**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу**

**Кафедра системного проектування**

**Курсова робота**

**з дисципліни «Об'єктно-орієнтоване програмування»**

**на тему: «Гра "Монополія"»**

Виконав: студент 2 курсу групи ДА-21

спеціальність 122 «Комп’ютерні науки»

Кузьмін Артем Ренатович

Керівник: Булах Б.В.

Київ – 2024 рік

**ЗМІСТ**

[**ВСТУП** 3](#_Toc168254609)

[**РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ** 4](#_Toc168254610)

[**1.1. Уточнена постановка задачі на розробку програмного забезпечення** 4](#_Toc168254611)

[**1.2. Обґрунтування вибору засобів розробки** 5](#_Toc168254612)

[**РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ТА ПОДІЛ РОЗРОБКИ ПРОЕКТУ** 6](#_Toc168254613)

[**2.1 Розподіл коду програми** 6](#_Toc168254614)

[**2.2 Реалізація коду** 7](#_Toc168254615)

[**ВИСНОВОК:** 24](#_Toc168254616)

[**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:** 26](#_Toc168254617)

# **ВСТУП**

Гра “Монополія” – легендарний настільний симулятор фінансового управління, який здобув популярність у кінці XIX століття і став відомим в усьому світі завдяки своїй захоплюючій геймплейній механіці та великому різноманіттю стратегій.

Ця ігра була вперше представлена в 1903 році Елізабет Магі та Чарльзом Дарроу як “The Landlord's Game” [1], але згодом була перероблена та розроблена як “Монополія” компанією Parker Brothers. Завдяки своїй розважальній інтригующій природі, “Монополія” не тільки надихає гравців на стратегічне планування та фінансовий аналіз, але й є чудовим способом відпочити та розважитися, допомагаючи відволіктися від повсякденних проблем.

Метою даної курсової роботи є розробка та реалізація програмного продукту для моделювання гри “Монополія”. Завдання роботи включає в себе аналіз різних існуючих стратегій гри, вибір оптимальних методів реалізації та розробку програмного забезпечення з зручним користувацьким інтерфейсом.

Практичне значення цього проекту полягає в глибшому розумінні принципів та алгоритмів програмування, що досліджуються в контексті гри “Монополія”, а також у отриманні навичок роботи з бібліотеками для графічного інтерфейсу. Виконання завдання допоможе розвивати аналітичне мислення, навички програмування та покращувати навички вибору найбільш оптимального та ефективного підходу до реалізації з урахуванням наявного досвіду.

У процесі виконання курсової роботи буде використано таке програмне забезпечення: середовище розробки IntelliJ IDEA 2024.1 (Ultimate Edition), операційна система MacOs Sonoma , та текстовий редактор MS Word 2021.

Пояснювальна записка складається зі вступу, трьох розділів, висновків та списку використаних джерел.

# **РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ**

## **1.1. Уточнена постановка задачі на розробку програмного забезпечення**

Реалізація гри “Монополія” з графічним інтерфейсом відбувалася у команді з чотирьх чоловік. Основні вимоги до функціоналу гри включають:

1. **Меню вибору гравців.** На цій сторінці користувач матиме можливість вибрати кількість гравців (від 2 до 6) та вказати їхні імена.

2. **Головна сторінка гри.** Це основний ігровий екран, на якому відображатиметься дошка гри, рухи гравців, їхні власності, грошові кошти та поточні активності гравців.

3. **Інформація про гравців.** На екрані повинна відображатись інформація про кожного гравця, зокрема їхні імена, грошові кошти, власності та статус (грає користувач чи ні).

4. **Взаємодія гравців з грою.** Гравці повинні мати можливість обирати ділянки для покупки, кидати кубики для визначення кроків, платити податки та виконувати інші дії, що визначаються правилами гри.

5. **Відображення інформації гри.** На екрані повинна бути відображена інформація для користувача про події в грі, такі як купівля нерухомості, отримання грошей, сплата податків, нове коло і тд.

6. **Можливість завершення гри.** Гра повинна завершуватися або якщо на ігровій дошці залишився лише один гравець в онлайні або коли було пройдено 100 кіл (тоді у кого більше грошей, той і виграв).

