Софийски университет "Св. Климент Охридски"
Факултет по математика и информатика

КУРСОВ ПРОЕКТ

по учебната дисциплина

„Обектно-ориентирано програмиране с Java"

Play Sudoku

Изготвил:

Никол Емануилова, 61987

Софтуерно инженерство, 3 курс

Научен ръководител:

д-р Евгений Кръстев

София, февруари 2019 год.

Съдържание

1. Цел на проекта

Целта на проекта е да се реализира потребителско клиент-сървър приложение, което представлява известната японска игра Судоку. Пъзелът е мрежа от 81 клетки (9x9), като всеки блок е подмрежа 3x3. Целта на играта е да се попълнят всички клетки, така че всеки ред, всяка колона и всеки блок да съдържат цифрите от 1 до 9 точно по веднъж. Чрез приложението потребителите могат да решават судоку с избрана от тях трудност в 3 нива, да проверяват дали решението им е правилно, както и да връщат напред и назад предишни действия. По всяко време потребител може да заяви нова игра, както и да види колко време му отнема да реши пъзела. Сървърната част реализира генерирането на валидно судоку (с единствено решение), проверката дали потребителя е решил правилно пъзела, както и връщането на правилното решение при неуспех. Също така се записват статистики за всеки потребител като ниво на трудност, брой решени и нерешени, и общо време прекарано в играта.

2. Архитектура

2.1. Сървърна част

Пакетът *server* от проекта *SudokuServer* съдържа имплементацията на RMI сървъра. Състои се от следните класове:

- *SudokuServerInterface* - дефинира интерфейса на сървъра и какви ще са неговите функционалности.
- *SudokuServer* - имплементира методите на дефинирания интерфейс. Осъществява генерирането на судоку и пращането му към клиент, връща решението на пъзел, както и проверява потребителско решение. Също така записва статистики във файл.
- *RegisterSudokuServer* - това е изпълнимият файл, от който се регистрира и стартира сървъра.

2.2. Клиентска част

Пакетът *usergui* от проекта *SudokuClient* съдържа имплементацията на графичния потребителски интерфейс, както и

функционалностите на клиентската част. Състои се от следните класове и файлове:

- *layout.fxml* - fxml файл, съдържащ XML описание на графичния интерфейс за клиента. Позволява въвеждането на цифри чрез бутони и съответно бутони за връщане назад и напред, избирането на ниво на трудност чрез радио бутони, както и бутони за нова игра, проверка на решение, показване на правилно решение и изход от играта.
- *style.css* - css файл, съдържащ дефиниции за промяна на външния вид на елементите от графичния интерфейс.
- *TimerManager.java* - това е помощен клас, който управлява таймера на приложението, чрез създаването на допълнителна нишка. Предоставя функционалност за лесно спиране и пускане на таймера от контролера на графичния интерфейс.
- *UndoRedoManager.java* - това е помощен клас, който осигурява подходяща структура за съхраняване на действията направени от потребителя. Този клас помага на контролера да имплементира функционалността за връщане напред и назад в решаването на пъзела.
- *UserController.java* - този клас представлява контролер на графичния интерфейс на приложението. Предоставя интерактивност на интерфейса и имплементира функционалността на клиентската част.
- *SudokuUser.java* - този клас е изпълнимият файл, от който се стартира клиентската част на приложението.

2.3. Помощен пакет

Пакетът c*ommon* от проекта *SudokuServer* съдържа допълнителни класове за имплементирането на логиката на приложението. Така се постига по-ясна и разграничена архитектура. Този пакет от класове се използва, както в сървърната част, така и в клиентската. Състои се от следните класове:

- *enum LevelType* - изброим тип, представящ възможните нива на трудност, броят празни клетки за това ниво (генерира се случайно число в даден интервал), както и начина, по който се избира кои да бъдат празните клетки.

