянь отриманий у результаті роботи обраного ним алгоритму розв’язку системи.
* Save solution to the file – має можливість зберігати розв’язок системи у текстовий файл.
* Input the name of the file –користувач може вводити назву файлу, у який буде збережено систему.

2.3 ОПИС АЛГОРИТМІВ

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

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| The\_system | Система лінійних рівнянь |
| Matrix | Матриця системи |
| How\_many | Вказує на кількість змінних у системі |
| Variables | Список змінних присутніх у системі. |
| Plus | Вказує на знак що стоїть перед змінною |
| Cnumber | Число яке ми занесемо у матрицю |
| Row | Рядок матриці з яким ми працюємо в даний момент |
| Coma | Вказує на наявність десяткового дробу |
|  |  |

## Загальний алгоритм

1. ПОЧАТОК
2. Зчитати систему у текстовому вигляді.
3. Зберегти систему у вигляді матриці системи, та списку змінних:
   1. Зчитати матрицю системи:
      1. Цикл проходу по всіх символах введеного рядка :
         * 1. ЯКЩО данний елемент – літера, яка не зустрічалась у рядку до цього – додати її до списку змінних.
   2. Цикл проходу по всіх символах введеного рядка:
      1. ЯКЩО даний елемент рядка – операція зберегти знак коефіціенту у змінну
      2. ЯКЩО даний елемент кома зберегти це у змінній, з можливістю подальшої обробки десяткового дробу.
      3. ЯКЩО даний елемент – число, додати його до сnumber з урахуванням коми і знаку.
      4. ЯКЩО даний елемент – літера, зберегти cnumber у клітинці що відповідає даній змінній. Та обнулити змінні використані до цього.
4. ЯКЩО обраний метод Гауса, ТО обробити дані згідно алгоритму методу Гауса(пункт 3.2)
5. ЯКЩО обраний метод Жордана-Гауса, ТО обробити дані згідно алгоритму методу Жордана-Гауса (пункт 3.3)
6. ЯКЩО обраний метод Обертання, ТО обробити дані згідно алгоритму методу Обертання (пункт 3.4).
7. ЯКЩО обрано графічний метод:

7.1 ЯКЩО система має 2 змінні, ТО обробити дані згідно алгоритму Графічного методу

1. Вивести результат роботи на екран.
2. ЯКЩО вибрано збереження результату в файл, зберегти результат у файл.
3. КІНЕЦЬ

## Алгоритм методу Гауса

1. ПОЧАТОК
2. Цикл по всіх рядах матриці (*Matrix*):
   1. ЦИКЛ по всіх рядках матриці починаючи з поточного
      1. ЯКЩО початковий елементрядка не рівний нулю, поставити його на поточне місце розділити даний рядок матриці на початковий елемент, закінчити цикл. (*Matrix*)
   2. Цикл по всіх рядках матриці після відповідного:
      1. Відняти від поточного рядка даний помножений на початковий елемент поточного рядка.
3. Цикл по всіх рядках матриці починаючи з останнього.
   1. ЯКЩО рядок останній, розділити рядок на елемент arr[y][y].
      1. Занести значення даної змінної до множини розв’язків.
   2. ЯКЩО радок не останній, ТО відняти від останнього елементу рядку занчення усіх змінних помножені на коефіціени матриці.
      1. Занести значення змінної до множини розв’язку.
4. Записати результат у вигляді рядкової змінної
5. КІНЕЦЬ

**с. Метод Обертання**

1. ПОЧАТОК
2. Цикл повсіх рядках матриці:
   1. Цикл по всіх рядках матриці:
      1. Знайти значення змінних S, C, для даної пари рядків.
      2. Помножити перший рядок на S додати до другого помноженого на С, записати у перший рядок.
      3. Помножитидругий рядок на S додати відняти від першого помноженого на С, записати у другий рядок.
3. Цикл по всіх рядках матриці починаючи з останнього.
   1. ЯКЩО рядок останній, розділити рядок на елемент arr[y][y].
      1. Занести значення даної змінної до множини розв’язків.
   2. ЯКЩО радок не останній, ТО відняти від останнього елементу рядку занчення усіх змінних помножені на коефіціени матриці.
      1. Занести значення змінної до множини розв’язку.
4. Записати результат у вигляді рядкової змінної
5. КІНЕЦЬ
6. Опис архітектури програмної системи

На рисунку 3.1 зображена взаємодія між класами.

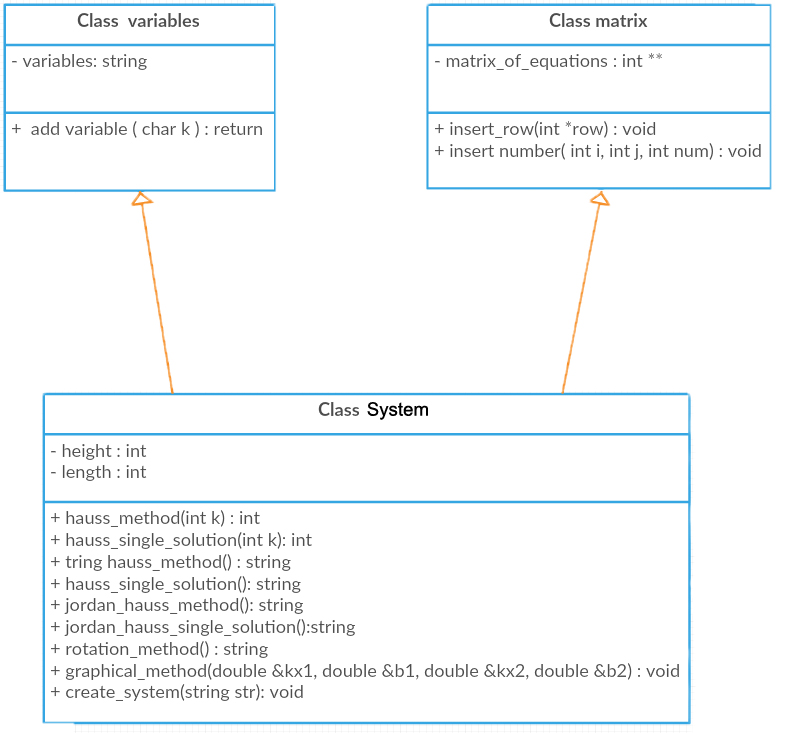


Рис 3.1 – Принцип взаємодії мі жкласами

За даною UML діаграмою видно, що клас System, що є інтерпретаціею системи лінйних алгебраїчних рівнянь, породжений від двох класів за допомогою множинного наслідування. При цьому батьківськими класами для нього є клас Variables I клас Matrix.

Variables – це клас який зберігає множину змінних що містяться у системі. Назви змінних – літери латинського алфавіту. Зберігаються вони у вигляді текстового рядка.

Matrix – це клас що зберігає матрицю системи та стовпець вільних членів. При цьому обидва вищезазначені елементи зберігаються у вигляді одного двовимірного масиву. Дана реалізація є доцільною з огляду на те, що алгоритми, які використовує дана програма виконують аналогічні дії як з матрицею системи так і з стовпцем вільних членів.

System – це власне клас, об’єкти якого зберігають у комп’ютерній пам’яті системи лінійних рівнянь. Він успадковує усі поля і методи батьківських класів, та має певну кількість своїх. Власні методи даного класу – це метод що будує систему алгебраїчних рівнянь з введеного користувачем рядка, та власне методи які перетворюють систему таким чином, щоб знайти розв’язок системи. Методи: hauss\_method, Jordan\_hauss\_method, rotation\_method – це методи, що розв’язують систему рівнянь. Вони повертають рядок, який на екран виводиться в текстове поле. Також його можливо зберегти в файл.

