**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»

Чупахін Микита Олексійович

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 мені потрібно реалізувати чотири класи Game, UserInterface, Tests та Controller. Розглянемо кожен клас, що я повинен реалізувати та їх роль у проекті:

**Клас “Game” :** Потрібно визначити поля клас та константи, що визначають параметри гри, такі як кількість квадратів на дошці, позиції тюрми, мінімальна та максимальна кількість гравців, список гравців у грі, мапа з квадратами на дошці гри та інші змінні, необхідні для ведення гри. Також потрібно ініціалізувати конструктори та створити методи для керування ходом гри, такі як отримання наступного гравця та оновлення поточного раунду.Крім того, потрібно реалізувати перерахування (enum), яке містить перелік різних локацій на дошці нашої гри. Кожна локація має своє унікальне ім'я, позицію на дошці, тип, ціну та орендну плату. Це дозволить створити структурований спосіб представлення різних локацій на дошці гри, що полегшить роботу з грою та надасть зручний доступ до інформації про кожну локацію.

**Інтерфейс “UserInterface” :** Потрібно описати поведінку (набір методів), яку будуть мати класи, що будуть реалазувати цей інтерфейс. У нашому випадку, методи для відображення різних аспектів гри, такі як кидання кубиків, переміщення гравця, покупка власності тощо. Також методи для повідомлення користувача про різні події в грі, наприклад, оголошення переможця, нагадування про сплату податків, запит вибору купівлі власності. В майбутньому цей інтерфейс повинен буде реалізовувати клас, який відповідає за графічний інтерфейс користувача або консольний.

**Клас “Tests” :** Потрібно реалізувати методи для тестування правильності роботи різних компонентів програми, наприклад, правильності сортування гравців тощо. Кожен тест повинен перевіряти конкретний аспект програми і гарантувати, що він працює правильно. Впевнетись у тому, що програма працює правильно та все виконується без багів через перевірку різних аспектів її роботи.

**Клас “Controller” :** Потрібно забезпечити правильну обробку подій натискання кнопок “Roll Dice”, “Ignore” та “Retire”, а також відображення повідомлень у текстовій області. Слід реалізувати логіку для оновлення стану гри залежно від натискання кнопок, зокрема зміни тексту кнопок після натискання. Також потрібно встановити та отримувати значення доступності кнопок, стану натиснення кнопок та графічних елементів, таких як зображення кубиків. Крім того, необхідно забезпечити можливість встановлення та отримання основної сцени та головного вікна програми, а також інтеграцію з основним ігровим класом “MonopolyStage” для керування логікою гри.

Також окрім реалізації класів, мені потрібно додати до проекту всі необхідні графічні елементи, такі як картинки полів гри, картинки кубиків та іконки гравців. Потрібно підібрати фон, картинку назви для монополії та табло. Ці графічні елементи відіграють важливу роль у створенні візуального інтерфейсу гри і допомагають користувачеві краще розуміти події та стан гри. Додатково вирішив оформити README файл для нашого проекту на гітхабі.

# **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 Game, UserInterface, Tests та enum MonopolyMap. Крім того, наведено скріншоти, фрагменти коду та таблички, що ілюструють певні аспекти реалізації.

**Клас “Game”** відповідає за управління самою грою “Монополія”. Усі поля та методи класу, які були реалізовані мною, описано в Таблиці 1 та Талиці 2 відповідно.

Таблиця 1 – Опис полів класу “Game”