- *enum GameOutcome* - изброим тип, представящ резултата от игра на судоку, който може да бъде решен пъзел или нерешен
- *enum Sequence* - изброим тип, представящ начина на обхождане на клетките при избор кои да бъдат празни. Начинът се определя от нивото на трудност.
- *class Cell* - клас, представящ най-малката единица в судокуто - клетка. Определя се от реда и колоната, в която се намира. Ако стойността на клетка е 0, тя се счита за празна.
- *class Board* - клас, представящ мрежата, в която се играе. Съдържа елементи от *class Cell.* Реализира създаването на мрежа и проверяването на условията за валидна стойност на клетка.
- *class Solver* - клас, който реализира решаването на судоку. По този начин се генерира пълна мрежа, както и се проверява дали даден пъзел е валиден, т.е. дали има единствено решение.
- *class SudokuGame* - клас, който представя игра на судоку. Реализира генерирането на валиден пъзел чрез алгоритми като Las Vegas и Backtracking, с помощта на *class Solver.* Също така помага на сървъра за проверка на потребителско решение и връщане на правилното решение при нужда.
- *class User* - клас, описващ един потребител на приложението. Всеки потребител се дефинира с потребителско име. За него се пази информация като на какви нива трудност е играл, колко пъзела е успял да реши и колко не, и колко време е прекарал в играта.

3. Структури от данни и алгоритми

3.1. Структури от данни

За реализацията на *class User* е използвана структура от данни HashMap, чрез който се пази информация за брой игра на всяко ниво, и втори HashMap за следенето на броя успешни и неуспешни игри.

В клиентската част е постигната функционалност за връщане назад и напред на действия чрез *class UndoRedoManager*, който реализира това чрез два Stack контейнера. Единият пази клетките, които трябва да бъдат върнати при операция *Undo,* а другия съответно тези за *Redo.* Всяка промяна по мрежата се запазва в *undoStack*, а при извършване на съответната операция се премества

в *redoStack,* и обратното. За да бъдат валидни операциите, при добавяне на ново действие, *redoStack* бива изчистен.

3.2. Решаване на судоку

Реализира се решаване на судоку в логиката на приложението, което помага при самото генериране на такова. За решаването се използва Backtracking алгоритъм. Обхожда се всяка клетка от мрежата, като съответно се пропускат вече запълнените. Празните места се запълват с валидна стойност от 1 до 9, ако отговаря на условията на играта. При достигане до невалидно решение, то правим стъпка назад и търсим друга възможна валидна стойност.

3.3. Генериране на судоку

Генерирането на судоку се постига на две стъпки. Първо се започва с празна мрежа от клетки. Използва се Las Vegas алгоритъм за генерирането на първоначални 11 случайни стойности, в случайни клетки, които трябва да са валидни. Така се получава начално състояние на мрежата, което се подава на *class Solver,* за да бъде решен пъзела и така да се получи пълна мрежа от клетки.