1. Опис програмного забезпечення

Опис специфікації функцій здійснюється у вигляді наведеному у таблиці 4.1

Таблиця 4.1 – Специфікація функцій

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | Variables | Add\_variable | Додає змінну до списку змінних | К – літера, що позначае змінну у системі рівнянь | - | System.h |
| 3 | Marix | Insert\_number | Вставляє в певну позицію чистеми певний індекс | Num – число яке необхідно встатити в систему. I номер рядка у який необхідно вставити систему. J – стовпчик в який необхідно вставити число | - | System.h |
| 2 | Matrix | Insert\_row | Додає до матриці рядок, що передається у функцію як покажчик | Row – покажчик на одновимірний цілочисельний масив, який деобхідно додати | - | System.h |
| 4 | - | Is\_letter | Перевіряє чи є символ літерою | Letter – символ який необхідно перевірти | Булеве значення правда чи ні | before\_solving.h |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 5 | - | How\_many\_wariables | Підраховує кількість змінних у рядку введеному користувачем | Raw\_system – текст введений користувачем, що має бути перетворений на систему рівнянь | Кількіть змінних у системі | before\_solving.h |
| 6 | - | How\_many\_equations | Підраховує кількість рядків у введеній системі, таким чином підраховує кількість рівнянь у системі | Raw\_system – текст введений користувачем, що має бути перетворений на систему рівнянь | Кількість рівнянь в системі | before\_solving.h |
| 7 | - | Is\_number | Перевіряє чи є даний символ числом | Letter – символ який еобхідно перевірити | Булеве значення чиє символ числом | before\_solving.h |
| 8 | Systema | Zero\_row | Перевіряє систему на наявність рядка що містить лише нулі(неврахоуючи стовпець вільних членів) | - | Булеве значення true, якщо такий рядок пристуній | system.h |
| 9 | Systema | Upper\_triangle\_matrix | Функція – перша частина методу Гауса(прямих хід методу Гауса) | - | - | system.h |
| 10 | Systema | Hauss\_method | Перевіряє чи можливе використання методу Гауса, у разі можливості, викликає функцію методу Гауса | - | Рядок з розв’язком системи | system.h |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 11 | Systema | Hauss\_method\_single\_solution | Функція, що розв’язує систему методом Гауса | - | Рядок з розв’язком системи | system.h |
| 12 | Systema | Jordan\_hauss\_method | Перевіряє чи можливе використання методу Жордана-Гауса, у разі можливості, викликає функцію методу Жордана-Гауса | - | Рядок з розв’язком системи | system.h |
| 13 | Systema | Jordan\_hauss\_single\_solution | Функція, що розв’язує систему методом Жордана-Гауса | - | Рядок з розв’язком системи | system.h |
| 14 | Systema | Rotation\_method | Розв’язує систему методом обертання | - | Рядок з розв’язком системи | system.h |
| 15 | Systema | Graphical\_method | Отримує коефіціенти рівняннь прямих для побудови їх у випадку вибору графічного методу | Kx1 – коефіцієнт при Х у першому рівнянні прямої b1- вільний член в першому рівнянні прямої kx2 – коефіцієнт при Х у другому рівнянні прямої b2 - вільний член в другому рівнянні прямої | - | system.h |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 16 | Systema | Create\_system | Будує систему лінійних рівнянь на основы тексту який введений користувачем з клавіатури. | Рядок введений користувачем з клавіатури у відповідне поле, на основі якого має бути побудовано системулінійних рівнянь | - | system.h |
| 17 | Systema | Matrix | Записує матрицю системи у рядкову змінну, для можливості подвльшого виведення її на екран | - | Рядкова змінна, що містить матрицю системи лінійних алгебраїчних рівнянь рівнянь | SelectionScreen.py |

1. Результати тестування програмного забезпечення

Тестування програмного забезпечення — це процес дослідження програми та її коду, з ціллю визначити якість продукту. Тестування включає в себе багато методів і процесів, від пошуку помилок логіки в коді програми до випробування програми непередбачуваними вхідними даними. Оцінюється:

* відповідність вимогам технічного завдання
* коректність результатів виконання обчислень
* час, витрачений на обчислення
* практичність
* сумісність з програмним забезпеченням та операційними системами

Тобто, тестування надає об'єктивну інформацію про якість програмного забезпечення і ризики його відмови.

Як правило, тестування проводиться протягом усього життєвого циклу програмного забезпечення – від початку його розробки до кінця підтримки.

Етапи тестування:

* Введення системи рівнянь не змінюючи порядок змінних у кожному рядку

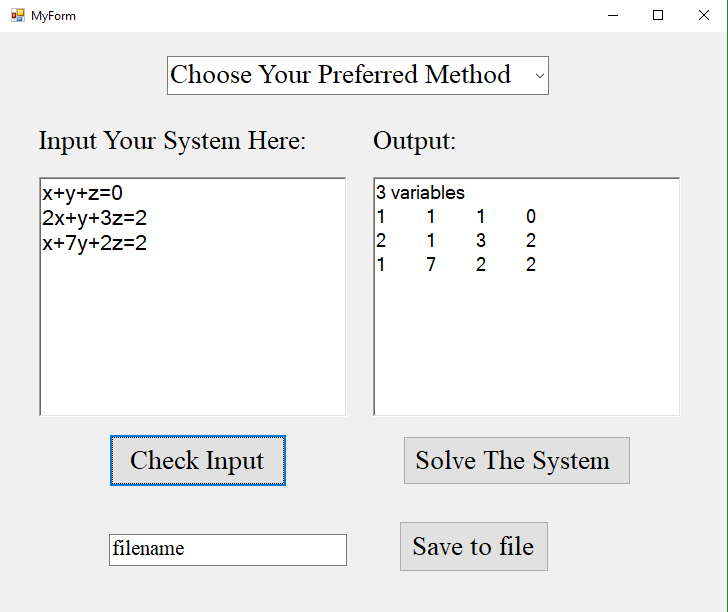


Рисунок 5.1 – Введення системи лінійних рівнянь, перевірка коректності зчитування

* Введення системи рівнянь змінюючи порядок слідування змінних

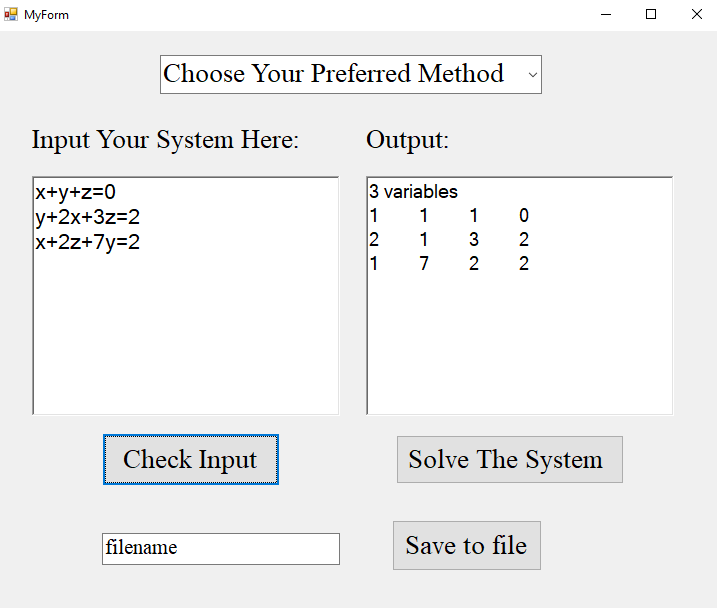


Рисунок 5.2 – Введення змінних у довільному поядку

* Перевірка опрацювання помилки введення – замало рівнянь.

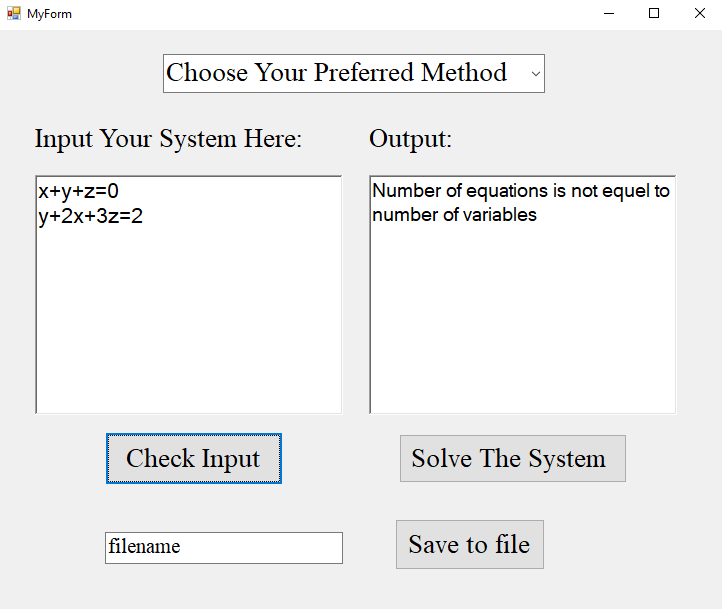


Рисунок 5.3 – повідомлення про помилку, введено недостатню кількість рівнянь

* Перевірка опрацювання помилки введення –забагато рівнянь.