|  |  |  |
| --- | --- | --- |
| № | Прототип поля | Призначення поля |
| 1 | private UserInterface UI; | Використовується для взаємодії з гравцями та відображення інформації на екрані |
| 2 | private final int FIRSTSQUARE = 1, LASTSQUARE = 20; | Визначають перше та останнє поле на дошці |
| 3 | private final int JAILPOSITION = 0, JAILNEXT = 7; | Визначають позиції на дошці, пов'язані з в'язницею |
| 4 | private final int MINPLAYERS = 2, MAXPLAYERS = 6; | Визначають мінімальну та максимальну кількість гравців |
| 5 | private final int FINE = 90, SALARY = 1500, TAXDIV = 10, ROUND = 5, JAILDAYS = 3,  GAMELENGTH = 100; | Визначають параметри гри, такі як величини штрафів, заробітних плат, податків, кількість раундів, кількість днів у в'язниці та тривалість гри |
| 6 | private int numberOfPlayers; | Зберігає кількість гравців, які беруть участь у грі |
| 7 | private List<Player> playerList; | Список об'єктів класу Player, який містить усіх гравців |
| 8 | private Map<Integer, Cell> squareSet; | Map, який відображає номер позиції на дошці гри в об'єкт класу Cell |
| 9 | private int round;  private int roundStep; | Змінні, яка відстежують поточний раунд гри та кількість кроків, зроблених у раунді |
| 10 | private Player currentPlayer; | Змінна, яка містить посилання на об'єкт класу Player, який зараз має право ходу |

Таблиця 2 – Опис методів класу “Game”

|  |  |  |
| --- | --- | --- |
| № | Прототип методів | Призначення методу |
| 1 | public List<Player> getPlayerList(); | Повертає список гравців, які беруть участь у грі |
| 2 | public Map<Integer, Cell> getSquareSet(); | Повертає колекцію, яка відображає номери позицій на дошці гри в об'єкти класу Cell, що представляють конкретні поля на дошці |
| 3 | private Player getNextPlayer(); | Визначає наступного гравця, який має право ходу |
| 4 | private void setNextPlayer(); | Встановлює наступного гравця, який має право ходу, та оновлює значення змінних, пов'язаних з раундом та кроком гри |
| 5 | private int getRandomNumber(int min, int max); | Генерує випадкове число в межах від min до max включно |
| 6 | private int[] rollDice(); | Симулює кидання двох гральних кубиків |
| 7 | private int randomChance(); | Генерує випадкове число, що представляє подію шансу у грі |
| 8 | private boolean isGameEnds(); | Перевіряє, чи закінчилась гра |
| 9 | private boolean askPayFine(); | Викликає відповідний метод з інтерфейсу користувача UI, запитуючи гравця, чи бажає він сплатити штраф |
| 10 | private boolean checkPlayerOut(Player player); | Перевіряє, чи повинен гравець вийти з гри |
| 11 | private void moveCurrentPlayer(int steps); | Переміщує поточного гравця на певну кількість кроків |
| 12 | public ArrayList getSortedPlayers(); | Повертає відсортований список гравців |
| 13 | private void showScoreBoard(); | Відображає результати гри |
| 14 | private void gameOver(); | Завершує гру, викликаючи метод notify з повідомленням про закінчення гри |

Діаграма класу Game з усіма полями та методами наведена на рис. 1.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1 – Diagram of class Game

Тепер давайте розглянемо реалізацію деяких основних методів. Почнемо з методу getNextPlayer(), код якого наведено на рис. 2.

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

Рисунок 2 – Код методу getNextPlayer()

Метод getNextPlayer() розпочинається з ініціалізації двох змінних: “i”, яка вказує на індекс поточного гравця в списку, і “foundCurrentPlayer”, яка слугує прапорцем для визначення, чи був знайдений поточний гравець. Після цього метод у циклі while перебирає гравців у списку. У цьому циклі відбувається перевірка, чи поточний гравець був знайдений. Якщо так, то встановлюється флаг “foundCurrentPlayer” на значення true. Після цього метод перевіряє, чи наступний гравець є онлайн. Якщо так, то цикл завершується і метод повертає наступного гравця. У разі, якщо наступний гравець не є онлайн, індекс “i” збільшується на одиницю, а потім залишається в межах допустимих значень списку за допомогою операції залишкового ділення за кількістю гравців у грі. Після цього цикл повторюється, і процес знову починається. Коли відбувається вихід з циклу, метод повертає наступного гравця, який має право ходу.

Тепер метод setNextPlayer(), реалізація якого наведена на рис. 3.

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

Рисунок 3 – Код методу setNextPlayer()

Метод setNextPlayer() встановлює наступного гравця для ходу в грі. Після виклику цього методу, поточний гравець змінюється на наступного в списку гравців. Якщо новий гравець має менший номер, ніж попередній, це означає, що раунд гри збільшується на одиницю.

