Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

**ГРА 2048**

**Текстовий звіт з завдання з навчальної практики**

**за спеціальністю „Інженерія прграмного забезпечення”**

Викладач Кирієнко О.В.

Виконали студенти 1-го курсу

Фомін В.О. і Філь А.В.

Київ 2017

ЗМІСТ

[ПОСТАНОВКА ЗАДАЧІ ТА РОЗПОДІЛ РОЛЕЙ 2](#_Toc485064871)

[Опис завдання 2](#_Toc485064872)

[Розподіл Ролей 2](#_Toc485064873)

[СТРУКТУРА ПРОГРАМИ 3](#_Toc485064874)

[Діаграма класів 3](#_Toc485064875)

[Опис методів та класів. 4](#_Toc485064876)

[РОБОТА ПРОГРАМИ 8](#_Toc485064877)

[ІНСТРУКЦІЯ КОРИСТУВАЧА 11](#_Toc485064878)

[ЛІСТИНГ ПРОГРАМНОГО КОДУ 15](#_Toc485064879)

[GameWindow 15](#_Toc485064880)

[StartScreen 15](#_Toc485064881)

[SettingsScreen 17](#_Toc485064882)

[DifficultyLevel 19](#_Toc485064883)

[Resolution 19](#_Toc485064884)

[PlayScreen 19](#_Toc485064885)

[GameBoard 21](#_Toc485064886)

[Tile 35](#_Toc485064887)

[User 36](#_Toc485064888)

[UserReader 36](#_Toc485064889)

[Game 37](#_Toc485064890)

**СПИСОК ІЛЮСТРАЦІЙ**

**Рис. 1 Діаграма класів 3**

**Рис. 2 Титульний екран 8**

**Рис. 3 Вікно з детальною інформацією про програму 8**

**Рис. 4 Вікно налаштувань гри 9**

**Рис. 5 Старт гри 9**

**Рис. 6 Програш 10**

**Рис. 7 Стартовий екран при першому запуску 11**

**Рис. 8 Екран налаштувань 12**

**Рис. 9 Інформація про програму 12**

**Рис. 10 Екран запущеної гри 13**

# **ПОСТАНОВКА ЗАДАЧІ ТА РОЗПОДІЛ РОЛЕЙ**

## Опис завдання

Зробити гру, де гравець має пересувати клітину по ігровому полю якоїсь розмірності і об'єднувати їх до тих пір, поки число клітини не буде рівне якомусь значенню, яке залежить від обраного рівню складності. Гравцю дано деякий час на хід, значення якого теж залежить від рівня складності, якщо його буде вичерпано, гравець програє. Іншою умовою програшу є заповнення ігрового поля клітинами, які не можна об'єднати. Гравець має можливість встановити свій нікнейм, рівень складності й роздільну здатність у полі налаштувань. Гравець може перезапускати гру довільну кількість разів. При запуску або перезапуску гри на полі створюються дві клітини з числом 2 або 4. Якщо хоча б одну клітину було пересунуто або об’єднано, то таймер ходу оновлюється й створюється нова клітка.

## Розподіл Ролей

1. Cтворення ігрового поля – Фомін В.
2. Можливість рухати клітинки з числами, прописання умов виграшу/програшу – Фомін В.
3. Регулювання рівню складності, збереження результату – Фомін В.
4. Графічний інтерфейс користувача, дизайн – Філь А.
5. Звіт до роботи – Фомін В., Філь А.

# 

# **СТРУКТУРА ПРОГРАМИ**

## Діаграма класів

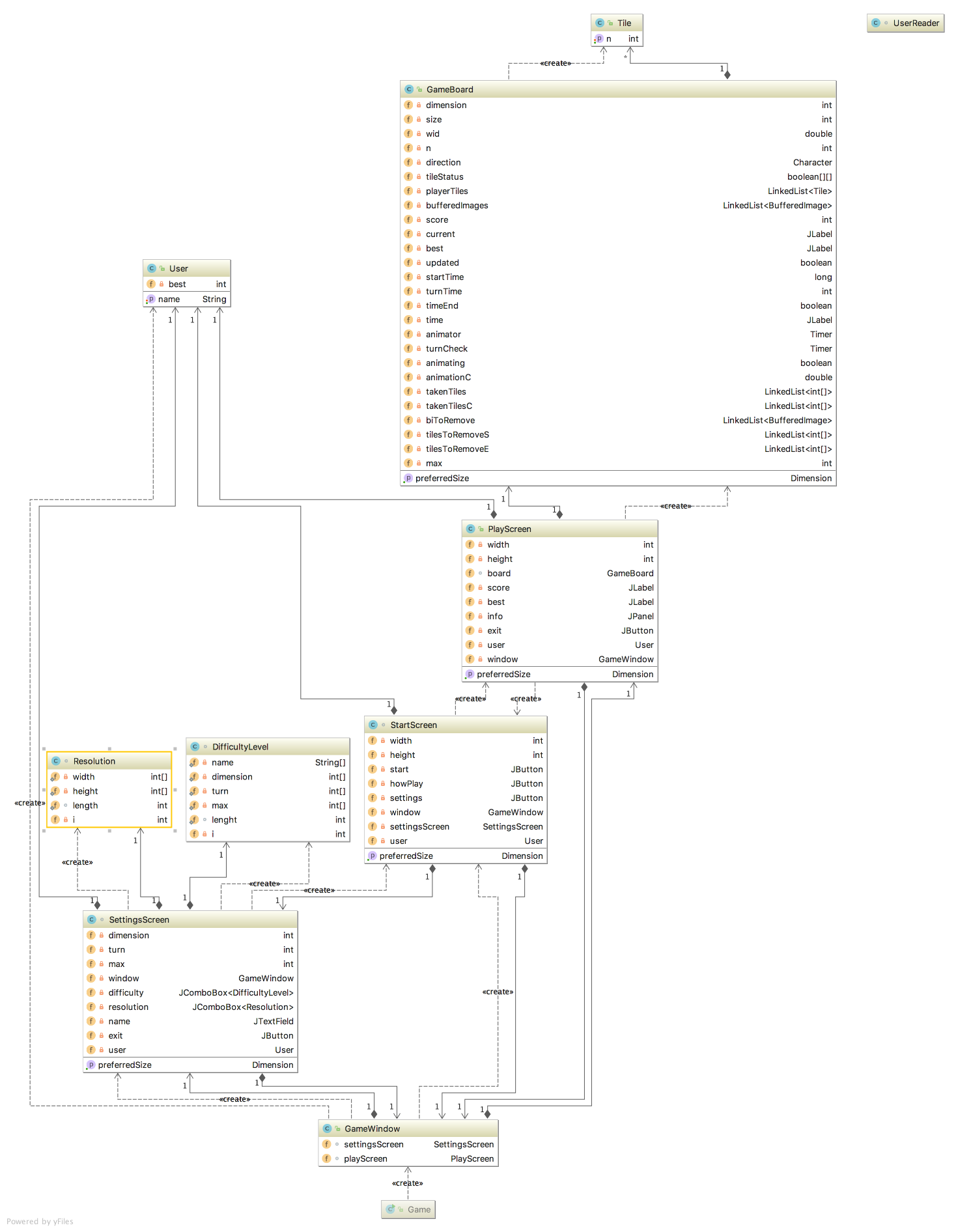


Рис. 1 Діаграма класів

## Опис методів та класів.

1. **GameWindow**

|  |  |
| --- | --- |
| (package private) void **initBoard**() | *Метод після виклику якого поле гри починає реагувати на натискання клавіш* |

1. **StartScreen**

|  |  |
| --- | --- |
| Dimension **getPreferredSize**() | *Встановлення розмірів панелі* |

1. **SettingsScreen**

|  |  |
| --- | --- |
| (package private) int **getDimension**() | *Повертає розмір ігрового поля в клітинах* |
| (package private) int **getMax**() | *Повертає число клітини, при досягненні якого гравець виграє* |
| Dimension **getPreferredSize**() | *Повертає розмір вікна* |
| (package private) int **getTurn**() | *Повертає час у мілісекундах відведений на хід гравця* |
| (package private) void **refresh**() | *Метод для перестворення панелі при зміні роздільної здатності* |

1. **Resolution**

|  |  |
| --- | --- |
| (package private) int **getHeight**() | *Повертає значення висоти вікна у режимі гри* |
| (package private) int **getWidth**() | *Повертає значення ширини вікна у режимі гри* |
| String **toString**() | *Визначає вигляд об’єкту цього класу в стрічці* |

1. **DifficultyLevel**

|  |  |
| --- | --- |
| (package private) int **getDimension**() | *Повертає значення розміру ігрового поля* |
| (package private) int **getMax**() | *Повертає число клітини, при досягненні якого гравець виграє* |
| (package private) String **getName**() | *Повертає назву рівня складності* |
| (package private) int **getTurn**() | *Повертає час у мілісекундах відведений на хід гравця* |
| String **toString**() | *Визначає вигляд об’єкту цього класу в стрічці* |

1. **PlayScreen**

|  |  |
| --- | --- |
| private int **getLast**(String s) | *Отримання позиції у стрічці з числом де відбувається перехід до десятої частини* |
| Dimension **getPreferredSize**() | *Встановлення розмірів панелі* |
| private void **initInfo**() | *Метод для додавання панелі зі статистичними даними користувача* |
| void **paintComponent**(Graphics g) | *Метод для малювання на панелі переданого об’єкту класу Graphics* |

1. **GameBoard**

|  |  |
| --- | --- |
| private void **copyPos**() | *Копіювання колекції зі значеннями зайнятих кліток* |
| private void **createTile**(Graphics2D g, int[] free, int i) | *Метод для створення нової клітинки* |
| private void **drawBackground**(Graphics2D g) | *Метод для малювання фону клітинок* |
| private void **empty**() | *Очищення клітин які було об'єднано з іншими* |
| private boolean **gameOver**() | *Метод, що дозволяє встановити чи програв гравець* |
| private int[] **getFreeTile**() | *Метод для отримання вільної клітинки* |
| private int **getLast**(String s) | *Отримання позиції у стрічці з числом де відбувається перехід до десятої частини* |
| private int **getMaxX**(int y) | *Метод, що дозволяє встановити максимальну вільну клітінку в заданому стовпчику* |
| private int **getMaxY**(int x) | *Метод, що дозволяє встановити максимальну вільну клітінку в заданому рядку* |
| private int **getMinX**(int y) | *Метод, що дозволяє встановити мінімальну вільну клітінку в заданому стовпчику* |
| private int **getMinY**(int x) | *Метод, що дозволяє встановити мінімальну вільну клітінку в заданому рядку* |
| public Dimension **getPreferredSize**() | *Метод який повертає бажані розміри панелі* |
| private boolean **moved**() | *Метод, що дозволяє встановити, чи було пересуното створені клітки* |
| private void **moveFreeTD**() | *Методи для пересування вже створених кліток в певному напрямку* |
| private void **moveFreeTL**() | *Методи для пересування вже створених кліток в певному напрямку* |
| private void **moveFreeTR**() | *Метод Метод для оновлення часу який лишився користувачу на хід*  *и для пересування вже створених кліток в певному напрямку* |
| private void **moveFreeTU**() | *Методи для пересування вже створених кліток в певному напрямку* |
| private void **moveTiles**(int c) | *Метод для об'єднання, переміщення клітинок й оновлення рахунку гравця в залежності від натиснутої клавіші* |
| void **paintComponent**(Graphics g) | *Метод для малювання графічних елементів на панелі* |
| private void **refreshTime**() | *Метод для оновлення часу який лишився користувачу на хід* |
| (package private) void **restart**() | *Метод для перезапуску гри* |
| private void **startAnimation**() | *Метод для старту анімації* |
| private void **unite**(int x1, int x2) | *Метод для об'єднання найближчих клітинок в графічному плані* |
| private void **uniteTD**() | *Метод для об'єднання найближчих клітинок при русі вниз* |
| private void **uniteTL**() | *Метод для об'єднання найближчих клітинок при русі вліво* |
| private void **uniteTR**() | *Метод для об'єднання найближчих клітинок при русі вправо* |
| private void **uniteTU**() | *Метод для об'єднання найближчих клітинок при русі вгору* |
| private boolean **updated**() | *Метод, що дозволяє встановити, чи був змінений рахунок* |
| private void **updateScore**() | *Метод для оновлення рахунку гравця* |
| private boolean **win**() | *Метод, що дозволяє встановити чи виграв гравець* |