7. **Можливість користувача покинути гру.** Гравець повинен мати можливість покинути гру, натиснувши на відповідну кнопку.

8. **Наявність юніт-тестів.** Необхідно створити набір юніт-тестів, які перевіряють правильність роботи окремих компонентів програми.

Для забезпечення зручного та зрозумілого взаємодії з користувачем необхідно врахувати чіткість інтерфейсу та зрозумілість підписів кнопок та функцій.

## **1.2. Обґрунтування вибору засобів розробки**

Для реалізації поставленого завдання було обрано мову програмування Java [2], з використанням JavaFx [3] та IntelliJ IDEA з наступних міркувань:

1. **Ефективність мови програмування Java.** Вибір мови Java обґрунтований її широким поширенням, простотою в освоєнні та великою кількістю готових бібліотек для розробки ігор. Java є платформонезалежною мовою, що означає, що гра, розроблена на Java, може запускатися на різних операційних системах без значних змін.

2. **Об'єктно-орієнтований підхід мови програмування Java для розробки ігор.** Java підтримує об'єктно-орієнтований підхід до програмування, що сприяє модульності та повторному використанню коду. Це дозволить розробникам легко керувати різними елементами гри та розширювати її функціональність [4].

3. **Зручність використання** **IntelliJ IDEA.** IntelliJ IDEA - це одне з найпопулярніших середовищ розробки для Java. Воно відоме своєю зручністю використання, багатим функціоналом та підтримкою різних інструментів для розробки програм на Java. IntelliJ IDEA забезпечує швидкий та ефективний процес розробки, а також підтримує використання JavaFx для створення графічного інтерфейсу.

4. **Підтримка графічного інтерфейсу користувача бібліотекою JavaFx.** JavaFx - це потужна бібліотека для розробки графічних інтерфейсів у програмах на Java [5]. Вона надає багатий набір інструментів для створення інтерактивних графічних інтерфейсів користувача, що робить його ідеальним вибором для реалізації гри “Монополія” з вимогами до графічного інтерфейсу.

Обрані засоби розробки дозволять ефективно та швидко реалізувати гру “Монополія” з відповідними функціональними та графічними можливостями, забезпечуючи високу якість та зручність користування.

# **РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ТА ПОДІЛ РОЗРОБКИ ПРОЕКТУ**

**2.1 Розподіл коду програми**

**1. Розробка Графічного Інтерфейсу:**

Було створено графічний інтерфейс користувача (GUI) для гри. Це включало розробку макету вікон, налаштування елементів керування та стилізації за допомогою JavaFX та FXML. Створено основний екран гри, включаючи ігрове поле, інформаційні панелі для гравців та інші інтерактивні елементи.

**2. Ініціалізація Ігрового Поля:**

Ініціалізація ігрового поля, яке включає налаштування всіх клітинок, таких як власність, в'язниця, стартова позиція та інші. За допомогою класу MonopolyMap було створено карту гри, що забезпечує коректне розташування та функціонування кожної клітинки.

**3. Ініціалізація та Запуск Гри:**

Метод initNewGame ініціалізує нову гру, створюючи ігрове поле та гравців. Конструктор класу Game приймає інтерфейс користувача і викликає метод run, який запускає гру та відображає повідомлення у разі невдачі ініціалізації.

**4. Тестування та Налагодження:**

Після розробки основних компонентів гри зроблено тестуванням та налагодженням, щоб забезпечити стабільну роботу програми. Це включало перевірку коректності ініціалізації, роботи з інтерфейсом та логікою гри, а також виправлення знайдених помилок.

**2.2 Реалізація коду**

Gamepad.fxml

**Основні Елементи**

* **HBox**: Основний контейнер, що визначає базову структуру інтерфейсу. Має ID "base".
* **GridPane**: Використовується для розміщення ігрового поля. Має розмір 635x605 пікселів. Включає колонкові та рядкові обмеження.
* **VBox**: Використовується для відображення клітинок ігрового поля та інших елементів інтерфейсу.