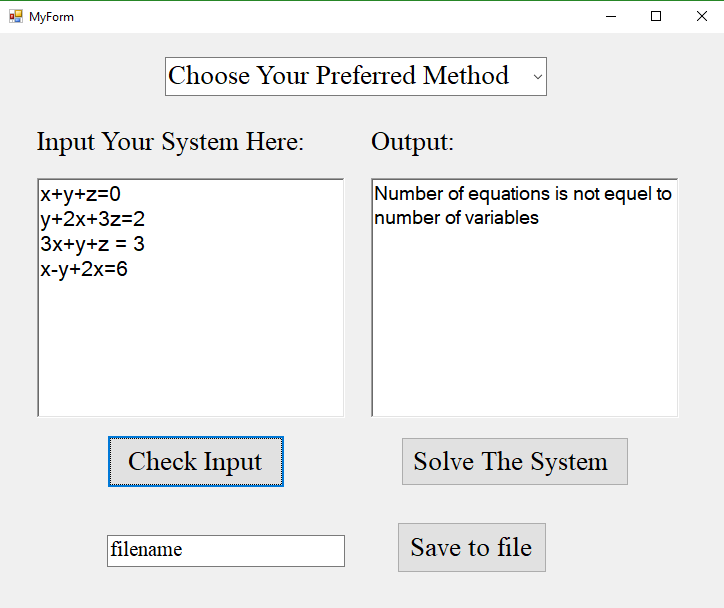


Рисунок 5.4 – повідомлення про помилку, введено завелику кількість рівнянь

* Перевірка роботи при лінійно залежній системі

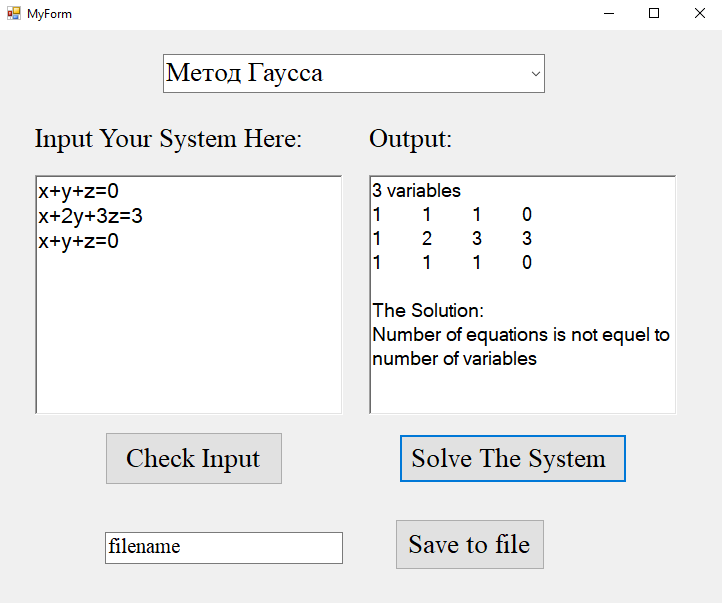


Рисунок 5.5 – повідомлення про лінійну залежність системи

* Обрання противника AI

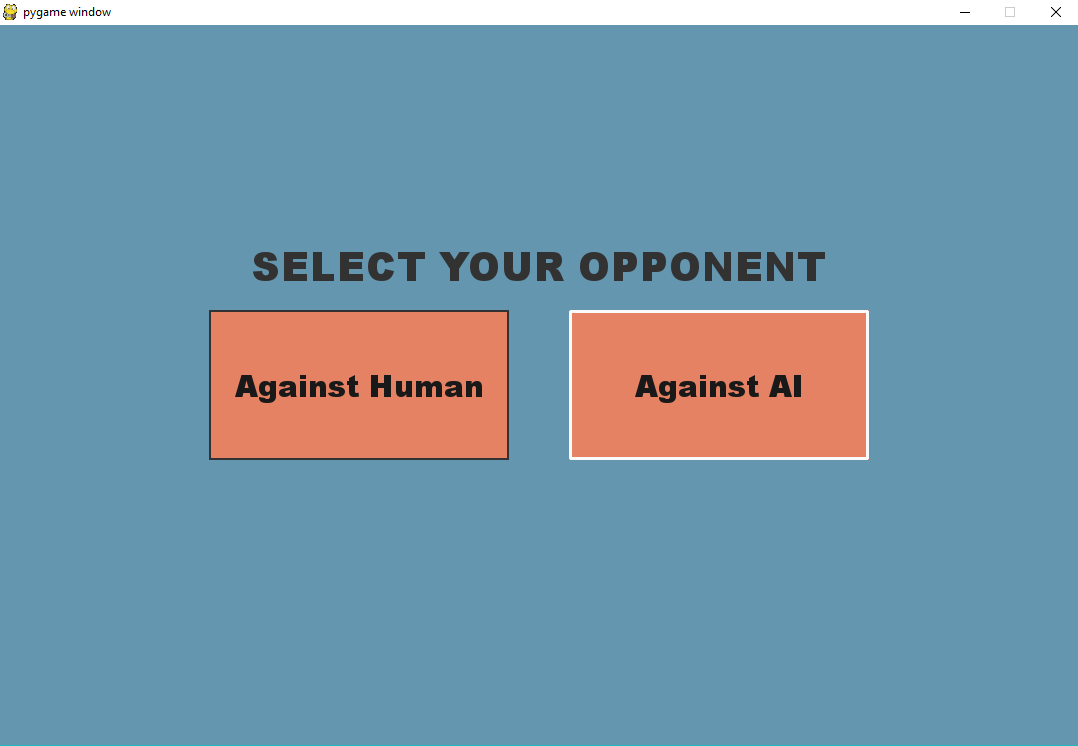


Рисунок 5.6 – обрання АІ

* Перевірка коректності роботи алгоритму на нескінченному полі

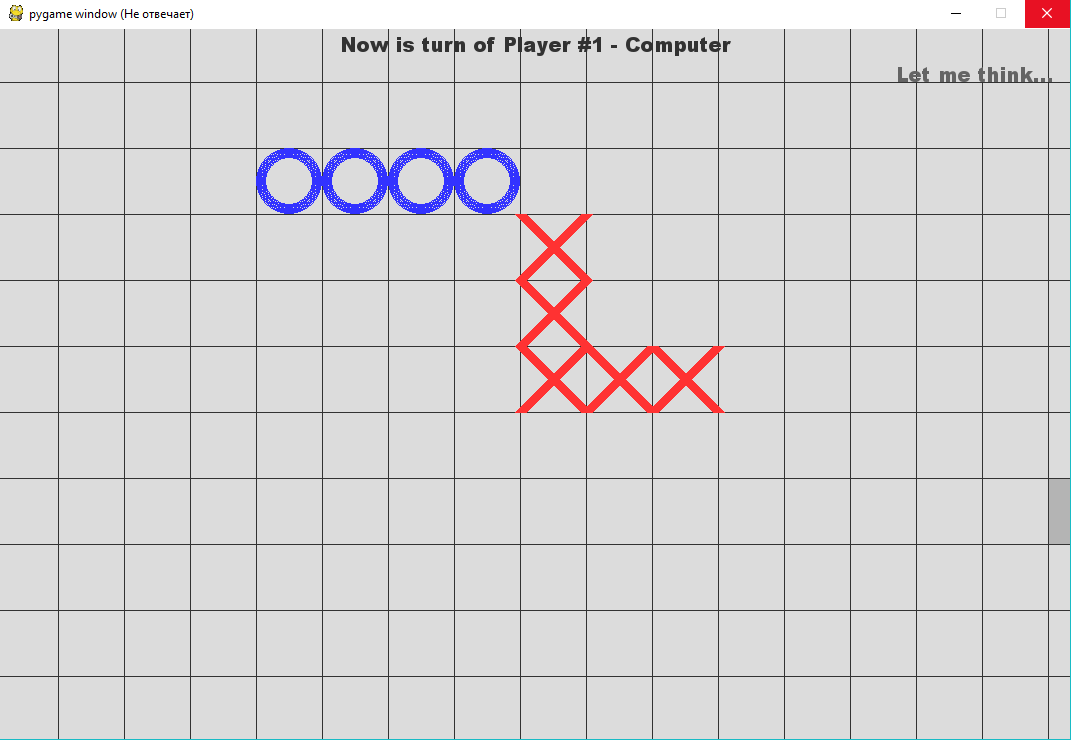


Рисунок 5.7 – коректність роботи

* Перевірка коректності обрання переможця

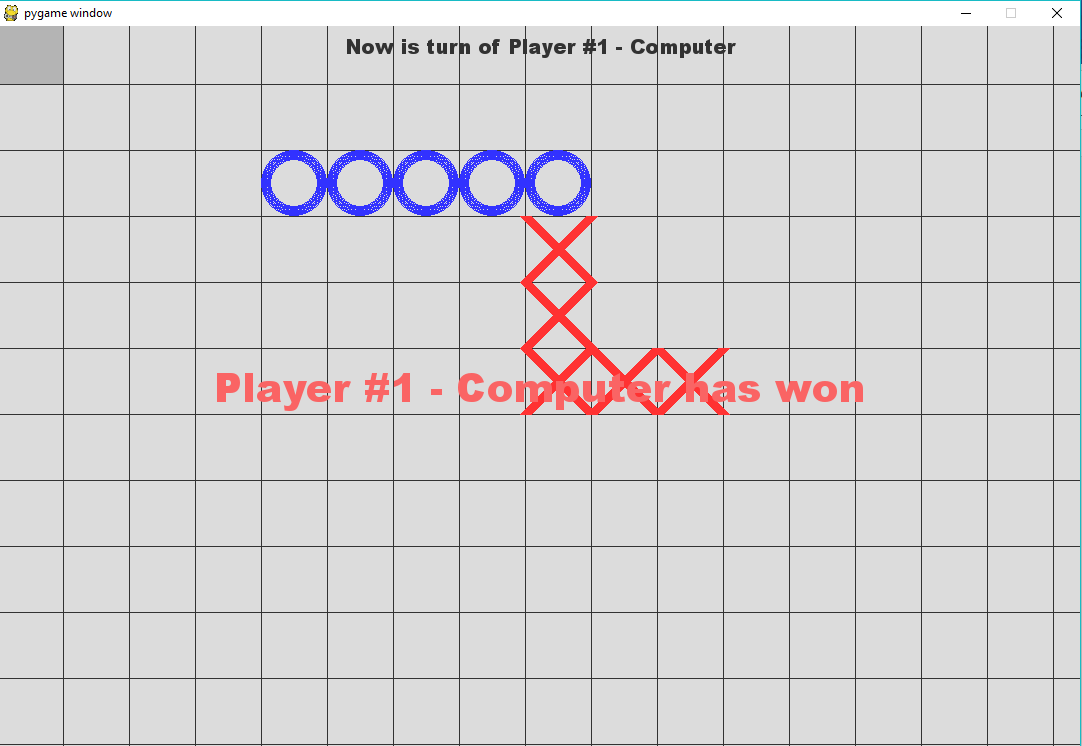


Рисунок 5.8 – обрання переможця

1. Інструкція користувача
2. Кроки виконання для успішної роботи програми:
3. Оберіть вид гри 3x3 або на нескінченній сітці

