**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

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

**Навчально-науковий 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»

Бойко Михайло Максимович

Керівник: Булах Б.В.

Київ – 2024

**ЗМІСТ**

[**ВСТУП** 3](#_Toc168244895)

[**РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ** 4](#_Toc168244896)

[**1.1. Уточнена постановка задачі на розробку програмного забезпечення** 4](#_Toc168244897)

[**1.2. Обґрунтування вибору засобів розробки** 5](#_Toc168244898)

[**РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ТА ПОДІЛ РОЗРОБКИ ПРОЕКТУ** 6](#_Toc168244899)

[**2.1. Проектування інтерфейсу** 6](#_Toc168244900)

[**2.2. Розподіл класів та алгоритмів для розробки** 7](#_Toc168244901)

[**РОЗДІЛ 3. ОПИС РОЗРОБЛЕНИХ КЛАСІВ У ПРОЕКТІ** 9](#_Toc168244902)

[**3.1. Опис класів, полів та методів програми** 9](#_Toc168244903)

[**ВИСНОВКИ** 29](#_Toc168244904)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 30](#_Toc168244905)

# **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. Проектування інтерфейсу**

Основною метою проектування є створення зручного та привабливого інтерфейсу, який дозволить користувачеві легко взаємодіяти з грою. Ми з командою можемо виділити основні положення, які повинні бути присутніми під час розробки проекту:

1. **Розташування елементів.** Оптимальне розташування ігрових компонентів на екрані забезпечить зручність користувача. Для гри “Монополія” важливо, щоб ігрове поле (ігрова дошка, гроші, власні ділянки гравців) було чітко розміщене, зручно використовувати та легко розпізнавати. Наприклад, гроші, інформація про гравців, інформація що відбувається у грі будуть розташовані в правій частині екрану, а ігрова дошка - у лівій частині.

2. **Дизайн інтерфейсу.** Важливо створити привабливий дизайн інтерфейсу, що стимулюватиме користувача до взаємодії з грою. Встановлення чіткого шрифту для тексту та вибір кольорової палітри для фону та елементів дозволить створити зручне та естетичне середовище для гри.

3. **Функціональність кнопок.** Кожна кнопка на інтерфейсі повинна мати визначену функціональність, яка відповідає логіці гри. Наприклад, кнопка “Roll Dice” повинна запускати процес підкидання гравця, а кнопка “Buy” - дозволяти гравцю придбати ділянку.

4. **Реакція інтерфейсу на дії користувача.** Інтерфейс повинен реагувати на дії користувача згідно з логікою гри. Наприклад, якщо користувач натискає на кнопку “Buy” для купівлі ділянки, то вона повинна підсвічуватися або відображати підтвердження операції.

5. **Тестування інтерфейсу.** Перед запуском гри важливо провести тестування інтерфейсу для виявлення будь-яких недоліків чи незручностей у взаємодії з користувачем. Тестування дозволить виправити всі можливі проблеми та забезпечити оптимальний досвід гри для кожного користувача.

Ретельне проектування інтерфейсу гри “Монополія” дозволить забезпечити зручну та привабливу взаємодію з грою, що підвищить задоволення користувачів від гри.

## **2.2. Розподіл класів та алгоритмів для розробки**

У нашому проекті мною було реалізовано такі класи: Cell, Player, Property, Game, MonopolyStage та GUI. Розглянемо детальніше кожен з них:

**Клас “Cell” :** Потрібно описати одну клітинку на полі для гри Монополія, а саме її тип який описується за допомогою enum – наприклад може бути клітинка в’язниці, шансу чи власності. Також у кожного поля має бути своя назва, координати на дошці та посилання на клітинку яка йде після неї для зручності ітерації по дошці.

**Клас “Player” :** Потрібно описати гравця, який буде задіяний у при виконанні програми та матиме наступні характеристики: статус – чи гравець наразі вибув з гри чи ні, ім’я гравця, кількість грошей на поточний момент часу, поле классу клітинки дошки яке вказує де зараз перебуває гравець, кількість днів у тюрмі для того щоб розуміти гравець здійснює переміщення по полю на даний момент часу чи ні, номер гравця, булеву змінну для перевірки чи виконується поточний зод автоматично чи рішення приймається вручну даним гравцем та змінна типу enum який визначає колір.

