## Зміст

Вступ

1. Загальна частина

* 1. Постановка задачі
  2. Дослідження і аналіз об’єкту програмування
  3. Використанні програмні засоби
  4. Вимоги до апаратного та програмного забезпечення

2. Практична частина

2.1 Створення та налагодження програми

2.2 Опис програми та її алгоритмів

2.3 Інструкція програміста

2.4 Інструкція оператора

Висновок

Список використаних джерел

Додаток А – Код програми

**Вступ**

Гра - вид непродуктивної діяльності, де мотив лежить як в її результаті, так і в самому процесі. Це розважальний вид занять, не пов'язаний з безпосередніми завданнями життєзабезпечення; один з невід'ємних елементів існування людини як виду, а також інших високоорганізованих тварин.

Незважаючи на те, що ігор розроблено вже велика кількість, актуальність їх не спадає, це пояснюється тим, що ігри є на різні платформи та різних жанрів, кожна гра майже кардинально відрізняється від іншої і тому вони завжди знаходять своїх користувачів та фанатів.

Серія версій операційної системи Windows має свій набір стандартних ігор, які є в кожній версії цієї ОС. Більшість з них це карточні ігри, але є також відомі ігри такі як «Шахмати», «Шашки», також є і гра «Сапер».

За основу був узятий аналог гри «Сапер» на Windows 2000. Ця гра вбудована в усі версії ОС Windows та с кожною версією її удосконалювали. Але версія яка була на Windows 2000 вважається стандартною та є більш популярною серед користувачів Windows.

В курсовій роботі розглядається детальна розробка гри «Сапер». Присутні приклади алгоритмів в програмі, діаграма класів; UML-діаграма прецедентів та є детальний розгляд кожного класу в програмі. Присутні картинки програм аналогів цієї гри та розробленої гри.

Дана курсова робота була на писана на мові програмування Java, тому опис кожного класу буде мати в собі діалект цієї мови.

1. **Загальна частина**
   1. **Постановка задачі**

Основне завдання полягає в створенні консольної та графічної   
версії гри «Сапер». Закінчена версія гри повинна мати:

* три рівня складності;
* відносно кожного рівня необхідно передбачити фіксовану кількість мін (червоного кольору);
* обов’язково передбачити механізм ідентифікації комірок за допомогою цифрового позначення (цифри від 1-го до 8-ми (різного кольору)), яке означає кількість мін поруч з коміркою;
* реалізувати відкриття порожнього поля клітинок при виборі порожньої клітинки.

Користувач повинен користуватися виключно мишкою, за допомогою лівої кнопки миші (ЛКМ) та правої кнопки миші (ПКМ). Для проходження гри користувач повинен відкрити усі комірки та не натрапити на міну.

Також користувачу надається вибір. Вибрати консольну версію гри, чи графічну. В кожній версії необхідно передбачити три рівня складності:

1. Легкий
2. Середній
3. Експерт

Рівні відрізняються розмірністю поля та кількістю мін.

Легкий рівень - поле: 9х9, кількість мін: 10.

Середній рівень - поле: 16х16, кількість мін: 40.

Рівень «Експерт» - поле: 30х16, кількість мін: 99.

Користувачі: діти від 6-ти років

В консольній версії. Вхідні дані: номер рядка, номер стовпця, чи є клітинка бомбою. Вихідні дані: поле з позначеними клітинками (порожні клітинки, цифри, бомби), програв користувач чи виграв.

В графічній версії. Вхідні дані: координати миші при натисканні ЛКМ чи ПКМ. Вихідні дані: поле з позначеними клітинками (порожні клітинки, цифри, бомби), програв користувач чи виграв.

* 1. **Опис предметної області**

Щоб навчити користувачів клікати мишкою, розробники компанії Microsoft створили гру «Сапер». Вона з'явилася як самостійний продукт в 1990 році (до того гри подібного жанру були відомі на ЕОМ ще в 60-х роках минулого століття), стала доповненням пакета розваг. А ось вже в Windows 3.1 гра стала незамінним атрибутом операційної системи. Написана спеціально для Windows Робертом Доннером і Куртом Джонсоном.

Ця гра була створена з метою ознайомити користувачів з правої і лівої кнопкою миші, а також підвищити швидкість використання даного маніпулятора. Як підсумок, гра «Сапер» стала дуже популярною, користувачі почали змагатися між собою в швидкості її проходження. Відповідно, люди почали використовувати мишу і поступово йти від використання «командного рядка».

Правила гри. Ігрове поле поділене на суміжні комірки, деякі з яких «заміновані»; кількість «замінованих» комірок відомо. Метою гри є відкриття всіх комірок, які не містять міни.

Гравець відкриває комірки, намагаючись не відкрити комірку з міною. Відкривши комірку з міною, він програє. Міни розставляються після першого ходу, тому програти на першому ж ході користувач не може. Якщо під відкритою коміркою міни немає, то в ній з'являється число, що показує, скільки комірок, що є сусідами з щойно відкритої, «заміновано»; використовуючи ці числа, гравець намагається розрахувати розташування мін. Якщо під сусідніми комірками теж немає мін, то відкривається деяка «Не замінована» область до комірок, в яких є цифри. «Заміновані» комірки гравець може помітити, щоб випадково не відкрити їх. Відкривши всі «Не заміновані» комірки, гравець виграє.

На рисунках 1.1 та 1.2 представлено приклад ігрового поля на початку гри та після першого кліку по полю.

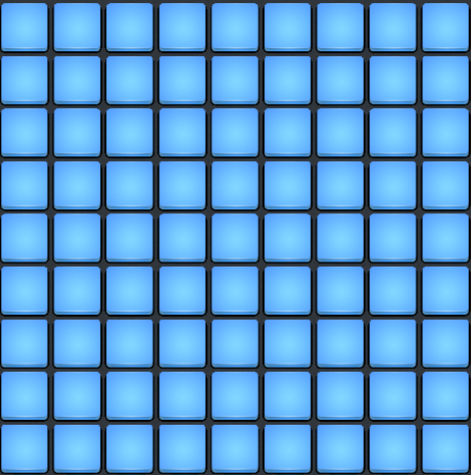


Рисунок 1.1 – Поле на початку гри

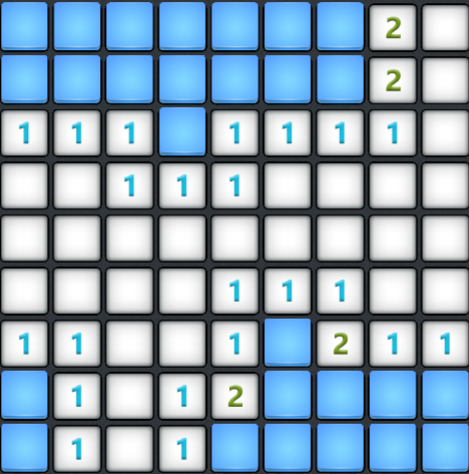


Рисунок 1.2 – Поле після першого кліку, який потрапив на порожню клітинку

В прикладі приведено два зображення стандартної реалізації гри «Сапер» на Windows 10.

На рисунку 1.1 зображено повністю НЕ відкрите поле

На рисунку 1.2 зображено на половину відкрите поле, де зображені цифри, які відповідають кількості бомб біля цієї клітинки

* 1. **Використані програмні засоби**

**Java** – це об'єктно-орієнтована мова програмування, що розробляється компанією Sun Microsystems з 1991 року і офіційно випущений 23 травня 1995 року.

Спочатку нова мова програмування мала назву Oak (James Gosling) і розроблялася для побутової електроніки, але згодом була переіменована й Java стала використовуватися для написання аплетів, додатків і серверного програмного забезпечення.

Програми на Java можуть бути трансльовані в байт-код, що виконується на віртуальній java-машині (JVM) - програмою, обробленої в байт-код і передавальної інструкції обладнанню, як інтерпретатор, але з тією відмінністю, що байт-код, на відміну від тексту, обробляється значно швидше.

Мова Java зародилася як частина проекту створення передового програмного забезпечення для різних побутових приладів. Реалізація проекту була розпочата на мові C++, але незабаром виник ряд проблем, найкращим засобом боротьби з якими була зміна самого інструмента - мови програмування. Стало очевидним, що необхідний вид платформи незалежної мови програмування, що дозволяє створювати програми, які не доводилося б компілювати окремо для кожної архітектури і можна було б використовувати на різних процесорах під різними операційними системами.

Мова Java потрібна була для створення інтерактивних продуктів для мережі Internet. Фактично, більшість архітектурних рішень, прийнятих при створенні Java, було продиктовано бажанням надати синтаксис, схожий з C і C++. У Java використовуються практично ідентичні угоди для оголошення змінних, передачі параметрів, операторів і для управління потоком виконанням коду. У Java додані всі хороші риси C++.

Три ключові елементи об'єдналися в технології мови Java.

Java надає для широкого використання свої аплети (applets) - невеликі, надійні, динамічні, які не залежать від платформи активні мережеві додатки, що вбудовуються в сторінки Web. Аплети Java можуть налаштовуватися і поширюватися споживачам з такою ж легкістю, як будь-які документи HTML.

Java вивільняє міць об'єктно-орієнтованої розробки додатків, поєднуючи простий і знайомий синтаксис з надійним і зручним в роботі середовищем розробки. Це дозволяє широкому колу програмістів швидко створювати нові програми і нові аплети.

**Java Development Kit (JDK).** Безкоштовно поширюваний компанією Oracle Corporation (раніше Sun Microsystems) комплект розробника додатків на мові Java, що включає в себе компілятор Java (javac), стандартні бібліотеки класів Java, приклади, документацію, різні утиліти і виконавчу систему Java (JRE). До складу JDK не входить інтегроване середовище розробки на Java, тому розробник, що використовує тільки JDK, змушений використовувати зовнішній текстовий редактор і компілювати свої програми, використовуючи утиліти командного рядка.

Всі сучасні інтегровані середовища розробки додатків на Java, такі, як JDeveloper, NetBeans IDE, Sun Java Studio Creator, IntelliJ IDEA, Borland JBuilder, Eclipse, спираються на послуги, що надаються JDK. Більшість з них для компіляції Java-програм використовують компілятор з комплекту JDK. Тому ці середовища розробки або включають в комплект поставки одну з версій JDK або вимагають для своєї роботи попередньої інсталяції JDK на машині розробника.

У ролі зовнішнього текстового редактора був використаний редактор Intellij IDEA.

**IntelliJ IDEA.** Інтегроване середовище розробки програмного забезпечення на багатьох мовах програмування, зокрема Java, JavaScript, Python, розроблена компанією JetBrains.

Перша версія IntelliJ IDEA з'явилася в січні 2001 року і швидко набула популярності, як перша Java IDE з широким набором інтегрованих інструментів для рефакторінга, які дозволяли програмістам швидко реорганізовувати вихідні тексти програм. Дизайн середовища орієнтований на продуктивність роботи програмістів, дозволяючи їм сконцентруватися на розробці функціональності, в той час як IntelliJ IDEA бере на себе виконання рутинних операцій.