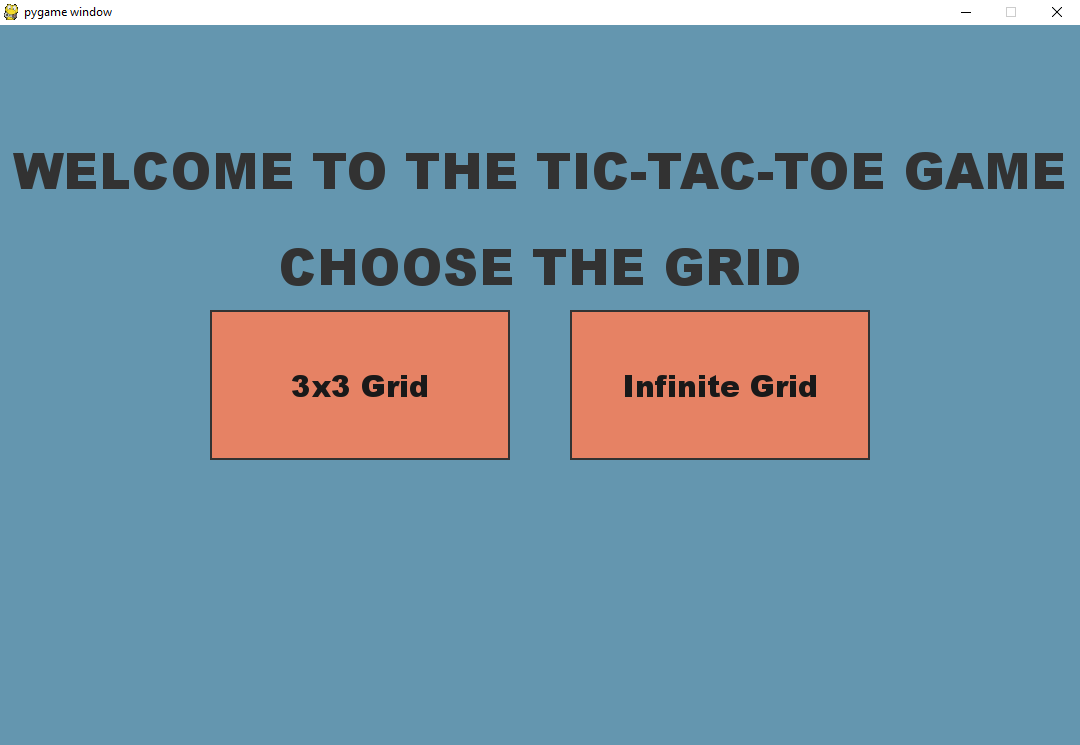


Рисунок 6.1 – обрання сітки

1. Вкажіть проти кого ви хочете грати людина або комп'ютер

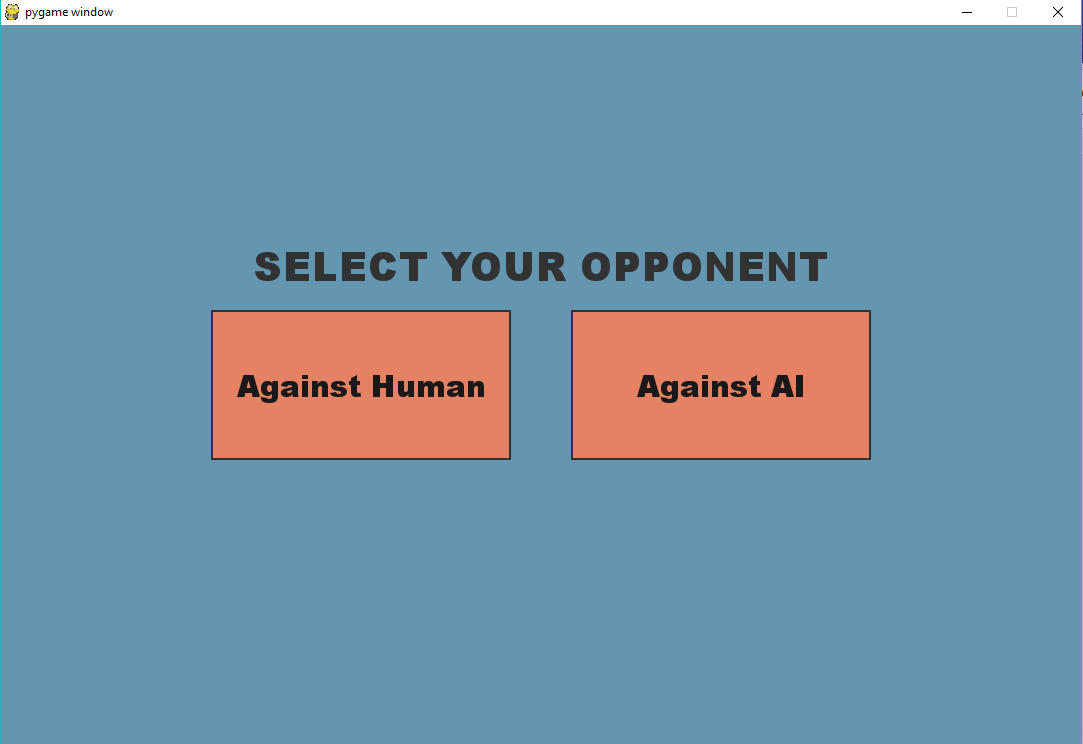


Рисунок 6.2 – обрання опонента

1. Оберіть клітинку в яку хочете поставити крестик або нулик

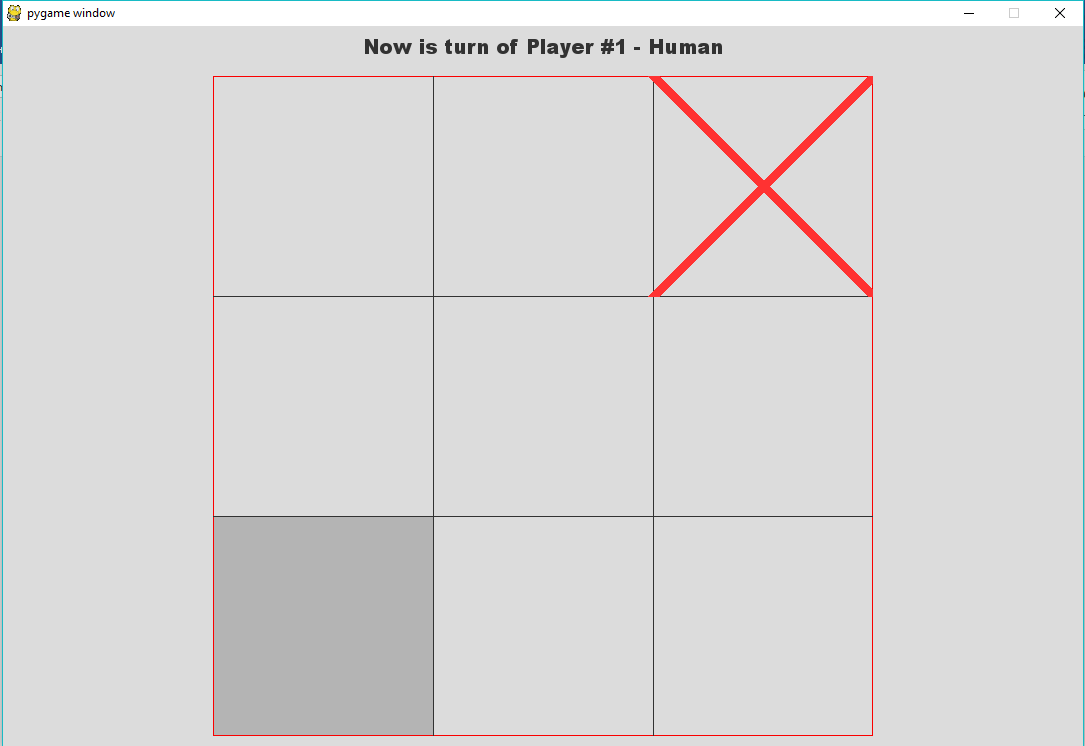


Рисунок 6.2 – вибрання клітинки для ходу

1. 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  MacOS 10.14.3 Mojave | Windows 7/Windows 8/  Windows 10  MacOS 10.14.3 Mojave |
| Процесор | Intel® Pentium® ІІІ 1.0 GHz або AMD Athlon™ 1.0 GHz | Intel® Core™ i3-3110M CPU @ 2.40 GHz або AMD Athlon™ 2.40 GHz |
| 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® HD Graphics 4000 або Radeon™ HD 8670M (або любий сумісний аналог) | |
| Дисплей | 1080х720 | 1600х900 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Python 3.4 та Pygame 1.9.2a0 або вище | |

Висновок

Під час розробки гри «Хрестики-Нулики» були досліджені нові методи побудови архітектури для гри для зручної подальшої розробки. Були використані три основні принципи ООП: інкапсуляція, поліморфізм, та успадкування. Були реалізовані інтерфейси для більш гнучкого використовування класів та поліморфізму, та було усунене дублювання методів у коді. Завдяки правильному використанню поліморфізму було досягнені більша швидкість роботи програми та зменшення часу на написання коду.

Також були розроблені та реалізовані два алгоритми штучного інтелекту. Для сітки 3х3 був розроблений штучний інтелект, який не дає перемогти гравцю ні в якому випадку. Тобто усі ігри зводяться або к нічиєї або к перемозі комп’ютера. Та для сітки 100х100 був також імплементований штучний інтелект, який аналізує попередні ходи гравця, та ставить фішку у найбільш виграшному варіанті. Штучний інтелект запобігає виграшу гравця, та намагається походити найоптимальнішим варіантом, який приведе до перемоги.

Перелік посилань

1. Albert Sweigart, Making Games with Python & Pygame, — 2012 р.
2. Albert Sweigart, Invent Your Own Computer Games with Python 3rd Edition — 2011 р.
3. Томас Кормен, Алгори́тмы: построе́ние и ана́лиз, — 2010 p.
4. Стюарт Рассел, Питер Норвиг, Искусственный интеллект: современный подход (AIMA-2), 2-е издание, — 2011 p.
5. Эндрю Роллингз, Дэйв Моррис, Проектирование и архитектура компьютерных игр, — 2016 p.
6. Алекс Дж. Шампандар, Искусственный интеллект в компьютерных играх — 2015 p.

Додаток А Технічне завдання

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

автоматизованих систем обробки інформації та управління

Затвердив

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

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

Виконавець:

Студент

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

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: Гра «Хрестики-нулики»

з дисципліни:

«Об’єктно-орієнтоване програмування»

Київ 2019

1. *Мета:* Метою курсової роботи є розробка програми, яка дозволяє грати на необмеженому полі в "хрестики-нулики".
2. *Дата початку роботи:* «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 201\_ р.
3. *Дата закінчення роботи:* «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 201\_ р.
4. *Вимоги до програмного забезпечення:*
   1. Функціональні вимоги:
      * Програма уявляє собою гру «Хрестики-нулики», де два гравця по черзі виконують хід, ставлячи фішку у відповідну клітинку на квадратному полі з деяким розміром: «Хрестик» або «Нулик»
      * Є два режими гри: З комп’ютером, та з іншим гравцем. Для кожного режиму є можливість обрати поле, на якому буде здійснюватися гра.
      * Перше поле надає наступні правила: гравець може перемогти тоді, коли набирає лінію з 5 фішек. Кількість клітинок – до 100.
      * Другий режим уявляє собою класичний режим «Хрестики-нулики» з деяким розміром поля 3х3, де кожен гравець повинен першим збудувати лінію довжиною в 3.
      * У якості другого гравця може виступати комп’ютер. Роблячи черговий хід, програма аналізує ситуацію, розраховуючи можливі ходи супротивника вперед та аналізує поточний стан поля, і в результаті проведеного аналізу ходить оптимальним чином.