**Клас “Property” :** Потрібно описати власність, яка розміщується на клітинці поля, наприклад певну вулицю. Кожна власність має такі властивості як ціна за яку її можна придбати, рента яка буде зніматися з гравців що попадуть на дане поле та поточний власник.

**Клас “Game” :** Потрібно сприектувати метод runGame() який служить для завершення гри та виконує два основні завдання. По-перше, він сповіщає користувача про завершення відображає відповідне повідомлення на інтерфейсі користувача. По-друге, метод показує підсумкову таблицю з результатами гри, зокрема рахунки гравців. Цей метод корисний, оскільки він забезпечує чистий та зрозумілий спосіб завершити гру, надавши гравцям необхідну інформацію про її підсумки. Крім того, виклик `gameOver()` у кінці методу `runGame()` гарантує, що завершення гри обробляється централізовано і всі необхідні дії виконуються послідовно.

**Клас “MonopolyStage” :** Потрібно сттворити клас який є основним для графічного інтерфейсу гри Monopoly. Він відповідає за налаштування і відображення вікон, обробку запитів від гравців та оновлення графічного інтерфейсу. Він містить методи для відображення повідомлень, запитів гравців щодо різних. Перечислення `Request` визначає типи запитів, які можуть виникати під час гри. Кожен запит має унікальний ідентифікатор (ID), що полегшує обробку запитів у класі `MonopolyStage`. Ці запити включають різні дії, такі як відображення повідомлення, запит на нову гру або завантаження гри, запит кількості гравців, введення імені гравця, збереження гри, кидання кубиків та інші дії. Комбінація класу `MonopolyStage` та перерахування `Request` забезпечує ефективну взаємодію між користувачем та грою, дозволяючи гравцям легко виконувати необхідні дії, отримувати інформацію та оновлювати стан гри.

**Клас “GUI”** : потрібно створити клас для графічного інтерфейсу користувача в грі Monopoly. Він обробляє повідомлення від гри через методи `notify`, `inform` та `displayMessage`, використовуючи об'єкт `MonopolyStage` для оновлення стану гри. Масив `reply` синхронізує дії між компонентами. Метод `notify` показує повідомлення користувачеві, аналогічно працює метод `displayMessage`. Метод `playerUpdateMoney` оновлює інформацію про гроші гравця. Метод `changePropertyOwner` оновлює інформацію про власника нерухомості та перезапускає інтерфейс. Метод `playerGotoJail` обробляє ситуацію, коли гравець потрапляє до в'язниці. Метод `rollDice` відображає результати кидка кубиків. Всі методи використовують синхронізацію для коректного оновлення інтерфейсу, запобігаючи конфліктам. Клас `GUI` забезпечує ефективне управління графічним інтерфейсом користувача в грі Monopoly.

**РОЗДІЛ 3. ОПИС РОЗРОБЛЕНИХ КЛАСІВ У ПРОЕКТІ**

## **3.1. Опис класів, полів та методів програми**

У цьому розділі буде описано поля та методи таких класів, як Cell, Player, Property, Game, MonopolyStage та GUI. Крім того, наведено скріншоти, фрагменти коду та таблички, що ілюструють певні аспекти реалізації.

**Клас “** **Cell ”** **:**

Таблиця 1 – Опис полів класу “Cell”

|  |  |  |
| --- | --- | --- |
| № | Прототип поля | Призначення поля |
| 1 | private Type type; | Тип комірки, що визначає її роль у грі (наприклад, стартова, нерухомість, подія тощо). |
| 2 | private String name; | Назва комірки, яка відображає її призначення або назву (наприклад, назва вулиці). |
| 3 | private int coordinates; | Координати комірки на ігровому полі, що визначають її розташування. |
| 4 | private Cell next; | Посилання на наступну комірку, що використовується для побудови зв'язного списку комірок на ігровому полі. |