* 1. **Вимоги до апаратного та програмного забезпечення**

Вимоги до апаратного забезпечення

* Операційна система: Windows XP/Vista/7/8/10;
* ОЗУ 32 МБ
* Наявність клавіатури та миші

Вимоги до програмного забезпечення

* JDK (1.5 та вище)
* Будь-який зовнішній текстовий редактор (Наприклад: JDeveloper, NetBeans IDE, Sun Java Studio Creator, IntelliJ IDEA, Borland JBuilder, Eclipse)

1. **Практична частина**
   1. **Створення та налагодження програми**

При створенні гри були використанні такі засоби об’єктно-орієнтованого програмування мови Java, як:

1. класи та їх екземпляри;
2. конструктори;
3. інкапсуляція;
4. поліморфізм;
5. композиція.

В графічній версії гри клас GUIAction реалізував два інтерфейси ActionListener та MouseListener, які належать бібліотеці AWT. Вона знаходиться в JDK. Вони потрібні для считування вхідних даних від користувача, коли він вибирає клітинку, та натискає ЛКМ чи ПКМ. Ці класи отримують координаті вибраної клітинки та передають їх для оброблення всіх даних поля.

* 1. **Опис програми та її алгоритмів**

Функціональні можливості системи відображено за допомогою UML-діаграми прецедентів, зображеної на рисунку 2.1.

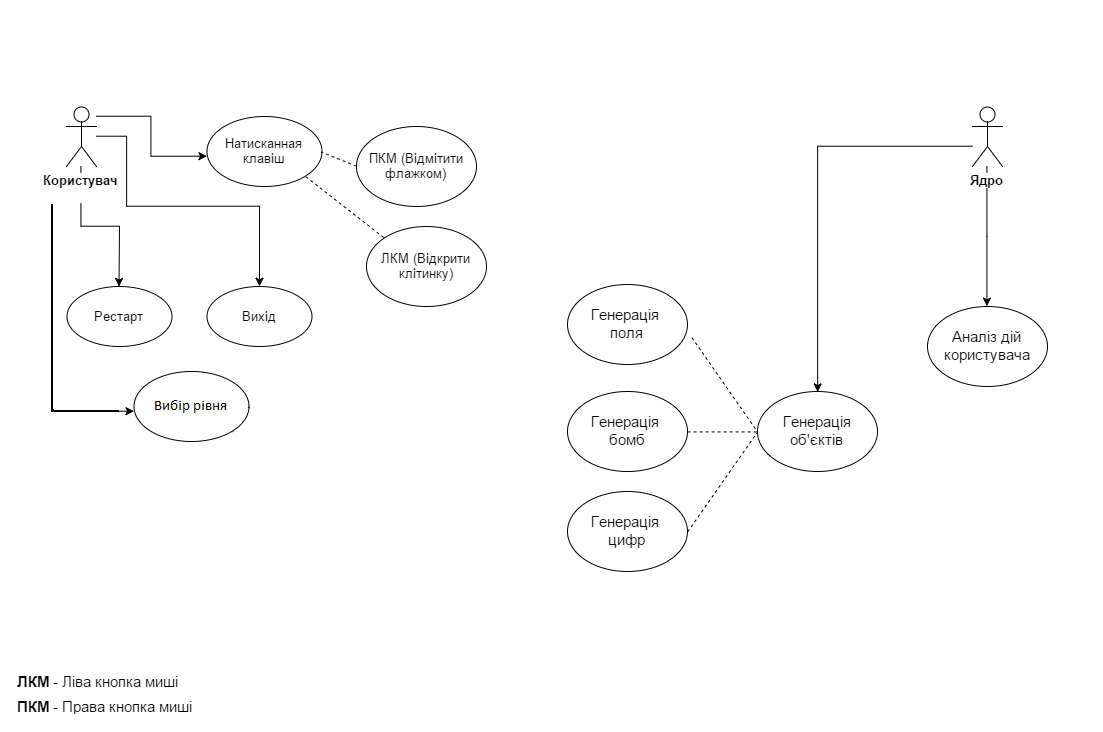
На діаграмі виділено актора – користувача, та ядро програми, яке генерує зовнішній вид.

Актор має наступні можливості:

* Натискання клавіш
  1. ПКМ – для того, щоб відмітити флажком клітинку, та зняти флажок з клітинки
  2. ЛКМ – для того, щоб відкрити клітинку
* Зміна рівня – користувач може вибрати три рівня гри, які відрізняються між собою розмірністю поля
* Рестарт – при бажанні користувач може в будь-який час почати нову гру та закінчити вже розпочату
* Вихід з гри

Ядро має наступні можливості:

* Аналіз дій користувача – аналізує дії користувача та оброблює їх згідно логіки гри
* Генерація об’єктів
  1. Генерація поля – генерує поле, відносно рівня гри
  2. Генерація бомб – генерує певну кількість бомб, яка залежить від вибраного рівня гри.
  3. Генерація цифр – генерує цифри, коли біля відкритої клітинки є певна кількість бомб (від 1 до 8).

****Рисунок 2.1 **-** UML-діаграма прецедентів

Для реалізації гри «Сапер» розроблено ієрархію класів, які взаємодіють між собою. Діаграма класів у термінах мови UML представлена на рисунку 2.2.

Ієрархія класів ділиться на три частини: класи ядра, класи, що реалізують консольну версію гри, класи, що реалізують віконну версію гри. Розглянемо кожну частину класів окремо.

Клас BaseAction – ніби то ядро програми. Він збирає усі дані (про клітинки, поле, логіку гри), після збору усіх даних він генерує поле. Також в ньому є метод, який в свою чергу приймає на вхід координати вибраної клітинки користувачем та викликає в потрібній послідовності методи, які реалізують логіку гри, за допомогою змінної, яка має посилання на інтерфейс. В ньому присутні три властивості, які мають посилання на інтерфейс (за допомогою них він і збирає усі дані), тому що при створенні екземпляру класу BaseAction, чи його дочірніх класів ми повинні передати три класи, які реалізують ці інтерфейси.

Класи, які реалізують консольну версію гри:

1. ConsoleCell – він реалізує інтерфейс ICell. Так як цей інтерфейс має узагальнений тип, ми передали на вхід PrintStream, таким чином з’являється можливість одразу виводити інформацію у консоль. Цей клас реалізує кожну клітинку в полі та за допомогою властивостей в ньому, йому передаються унікальні дані для кожної клітинки та враховуючи їх виводиться відповідний вигляд клітинки
2. ConsoleBoard – він реалізує інтерфейс IBoard та перемальовує поле кожний раз, після ходу користувача.
3. StandardLogicConsole – він реалізує одразу три інтерфейси (ISelectLevel, ILogic, ITheNumOfTheField). Цей клас реалізує стандарту логіку гри в консольній версії.
4. Runner – запускає гру в консольній версії.

Класи, які реалізують графічну версію гри:

1. GUIAction – цей клас розширює свою функціональність за допомогою класу BaseAction та реалізує інтерфейси ActionListener та MouseListener, які вбудовані у графічному бібліотеку JDK – AWT. Вони в свою чергу допомагають реалізувати дії у робочому вікні (клік миші и т. п.).
2. GUICell - цей клас працює таким же чином як і ConsoleCell, але на відміну від нього ми передаємо на вхід тип Graphics. Таким чином в нас з’являється можливість графічно малювати кожну клітинку.
3. GUIBoard – цей клас працює таким же чином як і ConsoleBoard, але на відміну від нього він реалізує графічний вид поля.
4. StandardLogicGUI – цей клас працює таким же чином як і StandardLogicConsole, але логіку реалізує для графічної версії гри.
5. Main - запускає гру в графічній версії.

Також є допоміжні класи (Рівні гри) – Easy, Medium, Expert. Ці класи реалізують інтерфейс ITheNumOfTheField та говорять тільки, якої розмірності повинно бути поле та скільки повинно бути бомб на полі.

****Рисунок 2.2 - Діаграма класів у термінах мови UML

Також в даній грі використаний алгоритм, який присвоює клітинці значення, цей алгоритм зображений на блок-схемі знизу (рисунок 2.3).



Рисунок 2.3 – Блок схема. Алгоритм присвоєння клітинці значення.

Спочатку ініціалізуємо дві змінні, які не дорівнюють одна одній. Потім починаємо цикл, який закінчиться коли ці змінні будуть дорівнювати одна одній. Після циклу ми присвоюємо значення однієї змінної іншій, таким чином вони стають рівні між собою. Потім починаємо 2 цикли, які будуть обходити усе поле та перевіряти що знаходиться біля обраної клітинки.

Після циклів обрана клітинка звірюється умовою (якщо ця клітинка пуста та не має в собі значення що вона вже провірена, тоді заходимо в умову та звіряємо далі. Якщо біля цієї клітинки немає ні однієї бомби, то відкриваємо сусідні клітинки та збільшуємо одну з двох змінних, які не повинні дорівнювати одна одній.

Таким чином обрана клітинка є порожньою тоді їй присвоюється значення, що вона провірена. Після цього присвоюємо клітинці значення, яке дорівнює кількості мін біля неї. Потім знову запитуємо, чи дорівнюють ті дві змінні одна одній. Якщо ні, то були відкриті нові клітинки і їх теж треба провірити і ми знову заходимо у цикл, інакше всі відкриті порожні клітинки ми вже перевірили та виходимо із циклу.

* 1. **Інструкція програміста**

Гра «Сапер» на основі класів написана на мові Java з використанням методів об`єктно-орієнтованого програмування.

Проект являє собою головну форму.

Мова програмування: Java.

Середовище програмування: Intellij IDEA.

Не потребує підключення до інтернету.

Не використовує базу даних.

* 1. **Інструкція оператора**

Для можливості користування грою на ПК повинні бути встановлені: JDK (версії 1.5 чи вище), будь-який зовнішній текстовий редактор для мови Java. Також комп`ютер повинен відповідати системним вимогам.

Для того щоб запустити програму, потрібно в зовнішньому текстовому редакторі відкрити проект з програмою, та для відображення консольної версії гри, потрібно запустити файл Runner.java, для графічної потрібно запустити файл Main.java.

В консольній версії гри після запуску зовнішній редактор відкриє консоль та виведе для користувача запитання про вибір рівня (рисунок 2.4).

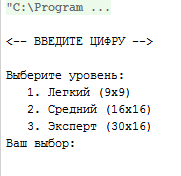


Рисунок 2.4 – Вибір рівня гри в консольній версії

Після вибору згенерує відповідного розміру поле та запитуватиме до кінця гри, яку клітинку користувач хоче вибрати (який рядок, який стовбець та чи вважає користувач цю клітинку бомбою), як зображено на рисунок 2.5, після вибору в консолі знову з’явиться поле, але вже з новими даними та з тими ж самими запитаннями (рисунок 2.6).