      * Додаток забезпечує початок нової гри на чистому полі, перевірку відповідності дій гравців правилами гри та умови закінчення гри.
   2. Нефункціональні вимоги:
      * Код програми повинен бути реалізован на мові Python3. В якості інтегрованої середовища розробки програми повинна бути використане середовище Pygame.
      * Вимоги до складу і параметрів технічних засобів:
        1. Операційні системи: Windows® 10, macOS, and Linux
        2. PIP and NumPy: Installed with PIP, Ubuntu\*, Python 3.6.2, NumPy 1.13.1, scikit-learn 0.18.2
        3. Windows: Python 3.6.2, PIP and NumPy 1.13.1, scikit-learn 0.18.2
        4. Встановлен Pygame 1.9.2 та версії Python3.2 (або вище)
      * 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.
5. *Стадії та етапи розробки:*
   1. Об’єктно-орієнтовний аналіз предметної області задачі (до \_\_.\_\_.201\_ р.)
   2. Об’єктно-орієнтовне проектування архітектури програмної системи (до \_\_.\_\_.201\_ р.)
   3. Розробка програмного забезпечення (до \_\_.\_\_.201\_ р.)
   4. Тестування розробленої програми (до \_\_.\_\_.201\_ р.)
   5. Розробка пояснювальної записки (до \_\_.\_\_.201\_ р.).
   6. Захист курсової роботи (до \_\_.\_\_.201\_ р.).
6. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв її оцінювання.

Додаток Б Тексти програмного коду

*студентів групи ІП-81 І курсу*

*Чернишова Д.Ю*

*Петроченко Д.С.*

(Найменування програми (документа))

*Тексти програмного коду для гри «Хрестики-Нулики»*

(Вид носія даних)

*CD-RW*

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

*22 арк, 126,6 Кб*

from abc import ABCMeta, abstractmethod

class Player:

def \_\_init\_\_(self, grid, id):

self.grid = grid

self.id = id

@abstractmethod

def on\_turn(self):

""" Methods that is called on each turn -> bool"""

pass

@abstractmethod

def make\_move(self):

""" Methods that is called to make a move on a grid """

pass

import pygame

import pygame.gfxdraw

from utils import \*

class Button(pygame.sprite.Sprite):

def \_\_init\_\_(self, position, size, color, text, text\_color=color.but\_text, text\_size=30, on\_click=None):

self.position = self.x, self.y = position

self.size = self.width, self.height = size

self.color = color

self.text = text

self.text\_color = text\_color

self.text\_size = text\_size

self.image = pygame.Surface(size)

self.rect = self.image.get\_rect()

self.rect.center = position

self.on\_click = on\_click

def update(self, mouse):

self.draw\_borders(mouse)

if self.hover(mouse):

self.check\_pressing(mouse)

def render(self, surface):

surface.blit(self.image, (self.x-self.width//2, self.y-self.height//2))

self.image.fill(color.button)

Utils.toast(self.image, self.text, self.text\_size, self.text\_color,

self.width/2, self.height/2)

def check\_pressing(self, mouse):

if mouse.pressed:

self.on\_click()

def draw\_borders(self, mouse):

if self.hover(mouse):

pygame.draw.rect(self.image, color.but\_hovered,

[1.5, 1.5, self.width-1.5, self.height-1.5], 3)

else:

pygame.draw.rect(self.image, color.but\_released,

[0, 0, self.width, self.height], 3)

def hover(self, mouse):

return pygame.sprite.collide\_rect(mouse, self)

from abc import ABCMeta, abstractmethod, abstractproperty

import pygame

import pygame.gfxdraw as pydraw

import pygame.draw as pygdraw

from utils import color

class Grid():

\_\_metaclass\_\_ = ABCMeta

def \_\_init\_\_(self, parent, columns, rows, cell\_size,

position, scale, grid\_color=color.grid):

self.cells = [[0 for i in range(columns)] for j in range(rows)]

self.initial\_size = cell\_size

self.scale = scale

self.columns = columns

self.rows = rows

self.x, self.y = position

self.last\_moved = None

self.parent = parent # parent surface

self.surface = pygame.Surface((self.width, self.height))

self.surface.fill(color.cell)

self.surface.set\_colorkey((0, 0, 0))

self.grid\_color = grid\_color

self.filled = 0

@abstractmethod

def update\_input(self, event):