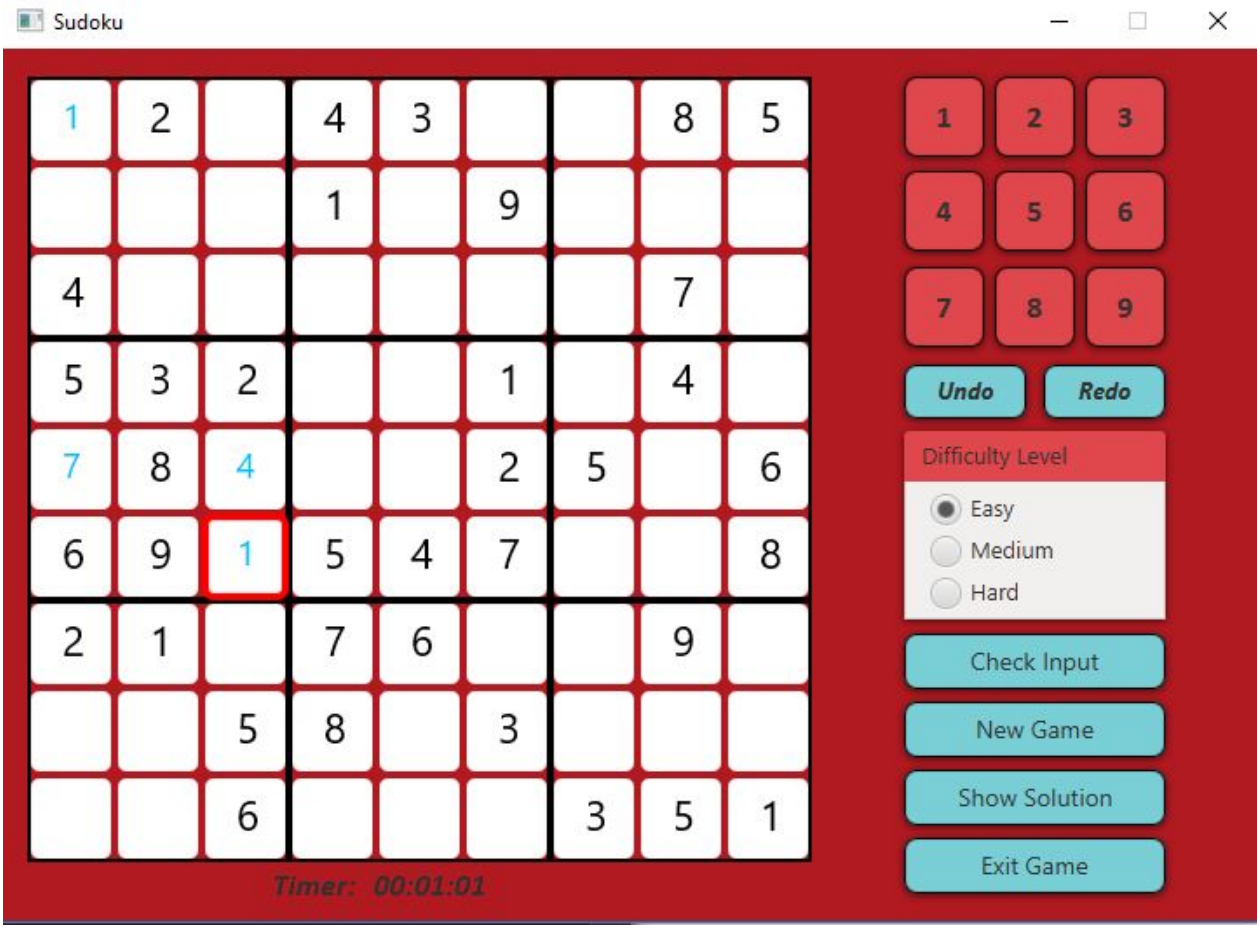
След като има завършен вид на пъзелът, то трябва да се реши къде да бъдат дупките (празните клетки). Клетките се обхождат като списък, който предварително е подреден според начина, по който трябва да се обходят те. Това се определя от нивото: за лесно ниво, те просто се разбъркват случайно; за средно ниво се обхождат линейно, а за трудно ниво - мрежата се обхожда във формата на “S”.

Правенето на дупки в мрежата се осъществява отново чрез Backtracking алгоритъм. За всяка направена дупка се проверява дали полученият пъзел има единствено решение - това е условие за валидно судоку. Ако има повече решения, връща се мрежата в предишното състояние и се пробва отново с друга клетка. Това се прави, докато се получи мрежа с нужния брой празни клетки според нивото и пъзелът е с едно уникално решение.

4. Възникнали и текущи проблеми

По време на разработката на приложението възникналите проблеми бяха предимно свързани с логиката и нейното имплементиране. На късен етап беше установено, че генерираните пъзели са с повече от едно решение. Този проблем беше отстранен с пренаписване на class Solver, който отговаря за решаването на пъзелите и определянето на броя решения.