Таблиця 2 – Опис методів класу “Cell”

|  |  |  |
| --- | --- | --- |
| № | Прототип методів | Призначення методу |
| 1 | public Type getType(); | Повертає тип комірки. |
| 2 | public String getName(); | Повертає назву комірки |
| 3 | public int getCoord(); | Повертає координати комірки на ігровому полі |
| 4 | public Cell getNext(); | Повертає посилання на наступну комірку |
| 5 | public void setNext(Cell next); | Встановлює посилання на наступну комірку |

**Клас “** **Player ”** **:**

Таблиця 3 – Опис полів класу “Player”

|  |  |  |
| --- | --- | --- |
| № | Прототип поля | Призначення поля |
| 1 | private boolean status; | Вказує, чи гравець знаходиться у грі (true), чи вийшов з гри або програв (false) |
| 2 | private String name; | Ім'я гравця |
| 3 | private int money; | Кількість грошей у гравця |
| 4 | private Cell location; | Поточне місцезнаходження гравця (комірка на ігровому полі) |
| 5 | private int jailDays; | Кількість днів, які гравець проведе у в'язниці |
| 6 | private int playerNo; | Номер гравця |
| 7 | private boolean auto; | Вказує, чи гравець керується автоматично (true), чи ні (false) |
| 8 | private Colors color; | Колір, яким позначено гравця |
| 9 | private final int ORIGINALMONEY = 2000; | Початкова кількість грошей у гравця |

Таблиця 4 – Опис методів класу “Player”

|  |  |  |
| --- | --- | --- |
| № | Прототип методу | Призначення методу |
| 1 | public String getName(); | Повертає ім'я гравця |
| 2 | public int getMoneyAmount(); | Повертає кількість грошей у гравця |
| 3 | public Cell getCoordiantes(); | Повертає поточну комірку (координати) гравця |
| 4 | public int getPlayerNum(); | Повертає номер гравця |
| 5 | public String getColor(); | Повертає колір гравця як назву кольору |
| 6 | public int getJailDays(); | Повертає кількість днів, які гравець проведе у в'язниці |

|  |  |  |
| --- | --- | --- |
| 7 | public boolean isOnline(); | Повертає статус гравця (онлайн чи офлайн) |
| 8 | public boolean isInJail(); | Повертає true, якщо гравець у в'язниці, і false інакше |
| 9 | public boolean isAuto(); | Повертає true, якщо гравець керується автоматично |
| 10 | public void updateMoney(int amount); | Оновлює кількість грошей гравця, додаючи вказану суму |
| 11 | public void setLocation(Cell location); | Встановлює нову поточну комірку для гравця |
| 12 | public void setJailDays(int jailDays); | Встановлює кількість днів, які гравець проведе у в'язниці |
| 13 | public void quitGame(); | Змінює статус гравця на офлайн, показуючи, що він вийшов з гри |

**Клас “** **Property ”** **:**

Таблиця 5 – Опис полів класу “Property”

|  |  |  |
| --- | --- | --- |
| № | Прототип поля | Призначення поля |
| 1 | private int price; | Ціна нерухомості |
| 2 | private int rent; | Розмір орендної плати за нерухомість |
| 3 | private Player owner; | Власник нерухомості, якщо є |

Таблиця 6 – Опис методів класу “Property”

|  |  |  |
| --- | --- | --- |
| № | Прототип методів | Призначення методу |
| 1 | public int getPrice(); | Повертає ціну нерухомості |
| 2 | public int getRent(); | Повертає розмір орендної плати за нерухомість |
| 3 | public Player getOwner(); | Повертає власника нерухомості, якщо він є |
| 4 | public void setOwner(Player owner); | становлює власника нерухомості |

**Клас “** **Game ”** **:**

У класі Game мною було реалізовано метод runGame(), лістинг якого наведено нижче:

public void runGame() {  
 int userChoice;  
 int[] dice;  
 while (!this.isGameEnds()) {  
 UI.displayMessage("\nRound " + this.round);  
 UI.showPlayerInfo(currentPlayer);  
  