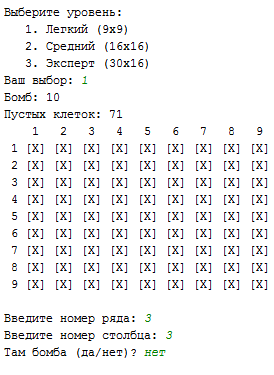


Рисунок 2.5 - Поле на початку гри в консольній версії

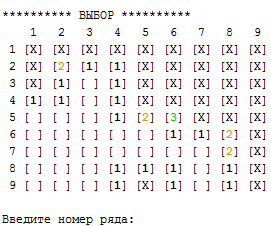


Рисунок 2.6 - Поле після першого вибору клітинки в консольній версії гри

В графічній версії гри після запуску з’явиться форма, в якій буде один пункт меню з назвою «Уровень», де можна буде обрати 1 з 3-х рівнів гри (рисунок 2.7).

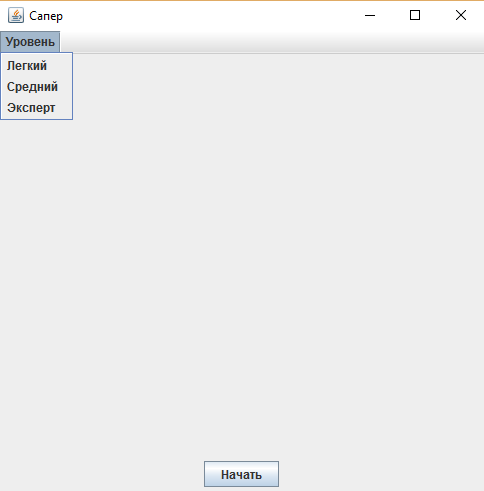
****

Рисунок 2.7 - Вибір рівня гри в графічній версії

Після чого одразу згенерує поле і розпочне гру (рисунок 2.8). Треба розуміти, що після вибору рівня, вже розпочата гра зітреться і розпочнеться нова.

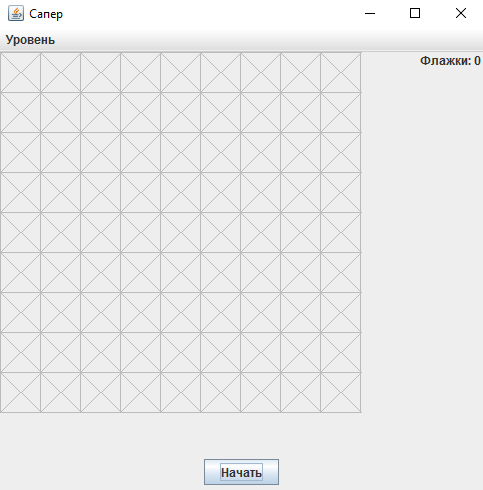
****

Рисунок 2.8 – Поле легкого рівня на початку гри в графічній версії

Коли користувач вже вибрав рівень гри та вікно вже згенерувало поле. Після першого кліку лівої кнопки миші відкриються вільні клітинки та користувач зможе орієнтуватись, які клітинки відкривати далі (рисунок 2.9).

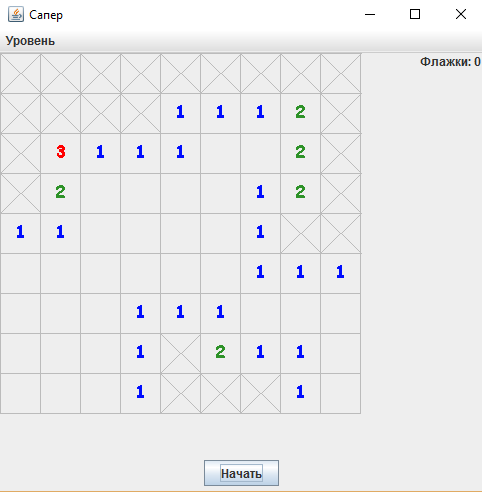
****

Рисунок 2.9 - Поле після першого кліку лівої кнопки миші

На рисунку 2.10 зображено, як виглядає поле на рівні «Експерт», в графічній версії гри.

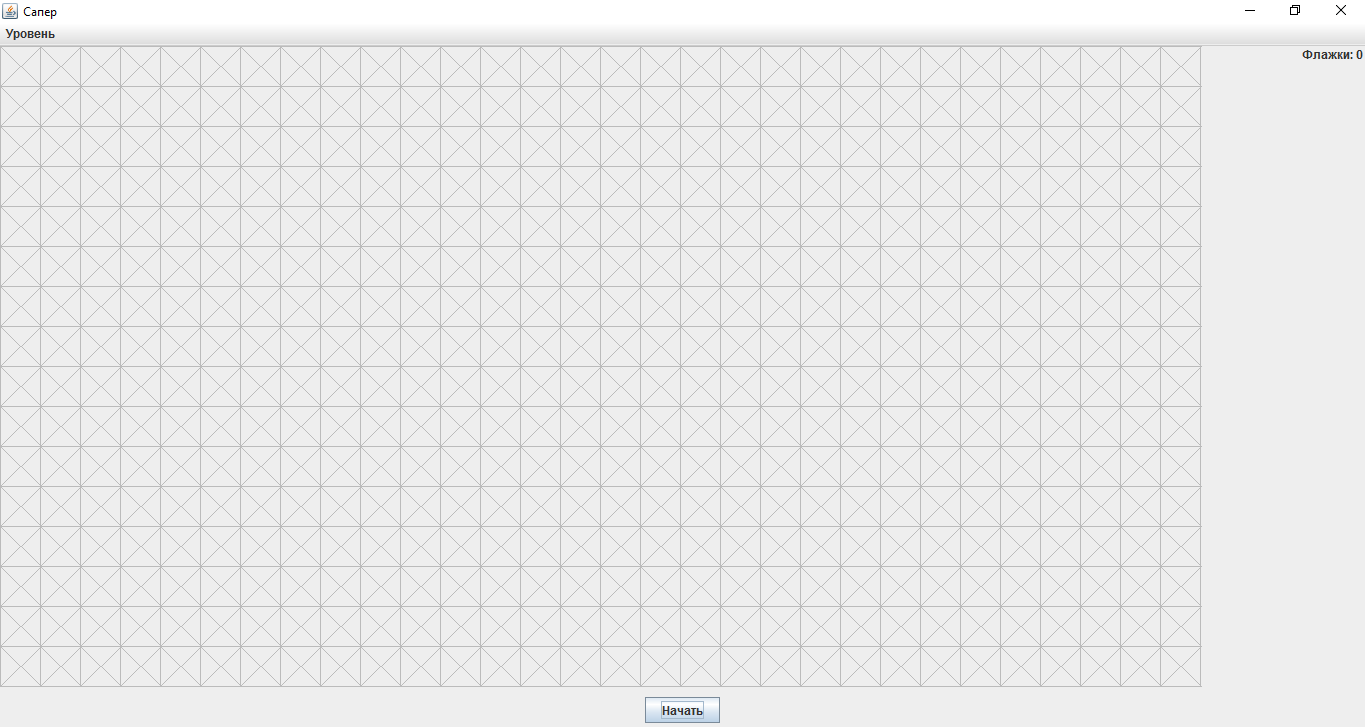


Рисунок 2.10 – Поле рівня експерт

**Висновок**

В ході написання курсової роботи була розроблена система класів для ігрових цілей мовою програмування Java з використанням засобів об’єктно-орієнтованого програмування.

На етапі розробки курсової роботи було проаналізовано програми-аналоги на Windows 2000 та Windows 10.

Для написання даної програми було обрано середовище програмування Intellij IDEA, яке включає в себе всі необхідні інструменти для реалізації поставленої задачі.

В межах курсової роботи було реалізовано консольний та графічний вид гри «Сапер» на основі класів.

Під час написання курсової роботи я закріпив навички та уміння програмувати на мові програмування Java з використанням об’єктно-орієнтованого програмування.

**Список використаних джерел**

1. Шилдт, Герберт. Java. Полное руководство, 8-е изд. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2012 – 1104 с.
2. <https://ru.wikipedia.org/wiki/Игра>
3. <https://ru.wiktionary.org/wiki/игра>
4. <https://ru.wikipedia.org/wiki/Java_Development_Kit>
5. <https://ru.wikipedia.org/wiki/IntelliJ_IDEA>
6. <http://progopedia.ru/language/java/>

**Додаток А**

**Код програми**

Інтерфейси:

**IBoard.java:**

*/\*\*  
 \* Описываем поведение доски  
 \*/***public interface** IBoard {  
  
 */\*\*  
 \* Количество помеченных бомб  
 \** ***@return*** *сколько помеченных бомб  
 \*/* **int** returnSumBomb();  
  
 */\*\*  
 \* Отменяем данную ячейку как помеченную  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \*/* **void** cancelSuggestBomb(**int** x, **int** y);  
  
 */\*\*  
 \* Помеченная ли ячейка  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \** ***@return*** *если данная ячейка помечена как бомба, то возвращаем true  
 \*/* **boolean** returnSuggestBomb(**int** x, **int** y);  
  
 */\*\*  
 \* Рисуем доску исходя из входящего массива ячеек  
 \** ***@param cells*** *массив ячеек  
 \*/* **void** drawBoard(ICell[][] cells);  
  
 */\*\*  
 \* Рисуем ячейку.  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \*/* **void** drawCell(**int** x, **int** y);  
  
 */\*\*  
 \* Рисуем взрыв всех бомб  
 \*/* **void** drawBang();  
  
 */\*\*  
 \* Рисуем поздравление когда игра выиграна  
 \*/* **void** drawCongratulate();  
  
 */\*\*  
 \* Рисуем проигрыш  
 \*/* **void** drawLosing();  
}

**ICell.java:**

*/\*\*  
 \* Описываем поведение ячейки  
 \*/***public interface** ICell<T> {  
  
 */\*\*  
 \* Является ли данная ячейка бомбой  
 \** ***@return*** *если данная ячейка бомба, возвращаем true  
 \*/* **boolean** isBomb();  
  
 */\*\*  
 \* Является ли данная ячейка помеченной как бомба  
 \** ***@return*** *если данная ячейка помечена как бомба, возвращаем true  
 \*/* **boolean** isSuggestBomb();  
  
 */\*\*  
 \* Является ли данная ячейка уже открытой  
 \** ***@return*** *если данная ячейка уже открыта, возвращаем true  
 \*/* **boolean** isSuggestEmpty();  
  
 */\*\*  
 \* Помечаем данную ячейку как открытую  
 \*/* **void** suggestEmpty();  
  
 */\*\*  
 \* Помечаем данную ячейку как бомбу  
 \*/* **void** suggestBomb();  
  
 */\*\*  
 \* Отменяем значение помеченной бомбы  
 \*/* **void** cancelSuggestBomb();  
  
 */\*\*  
 \* Является ли данная ячейка проверенной  
 \** ***@return*** *если данная ячейка является проверенной, возвращаем true  
 \*/* **boolean** isChecked();  
  
 */\*\*  
 \* Помечаем данную ячейку как проверенную  
 \*/* **void** checked();  
  