Тепер метод rollDice(), реалізація якого наведена на рис. 4.

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

Рисунок 4 – Код методу rollDice()

Метод rollDice() визначає мінімальне (DICEMIN = 1) та максимальне (DICEMAX = 6) значення для кубиків. Далі, за допомогою методу getRandomNumber(), генерує два випадкові числа у зазначеному діапазоні, що представляють результати кидка двох кубиків. Ці результати передаються методу UI.rollDice для оновлення інтерфейсу користувача. Метод повертає масив з двох чисел, що представляють значення на кубиках, що були кинуті.

Тепер метод checkPlayerOut(), реалізація якого наведена на рис. 5.

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание**

Рисунок 5 – Код методу checkPlayerOut()

Метод checkPlayerOut(Player player) перевіряє, чи гравець знаходиться онлайн. Якщо ні, то гравець одразу вибуває. Далі, якщо баланс грошей гравця не перевищує нуля, метод проходить через всі клітинки ігрового поля і перевіряє, чи володіє гравець будь-якою власністю. Якщо так, власність повертається у власність гри, і метод UI.changePropertyOwner оновлює інтерфейс користувача. Після цього гравець виходить з гри через метод quitGame(), і користувач отримує повідомлення про це через UI.notify. Якщо гравець вибуває з гри, метод повертає true.

Тепер метод moveCurrentPlayer(), реалізація якого наведена на рис. 6.

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

Рисунок 6 – Код методу moveCurrentPlayer()

Метод moveCurrentPlayer(int steps) починає з визначення поточної позиції гравця. Далі, метод проходить через клітинки, пересуваючи гравця на вказану кількість кроків вперед. Якщо при цьому гравець проходить через клітинку "GO", йому нараховується зарплата, що відображається у вигляді повідомлення через UI.displayMessage і оновлення балансу через UI.playerUpdateMoney. Нарешті, метод оновлює позицію гравця на нову клітинку і викликає UI.playerMove, щоб показати переміщення в інтерфейсі користувача.

**Інтерфейс “UserInterface”** відповідає за взаємодію з користувачем та відображення графічного інтерфейсу гри. Усі методи інтерфейсу описано в Таблиці 3.

Таблиця 3 – Опис методів інтерфейсу “UserInterface”

|  |  |  |
| --- | --- | --- |
| № | Прототип методу | Призначення методу |
| 1 | void notify(String msg); | Відображення сповіщень у графічному інтерфейсі |
| 2 | void displayMessage(String msg); | Відображення текстових повідомлень в графічному інтерфейсі |
| 3 | void displayResult(List<Player> playerList); | Відображає результат гри, включаючи інформацію про гравців та їхній стан |
| 4 | void rollDice(int dice1, int dice2); | Відображає результати кидка гравцем кубиків |
| 5 | void playerMove(Cell beg, Cell end, int steps); | Відображає переміщення гравця від початкової до кінцевої клітинки з вказаною кількістю кроків |
| 6 | void changePropertyOwner(Property property, Player player); | Відображення події, коли гравець стає власником певної власності |

Продовження Таблиці 3

|  |  |  |
| --- | --- | --- |
| 7 | void playerGotoJail(Player player); | Повідомляє про те, що гравець потрапив у в'язницю |
| 8 | void showPlayerInfo(Player player); | Відображає інформацію про конкретного гравця |
| 9 | void playerUpdateMoney(Player player, int amount); | Оновлює інформацію про гроші у гравця після купівлі, продажу або оплати податків |
| 10 | int askNumberOfPlayers(int MIN, int MAX); | Запитує у користувача про кількість гравців у грі |
| 11 | String askPlayerName(int playerNumber); | Запитує у користувача ім'я для гравця певного номеру |
| 12 | boolean askForBuying(Property property); | Запитує у гравця, чи він хоче придбати певну власність |
| 13 | boolean askPayFine(); | Запитує у гравця, чи він готовий сплатити штраф |
| 14 | int step1ChooseOperation(); | Запитує у гравця про вибір операції на першому кроці гри |

Діаграма інтерфейсу UserInterface з усіма методами наведена на рис.7.