**Внутрішні Елементи**

* **FlowPane**: Панель всередині кожної VBox для розміщення різних елементів, таких як кнопки, зображення і т.д.
* **Button**: Кнопки для різних дій у грі, таких як "Roll Dice", "Nothing" та "Retire". Мають клас "BigButton".
* **ImageView**: Використовується для відображення зображень, таких як кубики та логотип гри.
* **Label**: Текстові елементи для відображення інформації про гравців, такі як імена та кількість грошей.
* **TextArea**: Текстове поле для відображення повідомлень.

**Основні кнпоки та функціонал коду FXML:**

1. **Кнопка "Roll Dice":**

<Button fx:id="yesButton" alignment="CENTER" contentDisplay="CENTER" maxWidth="1.7976931348623157E308"

mnemonicParsing="false" onAction="#yesClick" prefHeight="50.0" prefWidth="100.0"

styleClass="BigButton" text="Roll Dice" GridPane.columnIndex="2" GridPane.rowIndex="4">

<GridPane.margin>

<Insets bottom="5.0" left="5.0" right="5.0" top="5.0"/>

</GridPane.margin>

</Button>

**Візуальная форма кнопки:**

**Изображение выглядит как текст, Шрифт, Графика, снимок экрана

Автоматически созданное описание**

**Рис. 1 – Кнопка “Roll dice”**

**Изображение выглядит как снимок экрана, Прямоугольник, Графика, Шрифт

Автоматически созданное описание**

**Рис. 2 – Кнопка “Roll dice” відносно інших моделей**

**CSS стиль для Кнопки "Roll Dice":**

#yesButton {  
 -fx-font-weight:bold;  
 -fx-font-size:20;  
 -fx-background-color: rgba(220, 220, 100, 0.8);  
}

1. **Кнопка "Nothing":**

<Button fx:id="noButton" alignment="CENTER" contentDisplay="CENTER" maxWidth="1.7976931348623157E308"

mnemonicParsing="false" onAction="#noClick" prefHeight="50.0" prefWidth="100.0"

styleClass="BigButton" text="Nothing" GridPane.columnIndex="3" GridPane.rowIndex="4">

<GridPane.margin>

<Insets bottom="5.0" left="5.0" right="5.0" top="5.0"/>

</GridPane.margin>

</Button>

**Візуальная форма кнопки:**

Изображение выглядит как Шрифт, снимок экрана, Графика, текст

Автоматически созданное описание

**Рис. 3 – Кнопка “Nothing”**

**Изображение выглядит как снимок экрана, Прямоугольник, Графика, Шрифт

Автоматически созданное описание**

**Рис. 4 – Кнопка “Nothing”відносно інших моделей**

**CSS Стиль для Кнопки "BigButton" для Кнопки "Nothing":**

#noButton {

-fx-font-weight: bold;

-fx-font-size: 20;

-fx-background-color: rgba(100, 220, 220, 0.8);

}

1. **Кнопка "Retire"**

<Button id="retireButton" fx:id="retireButton" minHeight="70.0" minWidth="230.0"

mnemonicParsing="false" onAction="#retireClick" prefHeight="70.0" prefWidth="70.0"

text="Retire">

<HBox.margin>

<Insets bottom="5.0" left="5.0" right="5.0" top="5.0"/>

</HBox.margin>

</Button>

**Візуальная форма кнопки:**

**Изображение выглядит как текст, визитная карточка, Шрифт, снимок экрана

Автоматически созданное описание**

**Рис. 5 – Кнопка “Retire”**

**CSS Стиль для Кнопки "Retire":**

#retireButton {

-fx-font-weight: bold;

-fx-font-size: 20;

-fx-background-color: rgba(150, 150, 128, 0.4);

}

1. **Кнопка “ Owner Display”:**

<Button fx:id="owner12" maxHeight="12.0" maxWidth="1.7976931348623157E308" minHeight="12.0"

mnemonicParsing="false" prefHeight="12.0" styleClass="OwnerDisplay" text=" "/>

**Візуальная форма кнопки:**

Изображение выглядит как текст, снимок экрана, Шрифт, Цвет электрик

Автоматически созданное описание

**Рис. 6 – Кнопка “Owner Display”**

**CSS Стиль для Кнопки Власника (Owner Display):**

.OwnerDisplay {

-fx-background-color: #000;