1. **Tile**

|  |  |
| --- | --- |
| public int **getN**() | *Повертає число клітини* |
| (package private) void **inc**(double inc, double width) | *Метод для збільшення розмірів клітини. Використовується для анімації* |
| public void **setN**(int n) | *Встановлює число клітини* |

1. **User**

|  |  |
| --- | --- |
| (package private) int **getBest**() | *Повертає значення найкращого рахунку гравця* |
| public String **getName**() | *Повертає ім’я гравця* |
| (package private) void **setBest**(int best) | *Встановлює найкращий рахунок гравцю* |
| public void **setName**(String name) | *Встановлює ім’я гравцю* |

1. **UserReader**

|  |  |
| --- | --- |
| (package private) static User **getUser**(File file) | *Метод для зчитування даних з файлу* |
| (package private) static void **saveUser**(User user, String path) | *Метод для запису даних у файл за вказаним шляхом* |

1. **Game**

|  |  |
| --- | --- |
| public static void **main**(String[] args) | *Метод для старту програми* |

# **РОБОТА ПРОГРАМИ**

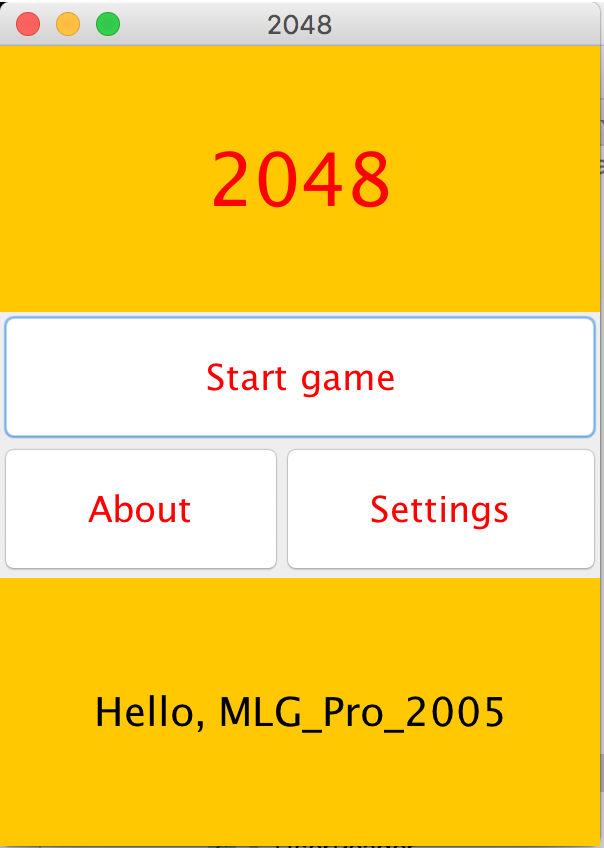


Рис. 2 Титульний екран

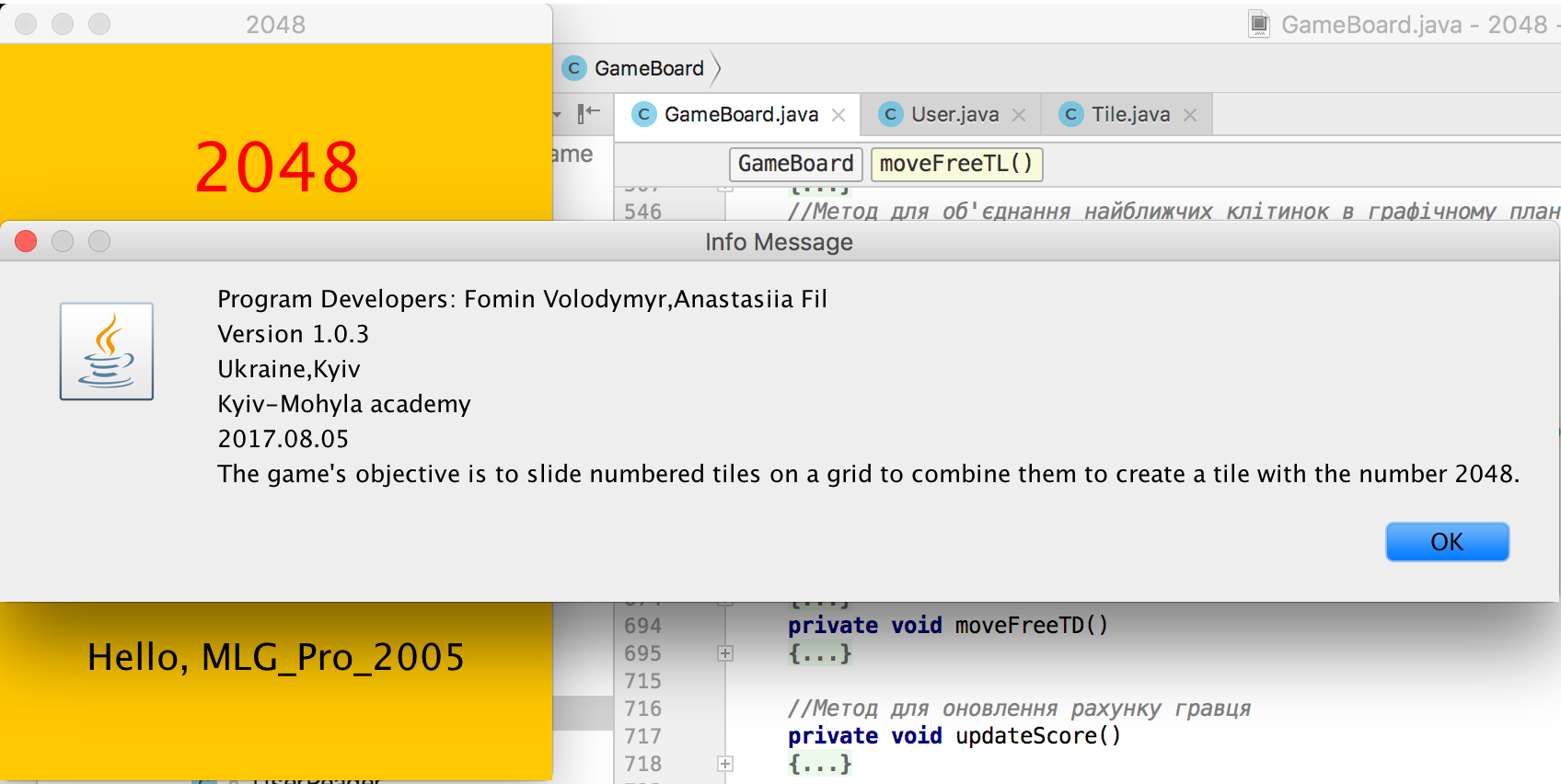


Рис. 3 Вікно з детальною інформацією про програму

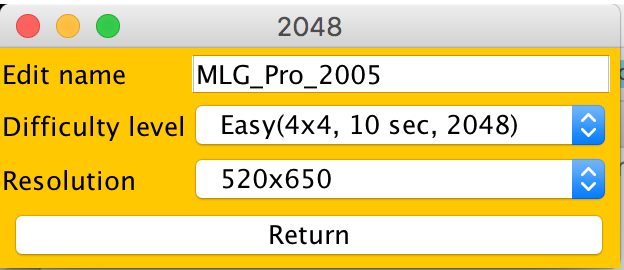


Рис. 4 Вікно налаштувань гри

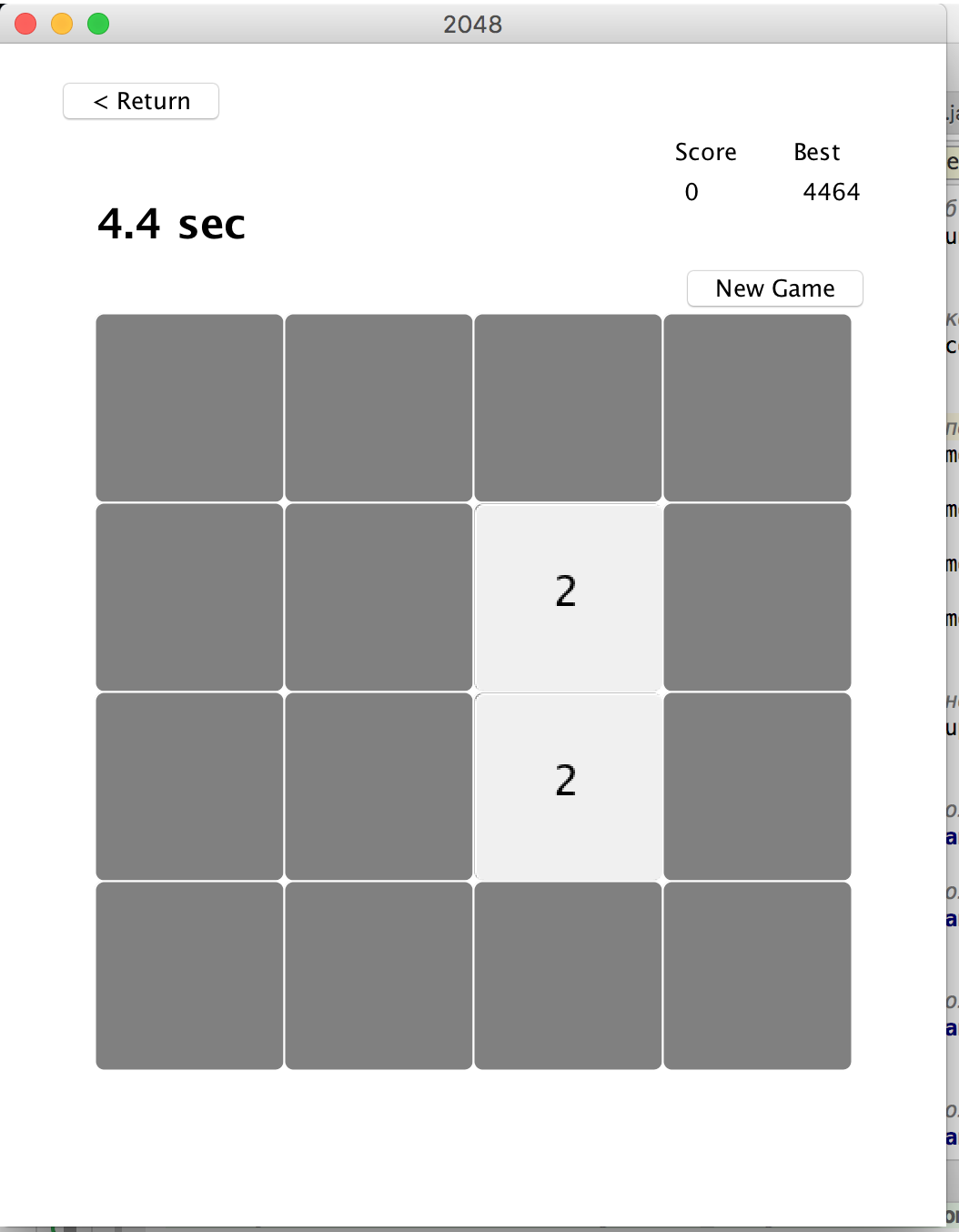


Рис. 5 Старт гри

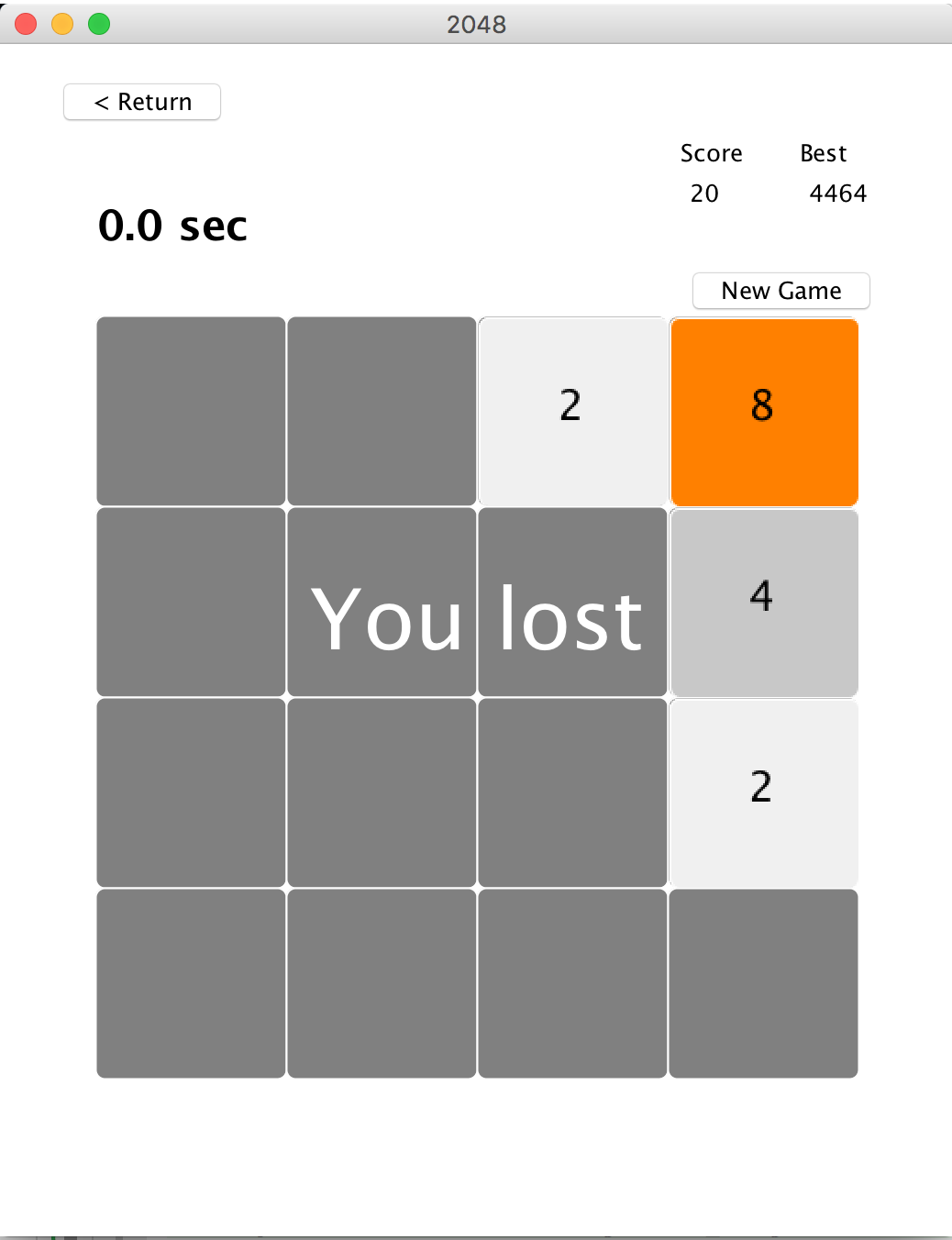


Рис. 6 Програш