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание**

Рисунок 7 – Diagram of interface UserInterface

**Клас “Tests”** відповідає за тестування різних компонентів та функціоналу гри за допомогою JUnit [6]. Усі поля та методи класу, описано в Таблиці 4.

Таблиця 4 – Опис полів та методів класу “Tests”

|  |  |  |
| --- | --- | --- |
| № | Прототип методу або поля | Призначення поля або методу |
| 1 | private PlayerComparator comparator; | Використовується для порівняння об'єктів типу Player |
| 2 | public void setUp(); | Ініціалізує полe comparator |
| 3 | public void testCompare\_OnePlayerOnline(); | Перевіряє, чи правильно працює метод порівняння гравців, коли один з них вийшов з гри |
| 4 | public void testSortingPlayers(); | Перевіряє правильність сортування списку гравців за допомогою методу порівняння comparator |

Продовження Таблиці 4

|  |  |  |
| --- | --- | --- |
| 5 | public void testPlayer(); | Перевіряє правильність роботи методів і конструктора класу Player, включаючи оновлення балансу, виходу з гри та інші параметри гравця |
| 6 | public void testProperty() | Перевіряє правильність роботи методів і конструктора класy Property, зокрема, правильність отримання та встановлення власника власності |
| 7 | public void testMonopolyMapValues() | Перевіряє правильність значень різних властивостей на дошці Monopoly, таких як назви, тип, ціна та позиція |

Діаграма класу Tests з усіма полями та методами наведена на рис. 8.

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание**

Рисунок 8 – Diagram of class Tests

Тепер розглянемо особливості реалізації усіх тестів. Почнемо з методу testCompare\_OnePlayerOnline(), реалізація якого наведено на рис. 9.

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание**

Рисунок 9 – Код методу testCompare\_OnePlayerOnline()

У цьому тесті створюються два гравця з різними балансами та статусами. Спочатку перевіряється, чи обидва гравці в онлайні. Якщо ні, то відповідно в залежності від того котрий із гравців не в онлайні, повертаємо 1 або -1. У даному випадку повертамо -1, бо другий гравець в оффлайні (детальніше можна глянути у реалізації класу PlayerComparator). Таким чином, тест перевіряє, чи коректно працює порівняння гравців, коли один з них вийшов з гри.

Тепер метод testSortingPlayers(), реалізація якого наведена на рис. 10.

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание**

Рисунок 10 – Код методу testSortingPlayers()

У цьому тесті створюються три гравці з різними балансами та випадковим порядком. Потім гравці перемішуються у випадковому порядку за допомогою *Collections.shuffle()*. Після цього вони сортуються за допомогою *Collections.sort()* та *comparator*. Загалом, тест перевіряє, чи гравці правильно впорядковані за спаданням балансу.

Тепер метод testPlayer(), реалізація якого наведена на рис. 11.

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание**

Рисунок 11 – Код методу testPlayer()

У цьому тесті створюється гравець з випадковим ім'ям та кольором. Далі проводиться перевірка правильності отримання різних параметрів гравця, таких як ім'я, баланс, кольори тощо. Потім гравець оновлює свій баланс та виходить з гри. Усі ці дії виконуються з метою перевірки правильності роботи методів класу Player.

Тепер метод testProperty(), реалізація якого наведена на рис. 12.

**Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание**

Рисунок 12 – Код методу testProperty()

У цьому тесті створюється власність з випадковими параметрами. Потім проводиться перевірка правильності отримання різних параметрів власності, таких як ім'я, ціна, власник тощо. Після цього власність змінює власника та зберігається в додатковому об'єкті. Усі ці кроки виконуються з метою перевірки коректності роботи методів класу Property.

Тепер метод testMonopolyMapValues(), реалізація якого наведена на рис. 13.