""" Updates input of user (zoom in/out, drag)"""

pass

@abstractmethod

def update(self):

""" Updates the state of grid """

pass

@abstractmethod

def \_check\_winner(self, grid):

""" Function that will compute the winner for certain grid """

pass

@abstractmethod

def render(self):

""" Renders grid surface, grid lines, cells """

pass

@abstractproperty

def is\_just\_pressed(self):

""" Returns False if not properly pressed, otherwise -> position"""

pass

def update\_score(self):

""" Updates score of each player """

result = self.\_check\_winner(self.cells)

if result == 0 and self.filled == self.columns \* self.rows:

return -2

return result

def convert(self, position):

""" Converts absolute position to array position and returns False if unsuccessful"""

x, y = position

if not (self.x < x < self.x + self.width) or \

not (self.y < y < self.y + self.height):

return False

return (self.grid\_to\_arr(self.surf\_to\_grid(position)))

def add(self, value, position):

""" Adds particular move to a position(relative to array), returns True if successful """

if not self.cells[position[0]][position[1]]:

self.cells[position[0]][position[1]] = value

self.last\_moved = position

self.filled += 1

return True

else:

return False

def render\_grid(self):

""" Render lines and box of the grid """

self.surface.fill(color.cell)

# Draws a box of grid

pydraw.rectangle(self.surface, self.surface.get\_rect(), color.grid\_box)

# Draws vertical lines

for i in range(self.columns - 1):

pydraw.vline(self.surface, (i+1)\*self.cell\_size, 0, self.height, self.grid\_color)

# Draws horizontal lines

for j in range(self.rows - 1):

pydraw.hline(self.surface, 0, self.width, (j+1)\*self.cell\_size, self.grid\_color)

def render\_cells(self):

# Renders all the cells acording to the content of the cell

for row in range(len(self.cells)):

for col in range(len(self.cells[row])):

if self.cells[row][col] == 1:

self.render\_cross((row, col))

elif self.cells[row][col] == -1:

self.render\_circle((row, col))

def render\_cross(self, position):

""" Renders a cross at given position (relative to array) """

# id = 1

x, y = self.arr\_to\_grid(position)

pygdraw.line(self.surface, color.cross,

(x, y),

(x + self.cell\_size, y + self.cell\_size), 12

)

pygdraw.line(self.surface, color.cross,

(x + self.cell\_size, y),

(x, y + self.cell\_size), 12)

def render\_circle(self, position):

""" Renders a circle at given position (relative to array) """

# id = -1

x, y = self.arr\_to\_grid(position)

self.\_draw\_circle(x + self.cell\_size//2, y + self.cell\_size//2,

r=self.cell\_size//4,

R=self.cell\_size//2,

color=color.circle)

#

def \_draw\_circle(self, x, y, r, R, color):

#for i in range(r, R+1):

pygdraw.circle(self.surface, color, (x, y), R, 10)

def grid\_to\_surf(self, pos):

""" Converts Grid surface coordinates to Parent surface coordinates"""

return (self.x + pos[0], self.y + pos[1])

def surf\_to\_grid(self, pos):

""" Converts Parent surface coordinates to Grid surface coordinates"""

return (pos[0] - self.x, pos[1] - self.y)

def arr\_to\_grid(self, pos):

""" Converts Array coordinates to Grid surface coordinates"""

return (pos[0] \* self.cell\_size, pos[1] \* self.cell\_size)

def grid\_to\_arr(self, pos):

""" Converts Grid surface coordinates to Array coordinates """

return (pos[0] // self.cell\_size, pos[1] // self.cell\_size)

def highlight(self, position):

""" Highlights certain at position(relative to parent) """

x, y = position

if not (self.x < x < self.x + self.width) or \

not (self.y < y < self.y + self.height):

return

x, y = self.surf\_to\_grid((x, y))

x = (x // self.cell\_size) \* self.cell\_size

y = (y // self.cell\_size) \* self.cell\_size

x, y = self.grid\_to\_surf((x, y))

# TODO: Change to surface related position

pygame.draw.rect(self.parent, color.cell\_hov,

[x, y, self.cell\_size, self.cell\_size])

@property

def width(self):

""" Current width of the grid """

return self.cell\_size \* self.columns

@property

def height(self):

""" Current height of the grid """

return self.cell\_size \* self.rows

@property

def size(self):

""" Current dimensions of the grid """

return (self.width, self.height)

@property

def cell\_size(self):

""" Current scaled size of cells """

return int(self.initial\_size \* self.scale)

@property

def position(self):

return (self.x, self.y)

from grids.Grid import Grid

import pygame.gfxdraw as pydraw

import pygame

class StaticGrid(Grid):

def \_\_init\_\_(self, parent, position, cell\_size, columns=3, rows=3):

super().\_\_init\_\_(parent, columns, rows, cell\_size, position, scale=1)

self.scores = [0, 0]

self.touched = False

self.last\_pos = None

def update\_input(self, event):

# Only handling the touching: grid is not resizable

# Handles each event in loop

if event.type == pygame.MOUSEBUTTONDOWN:

if event.button == 1:

self.last\_pos = pygame.mouse.get\_pos()

if event.type == pygame.MOUSEBUTTONUP:

if event.button == 1:

self.touched = True

def update(self):

# Nothing to update

pass

def \_check\_winner(self, grid):

# rows

for x in range(0, 3):

row = set([grid[x][0], grid[x][1], grid[x][2]])

if len(row) == 1 and grid[x][0] != 0:

return grid[x][0]

# columns

for x in range(0, 3):

column = set([grid[0][x], grid[1][x], grid[2][x]])

if len(column) == 1 and grid[0][x] != 0:

return grid[0][x]

# diagonals

diag1 = set([grid[0][0], grid[1][1], grid[2][2]])

diag2 = set([grid[0][2], grid[1][1], grid[2][0]])

if (len(diag1) == 1 or len(diag2) == 1) and grid[1][1] != 0:

return grid[1][1]

return 0

def render(self):

# Parent surface blits grid surface at the position

self.parent.blit(self.surface, (self.x, self.y))

self.render\_grid()

self.render\_cells()

self.touched = False

@property

def is\_just\_pressed(self):

# If no touched returns False

# Otherwise returns the position of touching

if not self.touched:

return False

return self.last\_pos

import pygame

class Mouse(pygame.sprite.Sprite):

def \_\_init\_\_(self):

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

self.surface = pygame.Surface((1, 1))

self.rect = self.surface.get\_rect()

self.pressed = False

def on\_button\_down(self):

self.pressed = True

def on\_button\_up(self):

self.pressed = False

def update(self):

self.rect.center = pygame.mouse.get\_pos()

@property

def position(self):

return self.rect.center

@property

def x(self):

return self.position[0]

@property

def y(self):

return self.position[1]

import pygame

import screens.screen\_manager as screen\_manager

from utils import \*

class TicTacToe:

def \_\_init\_\_(self, w, h):

self.size = self.width, self.height = self.w, self.h = w, h

self.window = pygame.display.set\_mode(self.size)

self.current\_screen = None

self.mainloop = None

pygame.init()

def run(self, screen):

self.current\_screen = screen

self.current\_screen.create()

self.mainloop = True

self.update()

def update(self):

while self.mainloop:

self.current\_screen.handle\_input()

self.current\_screen.update()

self.current\_screen.render()

def set(self, screen):

self.mainloop = False

self.run(screen)

def quit(self):

self.mainloop = False

Utils.quit\_game()

def screen(self, name):

return screen\_manager.screens[name]

def change\_color(self):

color.color\_scheme = ( color.color\_scheme + 1 ) % color.max\_colors

game = TicTacToe(1080, 720)

game.run(game.screen("MainMenuScreen")(game))

from grids.Grid import Grid

import pygame

class ExpandableGrid(Grid):

def \_\_init\_\_(self, parent, position, cell\_size, columns=100, rows=100):

super().\_\_init\_\_(parent, columns, rows, cell\_size, position, scale=(25/2)/columns)

#elf.cells[0][0] = 1

self.scores = [0, 0]

self.x, self.y = self.parent.x - self.x, self.parent.y - self.y

self.last\_pressed = None # last pressed point

self.first = None # first pressed point on the drag

self.full\_delta = None # full delta between last and first point

self.min\_scale = 0.15 \* (25/columns)

self.max\_scale = 0.5 \* (25/rows)

self.max\_points = 5

self.dragged = False

def update\_input(self, event):