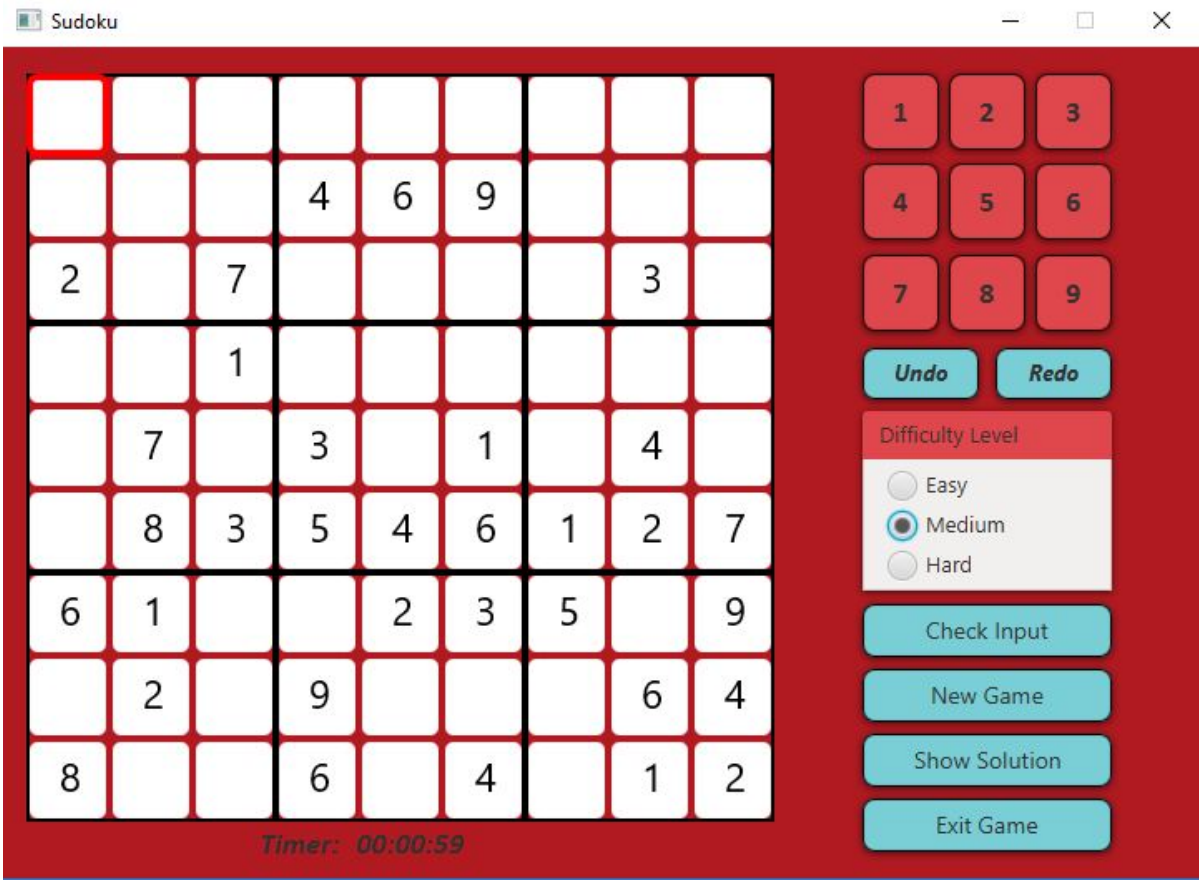
5. Графичен интерфейс



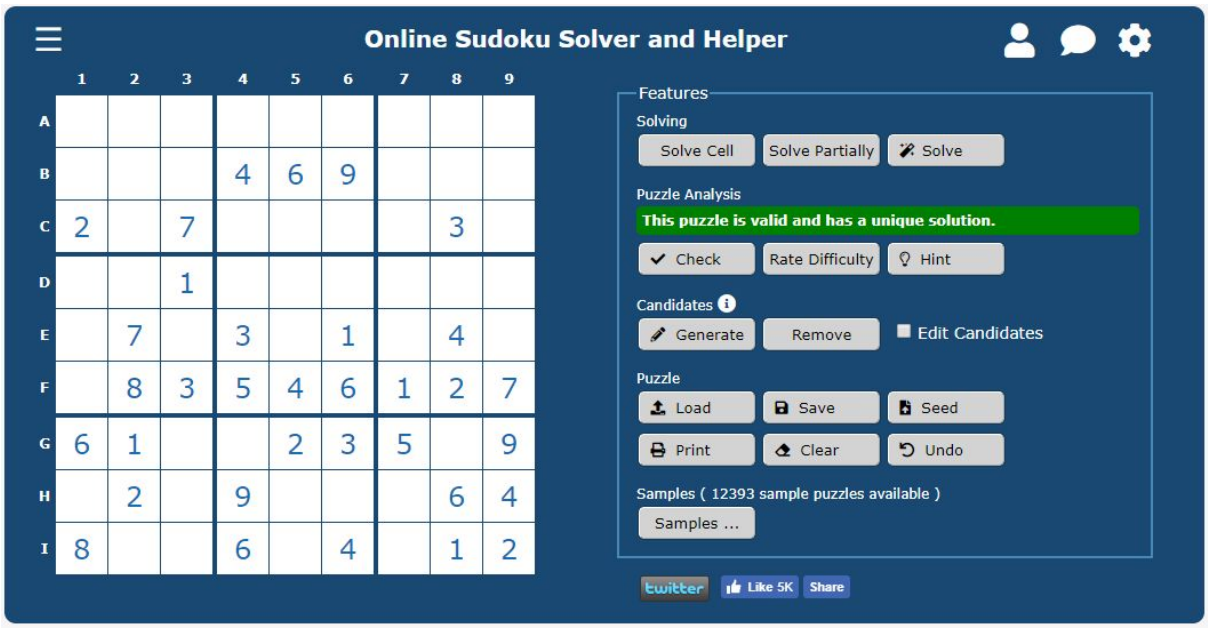
Фиг. 1 Графичен интерфейс на приложението. Въвеждане на стойности