 */\*\*  
 \* Является ли данная ячейка какой-то из этих цифр  
 \** ***@return*** *если данная ячейка является, то возвращаем true  
 \*/* **boolean** isSuggest1();  
 **boolean** isSuggest2();  
 **boolean** isSuggest3();  
 **boolean** isSuggest4();  
 **boolean** isSuggest5();  
 **boolean** isSuggest6();  
 **boolean** isSuggest7();  
 **boolean** isSuggest8();  
  
 */\*\*  
 \* Помечаем данную ячейку как цифру (от 1 до 8)  
 \*/* **void** suggest1();  
 **void** suggest2();  
 **void** suggest3();  
 **void** suggest4();  
 **void** suggest5();  
 **void** suggest6();  
 **void** suggest7();  
 **void** suggest8();  
  
 */\*\*  
 \* Рисуем ячейку  
 \** ***@param paint*** *Класс прорироски (консольный или графический)  
 \** ***@param real*** *рисовать реальное значение ячейки или же что предположил пользователь  
 \*/* **void** draw(T paint, **boolean** real);  
  
 */\*\*  
 \* Для GUI нужны доп. параметры, для нахождения координат данной ячейки  
 \*/* **void** draw(T paint, **int** x, **int** y, **boolean** real);  
}

**IGeneratorBoard.java:**

*/\*\*  
 \* Генерация игрового поля  
 \*/***public interface** IGeneratorBoard {  
 ICell[][] generate();  
}

**IUserAction.java:**

*/\*\*  
 \* Действия пользователя  
 \*/***public interface** IUserAction {  
  
 */\*\*  
 \* После того как пользователь выбрал уровень и мы сгенерировали поле,  
 \* передаем все значения для отрисовки поля  
 \*/* **void** initGame();  
  
 */\*\*  
 \* Ловим координаты, какие выбрал пользователь и ловим значение,  
 \* какой пользователь передал ячейке  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \** ***@param bomb*** *значение  
 \*/* **void** select(**int** x, **int** y, **boolean** bomb);  
}

**ITheNumOfTheField.java:**

*/\*\*  
 \* Содержит количество рядов, столбцов и количество бомб на поле  
 \*/***public interface** ITheNumOfTheField {  
  
 */\*\*  
 \* Количество бомб  
 \** ***@return*** *возвращаем количество бомб исходя от сложности выбранного уровня  
 \*/* **int** sumBombs();  
  
 */\*\*  
 \* Количество рядов  
 \** ***@return*** *возвращаем количество рядов исходя от сложности выбранного уровня  
 \*/* **int** sumRow();  
  
 */\*\*  
 \* Количество столбцов  
 \** ***@return*** *возвращаем количество столбцов исходя от сложности выбранного уровня  
 \*/* **int** sumColumn();  
}

**ILogic.java:**

*/\*\*  
 \* Логика игры  
 \*/***public interface** ILogic {  
  
 */\*\*  
 \* Загружаем поле игры  
 \*/* **void** loadBoard(ICell[][] cells);  
  
 */\*\*  
 \* Инициализируем двухмерный массив, за счет методов sumRow() и sumColumn()  
 \** ***@return*** *возвращаем размер всего поля в виде двухмерного массива  
 \*/* ICell[][] sizeField();  
  
 */\*\*  
 \* Взрыв бомбы  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \** ***@return*** *если пользователь выбрал клетку где бомба, то взрываемся  
 \*/* **boolean** shouldBang(**int** x, **int** y);  
  
 */\*\*  
 \* Конец игры  
 \** ***@return*** *если пользователь всё разгадал, возвращаем истину  
 \*/* **boolean** finish();  
  
 */\*\*  
 \* Предположения пользователя (Бомба или пустая клетка)  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \** ***@param bomb*** *есть ли бомба на клетке  
 \*/* **void** suggest(**int** x, **int** y, **boolean** bomb);  
  
 */\*\*  
 \* Проверка первого хода.  
 \** ***@return*** *если на поле нет бомб, возвращаем истину  
 \*/* **boolean** checkTheFirstMove();  
  
 */\*\*  
 \* Очистка вокруг ячейки при первом ходе  
 \* (Для того, чтобы у пользователя не открылась в начале игры только одна ячейка)  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \*/* **void** clearAroundCell(**int** x, **int** y);  
  
 */\*\*  
 \* Генерация бомб на поле  
 \*/* **void** bombsGeneration();  
  
 */\*\*  
 \* Возвращаем количество бомб вокруг ячейки  
 \** ***@param x*** *координата  
 \** ***@param y*** *координата  
 \** ***@return*** *количество бомб вокруг ячейки  
 \*/* **int** checkingAroundCell(**int** x, **int** y);  
  
 */\*\*  
 \* Открываем пустые ячейки  
 \*/* **void** openEmptyCells();  
}

**ISelectLevel.java:**

**import** ru.minesweeperLesson.minesweeper.levels.Easy;  
**import** ru.minesweeperLesson.minesweeper.levels.Expert;  
**import** ru.minesweeperLesson.minesweeper.levels.Medium;

*/\*\*  
 \* Выбор уровня игры  
 \*/***public interface** ISelectLevel {  
  
 */\*\*  
 \* Инициализируем лёгкий уровень  
 \** ***@return*** *возвращаем уровень  
 \*/* Easy easy();  
  
 */\*\*  
 \* Инициализируем средний уровень  
 \** ***@return*** *возвращаем уровень  
 \*/* Medium medium();  
  
 */\*\*  
 \* Инициализируем уровень эксперт  
 \** ***@return*** *возвращаем уровень  
 \*/* Expert expert();  
}

Рівні:

**Easy.java:**

**import** ru.minesweeperLesson.minesweeper.interfaces.ITheNumOfTheField;  
  
**public class** Easy **implements** ITheNumOfTheField {  
  
 @Override  
 **public int** sumBombs() {  
 **return** 10;  
 }  
  
 @Override  
 **public int** sumRow() {  
 **return** 9;  
 }  
  
 @Override  
 **public int** sumColumn() {  
 **return** 9;  
 }  
}

**Medium.java:**

**import** ru.minesweeperLesson.minesweeper.interfaces.ITheNumOfTheField;  
  
**public class** Medium **implements** ITheNumOfTheField {  
  
 @Override  
 **public int** sumBombs() {  
 **return** 40;  
 }  
  
 @Override  
 **public int** sumRow() {  
 **return** 16;  
 }  
  
 @Override  
 **public int** sumColumn() {  
 **return** 16;  
 }  
}

**Expert.java:**

**import** ru.minesweeperLesson.minesweeper.interfaces.ITheNumOfTheField;  
  
**public class** Expert **implements** ITheNumOfTheField {  
  
 @Override  
 **public int** sumBombs() {  
 **return** 99;  
 }  
  
 @Override  
 **public int** sumRow() {  
 **return** 30;  
 }  
  
 @Override  
 **public int** sumColumn() {  
 **return** 16;  
 }  
}

Базовий клас:

**BaseAction.java:**

**import** ru.minesweeperLesson.minesweeper.gui.Main;  
**import** ru.minesweeperLesson.minesweeper.interfaces.\*;  
  
*/\*\*  
 \* Базовые действия пользователя  
 \*/***public class** BaseAction **implements** IUserAction {  
  
 **private final** IGeneratorBoard **generator**;  
 **private final** IBoard **board**;  
 **private final** ILogic **logic**;  
  
 **public** BaseAction(**final** ILogic logic, **final** IBoard board, **final** IGeneratorBoard generator) {  
 **this**.**generator** = generator;  
 **this**.**board** = board;  
 **this**.**logic** = logic;  
 }  
  
 @Override  
 **public void** initGame() {  
 **final** ICell[][] cells = **generator**.generate();  
 **this**.**board**.drawBoard(cells);  
 **this**.**logic**.loadBoard(cells);  
 }  
  
 @Override  
 **public void** select(**int** x, **int** y, **boolean** bomb) {  
 **this**.**logic**.suggest(x, y, bomb);  
 **if** (**this**.**logic**.checkTheFirstMove() && !bomb) {  
 **this**.**logic**.clearAroundCell(x, y);  
 **this**.**logic**.bombsGeneration();  
 }  
 Main.*setLabel*(**"Флажки: "** + **this**.**board**.returnSumBomb() + **" "**);  
 **if** (!bomb)  
 **this**.**logic**.openEmptyCells();  
 **if** (**this**.**logic**.shouldBang(x, y)) {  
 **this**.**board**.drawBang();  
 **this**.**board**.drawLosing();  
 } **else  
 this**.**board**.drawCell(x, y);  
 **if** (**this**.**logic**.finish() && !**this**.**logic**.shouldBang(x, y))  
 **this**.**board**.drawCongratulate();  
 }  
}

Класи консольної версії гри:

**StandardLogicConsole.java:**

**import** ru.minesweeperLesson.minesweeper.console.ConsoleCell;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ICell;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ILogic;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ISelectLevel;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ITheNumOfTheField;  
**import** ru.minesweeperLesson.minesweeper.levels.Easy;  
**import** ru.minesweeperLesson.minesweeper.levels.Expert;  
**import** ru.minesweeperLesson.minesweeper.levels.Medium;  
  
**import** java.util.Random;  
  
*/\*\*  
 \* Стандартная логика игры  
 \*/***public class** StandardLogicConsole **implements** ISelectLevel, ILogic, ITheNumOfTheField {  
  
 **private** Easy **easy**;  
 **private** Medium **medium**;  
 **private** Expert **expert**;  
  
 **private** ICell[][] **cells**;  
  
 @Override  
 **public** Easy easy() {  
 **return easy** = **new** Easy();  
 }  
  
  
 @Override  
 **public** Medium medium() {  
 **return medium** = **new** Medium();  
 }  
  
 @Override  
 **public** Expert expert() {  
 **return expert** = **new** Expert();  
 }  
  
 @Override  
 **public void** loadBoard(ICell[][] cells) {  
 **this**.**cells** = cells;  
 }  
  
 @Override  
 **public int** sumBombs() {  
 **int** sum = 0;  
 **if** (**easy** == **null** && **medium** == **null** && **expert** != **null**)  
 sum = **expert**.sumBombs();  
 **else if** (**easy** == **null** && **expert** == **null** && **medium** != **null**)  
 sum = **medium**.sumBombs();  
 **else if** (**easy** != **null**)  
 sum = **easy**.sumBombs();  
 **return** sum;  
 }  
  
 @Override  
 **public int** sumRow() {  
 **int** row = 0;  
 **if** (**easy** == **null** && **medium** == **null** && **expert** != **null**)  
 row = **expert**.sumRow();  
 **else if** (**easy** == **null** && **expert** == **null** && **medium** != **null**)  
 row = **medium**.sumRow();  
 **else if** (**easy** != **null**)  
 row = **easy**.sumRow();  
 **return** row;  
 }  
  
 @Override  
 **public int** sumColumn() {  
 **int** column = 0;  
 **if** (**easy** == **null** && **medium** == **null** && **expert** != **null**)  
 column = **expert**.sumColumn();  
 **else if** (**easy** == **null** && **expert** == **null** && **medium** != **null**)  
 column = **medium**.sumColumn();  
 **else if** (**easy** != **null**)  
 column = **easy**.sumColumn();  
 **return** column;  
 }  
  
