Semestrálny projekt - Sudoku

Tereza Štefíková

Opis

- Program je schopný vygenerovať vlastné sudoku.
- Je na výber z 3 náročností:
  - **easy** - cca 25% políčok je prázdnych
  - **medium** - cca 35% políčok je prázdnych
  - **hard** - cca 45% políčok je prázdnych
- Je na výber z 3 kategórii:
  - **9x9**, dopĺňajú sa číslice 1-9
  - **6x6** – **horizontálne**, dopĺňajú sa číslice 1-6
  - **6x6** – **vertikálne**, dopĺňajú sa číslice 1-6
- Na začiatku hry si hráč **musí** vybrať typ sudoku, ktorý chce hrať.
- Potom si môže vybrať obtiažnosť, ak tak neurobí automaticky sa mu zvolí „easy“.
- Taktiež si vie vybrať sudoku ktoré chce hrať, podľa jeho čísla.
- Projekt bol schválený **Ing. Františkom Gyárfášom, PhD.**

Pomocné enum class -y

```
enum class SudokuDifficulty {
    Easy,
    Medium,
    Hard
};
```

- SudokuDifficulty - má v sebe uložené typy náročnosti, ktoré si bude môcť hráč vybrať

```
enum class SudokuType {
    Sudoku6x6Horizontal,
    Sudoku6x6Vertical,
    Sudoku9x9
};
```

- SudokuType - má v sebe uložené typy hracej plochy a to 9x9 alebo 6x6 (tu si môže vybrať akým spôsobom ju chce mať orientovanú)

```
SudokuType SudokuPlan::Type() {
    return this->sudokuType;
}
```

- Vráti typ sudoku - sudokuType

Class SudokuPlan

```
class SudokuPlan {
private:
    int** sudoku;
    SudokuType sudokuType;
    SudokuDifficulty sudokuDifficulty;
public:
    SudokuPlan();
    SudokuPlan(SudokuType type);
    //~SudokuPlan();

    int GetBoxIndex(int lineIndex, int columnIndex);
    int GetBoxLineIndex(int index);
    int GetBoxColumnIndex(int index);
    bool ContainsAnyDuplicates(int *index, int size);
    bool IsColumnValid(int &index);
    bool IsLineValid(int &index);
    bool IsBoxValid(int &index);

    int TotalLinesInBox();
    int TotalColumnsInBox();
    int TotalBoxesInLine();
    int TotalBoxesInColumn();

    int TotalBoxes();
    int TotalLines();
    int TotalColumns();

    int GetNextNumberFromCirle(int min, int current, int max);
```

- sudoku je 2-rozmerné pole ktoré je v podstate našou hracou plochou
- sudokuType je druh sudoku, ktorý sa má vytvoriť a je typu SudokuType

```
    void SwitchLines(int line1Index, int line2Index);
    void SwitchLinesInBox(int boxIndex, int line1Index, int line2Index);
    void SwitchColumns(int column1Index, int column2Index);
    void SwitchColumnsInBox(int boxIndex, int column1Index, int column2Index);
    void SwitchBoxes(int box1Index, int box2Index);

    void SetDifficulty(SudokuDifficulty difficulty);
    void DropNumbers(int count);
    void ShakeExistingSudoku(int seed);
    void CreateNew(int seed);

    SudokuType Type();

    int GetValueOf(int lineIndex, int columnIndex);
    void SetValueTo(int value, int lineIndex, int columnIndex);

    bool IsValid();
};
#endif //CVICENIE5_SUDOKUPLAN_H
```

1. Časť - Kontrola sudoku

```
void SudokuPlan::SetValueTo(int value, int lineIndex, int columnIndex) {
    sudoku[lineIndex][columnIndex] = value;
}
```

- Nastaví vybrané miesto v poli na túto hodnotu

```
int SudokuPlan::GetValueOf(int lineIndex, int columnIndex) {
    return sudoku[lineIndex][columnIndex];
}
```