 /\* Round Step 0 \*/  
 // [1] Continue [2] Report [3] Set Auto [4] Retire  
 if (roundStep == 0) { // Удалите условие currentPlayer.isAuto()  
 userChoice = UI.step1ChooseOperation();  
 switch (userChoice) {  
 case 2:  
 this.showScoreBoard();  
 continue;  
 case 3:  
 // Удалите функционал автопилота  
 // currentPlayer.setAuto(true);  
 break;  
 case 4:  
 currentPlayer.quitGame();  
 this.checkPlayerOut(currentPlayer);  
 UI.notify("Player " + currentPlayer.getName() + " quits game.");  
 this.setNextPlayer();  
 continue;  
 default:  
 break;  
 }  
 roundStep++;  
 }  
  
 /\* Round Step 1 \*/  
 // roll dice and move  
 if (roundStep == 1) {  
 dice = this.rollDice();  
 // If player is in jail  
 if (currentPlayer.isInJail()) {  
 UI.displayMessage(currentPlayer.getName() + " is in jail (" + currentPlayer.getJailDays() + " days left).");  
 if (dice[0] == dice[1]) {  
 currentPlayer.setJailDays(0);  
 } else if (this.askPayFine()) {  
 if (currentPlayer.getMoneyAmount() < FINE) {  
 UI.notify("You can't pay fine because your money is less than 90.");  
 } else {  
 currentPlayer.setJailDays(0);  
 currentPlayer.updateMoney(-FINE);  
 UI.playerUpdateMoney(currentPlayer, -FINE);  
 }  
 }  
 if (currentPlayer.getJailDays() == 1) {  
 UI.displayMessage(currentPlayer.getName() + " has to pay fine on the last day.");  
 currentPlayer.updateMoney(-FINE);  
 UI.playerUpdateMoney(currentPlayer, -FINE);  
 currentPlayer.setJailDays(0);  
 if (this.checkPlayerOut(currentPlayer)) {  
 this.setNextPlayer();  
 continue;  
 }  
 }  
 }  
 // Not in jail / just out of jail  
 if (!currentPlayer.isInJail()) {  
 this.moveCurrentPlayer(dice[0] + dice[1]);  
 }  
 roundStep++;  
 }  
  
 /\* Round Step 2 \*/  
 // Check the new square  
 if (roundStep == 2) {  
 switch (currentPlayer.getCoordiantes().getType()) {  
 case *TAX*:  
 int tax = (((currentPlayer.getMoneyAmount() + ROUND) / TAXDIV) + ROUND) / TAXDIV \* TAXDIV;  
 UI.displayMessage("Player " + currentPlayer.getName() + " should pay tax of HKD" + tax + ".");  
 currentPlayer.updateMoney(-tax);  
 UI.playerUpdateMoney(currentPlayer, -tax);  
 break;  
 case *JAIL*:  
 if (currentPlayer.isInJail()) currentPlayer.setJailDays(currentPlayer.getJailDays() - 1);  
 break;  
 case *CHANCE*:  
 int chanceMoney = this.randomChance();  
 UI.displayMessage("Player " + currentPlayer.getName() + " gets a chance!");  
 currentPlayer.updateMoney(chanceMoney);  
 UI.playerUpdateMoney(currentPlayer, chanceMoney);  
 if (this.checkPlayerOut(currentPlayer)) {  
 this.setNextPlayer();  
 continue;  
 }  
 break;  
 case *TO\_JAIL*:  
 UI.displayMessage("Player " + currentPlayer.getName() + " goes to jail.");  
 currentPlayer.setJailDays(JAILDAYS);  
 currentPlayer.setLocation(squareSet.get(JAILPOSITION));  
 UI.playerGotoJail(currentPlayer);  
 break;  
 case *PROPERTY*:  
 Property curProperty = (Property) currentPlayer.getCoordiantes();  
 UI.displayMessage("Player " + currentPlayer.getName() + " arrives at " + curProperty.getName() + " (Price:" + curProperty.getPrice() + ", Rent:" + curProperty.getRent() + ").");  
 if (curProperty.getOwner() != null && curProperty.getOwner() != currentPlayer) {  
 UI.displayMessage("This property belongs to " + curProperty.getOwner().getName() + ". " + currentPlayer.getName() + " should pay HKD " + curProperty.getRent() + ".");  
 currentPlayer.updateMoney(-curProperty.getRent());  
 curProperty.getOwner().updateMoney(curProperty.getRent());  
 UI.playerUpdateMoney(currentPlayer, -curProperty.getRent());  
 UI.playerUpdateMoney(curProperty.getOwner(), curProperty.getRent());  
 if (this.checkPlayerOut(currentPlayer)) {  
 this.setNextPlayer();  
 continue;  
 }  
 } else if (curProperty.getOwner() == null) {  
 boolean buy;  
  