 @Override  
 **public** ICell[][] sizeField() {  
 **return new** ICell[sumRow()][sumColumn()];  
 }  
  
 @Override  
 **public boolean** shouldBang(**int** x, **int** y) {  
 **final** ICell selected = **this**.**cells**[x][y];  
 *// Если это бомба, и пользователь не предположил что это бомба, то мы взрываемся* **return** selected.isBomb() && !selected.isSuggestBomb();  
 }  
  
 @Override  
 **public boolean** finish() {  
 **boolean** finish = **false**;  
 **int** check = 0;  
 **for** (ICell[] row : **this**.**cells**)  
 **for** (ICell cell : row)  
 **if** ((cell.isSuggestBomb() && cell.isBomb()) ||  
 (cell.isSuggestEmpty() && !cell.isBomb()) || (!cell.isSuggestBomb() && cell.isBomb())  
 || cell.isSuggest1() || cell.isSuggest2() || cell.isSuggest3() || cell.isSuggest4()  
 || cell.isSuggest5() || cell.isSuggest6() || cell.isSuggest7() || cell.isSuggest8()) {  
 check++;  
 }  
 **if** (check == (sumRow() \* sumColumn()))  
 finish = **true**;  
 **return** finish;  
 }  
  
 @Override  
 **public void** suggest(**int** x, **int** y, **boolean** bomb) {  
 **if** (!bomb)  
 **this**.**cells**[x][y].suggestEmpty();  
 **if** (bomb && !**this**.**cells**[x][y].isSuggestEmpty())  
 **this**.**cells**[x][y].suggestBomb();  
 **else if** (bomb && **this**.**cells**[x][y].isSuggestEmpty())  
 System.***out***.println(**"Вы уже открыли эту клетку!\n"**);  
 }  
  
 @Override  
 **public boolean** checkTheFirstMove() {  
 **boolean** check = **true**;  
 root: **for** (**int** i = 0; i < sumRow(); i++)  
 **for** (**int** j = 0; j < sumColumn(); j++)  
 **if** (**this**.**cells**[i][j].isBomb()) {  
 check = **false**;  
 **break** root;  
 }  
 **return** check;  
 }  
  
 @Override  
 **public void** clearAroundCell(**int** x, **int** y) {  
 **if** (**cells**.**length** > 3) {  
 **if** (y > 0) suggest(x, y - 1, **false**);  
 **if** (y + 1 < sumColumn()) suggest(x, y + 1, **false**);  
 **if** (x > 0) suggest(x - 1, y, **false**);  
 **if** (x + 1 < sumRow()) suggest(x + 1, y, **false**);  
 **if** (x > 0 && y > 0) suggest(x - 1, y - 1, **false**);  
 **if** (x + 1 < sumRow() && y + 1 < sumColumn()) suggest(x + 1, y + 1, **false**);  
 **if** (x > 0 && y + 1 < sumColumn()) suggest(x - 1, y + 1, **false**);  
 **if** (x + 1 < sumRow() && y > 0) suggest(x + 1, y - 1, **false**);  
 }  
 }  
  
 @Override  
 **public void** bombsGeneration() {  
 Random random = **new** Random();  
 **int** sumBombs = sumBombs();  
 **while** (sumBombs > 0) {  
 **int** row = random.nextInt(sumRow());  
 **int** column = random.nextInt(sumColumn());  
 **if** (!**this**.**cells**[row][column].isBomb() && !**this**.**cells**[row][column].isSuggestEmpty()) {  
 **this**.**cells**[row][column] = **new** ConsoleCell(**true**);  
 sumBombs--;  
 }  
 }  
 }  
  
 @Override  
 **public int** checkingAroundCell(**int** x, **int** y) {  
 **int** checking = 0;  
  
 **if** (y > 0 && **cells**[x][y - 1].isBomb()) checking++;  
 **if** (x > 0 && **cells**[x - 1][y].isBomb()) checking++;  
 **if** (y > 0 && x > 0 && **cells**[x - 1][y - 1].isBomb()) checking++;  
 **if** (y + 1 < sumColumn() && **cells**[x][y + 1].isBomb()) checking++;  
 **if** (x + 1 < sumRow() && **cells**[x + 1][y].isBomb()) checking++;  
 **if** (x + 1 < sumRow() && y + 1 < sumColumn() && **cells**[x + 1][y + 1].isBomb()) checking++;  
 **if** (x + 1 < sumRow() && y > 0 && **cells**[x + 1][y - 1].isBomb()) checking++;  
 **if** (x > 0 && y + 1 < sumColumn() && **cells**[x - 1][y + 1].isBomb()) checking++;  
  
 **return** checking;  
 }  
  
 @Override  
 **public void** openEmptyCells() {  
 **int** check = 1, sumEmpty = 0;  
  
 **while** (check != sumEmpty) {  
 check = sumEmpty;  
 *// Проходим поле и проверяем пустые ячейки* **for** (**int** i = 0; i < sumRow(); i++)  
 **for** (**int** j = 0; j < sumColumn(); j++) {  
 *// Если ячейка пустая и мы её ещё не проверяли* **if** (**cells**[i][j].isSuggestEmpty() && !**cells**[i][j].isChecked()) {  
  
 *// Если возле ячейки нет бомб* **if** (checkingAroundCell(i, j) == 0) {  
 sumEmpty++;  
 *// Открываем рядом стоящие ячейки* clearAroundCell(i, j);  
 *// Помечаем данную ячейку как просмотренную* **cells**[i][j].checked();  
 }  
  
 *// Выводим данную ячейку* **switch** (checkingAroundCell(i, j)) {  
 **case** 8: **this**.**cells**[i][j].suggest8();  
 **break**;  
 **case** 7: **this**.**cells**[i][j].suggest7();  
 **break**;  
 **case** 6: **this**.**cells**[i][j].suggest6();  
 **break**;  
 **case** 5: **this**.**cells**[i][j].suggest5();  
 **break**;  
 **case** 4: **this**.**cells**[i][j].suggest4();  
 **break**;  
 **case** 3: **this**.**cells**[i][j].suggest3();  
 **break**;  
 **case** 2: **this**.**cells**[i][j].suggest2();  
 **break**;  
 **case** 1: **this**.**cells**[i][j].suggest1();  
 **break**;  
 **default**: suggest(i, j, **false**);  
 }  
 }  
 }  
 }  
 }}

**ConsoleBoard.java:**

import ru.minesweeperLesson.minesweeper.interfaces.IBoard;  
import ru.minesweeperLesson.minesweeper.interfaces.ICell;  
  
  
*/\*\*  
 \* Реализация поля в консоли  
 \*/***public class** ConsoleBoard **implements** IBoard {  
  
 **private** ICell[][] **cells**;  
  
 @Override  
 **public int** returnSumBomb() {  
 **return** 0;  
 }  
  
 @Override  
 **public void** cancelSuggestBomb(**int** x, **int** y) {  
 }  
  
 @Override  
 **public boolean** returnSuggestBomb(**int** x, **int** y) {  
 **return false**;  
 }  
  
 @Override  
 **public void** drawBoard(ICell[][] cells) {  
 **this**.**cells** = cells;  
 **this**.redraw(**false**);  
 }  
  
 @Override  
 **public void** drawCell(**int** x, **int** y) {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*\*\* ВЫБОР \*\*\*\*\*\*\*\*\*\*"**);  
 **this**.redraw(**false**);  
 }  
  
 @Override  
 **public void** drawBang() {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*\*\* БУМ \*\*\*\*\*\*\*\*\*\*"**);  
 **this**.redraw(**true**);  
 }  
  
 @Override  
 **public void** drawCongratulate() {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*\*\* ВЫ ВЫИГРАЛИ! \*\*\*\*\*\*\*\*\*\*"**);  
 }  
  
 @Override  
 **public void** drawLosing() {  
 System.***out***.println(**"\*\*\*\*\*\*\*\*\*\* ВЫ ПРОИГРАЛИ! \*\*\*\*\*\*\*\*\*\*"**);  
 }  
  
 **private void** redraw(**boolean** real) {  
 **int** space, numRow = 0;  
 **for** (ICell[] row : **cells**) {  
 *// Нумерация столбцов* **if** (numRow < 1) {  
 **for** (**int** i = 1; i <= row.**length**; i++) {  
 **if** (i == 1) space = 4;  
 **else if** (i > 10) space = 2;  
 **else** space = 3;  
 **for** (**int** j = 0; j < space; j++)  
 System.***out***.print(**" "**);  
 System.***out***.print(i);  
 }  
 System.***out***.println();  
 }  
 *// Нумерация рядов* **if** (numRow < 9) System.***out***.print(**" "**);  
 System.***out***.print(++numRow + **" "**);  
  
 **for** (ICell cell : row) {  
 cell.draw(System.***out***, real);  
 }  
 System.***out***.println();  
 }  
 System.***out***.println();  
 }  
}

**ConsoleCell.java:**

**import** ru.minesweeperLesson.minesweeper.interfaces.ICell;  
  
**import** java.io.PrintStream;  
  
*/\*\*  
 \* Реализация ячейки в консоли  
 \*/  
  
// PrintStream - входящий поток, какой реализует нашу консоль***public class** ConsoleCell **implements** ICell<PrintStream> {  
  
 **private boolean bomb**;  
 **private boolean suggestBomb** = **false**;  
 **private boolean suggestEmpty** = **false**;  
  
 **private boolean checked** = **false**;  
  
 **private boolean suggest1** = **false**;  
 **private boolean suggest2** = **false**;  
 **private boolean suggest3** = **false**;  
 **private boolean suggest4** = **false**;  
 **private boolean suggest5** = **false**;  
 **private boolean suggest6** = **false**;  
 **private boolean suggest7** = **false**;  
 **private boolean suggest8** = **false**;  
  
 **public** ConsoleCell(**boolean** bomb) {  
 **this**.**bomb** = bomb;  
 }  
  
 @Override  
 **public boolean** isBomb() {  
 **return this**.**bomb**;  
 }  
  
 @Override  
 **public boolean** isSuggestBomb() {  
 **return this**.**suggestBomb**;  
 }  
  
 @Override  
 **public boolean** isSuggestEmpty() {  
 **return this**.**suggestEmpty**;  
 }  
  
 @Override  
 **public void** suggestEmpty() {  
 **this**.**suggestEmpty** = **true**;  
 }  
  
 @Override  
 **public void** suggestBomb() {  
 **this**.**suggestBomb** = **true**;  
 }  
  
 @Override  
 **public void** cancelSuggestBomb() {  
  
 }  
  
 @Override  
 **public boolean** isChecked() {  
 **return this**.**checked**;  
 }  
  
 @Override  
 **public void** checked() {  
 **this**.**checked** = **true**;  
 }  
  
 @Override  
 **public boolean** isSuggest1() {  
 **return this**.**suggest1**;  
 }  
  