# **ІНСТРУКЦІЯ КОРИСТУВАЧА**

Програма починає свою роботу зі стартового екрану, де відображено назву гри, привітання з іменем гравця (при першому запуску це обов’язково буде “*Player*”) і три кнопки.

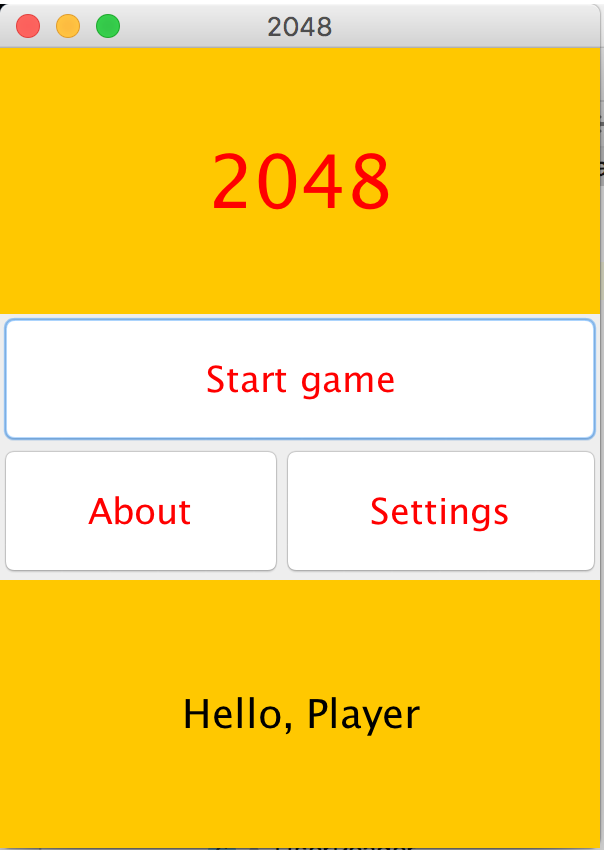


Рис. 7 Стартовий екран при першому запуску

Для запуску гри, користувач має натиснути на кнопку “*Start Game*”, гра запуститься у роздільній здатності *520х650* і на легкому рівні складності (10 секунд на хід, необхідне число клітини, яке треба набрати – 2048, поле 4х4). Аби змінити ім’я користувача, роздільну здатність і рівень складності користувач має натиснути на клавішу “*Settings”*, після чого він потрапить до екрану налаштувань.

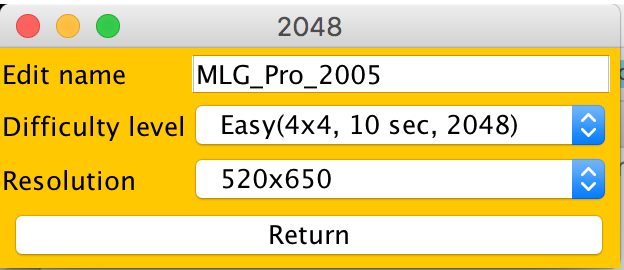


Рис. 8 Екран налаштувань

Тут користувач може встановити будь-яке ім’я у полі “*Edit name”,* обрати один з трьох рівней складності: “*Easy*” (Поле 4х4, число клітки для виграшу – 2048, час на хід – 10 сек), “*Normal*” (Поле 6х6, число клітки для виграшу – 4096, час на хід – 6 сек), “*Insane*” (Поле 8х8, число клітки для виграшу – 8192) та роздільну здатність вікна: 520x650, 800x1000, 1200x1500. Щоб повернутись до стартового екрану, користувач має натиснути на клавішу “*Return”*.

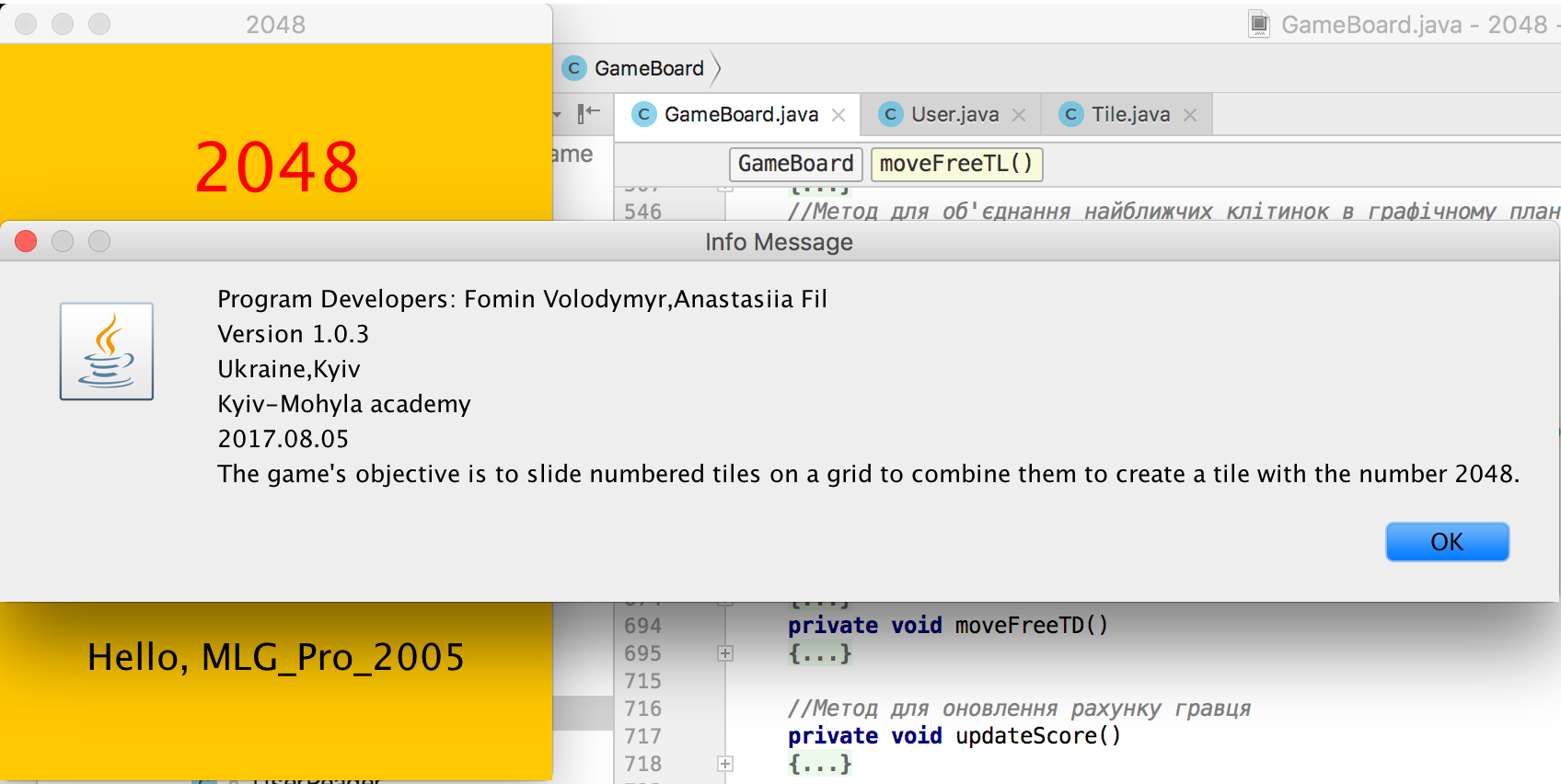


Рис. 9 Інформація про програму

При натисканні на стартовому екрані на кнопку “*About”,* користувачу буде виведено нове вікно з інформацією про розробників, версію гри, компанію розробників, дату створення й опис програми.