}

1. **Приклад Коду для Відображення Гравця 1:**

<HBox prefHeight="55.0" prefWidth="200.0">

<children>

<ImageView fx:id="playerIcon1" fitHeight="50.0" fitWidth="50.0" pickOnBounds="true" preserveRatio="true">

<HBox.margin>

<Insets left="5.0" right="5.0"/>

</HBox.margin>

</ImageView>

<Label fx:id="nameLabel1" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308"

prefWidth="100.0" styleClass="nameLabel" text="Player1">

<HBox.margin>

<Insets left="5.0" right="5.0"/>

</HBox.margin>

</Label>

<Label fx:id="moneyLabel1" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308"

minWidth="70.0" styleClass="moneyLabel" text="Money1">

<HBox.margin>

<Insets left="5.0" right="5.0"/>

</HBox.margin>

</Label>

</children>

<VBox.margin>

<Insets/>

</VBox.margin>

<padding>

<Insets left="5.0" right="5.0"/>

</padding>

</HBox>

**Візуальная форма об’єкта:**

Изображение выглядит как сердце, дизайн

Автоматически созданное описание

**Рис. 7 – об’єкт “Власник 1 ”**

**CSS Стиль для об’єкта Власника (Owner Display):**

.nameLabel {

-fx-font-size: 17;

-fx-font-style: Italic;

}

.moneyLabel {

-fx-font-weight: bold;

-fx-font-size: 16;

}

**З власниками 2-6 аналогічно.**

1. **HBox (Горизонтальна панель):**

<HBox id="base" fx:id="base" maxHeight="720.0" maxWidth="960.0" minHeight="720.0" minWidth="960.0" prefHeight="720.0" prefWidth="960.0" xmlns="http://javafx.com/javafx/8.0.141" xmlns:fx="http://javafx.com/fxml/1" fx:controller="GameLogicGUI.Controller">

<children>

<!-- GridPane та VBox є дочірніми елементами HBox -->

<GridPane maxHeight="635.0" maxWidth="605.0" minHeight="635.0" minWidth="605.0" prefHeight="635.0" prefWidth="605.0">

<!-- Інші елементи GridPane тут -->

</GridPane>

<VBox alignment="CENTER" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefWidth="320.0">

<!-- Інші елементи VBox тут -->

</VBox>

</children>

</HBox>

1. **GridPane (Таблична панель):**

<GridPane maxHeight="635.0" maxWidth="605.0" minHeight="635.0" minWidth="605.0" prefHeight="635.0" prefWidth="605.0">

<columnConstraints>

<ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0"/>

<!-- Додаткові налаштування стовпців тут -->

</columnConstraints>

<rowConstraints>

<RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES"/>

<!-- Додаткові налаштування рядків тут -->

</rowConstraints>

<children>

<VBox id="s11" alignment="BOTTOM\_CENTER" prefHeight="200.0" prefWidth="100.0">

<children>

<FlowPane fx:id="squarePane11" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefHeight="100.0" prefWidth="100.0"/>

</children>

</VBox>

<!-- Додаткові елементи GridPane тут -->

</children>

</GridPane>

1. **VBox** **(Вертикальна панель):**

<VBox alignment="CENTER" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefWidth="320.0">

<children>

<ImageView id="scoreBoardImage" fitHeight="70.0" fitWidth="200.0" pickOnBounds="true" preserveRatio="true">

<image>

<Image url="@img/scoreboard.png"/>

</image>

</ImageView>

<HBox prefHeight="55.0" prefWidth="200.0">

<children>

<ImageView fx:id="playerIcon1" fitHeight="50.0" fitWidth="50.0" pickOnBounds="true" preserveRatio="true"/>

<Label fx:id="nameLabel1" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefWidth="100.0" styleClass="nameLabel" text="Player1"/>

<Label fx:id="moneyLabel1" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" minWidth="70.0" styleClass="moneyLabel" text="Money1"/>