6. Тестване на приложението

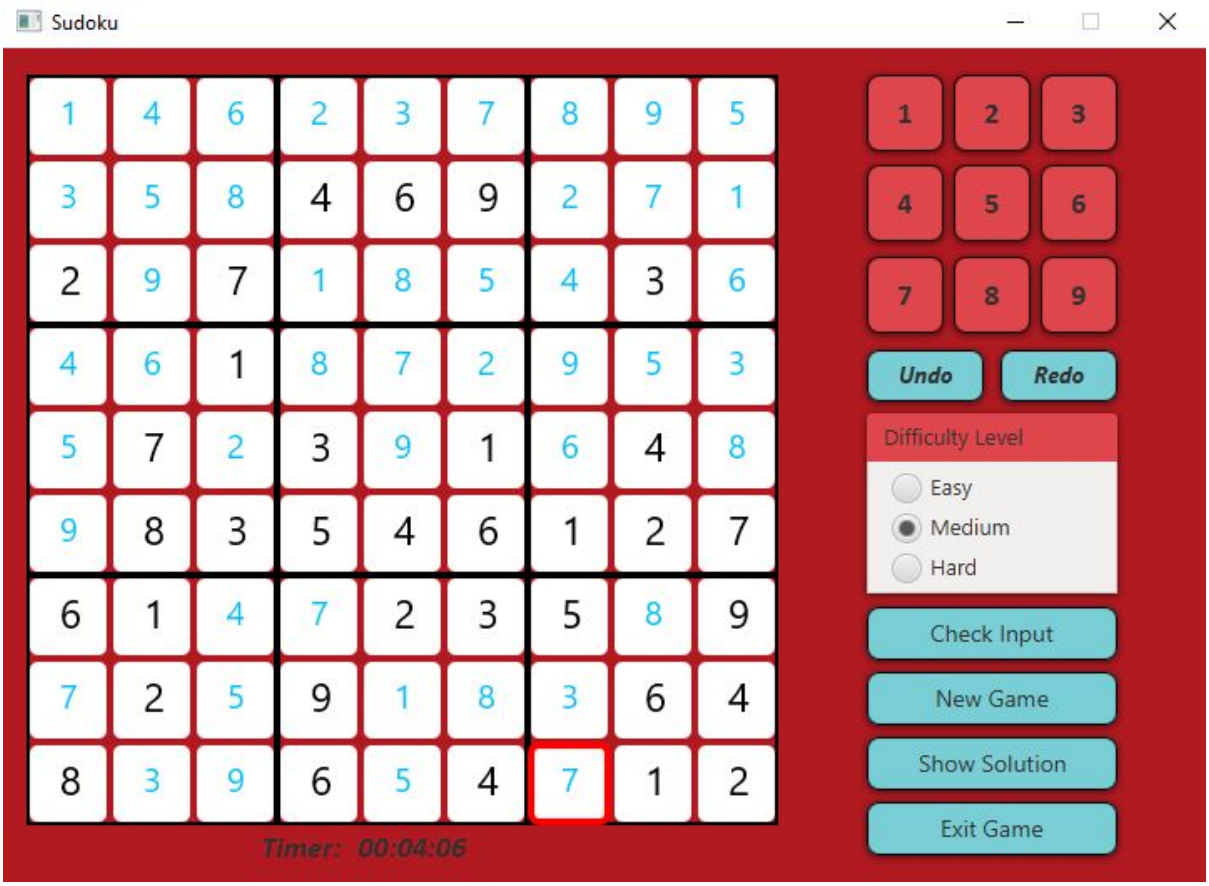
Приложението е тествано с помощта на външен Sudoku Solver - https://www.sudoku-solutions.com/. Въведени са генерираните судоку мрежи, за да се провери дали те са уникални с единствено решение. След което се въвеждат правилните стойности в приложението, и то показва съобщение за решен пъзел. Това означава, че играта генерира валидни пъзели, които могат да бъдат решени. При откриване на грешки приложението извежда съобщение за грешно решение и опитът продължава. Статистиките се записват от сървъра във файл statistics.txt, който се намира в папка SudokuServer.