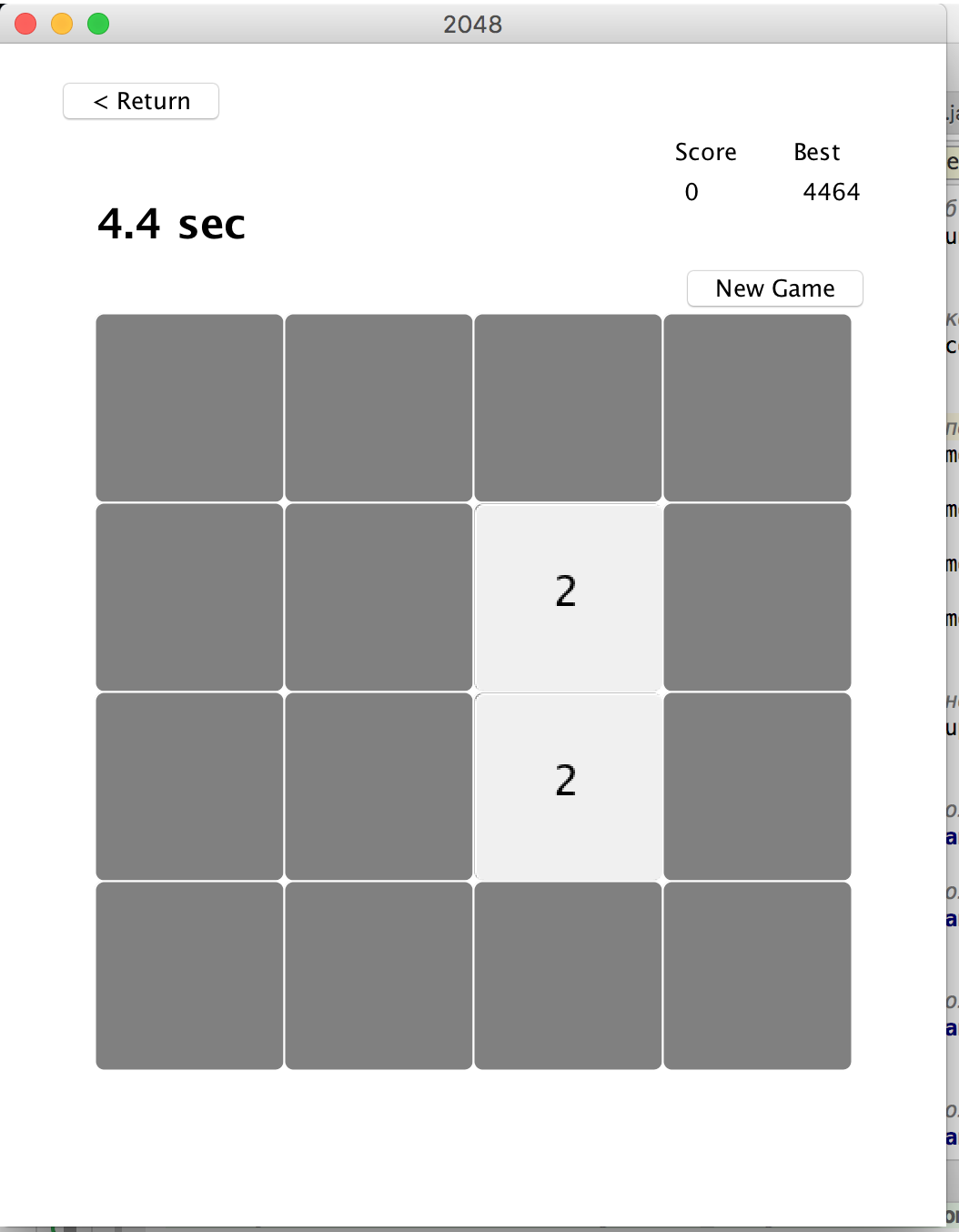


Рис. 10 Екран запущеної гри

Після натискання на кнопку “*Start Game*”, користувач побачить ігрове поле, розмірність якого визначає обраний рівень складності, по центру, набрану кількість очок, найкращий рахунок загалом і кнопку “*New Game*”, при натисканні якої відбувається перезапуск гри, у правому верхньому куті й скільки залишилось часу щоб зробити хід. При натисканні стрілок вправо, вліво, вгору, вниз на клавіатурі, корстувач задає напрямок в якому клітки має бути пересунуто або об’єднано. Якщо хоча б одну клітку було пересунуто або об’єднано з іншою, на полі з’явиться нова клітина з числом 2 або 4. Для того щоб виграти, гравець через об’єднання однакових кліток має створити клітку з числом, яке залежить від рівня складності (*Easy –* 2048*, Normal – 4096, Insane – 8192).* Гравець програє, якщо час на хід було вичерпано або якщо все поле зайнято клітками й жодну з них не можна об’єднати з іншою*.*

# **ЛІСТИНГ ПРОГРАМНОГО КОДУ**

## GameWindow

**import** javax.swing.\*;  
**import** java.io.File;  
  
*//Вікно програми***public class** GameWindow **extends** JFrame {  
 *//Панелі* SettingsScreen **settingsScreen**;  
 PlayScreen **playScreen**;  
  
 **public** GameWindow(){  
 **super**(**"2048"**);  
 *//Зчитується файл зі збереженими даними користувача* **new** File(File.*listRoots*()[0]+**"2048"**).mkdir();  
 User user = UserReader.*getUser*(**new** File(File.*listRoots*()[0] + **"2048"** + File.***separator*** + **"User.dat"**));  
 *//Якщо файлу ще не було створено, створюємо нового користувача* **if**(user ==**null**) user =**new** User(**"Player"**, 0);  
 *//Створюємо панель налаштувань* **settingsScreen**=**new** SettingsScreen(**this**, user);  
 *//Створюємо стартову панель* StartScreen startScreen = **new** StartScreen(**this**, **settingsScreen**, user, 520, 650);  
 **this**.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
 **this**.setVisible(**true**);  
 **this**.add(startScreen);  
 **this**.pack();  
 }  
 *//Метод після виклику якого поле гри починає реагувати на натискання клавіш* **void** initBoard(){  
 **playScreen**.**board**.setFocusable(**true**);  
 **playScreen**.**board**.requestFocusInWindow();  
 }  
}

## StartScreen

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
  
*//Панель яка дозволяє користувачу переглянути детальну інформацію про програму, почати гру або перейти до налаштувань***class** StartScreen **extends** JPanel {  
 *//Ширина панелі* **private int width**;  
 *//Висота панелі* **private int height**;  
 *//Кнопка для старту програми* **private** JButton **start**;  
 *//Кнопка для перегляду детальної інформації про програму* **private** JButton **howPlay**;  
 *//Кнопка для переходу до налаштувань програми* **private** JButton **settings**;  
 *//Вікно програми* **private** GameWindow **window**;  
 *//Панель з налаштуваннями* **private** SettingsScreen **settingsScreen**;  
 *//Гравець* **private** User **user**;  
 StartScreen(GameWindow window, **final** SettingsScreen settingsScreen, User user, **final int** width, **final int** height){  
 **super**();  
 *//Встановлення висоти й ширини панелі* **this**.**width**=(**int**)(width\*0.57692308);  
 **this**.**height**=(**int**)(height\*0.61538462);  
 *//Панель в якій буде збережено сукупність кнопок* **final** JPanel butPanel = **new** JPanel();  
 **this**.setLayout(**new** GridLayout(3,1));  
 butPanel.setLayout(**new** GridBagLayout());  
 GridBagConstraints c=**new** GridBagConstraints();  
 **this**.setBackground(Color.***ORANGE***);  
  
 **this**.**window** = window;  
 **this**.**user**=user;  
 **this**.**settingsScreen**=settingsScreen;  
  
 *//Назва гри* JLabel name = **new** JLabel(**"2048"**);  
 name.setFont(**new** Font(**"Colibri"**, Font.***PLAIN***, 37));  
 name.setHorizontalAlignment(JLabel.***CENTER***);  
 name.setForeground(Color.***red***);  
 **this**.add(name);  
 JLabel hello = **new** JLabel(**"Hello, "**+**this**.**user**.getName());  
 hello.setFont(**new** Font(**"Colibri"**, Font.***PLAIN***, 20));  
 hello.setHorizontalAlignment(JLabel.***CENTER***);  
  
 *//Створення кнопки для старту гри* **start**=**new** JButton(**"Start game"**);  
 **start**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 StartScreen.**this**.**window**.remove(StartScreen.**this**);  
 StartScreen.**this**.setVisible(**false**);  
 PlayScreen playScreen=**new** PlayScreen(StartScreen.**this**.**settingsScreen**.getDimension(), StartScreen.**this**.**settingsScreen**.getTurn(), StartScreen.**this**.**settingsScreen**.getMax(), StartScreen.**this**.**user**, StartScreen.**this**.**window**, width, height);  
 StartScreen.**this**.**window**.**playScreen**=playScreen;  
 StartScreen.**this**.**window**.add(playScreen);  
 StartScreen.**this**.**window**.pack();  
 StartScreen.**this**.**window**.initBoard();  
 }  
 });  
 **howPlay**=**new** JButton(**"About"**);  
 **howPlay**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JOptionPane.*showMessageDialog*(butPanel, **"Program Developers: Fomin Volodymyr,Anastasiia Fil \nVersion 1.0.3 \nUkraine,Kyiv \nKyiv-Mohyla academy \n2017.08.05\nThe game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048."**,**"Info Message"**,JOptionPane.***INFORMATION\_MESSAGE***);  
 }  
 });  
 **settings**=**new** JButton(**"Settings"**);  
 **settings**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 StartScreen.**this**.**window**.remove(StartScreen.**this**);  
 StartScreen.**this**.setVisible(**false**);  
 StartScreen.**this**.**settingsScreen**.setVisible(**true**);  
 StartScreen.**this**.**window**.add(StartScreen.**this**.**settingsScreen**);  
 StartScreen.**this**.**window**.pack();  
 StartScreen.**this**.**settingsScreen**.refresh();  
 }  
 });  
 **start**.setFont(**new** Font(**"Colibri"**, Font.***PLAIN***, 18));  
 **start**.setBackground(Color.***RED***);  
 **start**.setForeground(Color.***RED***);  
 **howPlay**.setBackground(Color.***RED***);  
 **howPlay**.setForeground(Color.***RED***);  
 **settings**.setBackground(Color.***RED***);  
 **settings**.setForeground(Color.***RED***);  
  
 **howPlay**.setFont(**new** Font(**"Colibri"**, Font.***PLAIN***, 18));  
 **settings**.setFont(**new** Font(**"Colibri"**, Font.***PLAIN***, 18));  
 c.**gridx**=0;  
 c.**gridy**=0;  
 c.**weighty**=1;  
 c.**weightx**=1;  
 c.**gridwidth**=2;  
 c.**fill**=GridBagConstraints.***BOTH***;  
 butPanel.add(**start**, c);  
 c.**gridy**=1;  
 c.**gridwidth**=1;  
 butPanel.add(**howPlay**, c);  
 c.**gridx**=1;  
 butPanel.add(**settings**, c);  
  
 **this**.add(butPanel);  
 **this**.add(hello);  
  
 }  
  
 *//Встановлення розмірів панелі* **public** Dimension getPreferredSize() {  
 **return new** Dimension(**width**, **height**);  
 }  
}

## SettingsScreen

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.io.File;  
  
*//Панель завдяки якій користувач може встановлювати рівень складності, змінювати свій нік і роздільну здатність***class** SettingsScreen **extends** JPanel {  
 *//Кількість клітин на полі* **private int dimension**;  
 *//Час на хід гравця* **private int turn**;  
 *//Число клітини, при якому гравець виграє* **private int max**;  
 *//Вікно програми* **private** GameWindow **window**;  
 *//Варіанти рівней складності* **private** JComboBox<DifficultyLevel> **difficulty**;  
 *//Варіанти роздільних здатностей* **private** JComboBox<Resolution> **resolution**;  
 *//Поле для редагування імені* **private** JTextField **name**;  
 *//Кнопка для переходу в головне меню* **private** JButton **exit**;  
 *//Гравець* **private** User **user**;  
 SettingsScreen(GameWindow window, User user){  
 **super**();  
 **this**.**window**=window;  
 **this**.**user**=user;  
 **difficulty**=**new** JComboBox<>();  
 **this**.setLayout(**new** GridBagLayout());  
 **this**.setBackground(Color.***ORANGE***);  
 **name**=**new** JTextField();  
 **name**.setText(**this**.**user**.getName());  
 **for**(**int** i = 0; i< DifficultyLevel.*lenght*; i++)**difficulty**.addItem(**new** DifficultyLevel(i));  
 **difficulty**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 DifficultyLevel i=(DifficultyLevel)**difficulty**.getSelectedItem();  
 **dimension**=i.getDimension();  
 **turn**=i.getTurn();  
 **max**=i.getMax();  
 }  
 });  
 *//За замовченням обрано перший рівень складності, аби гравець міг запустити гру не заходячи у налаштування* **difficulty**.setSelectedIndex(0);  
 **resolution**=**new** JComboBox<>();  
 **for**(**int** i=0;i<Resolution.*length*;i++)**resolution**.addItem(**new** Resolution(i));  
 **resolution**.setSelectedIndex(0);  
 *//Коли гравець натискає на цю кнопку, відбувається збереження його імені в файл і повернення в головне меню* **exit** = **new** JButton(**"Return"**);  
 **exit**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 SettingsScreen.**this**.**user**.setName(SettingsScreen.**this**.**name**.getText());  
 UserReader.*saveUser*(SettingsScreen.**this**.**user**, File.*listRoots*()[0]+**"2048"**+File.***separator***+**"User.dat"**);  
 SettingsScreen.**this**.setVisible(**false**);  
 SettingsScreen.**this**.**window**.remove(SettingsScreen.**this**);  
 SettingsScreen.**this**.**window**.add(**new** StartScreen(SettingsScreen.**this**.**window**, SettingsScreen.**this**, SettingsScreen.**this**.**user**,  
 ((Resolution)**resolution**.getSelectedItem()).getWidth(), ((Resolution)**resolution**.getSelectedItem()).getHeight() ));  
 SettingsScreen.**this**.**window**.pack();  
 }  
 });  
 }  
  