# Handles each event in for loop

if event.type == pygame.MOUSEBUTTONDOWN:

if event.button == 4: # mouse wheel rolled up

self.resize(+0.05, event.pos)

elif event.button == 5: # mouse wheel rolled down

self.resize(-0.05, event.pos)

elif event.button == 1 and not self.dragged:

self.dragged = True

self.last\_pressed = self.first = pygame.mouse.get\_pos()

if event.type == pygame.MOUSEBUTTONUP:

if event.button == 1:

if self.dragged:

self.dragged = False

self.full\_delta = self.first[0] - pygame.mouse.get\_pos()[0], self.first[1] - pygame.mouse.get\_pos()[1]

self.last\_pressed = None

def update(self):

if self.dragged:

self.on\_drag()

def \_check\_winner(self, grid):

pos = x, y = self.last\_moved

if pos is None:

return 0

total = []

# columns

for i in range(y-(self.max\_points-1), y+1):

if 0 <= i <= self.rows - 1 and 0 <= i + self.max\_points <= self.rows:

total.append(self.cells[x][i: i+self.max\_points])

#rows

for i in range(x-(self.max\_points-1), x+1):

row = []

for j in range(self.max\_points):

if 0 <= i+j <= self.columns - 1:

row.append(self.cells[i+j][y])

total.append(row)

#y=-x diagonal

for i in range(self.max\_points):

diag1 = []

for j in range(self.max\_points):

if 0 <= x-i+j <= self.columns - 1 and 0 <= y-i+j <= self.rows - 1:

diag1.append(self.cells[x-i+j][y-i+j])

total.append(diag1)

#y=x diagonal

for i in range(self.max\_points):

diag2 = []

for j in range(self.max\_points):

if 0 <= x-i+j <= self.columns - 1 and 0 <= y+i-j <= self.rows - 1:

diag2.append(self.cells[x-i+j][y+i-j])

total.append(diag2)

for line in total:

if len(line) == 5 and len(set(line)) == 1:

return line[0]

return 0

def render(self):

self.parent.blit(self.surface, (self.x, self.y))

self.render\_grid()

self.render\_cells()

self.full\_delta = None

# updating full\_delta

def on\_drag(self):

delta = pygame.mouse.get\_pos()[0] - self.last\_pressed[0], pygame.mouse.get\_pos()[1] - self.last\_pressed[1]

self.last\_pressed = pygame.mouse.get\_pos()

self.x = self.x + delta[0]

self.y = self.y + delta[1]

self.check\_boundaries()

def resize(self, delta\_scale, scale\_point):

if self.scale + delta\_scale < self.min\_scale \

or self.scale + delta\_scale > self.max\_scale:

return

point\_x, point\_y = self.surf\_to\_grid(scale\_point)

ratio\_x = point\_x / self.cell\_size

ratio\_y = point\_y / self.cell\_size

self.scale += delta\_scale

new\_point\_x = int(ratio\_x \* self.cell\_size)

new\_point\_y = int(ratio\_y \* self.cell\_size)

self.x += point\_x - new\_point\_x

self.y += point\_y - new\_point\_y

self.surface = pygame.transform.scale(self.surface, (self.width, self.height))

def check\_boundaries(self):

if self.width < self.parent.width or \

self.height < self.parent.height:

return

if self.x > 0:

self.x = 0

elif self.x < -self.width + self.parent.width:

self.x = -self.width + self.parent.width

if self.y > 0:

self.y = 0

elif self.y < -self.height + self.parent.height:

self.y = -self.height + self.parent.height

@property

def is\_just\_pressed(self):

# If not input provided or the delta is small return False

# Otherwise returns the position of touching

if not self.full\_delta:

return False

if self.full\_delta[0]\*\*2 + self.full\_delta[1]\*\*2 > 25\*\*2:

# Delta lenght is not more than 5 = sqrt(25) pixels

return False

return pygame.mouse.get\_pos()

import pickle

import base64

from sys import argv

from random import shuffle

def compute\_best\_move(grid, player, depth=2):

winner = compute\_winner(grid)

if winner == player:

return None, 1e7

elif winner == -player:

return None, -1e7

if depth == 0:

mark = estimation(grid, player)

return None, mark

else:

possible\_moves = compute\_possible\_moves(grid)

best\_move, best\_mark = None, -1e8

for current\_move in possible\_moves:

grid[current\_move[0]][current\_move[1]] = player

\_, current\_mark = compute\_best\_move(grid, -player, depth=depth - 1)

current\_mark = -current\_mark

if current\_mark > best\_mark:

best\_mark = current\_mark

best\_move = current\_move

grid[current\_move[0]][current\_move[1]] = 0

return best\_move, best\_mark

def estimation(grid, player):

score = 0

rows = make\_rows(grid)

for row in rows:

score += (

row.count('+') \* 1 +

row.count('++') \* 10 +

row.count('+++') \* 100 +

row.count('++++') \* 1000 +

row.count('+++++') \* 10000

)

score -= (

row.count('-') \* 1 +

row.count('--') \* 10 +

row.count('---') \* 100 +

row.count('----') \* 1000 +

row.count('-----') \* 10000

)

return score \* player

def compute\_winner(grid):

rows = make\_rows(grid)

for row in rows:

if '+++++' in row:

return 1

elif '-----' in row:

return -1

return None

def compute\_possible\_moves(grid):

moves = set()

for center\_row in range(len(grid)):

for center\_column in range(len(grid)):

if grid[center\_row][center\_column]:

for row in range(max(center\_row - 4, 0), min(center\_row + 5, len(grid))):

for column in range(max(center\_column - 4, 0), min(center\_column + 5, len(grid))):

if not grid[row][column]:

moves.add((row, column))

moves = list(moves)

shuffle(moves)

return moves

def to\_symbol(cell):

if cell > 0:

return '+'

elif cell < 0:

return '-'

else:

return ' '

def make\_rows(grid):

rows = []

rows.extend(''.join(to\_symbol(cell) for cell in row) for row in grid)

rows.extend(''.join(to\_symbol(cell) for cell in row) for row in zip(\*grid))

start\_points = (

[(0, column) for column in range(len(grid) - 4)] +

[(row, 0) for row in range(1, len(grid) - 4)]

)

for start\_point in start\_points:

row, column = start\_point

new\_row = []

while row < len(grid) and column < len(grid):

new\_row.append(grid[row][column])

row += 1

column += 1

rows.append(''.join(to\_symbol(cell) for cell in new\_row))

start\_points = (

[(0, column) for column in range(4, len(grid))] +

[(row, len(grid) - 1) for row in range(1, len(grid) - 4)]

)

for start\_point in start\_points:

row, column = start\_point

new\_row = []

while row < len(grid) and column >= 0:

new\_row.append(grid[row][column])

row += 1

column -= 1

rows.append(''.join(to\_symbol(cell) for cell in new\_row))

return rows

from players.Player import Player

import pygame

class Human(Player):

def \_\_init\_\_(self, grid, id):

super().\_\_init\_\_(grid, id)

self.pressed = False

def \_\_repr\_\_(self):

return "Human"

def on\_turn(self):

# Handle the input user and returns the position

pos = self.grid.is\_just\_pressed

if not pos:

return False

return self.make\_move(pos)

def make\_move(self, position):

# Convert position from absolute to cell position

# Returns true if move is successful and cell is empty

result\_pos = self.grid.convert(position)

if result\_pos:

return self.grid.add(self.id, result\_pos)

else:

return False

import doctest

import sys

import math

import os

import pickle

import base64

import subprocess

import platform

from random import choice

from copy import deepcopy

if sys.version\_info < (3, 5):

math.inf = float('inf')

infinity = math.inf

class AIComponent:

def \_\_init\_\_(self, id):

self.\_human = id \* -1

self.\_computer = id

@property

def human(self):

return self.\_human

@property

def computer(self):

return self.\_computer

def compute\_next\_move\_3x3(self, grid):

depth = len(self.\_\_empty\_cells(grid))

#here im returning True if comp won and False otherwise

if depth == 0 and self.\_\_is\_over(grid):

return self.\_\_is\_won(grid, self.\_computer)

if self.\_\_is\_over(grid):

return self.\_\_is\_won(grid, self.\_computer)

#here im returning None in case of draw

if depth == 0:

return

if depth == 9:

x = choice([0, 1, 2])

y = choice([0, 1, 2])

else:

move = self.\_\_minimax\_algorithm(grid, depth, self.\_computer)

x, y = move[0], move[1]