</children>

</HBox>

<!-- Додаткові елементи VBox тут -->

</children>

</VBox>

**Опис основних елементів fxml:**

* **HBox:** Горизонтальна панель для розміщення вкладених елементів в рядок.
* **id="base" та fx:id="base**": Встановлює ідентифікатор для панелі.
* **maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth:** Задає розміри панелі.
* **xmlns та xmlns:fx:** Визначає простори імен.
* **fx:controller:** Задає контролер для керування логікою програми.
* **Children:** Контейнер для вкладених елементів.
* **GridPane:** Сіткова панель для розміщення елементів в таблиці.
* **maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth:** Задає розміри сітки.
* **columnConstraints:**Визначає налаштування для стовпців у `GridPane`.
* **ColumnConstraints:** Встановлює обмеження для стовпців.
* **hgrow:** Вказує, коли стовпець може рости.
* minWidth, prefWidth: Встановлює мінімальну та бажану ширину стовпців.
* **rowConstraints:** Визначає налаштування для рядків у `GridPane`.
* **RowConstraints:** Встановлює обмеження для рядків.
* **minHeight, prefHeight:** Встановлює мінімальну та бажану висоту рядків.
* **vgrow:** Вказує, коли рядок може рости.
* **HBox.margin:** Встановлює зовнішні відступи для `HBox`.
* **Insets:** Вказує відступи (у цьому випадку немає відступів).
* **VBox:** Вертикальна панель для розміщення елементів у стовпець.
* **id="s11"та `alignment="BOTTOM\_CENTER"`:** Встановлює ідентифікатор і вирівнювання по центру знизу.
* **prefHeight, prefWidth:** Встановлює бажану висоту і ширину панелі.
* **GridPane.margin:** Встановлює зовнішні відступи для елемента всередині `GridPane`.
* **Insets:** Вказує відступи зверху та знизу по 3 пікселі.
* **children:** Контейнер для вкладених елементів всередині `VBox`.
* **FlowPane:** Потокова панель для послідовного розміщення елементів.
* **fx:id="squarePane11":** Встановлює ідентифікатор панелі.
* **maxHeight, maxWidth:** Встановлює максимальні розміри панелі.
* **prefHeight, prefWidth:** Встановлює бажану висоту і ширину панелі.
* **Padding:** Встановлює внутрішні відступи для `VBox`.
* **Insets:** Вказує внутрішні відступи: 7 пікселів знизу, 5 пікселів зліва, 8 пікселів справа і 5 пікселів зверху.

PlayerComparator

package GameLogic;  
  
import java.util.Comparator;  
  
  
public class PlayerComparator implements Comparator<Player> {  
  
 @Override  
 public int compare(Player arg0, Player arg1) {  
 if (!arg0.isOnline()) return 1;  
 if (!arg1.isOnline()) return -1;  
 int flag = arg1.getMoneyAmount() - arg0.getMoneyAmount();  
 if (flag == 0) {  
 return arg0.getPlayerNum() - arg1.getPlayerNum();  
 } else {  
 return flag;  
 }  
 }  
  
}

**Логіка порівняння**

if (!arg0.isOnline()) return 1; if (!ar

Клас **PlayerComparator** використовується для сортування гравців у списку за наступними критеріями:

1. Гравці, які онлайн, мають пріоритет над офлайн гравцями.
2. Якщо обидва гравці онлайн або офлайн, вони порівнюються за кількістю грошей.
3. Якщо кількість грошей однакова, вони порівнюються за своїм номером гравця (**PlayerNum**).