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание**

Рисунок 13 – Код методу testMonopolyMapValues()

У цьому тесті перевіряється правильність значень елементів у перехуванні MonopolyMap. Спочатку перевіряється правильність назви, типу, ціни, оренди та позиції "Jail". Потім перевіряється "Central" з аналогічними параметрами. Далі перевіряється "GO" на правильність її параметрів. Тест переконується, що всі ці значення збігаються з орієнтовними.

Насамкінець на рис. 14 продемонстровано, що всі вище наведені тести проходять та код працює коректно.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

Рисунок 14 – Скріншот проходження тестів

**Перерахування “MonopolyMap”** визначає різні елементи на дошці нашої гри. Кожен елемент має власну назву, позицію, тип, ціну та оренду. Оголошені елементи включають ті, що представляють тюрму, GO, різні вулиці, шанси, податки та інші елементи гри. Для кожного елементу визначені методи getters, які повертають його назву, тип, ціну, оренду та позицію. Крім того, визначені конструктори для ініціалізації полів кожного елементу з різним набором параметрів.

Діаграма перерахування MonopolyMap наведена на рис. 15.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 15 – Diagram of enum MonopolyMap

**Клас “Controller”** відповідає за зв'язок між елементами графічного інтерфейсу та ігровими процесами, обробляючи події, які виникають при натисканні кнопок, та оновлюючи стан інтерфейсу відповідно до змін у грі. Усі поля та методи класу описано в Таблиці 5 та Талиці 6 відповідно.

Таблиця 5 – Опис полів класу “Controller”

|  |  |  |
| --- | --- | --- |
| № | Прототип поля | Призначення поля |
| 1 | protected Button yesButton, noButton, retireButton; | Представляють кнопки інтерфейсу користувача, які відповідають за різні дії у грі, такі як підтвердження, відмова та вихід з гри |
| 2 | protected boolean yesButtonAvailable, noButtonAvailable, retireButtonAvailable; | Визначають, чи доступні відповідні кнопки для натискання в поточний момент часу |
| 3 | protected boolean yes, no, retire; | Визначають, чи була натиснута відповідна кнопка |
| 4 | protected TextArea messageArea; | Вивід повідомлень для користувача |
| 5 | protected ImageView dice1, dice2; | Зображення, що представляють два гральні кубики |
| 6 | private Stage primaryStage; | Головна сцена додатку, яка представляє основне вікно гри |
| 7 | private Scene scene; | Поточна сцена гри, яка відображає всі елементи інтерфейсу |
| 8 | private MonopolyStage monopoly; | Використовується для виклику методів, які відповідають за логіку гри |

Таблиця 6 – Опис методів класу “Controller”

|  |  |  |
| --- | --- | --- |
| № | Прототип методів | Призначення методу |
| 1 | public void clearAllButton(); | Cкидання стану кнопок перед новою дією користувач |
| 2 | public void yesClick() | Викликається при натисканні кнопки “Yes/Roll Dice” |
| 3 | public void noClick(ActionEvent event); | Викликається при натисканні кнопки “Nothing/Ignore” |
| 4 | public void retireClick(); | Викликається при натисканні кнопки “Retire” |
| 5 | public void setPrimaryStage(Stage primaryStage); | Встановлює основну стадію додатку |
| 6 | public void setScene(Scene scene); | Встановлює поточну сцену гри |
| 7 | public Stage getPrimaryStage(); | Повертає основну стадію додатку |
| 8 | public Scene getScene(); | Повертає поточну сцену гри |
| 9 | public void setMonopoly(MonopolyStage monopoly); | Встановлює об'єкт MonopolyStage, що представляє основну сцену гри “Монополія” |
| 10 | public TextArea getMessageArea(); | Повертає текстове поле для виводу повідомлень для користувача |
| 11 | public Button getYesButton(), getNoButton(),getRetireButton(); | Повертає кнопку “Yes/Roll Dice”, “Nothing/Ignore”, “Retire” |
| 12 | public void setYesButtonAvailable(), setNoButtonAvailable(),  setRetireButtonAvailable(); | Встановлює доступність кнопки “Yes/Roll Dice”, “Nothing/Ignore”, “Retire” |
| 13 | public boolean getRetire(), getNo(), getYes(); | Повертає стан кнопки “Yes/Roll Dice”, “Nothing/Ignore”, “Retire” |
| 14 | public void setNo(), setYes(), setRetire(); | Встановлює стан кнопки “Yes/Roll Dice”, “Nothing/Ignore”, “Retire” |
| 15 | public ImageView getDice1(),getDice2(); | Повертає зображення кубика 1, кубика 2 |