 @Override  
 **public boolean** isSuggest2() {  
 **return this**.**suggest2**;  
 }  
  
 @Override  
 **public boolean** isSuggest3() {  
 **return this**.**suggest3**;  
 }  
  
 @Override  
 **public boolean** isSuggest4() {  
 **return this**.**suggest4**;  
 }  
  
 @Override  
 **public boolean** isSuggest5() {  
 **return this**.**suggest5**;  
 }  
  
 @Override  
 **public boolean** isSuggest6() {  
 **return this**.**suggest6**;  
 }  
  
 @Override  
 **public boolean** isSuggest7() {  
 **return this**.**suggest7**;  
 }  
  
 @Override  
 **public boolean** isSuggest8() {  
 **return this**.**suggest8**;  
 }  
  
 @Override  
 **public void** suggest1() {  
 **this**.**suggest1** = **true**;  
 }  
  
 @Override  
 **public void** suggest2() {  
 **this**.**suggest2** = **true**;  
 }  
  
 @Override  
 **public void** suggest3() {  
 **this**.**suggest3** = **true**;  
 }  
  
 @Override  
 **public void** suggest4() {  
 **this**.**suggest4** = **true**;  
 }  
  
 @Override  
 **public void** suggest5() {  
 **this**.**suggest5** = **true**;  
 }  
  
 @Override  
 **public void** suggest6() {  
 **this**.**suggest6** = **true**;  
 }  
  
 @Override  
 **public void** suggest7() {  
 **this**.**suggest7** = **true**;  
 }  
  
 @Override  
 **public void** suggest8() {  
 **this**.**suggest8** = **true**;  
 }  
  
 @Override  
 **public void** draw(PrintStream paint, **boolean** real) {  
 **if** (real)  
 **if** (**this**.isBomb())  
 paint.print(**"[\033[1;31m\*\033[0m] "**);  
 **else** paint.print(**"[ ] "**);  
 **else** {  
 **if** (**this**.**suggestBomb**)  
 paint.print(**"[\033[1;31m?\033[0m] "**);  
  
 **else if** (**this**.**suggest8**)  
 paint.print(**"[\033[1;31m8\033[0m] "**);  
 **else if** (**this**.**suggest7**)  
 paint.print(**"[\033[1;31m7\033[0m] "**);  
 **else if** (**this**.**suggest6**)  
 paint.print(**"[\033[1;35m6\033[0m] "**);  
 **else if** (**this**.**suggest5**)  
 paint.print(**"[\033[1;36m5\033[0m] "**);  
 **else if** (**this**.**suggest4**)  
 paint.print(**"[\033[1;34m4\033[0m] "**);  
 **else if** (**this**.**suggest3**)  
 paint.print(**"[\033[1;32m3\033[0m] "**);  
 **else if** (**this**.**suggest2**)  
 paint.print(**"[\033[1;33m2\033[0m] "**);  
 **else if** (**this**.**suggest1**)  
 paint.print(**"[\033[1;30m1\033[0m] "**);  
  
 **else if** (**this**.**suggestEmpty**)  
 paint.print(**"[ ] "**);  
 **else** paint.print(**"[Х] "**);  
 }  
 }  
  
 @Override  
 **public void** draw(PrintStream graphics, **int** x, **int** y, **boolean** real) {  
 }  
}

**Runner.java:**

**import** ru.minesweeperLesson.minesweeper.BaseAction;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ICell;  
**import** ru.minesweeperLesson.minesweeper.interfaces.IGeneratorBoard;  
**import** ru.minesweeperLesson.minesweeper.StandardLogicConsole;  
  
**import** java.util.InputMismatchException;  
**import** java.util.Objects;  
**import** java.util.Scanner;  
  
*/\*\*  
 \* Запуск игры в консольной версии  
 \*/***public class** Runner {  
  
 *// Генерация поля* **public void** generateField(Scanner sc, StandardLogicConsole level, ICell[][] cells) {  
 **final** BaseAction action = **new** BaseAction(level, **new** ConsoleBoard(), **new** IGeneratorBoard() {  
 @Override  
 **public** ICell[][] generate() {  
 generateEmptyCells(level, cells);  
 System.***out***.println(**"Бомб: "** + level.sumBombs());  
 System.***out***.println(**"Пустых клеток: "** + ((level.sumRow() \* level.sumColumn()) - level.sumBombs()));  
 **return** cells;  
 }  
 });  
 action.initGame();  
 cycleInput(sc, action, level);  
 }  
  
 *// При первом ходе делаем все клетки пустыми* **public void** generateEmptyCells(StandardLogicConsole level, ICell[][] cells) {  
 **for** (**int** i = 0; i < level.sumRow(); i++)  
 **for** (**int** j = 0; j < level.sumColumn(); j++)  
 cells[i][j] = **new** ConsoleCell(**false**);  
 }  
  
 *// Вводим ряд и столбец. Работает до тех пор, пока пользователь не выиграет или не проиграет* **public void** cycleInput(Scanner sc, BaseAction action, StandardLogicConsole level) {  
 **int** row = 0, column = 0;  
 **try** {  
 **do** {  
 **try** {  
 System.***out***.print(**"Введите номер ряда: "**);  
 row = sc.nextInt();  
 System.***out***.print(**"Введите номер столбца: "**);  
 column = sc.nextInt();  
 System.***out***.print(**"Там бомба (да/нет)? "**);  
 } **catch** (InputMismatchException ignored) {}  
 String answer = sc.next();  
 System.***out***.println();  
 **if** (row - 1 < level.sumRow() && column - 1 < level.sumColumn() && row - 1 >= 0 && column - 1 >= 0) {  
 **if** (Objects.*equals*(answer, **"да"**))  
 action.select(row - 1, column - 1, **true**);  
 **else if** (Objects.*equals*(answer, **"нет"**))  
 action.select(row - 1, column - 1, **false**);  
 *// Создаем исключение, если пользователь ввел некорректные данные* } **else throw new** ArrayIndexOutOfBoundsException(**"Такого ряда или столбца не существует!\n"**);  
 } **while** (!level.finish() && !level.shouldBang(row - 1, column - 1));  
 *// Перехватываем и выводим исключение* } **catch** (ArrayIndexOutOfBoundsException e) {  
 System.***out***.println(e.getMessage());  
 cycleInput(sc, action, level);  
 }  
 }  
  
 */\*\*  
 \* Выбираем уровень и запускаем генерацию поля  
 \*/* **public static void** main(String[] args) {  
 Runner runner = **new** Runner();  
 StandardLogicConsole level = **new** StandardLogicConsole();  
 **try** (Scanner sc = **new** Scanner(System.***in***)) {  
 System.***out***.println(  
 **"\n<-- ВВЕДИТЕ ЦИФРУ -->\n\n"** + **"Выберите уровень: "** + **"\n 1. Легкий (9x9)"** + **"\n 2. Средний (16x16)"** + **"\n 3. Эксперт (30x16)"**);  
 System.***out***.print(**"Ваш выбор: "**);  
 **int** answer = sc.nextInt();  
 **switch** (answer) {  
 **case** 1: level.easy();  
 **break**;  
 **case** 2: level.medium();  
 **break**;  
 **case** 3: level.expert();  
 **break**;  
 **default**:  
 System.***out***.println(**"\nВы не выбрали уровень. По умолчанию запущен \"Легкий\"!\n"**);  
 level.easy();  
 }  
 ICell[][] cells = level.sizeField();  
 runner.generateField(sc, level, cells);  
 } **catch** (InputMismatchException ime) {  
 System.***out***.println(ime.getMessage());  
 }  
 }  
}

Класи графічної версії гри:

**StandardLogicGUI:**

**import** ru.minesweeperLesson.minesweeper.gui.GUICell;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ICell;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ILogic;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ISelectLevel;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ITheNumOfTheField;  
**import** ru.minesweeperLesson.minesweeper.levels.Easy;  
**import** ru.minesweeperLesson.minesweeper.levels.Expert;  
**import** ru.minesweeperLesson.minesweeper.levels.Medium;  
  
**import** java.util.Random;  
  
*/\*\*  
 \* Стандартная логика игры  
 \*/***public class** StandardLogicGUI **implements** ISelectLevel, ILogic, ITheNumOfTheField {  
  
 **private** Easy **easy**;  
 **private** Medium **medium**;  
 **private** Expert **expert**;  
  
 **private** ICell[][] **cells**;  
  
 @Override  
 **public** Easy easy() {  
 **medium** = **null**;  
 **expert** = **null**;  
 **return easy** = **new** Easy();  
 }  
  
 @Override  
 **public** Medium medium() {  
 **easy** = **null**;  
 **expert** = **null**;  
 **return medium** = **new** Medium();  
 }  
  
 @Override  
 **public** Expert expert() {  
 **easy** = **null**;  
 **medium** = **null**;  
 **return expert** = **new** Expert();  
 }  
  
 @Override  
 **public void** loadBoard(ICell[][] cells) {  
 **this**.**cells** = cells;  
 }  
  
 @Override  
 **public int** sumBombs() {  
 **int** sum = 0;  
 **if** (**easy** == **null** && **medium** == **null** && **expert** != **null**)  
 sum = **expert**.sumBombs();  
 **else if** (**easy** == **null** && **expert** == **null** && **medium** != **null**)  
 sum = **medium**.sumBombs();  
 **else if** (**easy** != **null**)  
 sum = **easy**.sumBombs();  
 **return** sum;  
 }  
  
 @Override  
 **public int** sumRow() {  
 **int** row = 0;  
 **if** (**easy** == **null** && **medium** == **null** && **expert** != **null**)  
 row = **expert**.sumRow();  
 **else if** (**easy** == **null** && **expert** == **null** && **medium** != **null**)  
 row = **medium**.sumRow();  
 **else if** (**easy** != **null**)  
 row = **easy**.sumRow();  
 **return** row;  
 }  
  
 @Override  
 **public int** sumColumn() {  
 **int** column = 0;  
 **if** (**easy** == **null** && **medium** == **null** && **expert** != **null**)  
 column = **expert**.sumColumn();  
 **else if** (**easy** == **null** && **expert** == **null** && **medium** != **null**)  
 column = **medium**.sumColumn();  
 **else if** (**easy** != **null**)  
 column = **easy**.sumColumn();  
 **return** column;  
 }  
  
 @Override  
 **public** ICell[][] sizeField() {  
 **return new** ICell[sumRow()][sumColumn()];  
 }  
  
 @Override  
 **public boolean** shouldBang(**int** x, **int** y) {  
 **final** ICell selected = **this**.**cells**[x][y];  
 *// Если это бомба, и пользователь не предположил что это бомба, то мы взрываемся* **return** selected.isBomb() && !selected.isSuggestBomb();  
 }  
  