- Získa hodnotu vybraného prvku z poľa

1. Kontrola sudoku

```
SudokuPlan::SudokuPlan() : sudokuType(SudokuType::Sudoku9x9) {
    this->sudoku = new int *[SudokuPlan::TotalLines()];
    for (int i = 0; i < TotalLines(); ++i) {
        this->sudoku[i] = new int[TotalColumns()];
    }
    SudokuPlan::CreateNew( seed: 0);
}
```

- Je to konštruktor
- sudokuType nastaví na Sudoku9x9
- Alokuje 2-rozmerné pole **sudoku
- Zavolá funkciu CreateNew s 0, ktorá nám naplní pole **sudoku

```
SudokuPlan::SudokuPlan(SudokuType type) : sudokuType(type) {
    this->sudoku = new int *[TotalLines()];
    for (int i = 0; i < TotalLines(); ++i) {
        this->sudoku[i] = new int[TotalColumns()];
    }
    SudokuPlan::CreateNew( seed: 0);
}
```

- Je to konštruktor
- Robí to isté ako minulý konštruktor, ale teraz si môžeme vybrať aký typ sudoku chceme vytvoriť

1. Kontrola sudoku

```
int SudokuPlan::TotalLines() {
    switch (Type()) {
        case SudokuType::Sudoku9x9: {
            return 9;
        }
        case SudokuType::Sudoku6x6Vertical:
        case SudokuType::Sudoku6x6Horizontal: {
            return 6;
        }
        default: {
            return -1;
        }
    }
}
```

- Vráti celkový počet riadkov v sudoku

```
int SudokuPlan::TotalBoxes() {
    switch (Type()) {
        case SudokuType::Sudoku9x9: {
            return 9;
        }
        case SudokuType::Sudoku6x6Vertical:
        case SudokuType::Sudoku6x6Horizontal: {
            return 6;
        }
        default: {
            return -1;
        }
    }
}
```

- Vráti celkový počet boxov v sudoku

```
int SudokuPlan::TotalColumns() {
    switch (Type()) {
        case SudokuType::Sudoku9x9: {
            return 9;
        }
        case SudokuType::Sudoku6x6Horizontal:
        case SudokuType::Sudoku6x6Vertical: {
            return 6;
        }
        default: {
            return -1;
        }
    }
}
```

- Vráti celkový počet stĺpcov v sudoku

1. Kontrola sudoku

```
int SudokuPlan::TotalBoxesInLine() {
    switch (Type()) {
        case SudokuType::Sudoku9x9: {
            return 3;
        }
        case SudokuType::Sudoku6x6Vertical: {
            return 2;
        }
        case SudokuType::Sudoku6x6Horizontal: {
            return 3;
        }
        default: {
            return -1;
        }
    }
}
```

- Vráti počet boxov v riadku sudoku, na základe jeho typu

```
int SudokuPlan::TotalBoxesInColumn() {
    switch (Type()) {
        case SudokuType::Sudoku9x9: {
            return 3;
        }
        case SudokuType::Sudoku6x6Vertical: {
            return 3;
        }
        case SudokuType::Sudoku6x6Horizontal: {
            return 2;
        }
        default: {
            return -1;
        }
    }
}
```

- Vráti počet boxov v stĺpci sudoku, na základe jeho typu

1. Kontrola sudoku

```
int SudokuPlan::TotalColumnsInBox() {
    switch (Type()) {
        case SudokuType::Sudoku9x9: {
            return 3;
        }
        case SudokuType::Sudoku6x6Vertical: {
            return 3;
        }
        case SudokuType::Sudoku6x6Horizontal: {
            return 2;
        }
        default: {
            return -1;
        }
    }
}
```

- Vráti počet stĺpcov v boxe na základe typu sudoku

```
int SudokuPlan::TotalLinesInBox() {
    switch (Type()) {
        case SudokuType::Sudoku9x9: {
            return 3;
        }
        case SudokuType::Sudoku6x6Vertical: {
            return 2;
        }
        case SudokuType::Sudoku6x6Horizontal: {
            return 3;
        }
        default: {
            return -1;
        }
    }
}
```