 *//Метод для перестворення панелі при зміні роздільної здатності* **void** refresh(){  
 **this**.removeAll();  
 GridBagConstraints c=**new** GridBagConstraints();  
 c.**gridx**=0;  
 c.**gridy**=0;  
 c.**fill**=GridBagConstraints.***BOTH***;  
 **this**.add(**new** JLabel(**"Edit name"**),c);  
 **name**.setText(**this**.**user**.getName());  
 c.**gridx**=1;  
 **this**.add(**name**, c);  
 c.**gridx**=0;  
 c.**gridy**=1;  
 **this**.add(**new** JLabel(**"Difficulty level"**), c);  
 c.**gridx**=1;  
 **this**.add(**difficulty**, c);  
 c.**gridx**=0;  
 c.**gridy**=2;  
 **this**.add(**new** JLabel(**"Resolution"**), c);  
 c.**gridx**=1;  
 **this**.add(**resolution**, c);  
 *//При нажатии на эту кнопку отрисовывается новый вступительный экран и измененный пользователь снова сохраняется в файл* c.**gridx**=0;  
 c.**gridy**=3;  
 c.**gridwidth**=2;  
 **this**.add(**exit**, c);  
 }  
  
 **int** getDimension() {  
 **return dimension**;  
 }  
  
 **int** getTurn() {  
 **return turn**;  
 }  
  
 **int** getMax() {  
 **return max**;  
 }  
  
 *//Повертає розмір вікна* **public** Dimension getPreferredSize() {  
 **return new** Dimension( (**int**)( ((Resolution)**resolution**.getSelectedItem()).getWidth() \* 0.59615385),  
 (**int**)( ((Resolution)**resolution**.getSelectedItem()).getHeight() \*0.16923077));  
 }  
}

## DifficultyLevel

//Клас з рівнями складності  
class DifficultyLevel {  
 private static String[] name={"Easy", "Normal", "Insane"};  
 private static int[] dimension={16, 36, 64};  
 private static int[] turn={10000, 6000, 3000};  
 private static int[] max={2048, 4096, 8192};  
 static int lenght=name.length;  
 private int i;  
 DifficultyLevel(int i){  
 this.i=i;  
 }  
 String getName() {  
 return name[i];  
 }  
  
 int getDimension() {  
 return dimension[i];  
 }  
  
 int getTurn() {  
 return turn[i];  
 }  
  
 int getMax() {  
 return max[i];  
 }  
 public String toString(){  
 return name[i]+"("+(int)Math.sqrt(dimension[i])+"x"+(int)Math.sqrt(dimension[i])+", "+turn[i]/1000+" sec, "+max[i]+")";  
 }  
}

## Resolution

//Клас із роздільними здатностями  
class Resolution{  
 private static int[] width={520, 800, 1200};  
 private static int[] height={650, 1000, 1500};  
 static int length=width.length;  
 private int i;  
 Resolution(int i){  
 this.i=i;  
 }  
 int getWidth() {  
 return width[i];  
 }  
  
 int getHeight() {  
 return height[i];  
 }  
  
 public String toString(){  
 return width[i]+"x"+height[i];  
 }  
}

## PlayScreen

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.io.File;  
  
*//Панель з полем гри й іншими статистичними даними***public class** PlayScreen **extends** JPanel {  
 *//Ширина й висота панелі* **private int width**;  
 **private int height**;  
 *//Ігрове поле* GameBoard **board**;  
 *//Набраний рахунок* **private** JLabel **score**;  
 *//Найкращий рахунок* **private** JLabel **best**;  
 *//Панель де зберігаються рахунки користувача* **private** JPanel **info**;  
 *//Кнопка для виходу з гри до головного меню, після натискання якої також відбувається запис найкращого рахунку користувача в файл* **private** JButton **exit**;  
 *//Користувач* **private** User **user**;  
 *//Вікно програми* **private** GameWindow **window**;  
  
 PlayScreen(**int** dimension, **int** turn, **int** max, User user, GameWindow window, **int** width, **int** height){  
 **super**();  
 **this**.**width**=width;  
 **this**.**height**=height;  
 **this**.setBackground(Color.***WHITE***);  
 **this**.setLayout(**new** GridBagLayout());  
 **this**.**window**=window;  
 **this**.**user**=user;  
 GridBagConstraints c=**new** GridBagConstraints();  
 initInfo();  
 c.**gridx**=0;  
 c.**gridy**=0;  
 c.**insets**=**new** Insets(0,(**int**)(**this**.**width** \*0.64),(**int**)(**this**.**height** \*0.7),0);  
 **this**.add(**info**,c);  
 *//Відображення часу який залишився в користувача на те, щоб зробити хід* JLabel time = **new** JLabel((**""** + (turn \* 0.001)).substring(0, getLast(**""** + (turn \* 0.001)))+**" sec"**);  
 c.**insets**=**new** Insets(0,-(**int**)(**this**.**width** \*0.64),(**int**)(**this**.**height** \*0.7),0);  
 time.setFont(**new** Font(Font.***SANS\_SERIF***, Font.***BOLD***, (**int**)(0.045\* **this**.**width**)));  
 **this**.add(time,c);  
 **board**=**new** GameBoard(dimension, **this**.**width**, **score**, **best**, time, turn, max);  
 c.**fill**=GridBagConstraints.***BOTH***;  
 c.**insets**=**new** Insets((**int**)(**this**.**height** \*0.2),(**int**)(**this**.**width** \*0.1),-(**int**)(**this**.**height** \*0.2),-(**int**)(**this**.**width** \*0.1));  
 **this**.add(**board**,c);  
 }  
  
 *//Метод для отримання позиції у стрічці з числом з десятою частиною* **private int** getLast(String s) {  
 **int** i=0;  
 **while**(s.charAt(i)!=**'.'**)i++;  
 i+=2;  
 **return** i;  
 }  
  
 *//Метод для додавання панелі зі статистичними даними користувача* **private void** initInfo(){  
 GridBagConstraints c=**new** GridBagConstraints();  
 **info**=**new** JPanel(**new** GridBagLayout());  
 **info**.setBackground(Color.***WHITE***);  
 c.**gridx**=0;  
 c.**gridy**=0;  
 c.**fill**=GridBagConstraints.***HORIZONTAL***;  
 c.**insets**=**new** Insets((**int**)(0.01\* **height**),(**int**)(0.01\* **width**),0,0);  
 **info**.add(**new** JLabel(**"Score"**),c);  
 c.**gridx**++;  
 c.**insets**=**new** Insets((**int**)(0.01\* **height**),(**int**)(0.06\* **width**),0,(**int**)(0.01\* **width**));  
 **info**.add(**new** JLabel(**"Best"**),c);  
 c.**gridx**--;  
 c.**gridy**++;  
 c.**insets**=**new** Insets((**int**)(0.01\* **height**),(**int**)(0.02\* **width**),0,0);  
 **score**=**new** JLabel(**"0"**);  
 **info**.add(**score**,c);  
 c.**gridx**++;  
 c.**insets**=**new** Insets((**int**)(0.01\* **height**),(**int**)(0.07\* **width**),0,(**int**)(0.01\* **width**));  
 **best**=**new** JLabel(**""**+**user**.getBest());  
 **info**.add(**best**,c);  
 c.**insets**=**new** Insets((**int**)(0.05\* **height**),(**int**)(0.01\* **width**),0,(**int**)(0.01\* **width**));  
 JButton newGame = **new** JButton(**"New Game"**);  
 newGame.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **board**.restart();  
 **board**.setFocusable(**true**);  
 **board**.requestFocusInWindow();  
 **score**.setText(**"0"**);  
 }  
 });  
 c.**gridx**--;  
 c.**gridy**++;  
 c.**gridwidth**=2;  
 **info**.add(newGame,c);  
 **exit**=**new** JButton(**"< Return"**);  
 **exit**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 PlayScreen.**this**.**user**.setBest(Integer.*parseInt*(**best**.getText()));  
 UserReader.*saveUser*(PlayScreen.**this**.**user**, File.*listRoots*()[0]+**"2048"**+File.***separator***+**"User.dat"**);  
 PlayScreen.**this**.setVisible(**false**);  
 **window**.remove(PlayScreen.**this**);  
 PlayScreen.**this**.**window**.add(**new** StartScreen(PlayScreen.**this**.**window**, PlayScreen.**this**.**window**.**settingsScreen**, PlayScreen.**this**.**user**, PlayScreen.**this**.**width**, PlayScreen.**this**.**height**));  
 PlayScreen.**this**.**window**.pack();  
 }  
 });  
 c.**gridy**=0;  
 c.**gridx**=0;  
 c.**fill**=GridBagConstraints.***NONE***;  
 c.**insets**=**new** Insets(0,-(**int**)(**width** \*0.7),(**int**)(**height** \*0.9),0);  
 **this**.add(**exit**, c);  
  
 }  
 **public void** paintComponent(Graphics g){  
 **super**.paintComponent(g);  
 ((Graphics2D)g).setRenderingHint ( RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON*** );  
 }  
 **public** Dimension getPreferredSize() {  
 **return new** Dimension(**width**, **height**);  
 }  
}

## GameBoard

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.KeyEvent;  
**import** java.awt.event.KeyListener;  
**import** java.awt.geom.RoundRectangle2D;  
**import** java.awt.image.BufferedImage;  
**import** java.util.LinkedList;  
  
*//Ігрове поле***public class** GameBoard **extends** JPanel {  
 *//Встановлює скільки клітин має бути створено* **private int dimension**;  
 *//Встановлює розміри панелі* **private int size**;  
 *//Встановлює розмір клітини* **private double wid**;  
 *//Свідчить про число клітин* **private int n**=1;  
 *//Свідчить в якому напрямі має бути пересунуто клітини* **private** Character **direction**=**null**;  
 *//Свідчить про зайнятість клітини за заданими кординатами* **private boolean**[][] **tileStatus**;  
 *//Колекція зі створеними клітинами* **private** LinkedList<Tile> **playerTiles**;  
 *//Колеція з зображеннями на яких буде намальовано клітини* **private** LinkedList<BufferedImage> **bufferedImages**;  
 *//Свідчить про кілкість очок які набрав гравець* **private int score**;  
 *//Відображає гравцю скільки він набрав очок* **private** JLabel **current**;  
 *//Відображає гравцю його найкращий рахунок* **private** JLabel **best**;  
 *//Свідчить, чи було змінено рахунок* **private boolean updated**;  
 *//Зберігає час у який було зроблено останній хід* **private long startTime**;  
 *//Зберігає час, який відведено користувачу на хід* **private int turnTime**;  
 *//Свідчить чи було вичерпано час* **private boolean timeEnd**=**false**;  
 *//Відображає час який лишився користувачу на хід* **private** JLabel **time**;  
 *//Таймер для анімації* **private** Timer **animator** = **null**;  
 *//Таймер для моніторингу ходу* **private** Timer **turnCheck**=**null**;  
 *//Свідчить, чи відбувається зараз анімація* **private boolean animating** = **false**;  
 *//Відповідає за ступінь анімації* **private double animationС** = 0;  
 *//Колекція з кординатами клітин, які зайняті* **private** LinkedList<**int**[]> **takenTiles**;  
 *//Копія колекції з кординатами клітин, які зайняті* **private** LinkedList<**int**[]> **takenTilesC**;  
 *//Клітини які були об'єдані з іншими* **private** LinkedList<BufferedImage> **biToRemove**;  
 **private** LinkedList<**int**[]> **tilesToRemoveS**;  
 **private** LinkedList<**int**[]> **tilesToRemoveE**;  
 *//Число клітини при отриманні якого гравець виграє* **private int max**;  
  