 @Override  
 **public boolean** finish() {  
 **boolean** finish = **false**;  
 **int** check = 0;  
 **for** (ICell[] row : **this**.**cells**)  
 **for** (ICell cell : row)  
 **if** ((cell.isSuggestBomb() && cell.isBomb()) ||  
 (cell.isSuggestEmpty() && !cell.isBomb()) */\*|| (!cell.isSuggestBomb() && cell.isBomb())\*/* || cell.isSuggest1() || cell.isSuggest2() || cell.isSuggest3() || cell.isSuggest4()  
 || cell.isSuggest5() || cell.isSuggest6() || cell.isSuggest7() || cell.isSuggest8()) {  
 check++;  
 }  
 **if** (check == (sumRow() \* sumColumn()))  
 finish = **true**;  
 **return** finish;  
 }  
  
 @Override  
 **public void** suggest(**int** x, **int** y, **boolean** bomb) {  
 **if** (!bomb)  
 **this**.**cells**[x][y].suggestEmpty();  
 **if** (bomb && !**this**.**cells**[x][y].isSuggestEmpty())  
 **this**.**cells**[x][y].suggestBomb();  
 **else if** (bomb && **this**.**cells**[x][y].isSuggestEmpty())  
 System.***out***.println(**"Вы уже открыли эту клетку!\n"**);  
 }  
  
 @Override  
 **public boolean** checkTheFirstMove() {  
 **boolean** check = **true**;  
 root: **for** (**int** i = 0; i < sumRow(); i++)  
 **for** (**int** j = 0; j < sumColumn(); j++)  
 **if** (**this**.**cells**[i][j].isBomb()) {  
 check = **false**;  
 **break** root;  
 }  
 **return** check;  
 }  
  
 @Override  
 **public void** clearAroundCell(**int** x, **int** y) {  
 **if** (**cells**.**length** > 3) {  
 **if** (y > 0) suggest(x, y - 1, **false**);  
 **if** (y + 1 < sumColumn()) suggest(x, y + 1, **false**);  
 **if** (x > 0) suggest(x - 1, y, **false**);  
 **if** (x + 1 < sumRow()) suggest(x + 1, y, **false**);  
 **if** (x > 0 && y > 0) suggest(x - 1, y - 1, **false**);  
 **if** (x + 1 < sumRow() && y + 1 < sumColumn()) suggest(x + 1, y + 1, **false**);  
 **if** (x > 0 && y + 1 < sumColumn()) suggest(x - 1, y + 1, **false**);  
 **if** (x + 1 < sumRow() && y > 0) suggest(x + 1, y - 1, **false**);  
 }  
 }  
  
 @Override  
 **public void** bombsGeneration() {  
 Random random = **new** Random();  
 **int** sumBombs = sumBombs();  
 **while** (sumBombs > 0) {  
 **int** row = random.nextInt(sumRow());  
 **int** column = random.nextInt(sumColumn());  
 **if** (!**this**.**cells**[row][column].isBomb() && !**this**.**cells**[row][column].isSuggestEmpty()) {  
 **this**.**cells**[row][column] = **new** GUICell(**true**);  
 sumBombs--;  
 }  
 }  
 }  
  
 @Override  
 **public int** checkingAroundCell(**int** x, **int** y) {  
 **int** checking = 0;  
  
 **if** (y > 0 && **cells**[x][y - 1].isBomb()) checking++;  
 **if** (x > 0 && **cells**[x - 1][y].isBomb()) checking++;  
 **if** (y > 0 && x > 0 && **cells**[x - 1][y - 1].isBomb()) checking++;  
 **if** (y + 1 < sumColumn() && **cells**[x][y + 1].isBomb()) checking++;  
 **if** (x + 1 < sumRow() && **cells**[x + 1][y].isBomb()) checking++;  
 **if** (x + 1 < sumRow() && y + 1 < sumColumn() && **cells**[x + 1][y + 1].isBomb()) checking++;  
 **if** (x + 1 < sumRow() && y > 0 && **cells**[x + 1][y - 1].isBomb()) checking++;  
 **if** (x > 0 && y + 1 < sumColumn() && **cells**[x - 1][y + 1].isBomb()) checking++;  
  
 **return** checking;  
 }  
  
 @Override  
 **public void** openEmptyCells() {  
 **int** check = 1, sumEmpty = 0;  
  
 **while** (check != sumEmpty) {  
 check = sumEmpty;  
 *// Проходим поле и проверяем пустые ячейки* **for** (**int** i = 0; i < sumRow(); i++)  
 **for** (**int** j = 0; j < sumColumn(); j++) {  
 *// Если ячейка пустая и мы её ещё не проверяли* **if** (**cells**[i][j].isSuggestEmpty() && !**cells**[i][j].isChecked()) {  
  
 *// Если возле ячейки нет бомб* **if** (checkingAroundCell(i, j) == 0) {  
 sumEmpty++;  
 *// Открываем рядом стоящие ячейки* clearAroundCell(i, j);  
 *// Помечаем данную ячейку как просмотренную* **cells**[i][j].checked();  
 }  
  
 *// Выводим данную ячейку* **switch** (checkingAroundCell(i, j)) {  
 **case** 8: **this**.**cells**[i][j].suggest8();  
 **break**;  
 **case** 7: **this**.**cells**[i][j].suggest7();  
 **break**;  
 **case** 6: **this**.**cells**[i][j].suggest6();  
 **break**;  
 **case** 5: **this**.**cells**[i][j].suggest5();  
 **break**;  
 **case** 4: **this**.**cells**[i][j].suggest4();  
 **break**;  
 **case** 3: **this**.**cells**[i][j].suggest3();  
 **break**;  
 **case** 2: **this**.**cells**[i][j].suggest2();  
 **break**;  
 **case** 1: **this**.**cells**[i][j].suggest1();  
 **break**;  
 **default**: suggest(i, j, **false**);  
 }  
 }  
 }  
 }  
 }  
}

**GUIAction.java:**

**import** ru.minesweeperLesson.minesweeper.BaseAction;  
**import** ru.minesweeperLesson.minesweeper.interfaces.IGeneratorBoard;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ILogic;  
  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.MouseEvent;  
**import** java.awt.event.MouseListener;  
  
**public class** GUIAction **extends** BaseAction **implements** ActionListener, MouseListener {  
  
 **private** GUIBoard **board**;  
  
 **public** GUIAction(ILogic logic, GUIBoard board, IGeneratorBoard generator) {  
 **super**(logic, board, generator);  
 **this**.**board** = board;  
 **this**.**board**.addMouseListener(**this**);  
 }  
  
 **public void** actionPerformed(ActionEvent e) {  
 **this**.initGame();  
 }  
  
 **public int** returnX(MouseEvent e) {  
 **return** e.getX() / GUIBoard.*getPADDING*();  
 }  
  
 **public int** returnY(MouseEvent e) {  
 **return** e.getY() / GUIBoard.*getPADDING*();  
 }  
  
 **public void** mouseClicked(MouseEvent e) {  
 System.***out***.println(**"X: "** + returnX(e) + **" Y: "** + returnY(e));  
 System.***out***.println();  
  
 *// Если ячейка уже помечена как бомба и пользователь говорит что в ячейке нет бомбы* **if** (**board**.returnSuggestBomb(returnX(e), returnY(e)) && e.getButton() != 1) {  
 **board**.cancelSuggestBomb(returnX(e), returnY(e));  
 } **else  
 if** (**board**.returnSuggestBomb(returnX(e), returnY(e)))  
 select(returnX(e), returnY(e), **true**);  
 **else** select(returnX(e), returnY(e), e.getButton() != 1);  
 **board**.repaint();  
 }  
  
 **public void** mousePressed(MouseEvent e) {  
 }  
  
 **public void** mouseReleased(MouseEvent e) {  
 }  
  
 **public void** mouseEntered(MouseEvent e) {  
 }  
  
 **public void** mouseExited(MouseEvent e) {  
 }  
}

**GUIBoard.java:**

**import** ru.minesweeperLesson.minesweeper.interfaces.IBoard;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ICell;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** GUIBoard **extends** JPanel **implements** IBoard {  
  
 **public static final int *PADDING*** = 40;  
  
 **public boolean real** = **false**;  
  
 **public** ICell<Graphics>[][] **cells**;  
  
 **public static int** getPADDING() {  
 **return *PADDING***;  
 }  
  
 @Override  
 **public int** returnSumBomb() {  
 **int** sumBomb = 0;  
 **for** (**int** x = 0; x != **cells**.**length**; x++) {  
 **for** (**int** y = 0; y != **cells**[0].**length**; y++) {  
 **if** (**cells**[x][y].isSuggestBomb()) sumBomb++;  
 }  
 }  
 **return** sumBomb;  
 }  
  
 @Override  
 **public void** cancelSuggestBomb(**int** x, **int** y) {  
 **this**.**cells**[x][y].cancelSuggestBomb();  
 }  
  
 @Override  
 **public boolean** returnSuggestBomb(**int** x, **int** y) {  
 **return this**.**cells**[x][y].isSuggestBomb();  
 }  
  
 @Override  
 **protected void** paintComponent(Graphics graphics) {  
 **super**.paintComponent(graphics);  
 **if** (**this**.**cells** != **null**) {  
 **for** (**int** x = 0; x != **cells**.**length**; x++) {  
 **for** (**int** y = 0; y != **cells**[0].**length**; y++) {  
 **cells**[x][y].draw(graphics, x, y, **real**);  
 }  
 }  
 }  
 }  
  
 @Override  
 **public void** drawBoard(ICell[][] cells) {  
 **this**.**cells** = cells;  
 **real** = **false**;  
 **this**.repaint();  
 }  
  
 @Override  
 **public void** drawCell(**int** x, **int** y) {  
 **this**.repaint();  
 }  
  
 @Override  
 **public void** drawBang() {  
 **real** = **true**;  
 **this**.repaint();  
 }  
  
 @Override  
 **public void** drawCongratulate() {  
 **this**.repaint();  
 Main.*setLosing*(**"Нажмите 'OK', чтобы закрыть окно."**, **" Поздравляем! Вы выиграли!"**);  
 }

@Override  
 **public void** drawLosing() {  
 **this**.repaint();  
 Main.*setLosing*(**"Нажмите 'OK', чтобы закрыть окно."**, **" К сожалению Вы проиграли"**);  
 }  
}

**GUICell.java:**

**import** ru.minesweeperLesson.minesweeper.interfaces.ICell;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** GUICell **extends** JPanel **implements** ICell<Graphics> {  
  
 **private boolean bomb**;  
 **private boolean suggestBomb** = **false**;  
 **private boolean suggestEmpty** = **false**;  
  
 **private boolean checked** = **false**;  
  
 **private boolean suggest1** = **false**;  
 **private boolean suggest2** = **false**;  
 **private boolean suggest3** = **false**;  
 **private boolean suggest4** = **false**;  
 **private boolean suggest5** = **false**;  
 **private boolean suggest6** = **false**;  
 **private boolean suggest7** = **false**;  
 **private boolean suggest8** = **false**;  
  
 **public** GUICell(**boolean** bomb) {  
 **this**.**bomb** = bomb;  
 }  
  
 @Override  
 **public boolean** isBomb() {  
 **return this**.**bomb**;  
 }  
  
 @Override  
 **public boolean** isSuggestBomb() {  
 **return this**.**suggestBomb**;  
 }  
  