#coordinates

return x, y

def \_\_minimax\_algorithm(self, grid, depth, player):

if player == self.\_computer:

best = [-1, -1, -infinity]

else:

best = [-1, -1, +infinity]

if depth == 0 or self.\_\_is\_over(grid):

score = self.\_\_change\_score(grid)

return [-1, -1, score]

empty\_cells = self.\_\_empty\_cells(grid)

for cell in empty\_cells:

x, y = cell[0], cell[1]

grid[x][y] = player

score = self.\_\_minimax\_algorithm(grid, depth - 1, -player)

grid[x][y] = 0

score[0], score[1] = x, y

if player == self.\_computer:

if score[2] > best[2]:

best = score

else:

if score[2] < best[2]:

best = score

return best

def \_\_empty\_cells(self, grid):

empty\_cells = []

for x, row in enumerate(grid):

for y, cell in enumerate(row):

if cell == 0:

empty\_cells.append([x, y])

return empty\_cells

def \_\_is\_won(self, grid, player):

win\_state = [

[grid[0][0], grid[0][1], grid[0][2]],

[grid[1][0], grid[1][1], grid[1][2]],

[grid[2][0], grid[2][1], grid[2][2]],

[grid[0][0], grid[1][0], grid[2][0]],

[grid[0][1], grid[1][1], grid[2][1]],

[grid[0][2], grid[1][2], grid[2][2]],

[grid[0][0], grid[1][1], grid[2][2]],

[grid[2][0], grid[1][1], grid[0][2]],

]

if [player, player, player] in win\_state:

return True

else:

return False

def \_\_is\_over(self, grid):

return self.\_\_is\_won(grid, self.\_human) or self.\_\_is\_won(grid, self.\_computer)

def \_\_change\_score(self, grid):

if self.\_\_is\_won(grid, self.\_computer):

score = +1

elif self.\_\_is\_won(grid, self.\_human):

score = -1

else:

score = 0

return score

def compute\_next\_move\_exp(self, grid):

grid\_pickled = pickle.dumps(grid)

grid\_pickled = base64.b64encode(grid\_pickled).decode()

player\_pickled = pickle.dumps(self.\_computer)

player\_pickled = base64.b64encode(player\_pickled).decode()

result\_pickled = None

if platform.system() == "Windows":

result\_pickled = subprocess.Popen(

[

os.path.join(

os.path.dirname(os.path.abspath(\_\_file\_\_)),

'pypywin', 'pypy3.exe'

),

os.path.join(

os.path.dirname(os.path.abspath(\_\_file\_\_)),

'unlimited.py'

),

grid\_pickled,

player\_pickled

],

stdout=subprocess.PIPE,

universal\_newlines=True

).communicate()[0]

else:

result\_pickled = subprocess.run(

[

os.path.join(

os.path.dirname(os.path.abspath(\_\_file\_\_)),

'pypymac', 'bin', 'pypy3'

),

os.path.join(

os.path.dirname(os.path.abspath(\_\_file\_\_)),

'unlimited.py'

),

grid\_pickled,

player\_pickled

],

stdout=subprocess.PIPE

).stdout

result\_pickled = base64.b64decode(result\_pickled)

result = pickle.loads(result\_pickled)

if result[0] != None:

return result[0][0], result[0][1]

else:

if result[1] == 1e7:

return True

else:

return False

if \_\_name\_\_ == '\_\_main\_\_':

doctest.testmod()

import pygame

from utils import \*

from Mouse import Mouse

from buttons.Button import Button

class SelectionScreen:

def \_\_init\_\_(self, game, type):

self.game = game

self.type = type

def create(self):

self.surface = pygame.Surface(self.game.size)

self.mouse = Mouse()

self.button\_hum = Button(

position=(self.game.w/3, self.game.h/2),

size=(300, 150),

color=color.button,

text="Against Human",

on\_click=lambda: self.game.set(self.game.screen("GameScreen")(self.game, self.type, 0)))

self.button\_ai = Button(

position=(self.game.w\*2/3, self.game.h/2),

size=(300, 150),

color=color.button,

text="Against AI",

on\_click=lambda: self.game.set(self.game.screen("GameScreen")(self.game, self.type, 1)))

def handle\_input(self):

for e in pygame.event.get():

if e.type == pygame.QUIT:

self.game.quit()

if e.type == pygame.MOUSEBUTTONDOWN:

self.mouse.on\_button\_down()

if e.type == pygame.MOUSEBUTTONUP:

self.mouse.on\_button\_up()

if e.type == pygame.KEYDOWN:

if e.key == pygame.K\_c:

self.game.change\_color()

if e.key == pygame.K\_ESCAPE:

self.game.quit()

def update(self):

self.mouse.update()

self.button\_hum.update(self.mouse)

self.button\_ai.update(self.mouse)

def render(self):

self.game.window.blit(self.surface, (0, 0))

self.surface.fill(color.background\_selection)

self.button\_hum.render(self.surface)

self.button\_ai.render(self.surface)

Utils.toast(self.surface, "SELECT YOUR OPPONENT", 40, color.text,

self.game.w/2, self.game.h/3)

pygame.display.flip()

from screens.GameScreen import GameScreen

from screens.MainMenuScreen import MainMenuScreen

from screens.SelectionScreen import SelectionScreen

screens = {

"MainMenuScreen": MainMenuScreen,

"GameScreen": GameScreen,

"SelectionScreen": SelectionScreen

}

import pygame

from utils import \*

from Mouse import Mouse

from buttons.Button import Button

class MainMenuScreen:

def \_\_init\_\_(self, game):

self.game = game

def create(self):

self.surface = pygame.Surface(self.game.size)

self.mouse = Mouse()

self.button\_3x3 = Button(

position=(self.game.w/3, self.game.h/2),

size=(300, 150),

color=color.button,

text="3x3 Grid",

on\_click=lambda: self.game.set(self.game.screen("SelectionScreen")(self.game, 0)))

self.button\_inf = Button(

position=(self.game.w\*2/3, self.game.h/2),

size=(300, 150),

color=color.button,

text="Infinite Grid",

on\_click=lambda: self.game.set(self.game.screen("SelectionScreen")(self.game, 1)))

def handle\_input(self):

for e in pygame.event.get():

if e.type == pygame.QUIT:

self.game.quit()

if e.type == pygame.MOUSEBUTTONDOWN:

self.mouse.on\_button\_down()

if e.type == pygame.MOUSEBUTTONUP:

self.mouse.on\_button\_up()

if e.type == pygame.KEYDOWN:

if e.key == pygame.K\_c:

self.game.change\_color()

if e.key == pygame.K\_ESCAPE:

self.game.quit()

def update(self):

self.mouse.update()

self.button\_inf.update(self.mouse)

self.button\_3x3.update(self.mouse)

def render(self):

self.game.window.blit(self.surface, (0, 0))

self.surface.fill(color.background\_mainmenu)

self.button\_inf.render(self.surface)

self.button\_3x3.render(self.surface)

Utils.toast(self.surface, 'WELCOME TO THE TIC-TAC-TOE GAME', 50, color.text,

self.game.w/2, self.game.h/5)

Utils.toast(self.surface, 'CHOOSE THE GRID', 50, color.text,

self.game.w/2, self.game.h/3)

pygame.display.flip()

import pygame

from random import randint

from utils import \*

from grids.ExpandableGrid import ExpandableGrid

from grids.StaticGrid import StaticGrid

from players.Human import Human

from players.Computer import Computer

from screens.surfaces.GameSurface import GameSurface

class GameScreen:

def \_\_init\_\_(self, game, type, mode):

self.game = game

self.type = type

self.mode = mode

# type == 0 - Static

# type == 1 - Expandable

# mode == 0 - Human

# mode == 1 - AI

def create(self):

self.surface = GameSurface((0, 0), self.game.size)

print("GameScreen size", self.surface.get\_size())

self.grid = [StaticGrid, ExpandableGrid][self.type](self.surface, (210, 50), cell\_size=220)

self.players = [Human(self.grid, 1), [Human, Computer][self.mode](self.grid, -1)]

# num = (id + 2) % 3

self.victory\_message = ""

self.currentPLayerNum = 0 if self.mode and self.type else randint(0, 1)

self.update\_current()

def handle\_input(self):

for e in pygame.event.get():

self.grid.update\_input(e)

if e.type == pygame.QUIT:

self.game.quit()

if e.type == pygame.KEYDOWN:

if e.key == pygame.K\_c:

self.game.change\_color()

if e.key == pygame.K\_ESCAPE:

self.game.set(self.game.screen("MainMenuScreen")(self.game))

def update(self):

if self.victory\_message:

Utils.toast(self.surface,

self.victory\_message,

40, color.victory\_text, self.game.w/2, self.game.h/2)

return

self.update\_current()

self.grid.update()

if not self.current.on\_turn():

pass

else:

result = self.grid.update\_score()

if result != 0:

if result != -2: # not draw

self.victory\_message = "Player #{} - {} has won".format(self.currentPLayerNum, self.current)

else:

self.victory\_message = "Draw!"

else:

self.currentPLayerNum = 1 - self.currentPLayerNum

def render(self):

self.surface.render(self.game.window)

self.surface.fill(color.background\_game)

self.grid.highlight(pygame.mouse.get\_pos())

self.grid.render()

self.renderMessages()

pygame.display.flip()

def renderMessages(self):

Utils.toast(self.surface,

"Now is turn of Player #{} - {}".format(self.currentPLayerNum, self.current),

20, color.text, self.game.w/2, 20)

def update\_current(self):

self.current = self.players[self.currentPLayerNum]