Game

private boolean initNewGame() {  
 // Init Map  
 this.squareSet = new HashMap<Integer, Cell>();  
 for (MonopolyMap mapinfo : MonopolyMap.*values*()) {  
 switch (mapinfo.getType()) {  
 case *PROPERTY*:  
 squareSet.put(mapinfo.getPosition(), new Property(mapinfo.getType(), mapinfo.getName(), mapinfo.getPosition(), null, mapinfo.getPrice(), mapinfo.getRent()));  
 break;  
 default:  
 squareSet.put(mapinfo.getPosition(), new Cell(mapinfo.getType(), mapinfo.getName(), mapinfo.getPosition(), null));  
 break;  
 }  
 }  
 for (int i = 1; i <= LASTSQUARE; i++) {  
 squareSet.get(i).setNext(squareSet.get((i + 1) % (LASTSQUARE + 1)));  
 }  
 squareSet.get(LASTSQUARE).setNext(squareSet.get(1));  
 squareSet.get(JAILPOSITION).setNext(squareSet.get(JAILNEXT));  
 // Init Players  
 this.playerList = new ArrayList<Player>();  
 this.numberOfPlayers = UI.askNumberOfPlayers(MINPLAYERS, MAXPLAYERS);  
 List<Colors> color = new ArrayList<Colors>();  
 color.add(Colors.*RED*);  
 color.add(Colors.*BLUE*);  
 color.add(Colors.*YELLOW*);  
 color.add(Colors.*GREEN*);  
 color.add(Colors.*PURPLE*);  
 color.add(Colors.*BLACK*);  
 String playerName;  
 for (int i = 0; i < this.numberOfPlayers; i++) {  
 playerName = UI.askPlayerName(i + 1);  
 playerList.add(new Player(playerName, color.get(i), squareSet.get(1)));  
 }  
 // Init parameters  
 this.round = 1;  
 this.roundStep = 0;  
 this.currentPlayer = playerList.get(0);  
 return true;  
}

* **Ініціалізація карти**: Створюється **HashMap** для зберігання всіх клітинок ігрового поля. Для кожного елемента **MonopolyMap** визначаються його тип і позиція, і відповідно створюється або **Property**, або загальна клітинка **Cell**. Всі клітинки з'єднуються в циклічний список.
* **Ініціалізація** **гравців**: Створюється список гравців **playerList** та визначається кількість гравців за допомогою інтерфейсу користувача. Далі, для кожного гравця запитується його ім'я і встановлюється колір, після чого гравці додаються до списку і розміщуються на стартовій позиції.
* **Ініціалізація** **параметрів**: Встановлюються початкові значення для параметрів гри, таких як номер раунду (**round**), крок раунду (**roundStep**), та поточний гравець (**currentPlayer**).
* public Game(UserInterface UI) {  
   this.UI = UI;  
  }  
    
  @Override  
  public void run() {  
   boolean initSuccessfully = this.initNewGame();  
   if (!initSuccessfully) {  
   UI.notify("Failed to initialize new game.");  
   return;  
   }  
   runGame();  
  }
* **Конструктор**: Зберігає інтерфейс користувача **UI**.
* **Метод run**: Запускає гру. Спочатку викликається метод **initNewGame** для ініціалізації нової гри. Якщо ініціалізація не вдалася, виводиться повідомлення про помилку. Якщо все успішно, гра починає свою роботу методом **runGame**.

GUI

Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описание

Рис.8 – Діаграма блоку GUI

@Override  
public void playerMove(Cell beg, Cell end, int steps) {  
 this.displayMessage("Move " + steps + " steps.");  
 waitForRequestRefresh();  
}

**Логіка фукнції:**

Відображає повідомлення про кількість кроків, які повинен зробити гравець, і чекає на оновлення запиту.

@Override  
public void showPlayerInfo(Player player) {  
 this.displayMessage("Now player #" + player.getPlayerNum() + " " + player.getName() + " is at #" + player.getCoordinates().getCoord() + " " + player.getCoordinates().getName() + ".");  
}

**Логіка фукнції:**

Відображає інформацію про поточного гравця, включаючи його номер, ім'я та місце розташування.

@Override  
public void displayResult(List<Player> playerList) {  
 waitForRequestRefresh();  
}

**Логіка фукнції:**

Чекає на оновлення запиту для відображення результатів гри.

@Override  
public int askNumberOfPlayers(int MIN, int MAX) {  
 final int ID = Request.*AskNumberOfPlayers*.getID();  
 int s = 0;  
 synchronized (this) {  
 try {  
 while (reply[ID].equals(-1)) {  
 this.gameStage.setRequests(ID, true);  
 this.wait();  
 }  
 if (reply[ID] instanceof Integer) s = (Integer) reply[ID];  
 else s = 0;  
 this.notifyAll();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println("Number of players DONE. - " + s);  
 reply[ID] = -1;  
 return s;  
}

**Логіка фукнції:**

Запитує у користувача кількість гравців для гри, чекає відповіді та повертає її.