Майже всі методи у даному класі є дефолтними getters and setters методи, які повертають та вставновлюють значення полів класу відповідно. Тепер давайте розглянемо реалізацію деяких основних методів. Почнемо з методу clearAllButton(), код якого наведено на рис. 16.

**Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание**

Рисунок 16 – Код методу clearAllButton()

Метод clearAllButton() служить для скидання стану всіх кнопок. Він встановлює значення змінних yes, no і retire у false. Це необхідно для того, щоб перед кожною новою дією користувача всі попередні стани кнопок були скинуті, забезпечуючи коректну обробку нових подій.

Тепер метод yesClick(), реалізація якого наведена на рис. 17. Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 17 – Код методу yesClick()

Метод yesClick() викликає метод clearAllButton(), який скидає стан усіх кнопок. Потім перевіряє, чи кнопка “Yes/ Roll Dice” доступна для натискання (yesButtonAvailable). Якщо так, то змінює стан змінної yes на true і змінює текст кнопки на “####”, інакше змінна yes залишається false. Після цього викликається метод monopoly.checkRequests(), який перевіряє запити гри та виконує відповідні дії.

Тепер метод noClick(), реалізація якого наведена на рис. 18. Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 18 – Код методу noClick()

Метод noClick() викликає метод clearAllButton(), щоб скинути стан усіх кнопок. Потім перевіряє доступність кнопки “Nothing/Ignore” (noButtonAvailable). Якщо кнопка доступна, змінює стан змінної no на true і змінює текст кнопки на “####”, інакше змінна no залишається false.

Тепер метод retireClick(), реалізація якого наведена на рис. 19. Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 19 – Код методу retireClick()

Метод retireClick() викликає метод clearAllButton(), щоб скинути стан усіх кнопок. Далі перевіряє, чи доступна кнопка “Retire” (retireButtonAvailable). Якщо так, змінює стан змінної retire на true і змінює текст кнопки на “####”, інакше змінна retire залишається false.

Також до проекту були додані всі графічні елементи, що використовуються для відображення ігрового поля, іконок гравців, зображень кубиків, назви гри та рахунку. Всі елементи зберігаються в пакеті img, як показано на рис. 20.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 20 – Скріншот графічних елементів проекту

# **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.

Моя робота включала розробку різних класів, таких як “Game” (включаючи перерахування “MonopolyMap”), “UserInterface”, “Tests” та “Controller”, кожен з яких виконував свої функції у системі гри. Клас “Game” був відповідальний за логіку гри, управління гравцями та ігровим полем, включаючи реалізацію руху гравців, купівлі та обміну власностей, а також взаємодії з інтерфейсом користувача. Інтерфейс “UserInterface” визначав методи, які відповідали за відображення графічного інтерфейсу гри, сповіщення гравців та взаємодію з ними. Клас “Tests” містив набір юніт-тестів для перевірки функціональності різних частин гри. Клас “Controller” забезпечував зв'язок між елементами графічного інтерфейсу та ігровими процесами, обробляючи події, які виникають при натисканні кнопок, та оновлюючи стан інтерфейсу відповідно до змін у грі.

Крім того, в рамках роботи я також виконував задачу з розробки графічного інтерфейсу для гри “Монополія”, включаючи дизайн інтерфейсу, розташування елементів та налагодження стилів. Використання вищезазначених інструментів та підходів дозволило ефективно реалізувати поставлені завдання та створити функціональну та зручну для користувача гру "Монополія" з графічним інтерфейсом.

Підсумовуючи, можна сказати, що розроблена програма “Монополія” надає користувачам можливість насолоджуватися грою на комп'ютері з відтвореним функціоналом та інтерфейсом класичної настільної гри.

# **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/>