 buy = UI.askForBuying(curProperty);  
 if (buy) {  
 if (currentPlayer.getMoneyAmount() > curProperty.getPrice()) {  
 UI.changePropertyOwner(curProperty, currentPlayer);  
 UI.playerUpdateMoney(currentPlayer, -curProperty.getPrice());  
 curProperty.setOwner(currentPlayer);  
 currentPlayer.updateMoney(-curProperty.getPrice());  
 } else {  
 UI.displayMessage("Failed because money is not enough.");  
 }  
 }  
 } else {  
 UI.displayMessage("Nothing happens.");  
 }  
 break;  
 default:  
 break;  
 }  
 roundStep = 0;  
 }  
 UI.showPlayerInfo(currentPlayer);  
 this.setNextPlayer();  
 }  
 this.gameOver();  
}

Метод `runGame` відповідає за виконання основного циклу гри та контроль за ходом подій.

**1.** Початок циклу гри:

- Цикл продовжується, поки гра не завершиться (поки `isGameEnds()` повертає `false`).

**2.** Оголошення змінних:

- Оголошуються локальні змінні `userChoice` для вибору користувача та `dice` для результатів кидка кубиків.

**3.** Виведення інформації про раунд та гравця:

- Викликається метод `displayMessage` для відображення інформації про поточний раунд та гравця за допомогою `UI`.

**4. Етап 0:** Обробка вибору гравця:

- Гравець обирає операцію для виконання: продовжити гру, показати звіт, вийти з гри тощо.

- Залежно від вибору гравця виконуються відповідні дії, такі як показ звіту чи вихід з гри.

**5. Етап 1:** Кидок кубиків та рух гравця:

- Гравець кидає кубики.

- Якщо гравець у в'язниці, розглядається можливість виплати штрафу або виходу з в'язниці.

- Гравець рухається на відповідну кількість комірок на дошці.

**6. Етап 2:** Перевірка нової комірки:

- Перевіряється тип нової комірки та виконуються відповідні дії: сплатити податки, перейти в в'язницю, отримати шанс, купити нерухомість тощо.

- Грошові та ігрові зміни відображаються за допомогою `UI`.

**7.** Оновлення стану гри:

- Виводиться інформація про гравця.

- Перехід до наступного гравця.

**8.** Завершення гри:

- По закінченню гри викликається метод `gameOver()`.

Цей метод забезпечує керування головним алгоритмом гри, включаючи обробку вибору гравця, рух по дошці, взаємодію з комірками та оновленням стану гравців.

**Клас “** **MonopolyStage ”** **:**

Таблиця 9 – Опис полів класу “MonopolyStage”

|  |  |  |
| --- | --- | --- |
| № | Прототип поля | Призначення поля |
| 1 | private final int WindowWIDTH = 960; | Ширина вікна гри |
| 2 | private final int WindowHEIGHT=720; | Висота вікна гри |
| 3 | private final int PLAYERS = 20; | Максимальна кількість гравців у грі |
| 4 | private ArrayList<Node> playerIcon; | Список графічних об'єктів (ImageView), що представляють гравців |
| 5 | private ArrayList<ImageView> playerLocation; | Список іконок гравців на полі гри |
| 6 | private ArrayList<Label> playerName; | Список міток імен гравців |
| 7 | private ArrayList<Label> playerMoney; | Список міток грошей гравців |
| 8 | private Map<Integer,Button> ownerDisplay; | Мапа кнопок для відображення власності комірки |
| 9 | private Map<Integer,FlowPane> squarePane; | Мапа об'єктів FlowPane, що представляють кожну комірку гри |
| 10 | private Controller controller; | Екземпляр контролера гри |
| 11 | private GUI gui; | Об'єкт графічного інтерфейсу користувача |
| 12 | private Parent root; | Батьківський вузол сцени гри |
| 13 | private Stage primaryStage; | Головна сцена гри |
| 14 | private Thread gameThread; | Потік гри |
| 15 | private Game game; | Об'єкт гри |
| 16 | private Object[] requests; | Масив запитів |
| 17 | private final int SQUARES = 6; | Кількість комірок на дошці гри |