@Override  
public String askPlayerName(int playerNumber) {  
 final int ID = Request.*AskPlayerName*.getID();  
 String s = "null";  
 synchronized (this) {  
 try {  
 while (reply[ID].equals(-1)) {  
 this.gameStage.setRequests(ID, playerNumber);  
 this.wait();  
 }  
 if (reply[ID] instanceof String) s = (String) reply[ID];  
 else s = "null";  
 this.notifyAll();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 if (s.length() == 0) s = "Player " + playerNumber;  
 System.*out*.println("Name of player " + playerNumber + " DONE. - " + s);  
 reply[ID] = -1;  
 return s;  
}

**Логіка фукнції:**

Запитує у користувача ім'я для кожного гравця, чекає відповіді та повертає її.

@Override  
public boolean askForBuying(Property property) {  
 final int ID = Request.*AskForBuying*.getID();  
 boolean s = false;  
 reply[ID] = -1;  
  
 synchronized (this) {  
 try {  
 while (reply[ID].equals(-1)) {  
 this.gameStage.setRequests(ID, true);  
 this.wait();  
 }  
 s = reply[ID].equals(1);  
 this.notifyAll();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println("Ask for buying DONE. - " + s);  
 reply[ID] = -1;  
 return s;  
}

**Логіка фукнції:**

Запитує у користувача, чи хоче він купити певну власність, чекає відповіді та повертає результат.

@Override  
public boolean askPayFine() {  
 final int ID = Request.*AskPayFine*.getID();  
 boolean s = false;  
 reply[ID] = -1;  
  
 synchronized (this) {  
 try {  
 while (reply[ID].equals(-1)) {  
 this.gameStage.setRequests(ID, true);  
 this.wait();  
 }  
 s = reply[ID].equals(1);  
 this.notifyAll();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println("Ask pay fine DONE. - " + s);  
 reply[ID] = -1;  
 return s;  
}

**Логіка фукнції:**

Запитує у користувача, чи хоче він сплатити штраф, чекає відповіді та повертає результат.

@Override  
public int step1ChooseOperation() {  
 final int ID = Request.*StepOne*.getID();  
 int s = 1;  
 reply[ID] = -1;  
  
 synchronized (this) {  
 try {  
 while (reply[ID].equals(-1)) {  
 this.gameStage.setRequests(ID, true);  
 this.wait();  
 }  
 if (reply[ID] instanceof Integer) s = (Integer) reply[ID];  
 else s = 1;  
 this.notifyAll();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println("Step one DONE. - " + s);  
 reply[ID] = -1;  
 return s;  
}

**Логіка фукнції:**

Запитує у користувача, яку операцію він хоче виконати на першому кроці, чекає відповіді та повертає результат.

public void setReply(int index, Object value) {  
 synchronized (this) {  
 if (index < NUMBEROFREPLIES) {  
 reply[index] = value;  
 }  
 this.notifyAll();  
 }  
}

**Логіка фукнції:**

Встановлює відповідь для запиту на основі індексу та значення.

public GUI(MonopolyStage gameStage) {  
 this.gameStage = gameStage;  
 this.reply = new Object[NUMBEROFREPLIES];  
 Arrays.*fill*(reply, -1);  
}

**Логіка фукнції:**

Ініціалізує об'єкт GUI з заданим ігровим етапом та масивом відповідей.

**ВИСНОВОК:**

У рамках цієї курсової роботи було розроблено гру "Монополія" на мові програмування Java із використанням технологій JavaFX для створення графічного інтерфейсу користувача. Виконана робота включає декілька ключових аспектів:

**Розробка графічного інтерфейсу (GUI):**

* Створено інтуїтивно зрозумілий графічний інтерфейс користувача, який включає основне ігрове поле, панелі інформації про гравців, кнопки для взаємодії та інші інтерактивні елементи.
* Використання FXML дозволило ефективно організувати структуру інтерфейсу, спростивши процес його розробки та підтримки.