 *//Конструктор панелі* GameBoard(**int** dimension, **int** width, JLabel current, JLabel best, JLabel time, **int** turnTime, **int** max)  
 {  
 **super**();  
 **score**=0;  
 **this**.**current**=current;  
 **this**.**best**=best;  
 **playerTiles**=**new** LinkedList<>();  
 **bufferedImages**= **new** LinkedList<>();  
 **takenTiles** =**new** LinkedList<>();  
 **takenTilesC** =**new** LinkedList<>();  
 **biToRemove**=**new** LinkedList<>();  
 **tilesToRemoveS** =**new** LinkedList<>();  
 **tilesToRemoveE** =**new** LinkedList<>();  
 **this**.**dimension**=dimension;  
 **tileStatus** =**new boolean**[ (**int**)Math.*sqrt*(dimension) ][( **int**)Math.*sqrt*(dimension) ];  
 **for**(**int** j = 0; j< **tileStatus**.**length**; j++) **for**(**int** i = 0; i< **tileStatus**[j].**length**; i++) **tileStatus**[j][i]=**true**;  
 **this**.**size** =width;  
 **wid**=(0.8/Math.*sqrt*(dimension))\***size**;  
 **this**.setBackground(Color.***WHITE***);  
 **this**.addKeyListener(**new** KeyListener()  
 {  
 @Override  
 **public void** keyTyped(KeyEvent e)  
 {  
  
 }  
  
 @Override  
 **public void** keyPressed(KeyEvent e)  
 {  
  
 }  
  
 @Override  
 **public void** keyReleased(KeyEvent e)  
 {  
 moveTiles(e.getKeyCode());  
 }  
 });  
 **this**.**turnTime**=turnTime;  
 **this**.**time**=time;  
 **this**.**max**=max;  
 refreshTime();  
 restart();  
 }  
  
 *//Метод для старту анімації* **private void** startAnimation ()  
 {  
 **animating** = **true**;  
 **animationС** = 0;  
 **animator** = **new** Timer ( 18, **new** ActionListener()  
 {  
 **public void** actionPerformed ( ActionEvent e )  
 {  
 **if** ( **animationС** <1 )  
 {  
 **animationС** += 0.05;  
 GameBoard.**this**.repaint ();  
 }  
 **else** {  
 **animating** = **false**;  
 **animationС** = 0;  
 **if** (moved() || **updated** || **takenTiles**.size()==2) {  
 GameBoard.**this**.**startTime** = System.*currentTimeMillis*();  
 }  
 **turnCheck**.start();  
 **animator**.stop ();  
 }  
 }  
 });  
 **animator**.start ();  
 }  
  
 *//Метод для малювання графічних елементів на панелі* **public void** paintComponent(Graphics g)  
 {  
 **super**.paintComponent(g);  
 ((Graphics2D)g).setRenderingHint ( RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON*** );  
 *//Спочатку малюємо фон* drawBackground((Graphics2D)g);  
 **if**(**animating** || **timeEnd**) {  
 *//При початку гри ствоюємо тільки дві клітини* **if** (**n** <= 1 && **direction** == **null**) {  
 **if** (**takenTiles**.size()==0) {  
 **takenTiles**.add(getFreeTile());  
 **takenTiles**.add(getFreeTile());  
 }  
 createTile((Graphics2D) g, **takenTiles**.get(**n** - 1), **n** - 1);  
 createTile((Graphics2D) g, **takenTiles**.get(**n**), **n**);  
 } **else** {  
 **switch** (**direction**) {  
 *//Якщо ні, то спочатку пересуваємо об'єднані клітини й вже створені* **case 'l'**:  
 **if** (**animationС** < 1 && !**biToRemove**.isEmpty()) **for** (**int** i = 0; i < **biToRemove**.size(); i++)  
 g.drawImage(**biToRemove**.get(i),  
 (**int**) ((**tilesToRemoveE**.get(i)[0] \* **wid**) + (1 - **animationС**) \* ((**tilesToRemoveS**.get(i)[0] - **tilesToRemoveE**.get(i)[0]) \* **wid**)),  
 (**int**) (**tilesToRemoveS**.get(i)[1] \* **wid**), **null**);  
  
 **for** (**int** i = 0; i < **bufferedImages**.size(); i++)  
 g.drawImage(**bufferedImages**.get(i),  
 (**int**) ((**takenTiles**.get(i)[0] \* **wid**) + (1 - **animationС**) \* ((**takenTilesC**.get(i)[0] - **takenTiles**.get(i)[0]) \* **wid**)),  
 (**int**) (**takenTiles**.get(i)[1] \* **wid**), **null**);  
 **break**;  
 **case 'r'**:  
 **if** (**animationС** < 1 && !**biToRemove**.isEmpty()) **for** (**int** i = 0; i < **biToRemove**.size(); i++)  
 g.drawImage(**biToRemove**.get(i),  
 (**int**) ((**tilesToRemoveE**.get(i)[0] \* **wid**) + (1 - **animationС**) \* ((**tilesToRemoveS**.get(i)[0] - **tilesToRemoveE**.get(i)[0]) \* **wid**)),  
 (**int**) (**tilesToRemoveS**.get(i)[1] \* **wid**), **null**);  
  
 **for** (**int** i = 0; i < **bufferedImages**.size(); i++)  
 g.drawImage(**bufferedImages**.get(i),  
 (**int**) ((**takenTiles**.get(i)[0] \* **wid**) + (1 - **animationС**) \* ((**takenTilesC**.get(i)[0] - **takenTiles**.get(i)[0]) \* **wid**)),  
 (**int**) (**takenTiles**.get(i)[1] \* **wid**), **null**);  
  
 **break**;  
 **case 'u'**:  
 **if** (**animationС** < 1 && !**biToRemove**.isEmpty()) **for** (**int** i = 0; i < **biToRemove**.size(); i++)  
 g.drawImage(**biToRemove**.get(i),  
 (**int**) (**tilesToRemoveE**.get(i)[0] \* **wid**),  
 (**int**) (**tilesToRemoveE**.get(i)[1] \* **wid** + (1 - **animationС**) \* ((**tilesToRemoveS**.get(i)[1] - **tilesToRemoveE**.get(i)[1]) \* **wid**)), **null**);  
  
 **for** (**int** i = 0; i < **bufferedImages**.size(); i++)  
 g.drawImage(**bufferedImages**.get(i),  
 (**int**) (**takenTiles**.get(i)[0] \* **wid**),  
 (**int**) (**takenTiles**.get(i)[1] \* **wid** + (1 - **animationС**) \* ((**takenTilesC**.get(i)[1] - **takenTiles**.get(i)[1]) \* **wid**)), **null**);  
  
 **break**;  
 **case 'd'**:  
 **if** (**animationС** < 1 && !**biToRemove**.isEmpty()) **for** (**int** i = 0; i < **biToRemove**.size(); i++)  
 g.drawImage(**biToRemove**.get(i),  
 (**int**) (**tilesToRemoveE**.get(i)[0] \* **wid**),  
 (**int**) (**tilesToRemoveE**.get(i)[1] \* **wid** + (1 - **animationС**) \* ((**tilesToRemoveS**.get(i)[1] - **tilesToRemoveE**.get(i)[1]) \* **wid**)), **null**);  
  
 **for** (**int** i = 0; i < **bufferedImages**.size(); i++)  
 g.drawImage(**bufferedImages**.get(i),  
 (**int**) (**takenTiles**.get(i)[0] \* **wid**),  
 (**int**) (**takenTiles**.get(i)[1] \* **wid** + (1 - **animationС**) \* ((**takenTilesC**.get(i)[1] - **takenTiles**.get(i)[1]) \* **wid**)), **null**);  
  
 }  
 *//Якщо клітини було пересунуто або об'єднано, то ще створюємо нову клітину* **if** (moved() || **updated**) {  
 **if** (**animationС** <= 0.05) **takenTiles**.add(getFreeTile());  
 createTile((Graphics2D) g, **takenTiles**.get(**n**), **n**);  
 }  
  
 }  
 **if** (win()) {  
 g.setFont(**new** Font(Font.***SANS\_SERIF***, Font.***PLAIN***, (**int**)(0.09\***size**)));  
 g.drawString(**"You won!"**, (**int**) (**size**\*0.2), (**int**) (**size** \* 0.45));  
 } **else** {  
 **if** (gameOver()) {  
 g.setFont(**new** Font(Font.***SANS\_SERIF***, Font.***PLAIN***, (**int**)(0.09\***size**)));  
 g.drawString(**"You lost"**, (**int**) (**size**\*0.225), (**int**) (**size** \* 0.35));  
 }  
 }  
 }  
 }  
  
 *//Метод для малювання фону клітинок* **private void** drawBackground(Graphics2D g)  
 {  
 **double** x=Math.*sqrt*(**dimension**);  
 **for**(**int** j=0;j<x;j++)  
 {  
 **for**(**int** i=0;i<x;i++)  
 {  
 RoundRectangle2D r=**new** RoundRectangle2D.Double((**int**)(i\***wid**),(**int**)(j\***wid**),(**int**)**wid**,(**int**)**wid**,(**int**)(**wid**\*0.1),(**int**)(**wid**\*0.1));  
 g.setColor(Color.***GRAY***);  
 g.fill(r);  
 g.setColor(Color.***WHITE***);  
 g.draw(r);  
 }  
 }  
 }  
  
 *//Метод для створення нової клітинки* **private void** createTile(Graphics2D g, **int**[] free, **int** i)  
 {  
 **if**(**animating** && **animationС** <=0.05)**playerTiles**.add(**new** Tile((**int**)(0.5 \* **wid**), (**int**)(0.5 \* **wid**), 0, 0, 0, 0));  
 BufferedImage bi = **new** BufferedImage ( (**int**)**wid**, (**int**)**wid**, BufferedImage.***TYPE\_INT\_ARGB*** );  
 Graphics2D g2d = bi.createGraphics ();  
 g2d.setRenderingHint(RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON***);  
 **playerTiles**.get(i).inc(**animationС**, **wid**);  
 **if** (**playerTiles**.get(i).getN() == 2) g2d.setColor(**new** Color(240,240,240));  
 **else** g2d.setColor(**new** Color(200,200,200));  
 g2d.fill(**playerTiles**.get(i));  
 g2d.setColor(Color.***WHITE***);  
 g2d.draw(**playerTiles**.get(i));  
 g2d.setColor(Color.***BLACK***);  
 g2d.setFont(**new** Font(Font.***SANS\_SERIF***, Font.***PLAIN***, (**int**)(((4\*0.045\***size**)/Math.*sqrt*(**dimension**))\* **animationС**)));  
 g2d.drawString(**""** + **playerTiles**.get(i).getN(), (**int**) (**wid** \* 0.43), (**int**) (**wid** \* 0.55));  
 g2d.setComposite ( AlphaComposite.*getInstance* ( AlphaComposite.***SRC\_IN*** ) );  
 **if**(**animationС** >=1)**bufferedImages**.add(bi);  
 g.drawImage ( bi, (**int**) (free[0] \* **wid**), (**int**) (free[1] \* **wid**), **null** );  
 }  
  