 @Override  
 **public boolean** isSuggestEmpty() {  
 **return this**.**suggestEmpty**;  
 }  
  
 @Override  
 **public void** suggestEmpty() {  
 **this**.**suggestEmpty** = **true**;  
 }  
  
 @Override  
 **public void** suggestBomb() {  
 **this**.**suggestBomb** = **true**;  
 }  
  
 @Override  
 **public void** cancelSuggestBomb() {  
 **this**.**suggestBomb** = **false**;  
 }  
  
 @Override  
 **public boolean** isChecked() {  
 **return this**.**checked**;  
 }  
  
 @Override  
 **public void** checked() {  
 **this**.**checked** = **true**;  
 }  
  
 @Override  
 **public boolean** isSuggest1() {  
 **return this**.**suggest1**;  
 }  
  
 @Override  
 **public boolean** isSuggest2() {  
 **return this**.**suggest2**;  
 }  
  
 @Override  
 **public boolean** isSuggest3() {  
 **return this**.**suggest3**;  
 }  
  
 @Override  
 **public boolean** isSuggest4() {  
 **return this**.**suggest4**;  
 }  
  
 @Override  
 **public boolean** isSuggest5() {  
 **return this**.**suggest5**;  
 }  
  
 @Override  
 **public boolean** isSuggest6() {  
 **return this**.**suggest6**;  
 }  
  
 @Override  
 **public boolean** isSuggest7() {  
 **return this**.**suggest7**;  
 }  
  
 @Override  
 **public boolean** isSuggest8() {  
 **return this**.**suggest8**;  
 }  
  
 @Override  
 **public void** suggest1() {  
 **this**.**suggest1** = **true**;  
 }  
  
 @Override  
 **public void** suggest2() {  
 **this**.**suggest2** = **true**;  
 }  
  
 @Override  
 **public void** suggest3() {  
 **this**.**suggest3** = **true**;  
 }  
  
 @Override  
 **public void** suggest4() {  
 **this**.**suggest4** = **true**;  
 }  
  
 @Override  
 **public void** suggest5() {  
 **this**.**suggest5** = **true**;  
 }  
  
 @Override  
 **public void** suggest6() {  
 **this**.**suggest6** = **true**;  
 }  
  
 @Override  
 **public void** suggest7() {  
 **this**.**suggest7** = **true**;  
 }  
  
 @Override  
 **public void** suggest8() {  
 **this**.**suggest8** = **true**;  
 }  
  
 @Override  
 **public void** draw(Graphics paint, **boolean** real) {  
 }  
  
 @Override  
 **public void** draw(Graphics paint, **int** x, **int** y, **boolean** real) {  
 paint.setFont(**new** Font(**"Comic Sans MS"**, Font.***BOLD***, 17));  
  
 paint.setColor(**new** Color(187,187,187));  
 paint.drawRect(x \* GUIBoard.*getPADDING*(), y \* GUIBoard.*getPADDING*(),  
 GUIBoard.*getPADDING*(), GUIBoard.*getPADDING*());  
  
 x = x \* GUIBoard.*getPADDING*() + 16;  
 y = y \* GUIBoard.*getPADDING*() + 25;  
  
 **if** (real) {  
 **if** (**this**.isBomb()) {  
 paint.setFont(**new** Font(**"Comic Sans MS"**, Font.***BOLD***, 35));  
 paint.setColor(Color.***red***);  
 paint.drawString(**"\*"**, x - 4, y + 15);  
 }  
  
  
  
 **else if** (**this**.**suggest8**) {  
 paint.setColor(Color.***black***);  
 paint.drawString(**"8"**, x, y);  
 }  
 **else if** (**this**.**suggest7**) {  
 paint.setColor(Color.***black***);  
 paint.drawString(**"7"**, x, y);  
 }  
 **else if** (**this**.**suggest6**) {  
 paint.setColor(Color.***black***);  
 paint.drawString(**"6"**, x, y);  
 }  
 **else if** (**this**.**suggest5**) {  
 paint.setColor(**new** Color(136,96,44));  
 paint.drawString(**"5"**, x, y);  
 }  
 **else if** (**this**.**suggest4**) {  
 paint.setColor(**new** Color(34,44,118));  
 paint.drawString(**"4"**, x, y);  
 }  
 **else if** (**this**.**suggest3**) {  
 paint.setColor(**new** Color(255,0,0));  
 paint.drawString(**"3"**, x, y);  
 }  
 **else if** (**this**.**suggest2**) {  
 paint.setColor(**new** Color(45,146,40));  
 paint.drawString(**"2"**, x, y);  
 }  
 **else if** (**this**.**suggest1**) {  
 paint.setColor(**new** Color(0,13,255));  
 paint.drawString(**"1"**, x, y);  
 }  
  
 } **else** {  
 **if** (**this**.**suggestBomb**) {  
 paint.drawLine(x - 16, y - 25, x + 25, y + 15);  
 paint.drawLine(x + 25, y - 25, x - 16, y + 15);  
 paint.setColor(Color.***black***);  
 paint.drawString(**"?"**, x, y);  
 }  
  
 **else if** (**this**.**suggest8**) {  
 paint.setColor(Color.***black***);  
 paint.drawString(**"8"**, x, y);  
 }  
 **else if** (**this**.**suggest7**) {  
 paint.setColor(Color.***black***);  
 paint.drawString(**"7"**, x, y);  
 }  
 **else if** (**this**.**suggest6**) {  
 paint.setColor(Color.***black***);  
 paint.drawString(**"6"**, x, y);  
 }  
 **else if** (**this**.**suggest5**) {  
 paint.setColor(**new** Color(136,96,44));  
 paint.drawString(**"5"**, x, y);  
 }  
 **else if** (**this**.**suggest4**) {  
 paint.setColor(**new** Color(34,44,118));  
 paint.drawString(**"4"**, x, y);  
 }  
 **else if** (**this**.**suggest3**) {  
 paint.setColor(**new** Color(255,0,0));  
 paint.drawString(**"3"**, x, y);  
 }  
 **else if** (**this**.**suggest2**) {  
 paint.setColor(**new** Color(45,146,40));  
 paint.drawString(**"2"**, x, y);  
 }  
 **else if** (**this**.**suggest1**) {  
 paint.setColor(**new** Color(0,13,255));  
 paint.drawString(**"1"**, x, y);  
 }  
  
 **else if** (**this**.**suggestEmpty**) {  
 paint.drawString(**""**, x - 2, y);  
 }  
  
 **else** {  
 paint.drawLine(x - 16, y - 25, x + 25, y + 15);  
 paint.drawLine(x + 25, y - 25, x - 16, y + 15);  
 }  
 }  
 }  
}

**Main.java:**

**import** ru.minesweeperLesson.minesweeper.StandardLogicGUI;  
**import** ru.minesweeperLesson.minesweeper.interfaces.ICell;  
**import** ru.minesweeperLesson.minesweeper.interfaces.IGeneratorBoard;  
  
**import** javax.swing.\*;  
**import** javax.swing.border.EmptyBorder;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.WindowAdapter;  
**import** java.awt.event.WindowEvent;  
  
*/\*\*  
 \* Запуск игры в графической версии  
 \*/***public class** Main {  
 **private static final** JPanel ***controlPanel*** = **new** JPanel();  
 **private static** JLabel *label* = **new** JLabel(**""**);  
 **private static final** GUIBoard ***board*** = **new** GUIBoard();  
  
 **public static void** main(String[] arg) {  
 SwingUtilities.*invokeLater*(**new** Runnable() {  
 **public void** run() {  
 **final** JFrame frame = **new** JFrame();  
 frame.setTitle(**"Сапер"**);  
 frame.setLayout(**new** BorderLayout());  
 frame.setSize(500, 500);  
 frame.add(***board***, BorderLayout.***CENTER***);  
 ***board***.setBorder(**new** EmptyBorder(10, 10, 10, 10));  
 frame.add(***controlPanel***, BorderLayout.***PAGE\_END***);  
  
 *label*.setVerticalAlignment(SwingConstants.***TOP***);  
 frame.add(*label*, BorderLayout.***EAST***);  
  
 ***controlPanel***.setLayout(**new** FlowLayout());  
 **final** JButton generate = **new** JButton(**"Начать"**);  
 StandardLogicGUI level = **new** StandardLogicGUI();  
 level.easy();  
 */\*\*  
 \* ПЕРЕКЛЮЧЕНИЕ УРОВНЕЙ.  
 \*/* JMenuBar menuBar = **new** JMenuBar(); *// Верхняя полоска с элементами меню* frame.setJMenuBar(menuBar);  
 JMenu menu = **new** JMenu(**"Уровень"**); *// Меню* menuBar.add(menu);  
  
 JMenuItem easy = **new** JMenuItem(**"Легкий"**); *// Подменю* menu.add(easy);  
 easy.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 level.easy();  
 generate.doClick();  
 }  
 });  
  
 JMenuItem medium = **new** JMenuItem(**"Средний"**); *// Подменю* menu.add(medium);  
 medium.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 level.medium();  
 generate.doClick();  
 }  
 });  
  
 JMenuItem expert = **new** JMenuItem(**"Эксперт"**); *// Подменю* menu.add(expert);  
 expert.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 level.expert();  
 generate.doClick();  
 }  
 });  
  
 generate.addActionListener(**new** GUIAction (level, ***board***, **new** IGeneratorBoard() {  
 @Override  
 **public** ICell[][] generate() {  
 ICell[][] cells = level.sizeField();  
 *label*.setText(**"Флажки: 0 "**);  
 **for** (**int** i = 0; i < level.sumRow(); i++) {  
 **for** (**int** j = 0; j < level.sumColumn(); j++) {  
 cells[i][j] = **new** GUICell(**false**);  
 }  
 }  
 **return** cells;  
 }  
 }  
 ));  
 ***controlPanel***.add(generate);  
 *center*(frame);  
 frame.setDefaultCloseOperation(JFrame.***DO\_NOTHING\_ON\_CLOSE***);  
 frame.addWindowListener(**new** WindowAdapter() {  
 @Override  
 **public void** windowClosing(WindowEvent e) {  
 *closePerform*(frame);  
 }  
 });  
 frame.setVisible(**true**);  
 }  
 });  
 }  
  
 **public static void** center(Window w) {  
 Dimension us = w.getSize();  
 Dimension them = Toolkit.*getDefaultToolkit*().getScreenSize();  
 **int** newX = (them.**width** - us.**width**) / 2;  
 **int** newY = (them.**height** - us.**height**) / 2;  
 w.setLocation(newX, newY);  
 }  
  
 **public static void** closePerform(JFrame frame) {  
 frame.setVisible(**false**);  
 frame.dispose();  
 System.*exit*(0);  
 }  
  
 **public static void** setLabel(String textOnLabel) {  
 *label*.setText(textOnLabel);  
 *label*.repaint();  
 }  
  
 **public static void** setLosing(String body, String title) {  
 JOptionPane.*showMessageDialog*(**null**, body, title, JOptionPane.***INFORMATION\_MESSAGE***);  
 }  
}