Таблиця 10 – Опис методів класу “MonopolyStage”

|  |  |  |
| --- | --- | --- |
| № | Прототип методів | Призначення методу |
| 1 | public void simpleNoticeWindow(String msg); | Відображає просте вікно повідомлення з одним кнопковим елементом "OK" |
| 2 | public void showNotice(); | Викликає метод simpleNoticeWindow для відображення повідомлення |
| 3 | public void displayMessage(); | Відображає повідомлення в області повідомлень |
| 4 | public void askNewOrLoadGame(); | Відображає діалогове вікно для вибору нової гри або завантаження попередньої |
| 5 | public void askNumberOfPlayers(); | Відображає діалогове вікно для вибору кількості гравців |
| 6 | public void askPlayerName(); | Відображає вікно для введення імені гравця |
| 7 | private String simpleInputWindow(String labelText,String title); | Відображає вікно введення з міткою та заголовком |
| 8 | public void loadGame(); | Викликає simpleInputWindow для введення імені файлу гри для завантаження |
| 9 | public void saveGame(); | Викликає simpleInputWindow для введення імені файлу гри для збереження |
| 10 | public boolean stepOne(); | Перша крок гри, що включає головне вікно та кнопки |
| 11 | public void rollDice(); | Кидає кубики та оновлює відповідь у GUI |
| 12 | public boolean askForBuying(); | Питає гравця, чи хоче він купити комірку |
| 13 | public boolean askPayFine(); | Питає гравця, чи хоче він сплатити штраф |
| 14 | private void refreshPad(); | Оновлює відображення комірок, гравців та власностей |
| 15 | public void checkRequests(); | Перевіряє наявність запитів та виконує їх обробку |
| 16 | private boolean initWidgets(Parent root); | Ініціалізує віджети графічного інтерфейсу |
| 17 | public void start(Stage primaryStage); | Запускає головний метод гри |
| 18 | public void init(); | Ініціалізує графічний інтерфейс та гру |
| 19 | public static void main(String[] args); | Встановлює значення запиту за індексом |
| 20 | public void setRequests(int index, Object value); | Повертає інтерфейс користувача |
| 21 | public GUI getGUIinterface(); | Точка входу у програму |

**Клас “** **GUI ”** **:**

Таблиця 11 – Опис полів класу “GUI”

|  |  |  |
| --- | --- | --- |
| № | Прототип поля | Призначення поля |
| 1 | private MonopolyStage gameStage; | Поле, що зберігає об'єкт типу MonopolyStage, який представляє собою сцену гри Monopoly |
| 2 | private final int NUMBEROFREPLIES = 20; | Кількість відповідей, що зберігаються у масиві reply. Значення є фіксованим та дорівнює 20 |
| 3 | private Object[] reply; | Масив об'єктів, який зберігає відповіді на запити для синхронізації дій між різними компонентами гри. Кожен елемент масиву відповідає конкретному запиту та містить об'єкт, що представляє відповідь на цей запит |