 *//Метод для перезапуску гри* **void** restart()  
 {  
 **if**(!**animating**)  
 {  
 **time**.setText((**""**+(**turnTime**\*0.001)).substring(0,getLast(**""**+(**turnTime**\*0.001)))+**" sec"**);  
 **n**=1;  
 **direction**=**null**;  
 **timeEnd**=**false**;  
 **turnCheck**.stop();  
 **score**=0;  
 **for** (**int** j = 0; j < **tileStatus**.**length**; j++) **for** (**int** i = 0; i < **tileStatus**[j].**length**; i++) **tileStatus**[j][i] = **true**;  
 **while** (!**playerTiles**.isEmpty()) **playerTiles**.removeFirst();  
 **while** (!**bufferedImages**.isEmpty()) **bufferedImages**.removeFirst();  
 **while** (!**takenTiles**.isEmpty()) **takenTiles**.removeFirst();  
 **while** (!**takenTilesC**.isEmpty()) **takenTilesC**.removeFirst();  
 **while** (!**biToRemove**.isEmpty()) **biToRemove**.removeFirst();  
 **while** (!**tilesToRemoveS**.isEmpty()) **tilesToRemoveS**.removeFirst();  
 **while** (!**tilesToRemoveE**.isEmpty()) **tilesToRemoveE**.removeFirst();  
 startAnimation();  
 }  
 }  
 *//Метод для об'єднання, переміщення клітинок й оновлення рахунку гравця в залежності від натиснутої клавіші* **private void** moveTiles(**int** c)  
 {  
 **if**(!win() && !gameOver() && !**animating**)  
 {  
 **switch** (c)  
 {  
 **case** KeyEvent.***VK\_LEFT***:  
 *//Встановлюємо напрям руху клітин* **direction**=**'l'**;  
 *//Очищення колекцій об'єднаних клітин* empty();  
 *//Об'єднання клітин* uniteTL();  
 *//Копіювання стартових позицій клітин які треба пересунути* copyPos();  
 *//Пересування клітин(встановлення нової позиції)* moveFreeTL();  
 *//Якщо було об'єднано або пересунуто клітини, то оновлюємо таймер ходу й збільшуємо кількість клітин* **updated**=updated();  
 **if**(moved() || **updated**){  
 **n**++;  
 **turnCheck**.stop();  
 }  
 *//Стартуємо анімацію* startAnimation();  
 *//Оновлюємо рахунок* updateScore();  
 **break**;  
 **case** KeyEvent.***VK\_RIGHT***:  
 **direction**=**'r'**;  
 empty();  
 uniteTR();  
 copyPos();  
 moveFreeTR();  
 **updated**=updated();  
 **if**(moved() || **updated**){  
 **n**++;  
 **turnCheck**.stop();  
 }  
 startAnimation();  
 updateScore();  
 **break**;  
 **case** KeyEvent.***VK\_UP***:  
 **direction**=**'u'**;  
 empty();  
 uniteTU();  
 copyPos();  
 moveFreeTU();  
 **updated**=updated();  
 **if**(moved() || **updated**){  
 **n**++;  
 **turnCheck**.stop();  
 }  
 startAnimation();  
 updateScore();  
 **break**;  
 **case** KeyEvent.***VK\_DOWN***:  
 **direction**=**'d'**;  
 empty();  
 uniteTD();  
 copyPos();  
 moveFreeTD();  
 **updated**=updated();  
 **if**(moved() || **updated**){  
 **n**++;  
 **turnCheck**.stop();  
 }  
 startAnimation();  
 updateScore();  
 }  
 }  
 }  
  
 *//Очищення клітин які було об'єднано з іншими* **private void** empty(){  
 **while** (!**biToRemove**.isEmpty()) **biToRemove**.removeFirst();  
 **while** (!**tilesToRemoveS**.isEmpty()) **tilesToRemoveS**.removeFirst();  
 **while** (!**tilesToRemoveE**.isEmpty()) **tilesToRemoveE**.removeFirst();  
 }  
 *//Метод для отримання вільної клітинки* **private int**[] getFreeTile()  
 {  
 **int**[] freeT=**new int**[2];  
 **int** x;  
 **int** y;  
 **while**(**true**)  
 {  
 x=(**int**)(Math.*random*()\*Math.*sqrt*(**dimension**));  
 y=(**int**)(Math.*random*()\*Math.*sqrt*(**dimension**));  
 **if**(**tileStatus**[y][x])  
 {  
 **tileStatus**[y][x]=**false**;  
 freeT[0]=x;  
 freeT[1]=y;  
 **break**;  
 }  
 }  
 **return** freeT;  
 }  
 *//Метод для об'єднання найближчих клітинок при русі вліво* **private void** uniteTL()  
 {  
 **boolean** exit;  
 **for**(**int** j=0;j<Math.*sqrt*(**dimension**);j++)  
 {  
 **for**(**int** i1=0;i1<Math.*sqrt*(**dimension**);i1++)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**){  
 **if**(**takenTiles**.get(x1)[0]==i1 && **takenTiles**.get(x1)[1]==j)  
 {  
 exit=**false**;  
 **for**(**int** i2=i1;i2<Math.*sqrt*(**dimension**);i2++)  
 {  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++)  
 {  
 **if**(x2<=**n**&&x1!=x2)  
 {  
 **if**(**takenTiles**.get(x2)[0]==i2 && **takenTiles**.get(x2)[1]==j)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN())  
 {  
 unite(x1,x2);  
 **if** (x1 > x2) x1--;  
 }  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 *//Метод для об'єднання найближчих клітинок при русі вправо* **private void** uniteTR()  
 {  
 **boolean** exit;  
 **for**(**int** j=0;j<Math.*sqrt*(**dimension**);j++)  
 {  
 **for**(**int** i1=(**int**)Math.*sqrt*(**dimension**)-1;i1>=0;i1--)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**){  
 **if**(**takenTiles**.get(x1)[0]==i1 && **takenTiles**.get(x1)[1]==j)  
 {  
 exit=**false**;  
 **for**(**int** i2=i1;i2>=0;i2--){  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++)  
 {  
 **if**(x2<=**n**&&x1!=x2){  
 **if**(**takenTiles**.get(x2)[0]==i2 && **takenTiles**.get(x2)[1]==j)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN())  
 {  
 unite(x1,x2);  
 **if** (x1 > x2) x1--;  
 }  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 *//Метод для об'єднання найближчих клітинок при русі вгору* **private void** uniteTU()  
 {  
 **boolean** exit;  
 **for**(**int** i=0;i<Math.*sqrt*(**dimension**);i++)  
 {  
 **for**(**int** j1=0;j1<(**int**)Math.*sqrt*(**dimension**);j1++)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**){  
 **if**(**takenTiles**.get(x1)[0]==i && **takenTiles**.get(x1)[1]==j1)  
 {  
 exit=**false**;  
 **for**(**int** j2=j1;j2<(**int**)Math.*sqrt*(**dimension**);j2++)  
 {  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++){  
 **if**(x2<=**n**&&x1!=x2){  
 **if**(**takenTiles**.get(x2)[0]==i && **takenTiles**.get(x2)[1]==j2)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN())  
 {  
 unite(x1,x2);  
 **if** (x1 > x2) x1--;  
 }  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 *//Метод для об'єднання найближчих клітинок при русі вниз* **private void** uniteTD()  
 {  
 **boolean** exit;  
 **for**(**int** i=0;i<Math.*sqrt*(**dimension**);i++)  
 {  
 **for**(**int** j1=(**int**)Math.*sqrt*(**dimension**)-1;j1>=0;j1--)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**)  
 {  
 **if**(**takenTiles**.get(x1)[0]==i && **takenTiles**.get(x1)[1]==j1)  
 {  
 exit=**false**;  
 **for**(**int** j2=j1;j2>=0;j2--)  
 {  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++)  
 {  
 **if**(x2<=**n**&&x1!=x2)  
 {  
 **if**(**takenTiles**.get(x2)[0]==i && **takenTiles**.get(x2)[1]==j2)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN())  
 {  
 unite(x1,x2);  
 **if** (x1 > x2) x1--;  
 }  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 *//Метод для об'єднання найближчих клітинок в графічному плані* **private void** unite(**int** x1, **int** x2)  
 {  
 **playerTiles**.get(x1).setN(**playerTiles**.get(x1).getN() \* 2);  
 **score**+=**playerTiles**.get(x1).getN();  
 Graphics2D g = (Graphics2D)**bufferedImages**.get(x1).getGraphics();  
 g.setRenderingHint(RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON***);  
 **double** c=0.344/Math.*sqrt*(**dimension**);  
 **double** r=0.056/Math.*sqrt*(**dimension**);  
 **switch**(**playerTiles**.get(x1).getN())  
 {  
 **case** 4:  
 g.setColor(**new** Color(200,200,200));  
 **break**;  
 **case** 8:  
 g.setColor(**new** Color(255,128,0));  
 **break**;  
 **case** 16:  
 g.setColor(**new** Color(255,192,0));  
 c-=r;  
 **break**;  
 **case** 32:  
 g.setColor(**new** Color(255,160,0));  
 c-=r;  
 **break**;  
 **case** 64:  
 g.setColor(**new** Color(255,0,0));  
 c-=r;  
 **break**;  
 **case** 128:  
 g.setColor(**new** Color(255,224,0));  
 c-=2\*r;  
 **break**;  
 **case** 256:  
 g.setColor(**new** Color(255,255,0));  
 c-=2\*r;  
 **break**;  
 **case** 512:  
 g.setColor(**new** Color(255,224,0));  
 c-=2\*r;  
 **break**;  
 **case** 1024:  
 g.setColor(**new** Color(255,224,0));  
 c-=3\*r;  
 **break**;  
 **case** 2048:  
 g.setColor(**new** Color(255,255,0));  
 c-=3\*r;  
 **case** 4096:  
 g.setColor(**new** Color(192,0,192));  
 c-=3\*r;  
 **case** 8192:  
 g.setColor(**new** Color(255,0,255));  
 c-=3\*r;  
 }  
 g.fill(**playerTiles**.get(x1));  
 g.setColor(Color.***WHITE***);  
 g.draw(**playerTiles**.get(x1));  
 g.setColor(Color.***BLACK***);  
 g.setFont(**new** Font(Font.***SANS\_SERIF***, Font.***PLAIN***, (**int**)((4\*0.045\***size**)/Math.*sqrt*(**dimension**))));  
 g.drawString(**""** + **playerTiles**.get(x1).getN(), (**int**) (**size** \* c), (**int**) (**size** \* (0.44/Math.*sqrt*(**dimension**))));  
 **tileStatus**[**takenTiles**.get(x2)[1]][**takenTiles**.get(x2)[0]] = **true**;  
 **playerTiles**.remove(x2);  
 **biToRemove**.add(**bufferedImages**.get(x2));  
 **tilesToRemoveS**.add(**takenTiles**.get(x2));  
 **tilesToRemoveE**.add(**takenTiles**.get(x1));  
 **bufferedImages**.remove(x2);  
 **takenTiles**.remove(x2);  
 **n**--;  
 }  
  
 *//Копіювання колекції зі значеннями зайнятих кліток* **private void** copyPos()  
 {  
 **while** (!**takenTilesC**.isEmpty()) **takenTilesC**.removeFirst();  
 **int**[] t;  
 **for**(**int**[] a: **takenTiles**)  
 {  
 t=**new int**[a.**length**];  
 System.*arraycopy*(a,0,t,0,a.**length**);  
 **takenTilesC**.add(t);  
 }  
 }  
  