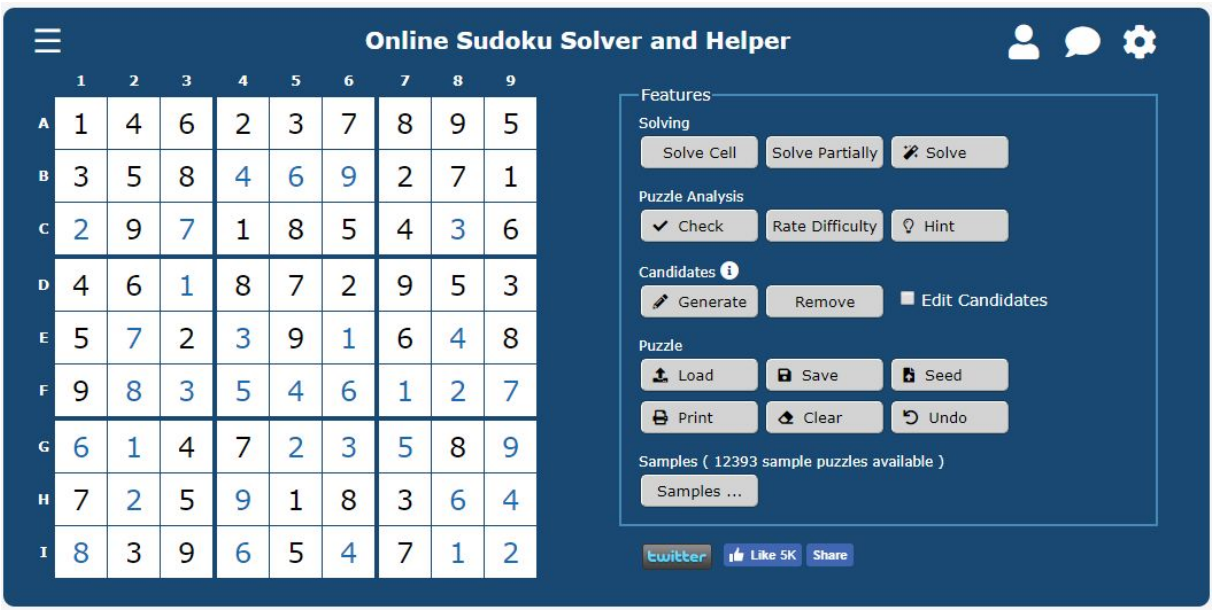
Фиг.2 Генериран пъзел от сървъра



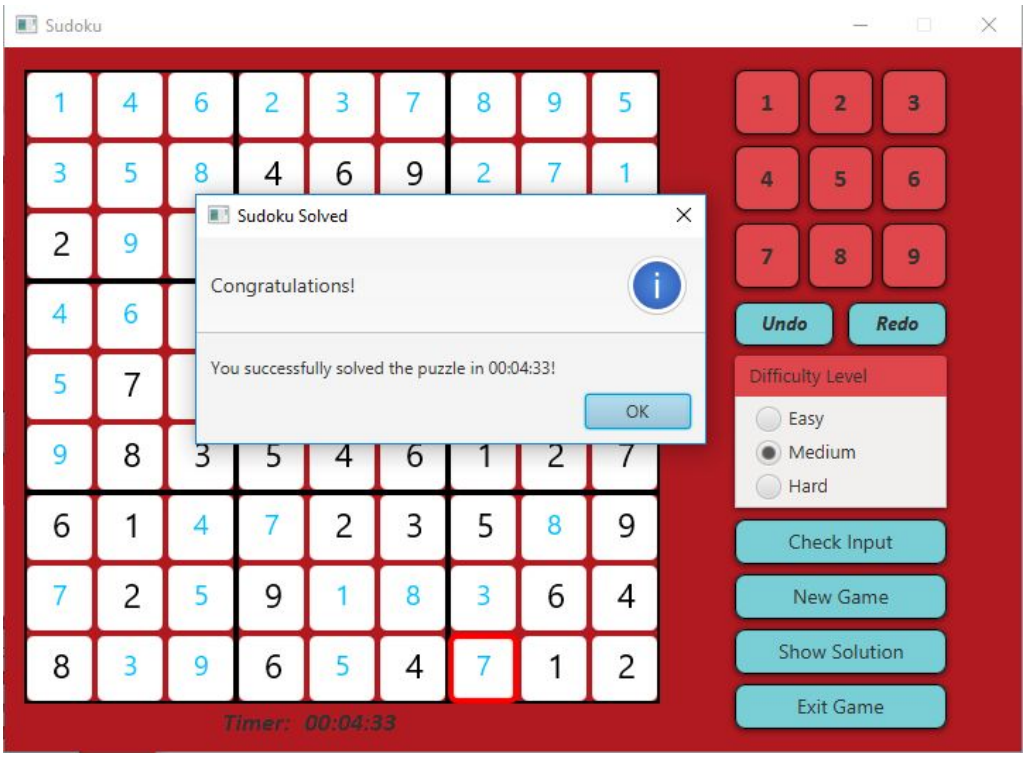
Фиг. 3 Потвърждение, че генерираният пъзел е валиден