Таблиця 12 – Опис методів класу “GUI”

|  |  |  |
| --- | --- | --- |
| № | Прототип методів | Призначення методу |
| 1 | public void notify(String msg) | Метод, який виводить повідомлення для користувача. Виконується у потоці гри та синхронізується з масивом відповідей. Відображає повідомлення та очікує, поки воно буде оброблене |
| 2 | public void inform(String msg) | Метод, який призначений для інформування користувача. Залишений без реалізації |
| 3 | public void displayMessage(String msg) | Метод, що відображає повідомлення на екрані гри. Синхронізований з масивом відповідей. Відображає повідомлення та очікує, поки воно буде оброблене |
| 4 | public void playerUpdateMoney(Player player, int amount) | Метод для оновлення кількості грошей у гравця. Викликає метод displayMessage для відображення повідомлення про зміну кількості грошей |
| 5 | public void changePropertyOwner(Property property, Player player) | Метод, що вказує, що власником властивості тепер є певний гравець. Викликає displayMessage для відображення цього факту та refresh для оновлення інтерфейсу |
| 6 | public void playerGotoJail(Player player) | Метод, який вказує, що гравець потрапив у в'язницю. Викликає refresh для оновлення інтерфейсу |
| 7 | public void rollDice(int dice1, int dice2) | Метод, що відображає результати кидка кубиків на екрані гри. Синхронізований з масивом відповідей. Відображає результати та очікує їх оброблення |

# 

# **ВИСНОВКИ**

У ході виконання курсової роботи я з командою розробляв комп'ютерну гру “Монополія” з графічним інтерфейсом, використовуючи мову програмування Java, бібліотеку JavaFX та середовище розробки IntelliJ IDEA.

В ході розробки гри "Монополія" я створив кілька класів, кожен з яких відігравав важливу роль у функціональності та інтерфейсі гри.

Клас "Cell" відображав окрему клітинку на полі гри "Монополія". Кожна клітинка мала свій тип, описаний за допомогою enum (наприклад, "в'язниця", "шанс", "власність"), а також назву, координати на дошці та посилання на наступну клітинку для зручної ітерації по полю.

Клас "Player" представляв гравця гри з його характеристиками, такими як статус (чи гравець зараз вибув з гри), ім'я, кількість грошей, поточна клітинка, на якій знаходиться гравець, кількість днів у в'язниці, номер гравця, тип рішення (автоматично чи вручну) та колір гравця, визначений enum.

Клас "Property" відображав власність на клітинці поля, таку як певна вулиця. Кожна власність мала ціну, ренту та поточного власника.

Клас "Game" мав метод "runGame()", який завершував гру та показував підсумкову таблицю з результатами гри. Цей метод також сповіщав користувача про завершення гри.

Клас "MonopolyStage" відповідав за графічний інтерфейс гри "Монополія", налаштування вікон, обробку запитів від гравців та оновлення графічного інтерфейсу. Він містив методи для відображення повідомлень, запитів гравців та оновлення інтерфейсу.

Клас "GUI" слугував для управління графічним інтерфейсом користувача в грі "Монополія", обробки повідомлень від гри та оновлення стану інтерфейсу. Він забезпечував синхронізацію дій для коректного оновлення інтерфейсу та запобігав конфліктам.

Ці класи разом дозволили ефективно реалізувати функціонал гри "Монополія" з графічним інтерфейсом, надаючи користувачам можливість грати в класичну настільну гру на комп'ютері.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Інтернет-енциклопедія “Вікіпедія”. Monopoly (game)

URL: <https://en.wikipedia.org/wiki/Monopoly_(game)>

1. Cay S. Horstmann, "Core Java, Volume 1: Fundamentals, 11th Edition" Pearson Education, 2018

URL: <https://studylib.net/doc/26342828/core-java-volume-1--fundamentals--11th-edition--cay-s.-ho>

1. Інтернер-ресурс “Oracle”. JavaFX Documentation

URL: <https://openjfx.io>

1. Інтернер-ресурс “MoldStud”. The Role of Java in Game Development: Relevance for University Applicants

URL: <https://moldstud.com/articles/p-the-role-of-java-in-game-development-relevance-for-university-applicants>

1. Інтернер-ресурс “JavatPoint”. JavaFX Tutorial

URL: <https://www.javatpoint.com/javafx-tutorial>

1. Інтернер-ресурс “JUnits”. JUnit 4 Documentation

URL: <https://junit.org/junit4/>