 *//Методи для пересування вже створених кліток в певному напрямку* **private void** moveFreeTL()  
 {  
 **int** min;  
 **for**(**int** j=0;j<Math.*sqrt*(**dimension**);j++)  
 {  
 min=getMinX(j);  
 **for**(**int** i=min;i<Math.*sqrt*(**dimension**);i++)  
 {  
 **for** (**int**[] aFreeT : **takenTiles**)  
 {  
 **if** (aFreeT[1] == j && aFreeT[0] == i)  
 {  
 aFreeT[0] = min;  
 **tileStatus**[j][i] = **true**;  
 **tileStatus**[j][min] = **false**;  
 min++;  
 }  
 }  
 }  
 }  
 }  
 **private void** moveFreeTR()  
 {  
 **int** max;  
 **for**(**int** j=0;j<Math.*sqrt*(**dimension**);j++)  
 {  
 max=getMaxX(j);  
 **for**(**int** i=max;i>=0;i--)  
 {  
 **for** (**int**[] aFreeT : **takenTiles**)  
 {  
 **if** (aFreeT[1] == j && aFreeT[0] == i)  
 {  
 aFreeT[0] = max;  
 **tileStatus**[j][i] = **true**;  
 **tileStatus**[j][max] = **false**;  
 max--;  
 }  
 }  
 }  
 }  
 }  
 **private void** moveFreeTU()  
 {  
 **int** min;  
 **for**(**int** i=0;i<Math.*sqrt*(**dimension**);i++)  
 {  
 min=getMinY(i);  
 **for**(**int** j=min;j<Math.*sqrt*(**dimension**);j++)  
 {  
 **for** (**int**[] aFreeT : **takenTiles**)  
 {  
 **if** (aFreeT[0] == i && aFreeT[1] == j)  
 {  
 aFreeT[1] = min;  
 **tileStatus**[j][i] = **true**;  
 **tileStatus**[min][i] = **false**;  
 min++;  
 }  
 }  
 }  
 }  
 }  
 **private void** moveFreeTD()  
 {  
 **int** max;  
 **for**(**int** i=0;i<Math.*sqrt*(**dimension**);i++)  
 {  
 max=getMaxY(i);  
 **for**(**int** j=max;j>=0;j--)  
 {  
 **for** (**int**[] aFreeT : **takenTiles**)  
 {  
 **if** (aFreeT[1] == j && aFreeT[0] == i)  
 {  
 aFreeT[1] = max;  
 **tileStatus**[j][i] = **true**;  
 **tileStatus**[max][i] = **false**;  
 max--;  
 }  
 }  
 }  
 }  
 }  
  
 *//Метод для оновлення рахунку гравця* **private void** updateScore()  
 {  
 **current**.setText(**""**+**score**);  
 **if**(Integer.*parseInt*( **best**.getText() ) <= **score**) **best**.setText( **""**+**score** );  
  
 }  
  
 *//Метод, що дозволяє встановити, чи був змінений рахунок* **private boolean** updated()  
 {  
 **return** Integer.*parseInt*( **current**.getText() ) != **score**;  
 }  
  
 *//Метод, що дозволяє встановити, чи було пересуното створені клітки* **private boolean** moved()  
 {  
 **for**(**int** i = 0; i< **takenTilesC**.size(); i++)**if**(**takenTiles**.get(i)[0]!= **takenTilesC**.get(i)[0] || **takenTiles**.get(i)[1]!= **takenTilesC**.get(i)[1])**return true**;  
 **return false**;  
 }  
  
 *//Метод, що дозволяє встановити чи програв гравець* **private boolean** gameOver()  
 {  
 **if**(**timeEnd**) **return true**;  
 **if**(**n**==**dimension**-1)  
 {  
 **boolean** exit;  
 **for**(**int** j=0;j<Math.*sqrt*(**dimension**);j++)  
 {  
 **for**(**int** i1=0;i1<Math.*sqrt*(**dimension**);i1++)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**)  
 {  
 **if**(**takenTiles**.get(x1)[0]==i1 && **takenTiles**.get(x1)[1]==j)  
 {  
 exit=**false**;  
 **for**(**int** i2=i1;i2<Math.*sqrt*(**dimension**);i2++)  
 {  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++)  
 {  
 **if**(x2<=**n**&&x1!=x2)  
 {  
 **if**(**takenTiles**.get(x2)[0]==i2 && **takenTiles**.get(x2)[1]==j)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN()) **return false**;  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 **for**(**int** j=0;j<Math.*sqrt*(**dimension**);j++)  
 {  
 **for**(**int** i1=(**int**)Math.*sqrt*(**dimension**)-1;i1>=0;i1--)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**)  
 {  
 **if**(**takenTiles**.get(x1)[0]==i1 && **takenTiles**.get(x1)[1]==j)  
 {  
 exit=**false**;  
 **for**(**int** i2=i1;i2>=0;i2--)  
 {  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++)  
 {  
 **if**(x2<=**n**&&x1!=x2)  
 {  
 **if**(**takenTiles**.get(x2)[0]==i2 && **takenTiles**.get(x2)[1]==j)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN()) **return false**;  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 **for**(**int** i=0;i<Math.*sqrt*(**dimension**);i++)  
 {  
 **for**(**int** j1=0;j1<(**int**)Math.*sqrt*(**dimension**);j1++)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**)  
 {  
 **if**(**takenTiles**.get(x1)[0]==i && **takenTiles**.get(x1)[1]==j1)  
 {  
 exit=**false**;  
 **for**(**int** j2=j1;j2<(**int**)Math.*sqrt*(**dimension**);j2++)  
 {  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++)  
 {  
 **if**(x2<=**n**&&x1!=x2)  
 {  
 **if**(**takenTiles**.get(x2)[0]==i && **takenTiles**.get(x2)[1]==j2)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN()) **return false**;  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 **for**(**int** i=0;i<Math.*sqrt*(**dimension**);i++)  
 {  
 **for**(**int** j1=(**int**)Math.*sqrt*(**dimension**)-1;j1>=0;j1--)  
 {  
 **for**(**int** x1 = 0; x1< **takenTiles**.size(); x1++)  
 {  
 **if**(x1<=**n**){  
 **if**(**takenTiles**.get(x1)[0]==i && **takenTiles**.get(x1)[1]==j1)  
 {  
 exit=**false**;  
 **for**(**int** j2=j1;j2>=0;j2--)  
 {  
 **for**(**int** x2 = 0; x2< **takenTiles**.size(); x2++)  
 {  
 **if**(x2<=**n**&&x1!=x2){  
 **if**(**takenTiles**.get(x2)[0]==i && **takenTiles**.get(x2)[1]==j2)  
 {  
 **if**(**playerTiles**.get(x2).getN()==**playerTiles**.get(x1).getN()) **return false**;  
 exit=**true**;  
 **break**;  
 }  
 }  
 }  
 **if**(exit)**break**;  
 }  
 }  
 }  
 }  
 }  
 }  
 **return true**;  
 }  
 **return false**;  
 }  
  
 *//Метод, що дозволяє встановити чи виграв гравець* **private boolean** win()  
 {  
 **for**(Tile a:**playerTiles**) **if**(a.getN()==**max**) **return true**;  
 **return false**;  
 }  
  
 *//Метод, що дозволяє встановити максимальну вільну клітінку в заданому стовпчику* **private int** getMaxX(**int** y)  
 {  
 **int** max=(**int**)Math.*sqrt*(**dimension**)-1;  
 **for**(**int**[] a: **takenTiles**) **if**(a[1]==y && a[0]==max) max--;  
 **return** max;  
 }  
  
 *//Метод, що дозволяє встановити мінімальну вільну клітінку в заданому стовпчику* **private int** getMinX(**int** y)  
 {  
 **int** min=0;  
 **for**(**int**[] a: **takenTiles**) **if**(a[1]==y && a[0]==min) min++;  
 **return** min;  
 }  
  
 *//Метод, що дозволяє встановити максимальну вільну клітінку в заданому рядку* **private int** getMaxY(**int** x)  
 {  
 **int** max=(**int**)Math.*sqrt*(**dimension**)-1;  
 **for**(**int**[] a: **takenTiles**) **if**(a[0]==x && a[1]==max) max--;  
 **return** max;  
 }  
  
 *//Метод, що дозволяє встановити мінімальну вільну клітінку в заданому рядку* **private int** getMinY(**int** x)  
 {  
 **int** min=0;  
 **for**(**int**[] a: **takenTiles**) **if**(a[0]==x && a[1]==min) min++;  
 **return** min;  
 }  
  
 *//Метод для оновлення часу який лишився користувачу на хід* **private void** refreshTime(){  
 **turnCheck**=**new** Timer(1, **new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 *//Якщо час закінчився гравець програє* **if** (**turnTime** + **startTime** <= System.*currentTimeMillis*()) {  
 **timeEnd** = **true**;  
 **animationС** = 1;  
 copyPos();  
 repaint();  
 **turnCheck**.stop();  
 }  
 *//Оновлення часу який лишився користувачу на хід* GameBoard.**this**.**time**.setText((**""** + Math.*abs*((**turnTime** - (System.*currentTimeMillis*() - **startTime**))\*0.001)).substring(0,getLast(**""** + Math.*abs*((**turnTime** - (System.*currentTimeMillis*() - **startTime**))\*0.001)))+**" sec"**);  
 }  
 });  
 }  
 *//Отримання позиції у стрічці з числом де відбувається перехід до десятої частини* **private int** getLast(String s) {  
 **int** i=0;  
 **while**(s.charAt(i)!=**'.'**)i++;  
 i+=2;  
 **return** i;  
 }  
 *//Метод який повертає бажані розміри панелі* **public** Dimension getPreferredSize()  
 {  
 **return new** Dimension(**size**, **size**);  
 }  
}

## Tile

**import** java.awt.geom.RoundRectangle2D;  
  
*//Клас клітини з числом, який буде намальовано на ігровому полі***public class** Tile **extends** RoundRectangle2D.Double {  
 *//Зберігається число клітини* **private int n**;  
 Tile(**double** x, **double** y, **double** width, **double** height, **double** arcw, **double** arch){  
 **super**(x, y, width, height, arcw, arch);  
 **n**= (Math.*random*() < 0.9) ? 2 : 4;  
 }  
 *//Метод для збільшення розмірів клітини. Використовується для анімації* **void** inc(**double** inc, **double** width){  
 **this**.**x**=(1.024-inc)\*0.5\*width;  
 **this**.**y**=(1.02-inc)\*0.5\*width;  
 **this**.**width**=inc\*width\*0.99;  
 **this**.**height**=inc\*width\*0.99;  
 **this**.**arcwidth**=inc\*width\*0.099;  
 **this**.**archeight**=inc\*width\*0.099;  
 }  
 **public int** getN(){  
 **return n**;  
 }  
 **public void** setN(**int** n){  
 **this**.**n**=n;  
 }  
}

## User

**import** java.io.Serializable;  
  
*//Клас в якому зберігаються дані користувача***public class** User **implements** Serializable{  
 *//Ім'я користувача* **private** String **name**;  
 *//Найкращій рахунок користувача* **private int best**;  
 User(String name, **int** best){  
 **this**.**name**=name;  
 **this**.**best**=best;  
 }  
 **public** String getName() {  
 **return name**;  
 }  
  
 **int** getBest() {  
 **return best**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **void** setBest(**int** best) {  
 **this**.**best** = best;  
 }  
}

## UserReader

**import** java.io.\*;  
  
*//Клас для зчитування/збереження файлу з даними користувача***class** UserReader **implements** Serializable {  
 *//Метод для зчитування даних з файлу* **static** User getUser(File file) {  
 **try** {  
 **if** (file.exists()) {  
 FileInputStream in = **new** FileInputStream(file);  
 ObjectInputStream os = **new** ObjectInputStream(in);  
 **try** {  
 **return** (User) os.readObject();  
 } **catch** (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 os.close();  
 in.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
  
 *//Метод для запису даних у файл за вказаним шляхом* **static void** saveUser(User user, String path) {  
 **try** {  
 **if** (path.endsWith(File.***separator***)) path += **"User"**;  
 **if** (!path.contains(**".dat"**)) path += **".dat"**;  
 File file = **new** File(path);  
 FileOutputStream out = **new** FileOutputStream(file);  
 ObjectOutputStream oout = **new** ObjectOutputStream(out);  
 oout.writeObject(user);  
 oout.close();  
 out.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
  
 }  
 }  
}

## Game

**import** Tasks.Game2048.GameWindow;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** Game {  
 **public static void** main(String[] args){  
 EventQueue.*invokeLater*(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **try** {  
 UIManager.*setLookAndFeel*(UIManager.*getSystemLookAndFeelClassName*());  
 } **catch** (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException ex) {  
 ex.printStackTrace();  
 }  
 **new** GameWindow();  
 }  
 });  
 }  
}