**Ініціалізація ігрового поля:**

* Ігрове поле було ініціалізовано відповідно до стандартних правил гри "Монополія". Кожна клітинка на полі отримала свою функціональність (власність, в'язниця тощо).
* Розроблено логіку для коректного пересування гравців по ігровому полю.

**Ініціалізація гравців:**

* Реалізовано процес ініціалізації гравців, включаючи запит щодо кількості гравців, введення імен та призначення унікальних кольорів.
* Забезпечено зберігання всіх необхідних параметрів гравців.

**Реалізація логіки гри:**

* Розроблено основну логіку гри, включаючи механіку ходів, обробку подій на клітинках та управління чергою гравців.
* Запроваджено систему для визначення поточного гравця та обробки спеціальних умов, таких як потрапляння до в'язниці та купівля власності.

**Розширюваність та підтримка:**

* Проект організовано з урахуванням об'єктно-орієнтованих принципів програмування, що забезпечує легку розширюваність та підтримку.
* Код є добре структурованим та задокументованим, що полегшує його читання та розвиток.

**Тестування та налагодження:**

* Проведено ретельне тестування всіх компонентів гри, включаючи перевірку коректності ініціалізації, роботи з інтерфейсом та основної логіки гри.
* Виявлено та виправлено помилки, що забезпечило стабільну роботу програми.

Результати роботи демонструють успішну реалізацію гри "Монополія" з використанням сучасних технологій програмування. Це дозволило поглибити знання у розробці ігор, об'єктно-орієнтованому програмуванні та створенні графічних інтерфейсів, а також отримати цінний практичний досвід у проектуванні та реалізації складних програмних систем.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:**

1. Ulla, Paul. "Getting Started with FXML." Oracle Documentation. URL: <https://docs.oracle.com/javase/8/javafx/get-started-tutorial/fxml_tutorial.htm> (дата звернення: 01.05.2024).
2. Oracle. "Using FXML to Create a User Interface in JavaFX." Oracle Documentation. URL: <https://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm> (дата звернення: 04.05.2024).
3. Oracle. "The Java Tutorials." Oracle Documentation. URL: <https://docs.oracle.com/javase/tutorial/> (дата звернення: 07.05.2024).
4. GeeksforGeeks. "Java Programming Language." GeeksforGeeks. URL: <https://www.geeksforgeeks.org/java/> (дата звернення: 09.05.2024).
5. Oracle. "Java Platform, Standard Edition Documentation." Oracle Documentation. URL: <https://docs.oracle.com/en/java/javase/14/> (дата звернення: 10.05.2024).
6. MDN Web Docs. "CSS: Cascading Style Sheets." Mozilla. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS> (дата звернення: 13.05.2024).
7. W3Schools. "CSS Tutorial." W3Schools. URL: <https://www.w3schools.com/css/> (дата звернення: 14.05.2024).
8. CSS-Tricks. "A Complete Guide to Flexbox." CSS-Tricks. URL: [https://css-tricks.com/snippets/css/a-guide-to-flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)/ (дата звернення: 15.05.2024).
9. CSS-Tricks. "CSS Basics." CSS-Tricks. URL: https://css-tricks.com/almanac/properties/a/align-items/ (дата звернення: 18.05.2024).
10. Baeldung. "A Guide to Java Annotations." Baeldung. URL: https://www.baeldung.com/java-annotations (дата звернення: 19.05.2024).
11. Oracle. "Annotations in Java." Oracle Documentation. URL: <https://docs.oracle.com/javase/tutorial/java/annotations/> (дата звернення: 20.05.2024).
12. Baeldung. "Custom Annotations in Java." Baeldung. URL: https://www.baeldung.com/java-custom-annotation (дата звернення: 21.05.2024).
13. Oracle. "JavaFX Button Example with FXML." Oracle Documentation. URL: <https://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm#AJI8C> (дата звернення: 28.05.2024).
14. Jenkov, Jakob. "JavaFX FXML Tutorial: Buttons." Jenkov. URL: [http://tutorials.jenkov.com/javafx/fxml.html#buttons](https://jenkov.com/tutorials/javafx/fxml.html) (дата звернення: 30.05.2024).