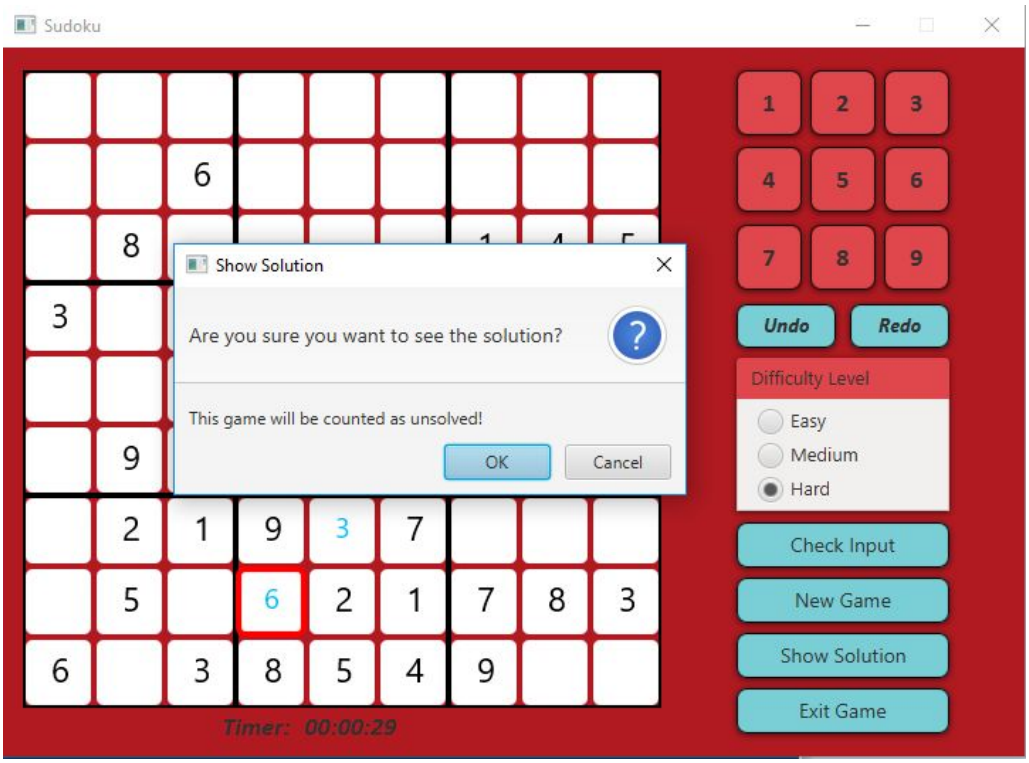
Фиг. 4 Попълнен пъзел от потребител



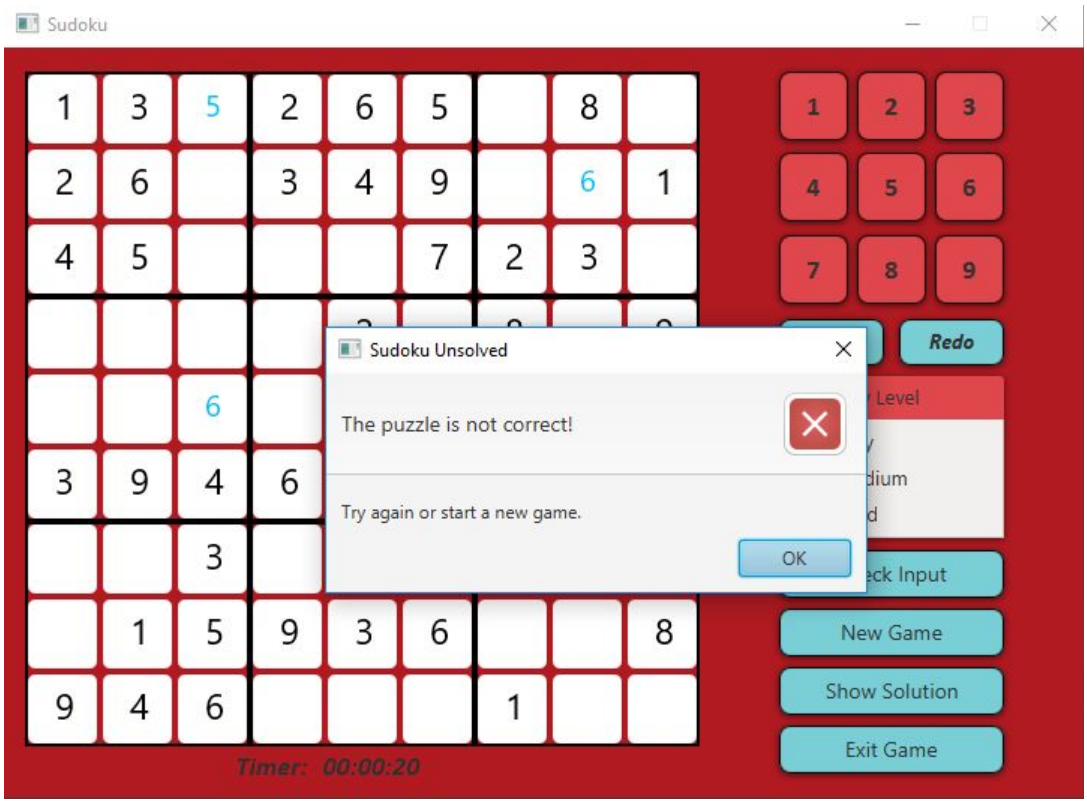
Фиг. 5 Потвърждение, че решението е валидно



Фиг. 6 Приложението отчита решението за вярно и съобщава на потребителя



Фиг. 7 Заявка за показване на решението на пъзела



Фиг. 8 При проверка, приложението извежда съобщение за грешно решение

7. Използвана литература

1) Евгений Кръстев, Lecture11c.pdf, Lecture14bFX.pdf
2) Bruce Eckel, “Thinking in Java”
3) http://zhangroup.aporc.org/images/files/Paper_3485.pdf
4) https://github.com/jcollard/captaincoder/tree/master/Java/sudoku-javafx