- Vráti počet riadkov v boxe na základe typu sudoku

1. Kontrola sudoku

```
int SudokuPlan::GetBoxIndex(int lineIndex, int columnIndex) {
    int boxLineIndex = lineIndex / TotalLinesInBox();
    int orderOfBox = columnIndex / TotalColumnsInBox();
    return boxLineIndex * TotalBoxesInLine() + orderOfBox;
}
```

- Získa index boxu na základe indexov riadka a stĺpca

```
int SudokuPlan::GetBoxLineIndex(int index) {
    return (index / TotalBoxesInLine()) * TotalLinesInBox();
}
```

- Z indexu boxu, získa index jeho prvého riadka

```
int SudokuPlan::GetBoxColumnIndex(int index) {
    return (index % TotalBoxesInLine()) * TotalColumnsInBox();
}
```

- Z indexu boxu, získa index jeho prvého stĺpca

1. Kontrola sudoku

```
bool SudokuPlan::ContainsAnyDuplicates(int *index, int size) {
    bool result = false;
    for (int number = 1; number <= size; ++number) {
        for (int i = 0; i < size; ++i) {
            if (result) {
                if (index[i] == number) {
                    return result;
                }
            } else {
                result = index[i] == number;
            }
        }
        result = false;
    }

    return result;
}
```

- Načíta pole
- Prebehne pole v cykle a kontroluje či sa v ňom opakujú prvky, ak áno vráti true inak vráti false

1. Prvý cyklus ide od 1 po posledné číslo, čo sa môže v sudoku nachádzať (napr. 1-9)
2. V ňom vnorený cyklus už ide cez indexy prvkov preto ide od 0
3. Kým result == false, kontrolujeme či sa number zhoduje s číslom z pola.
4. Ak sa zhoduje nastavíme result na true a ideme ďalej vo vnorenom for-cykle.
   - Ak result == true, vôjdeme do podmienky a kontrolujeme či sa ďalší prvok rovná s number, ak as rovná znamená to že máme duplicitu. Vrátime result – ten je teraz true.
   - Ak sa nerovná ideme na ďalší prvok až po size-1.
5. Inak ak prejdeme celé pole, pre všetky čísla čo sa tam môžu nacháďzať a nenájdeme žiadnu duplicitu vrátime false.

1. Kontrola sudoku

```
bool SudokuPlan::IsColumnValid(int &index) {
    int size = TotalLines();
    int numberStore[size];

    for (int i = 0; i < size; ++i) {
        numberStore[i] = GetValueOf(i, index);
    }

    return !ContainsAnyDuplicates(numberStore, size);
}
```

- Vráti true ak je stĺpec valídny, false ak nie je

1. Načíta index stĺpca ktorý chceme skontrolovať
2. Zadefinuje si pomocné pole **numberStore** o veľkosti počtu riadkov v sudoku (**TotalLines()**)
3. Prebehne pole v cykle a postupne si naplní **numberStore** hodnotami zo stĺpca
4. Zavolá **ContainsAnyDuplicates** a jeho bool hodnotu zneguje keďže tu chceme jeho opak

1. Kontrola sudoku

```
bool SudokuPlan::IsLineValid(int &index) {
    int size = TotalColumns();
    int numberStore[size];

    for (int i = 0; i < size; ++i) {
        numberStore[i] = GetValueOf(index, i);
    }

    return !ContainsAnyDuplicates(numberStore, size);
}
```

- Vráti true ak je riadok valídny, false ak nie je

1. Načíta index riadka ktorý chceme skontrolovať
2. Zadefinuje si pomocné pole **numberStore** o veľkosti počtu stĺpcov v sudoku (**TotalLines()**)
3. V cykle postupne naplní pole **numberStore** hodnotami z riadka
4. Zavolá **ContainsAnyDuplicates** a jeho bool hodnotu zneguje keďže tu chceme jeho opak

1. Kontrola sudoku

```
bool SudokuPlan::IsBoxValid(int &index) {
    int size = TotalLinesInBox() * TotalColumnsInBox();
    int numberStore[size];
    int numberStoreIndex = 0;

    int minLineBoxIndex = GetBoxLineIndex(index);
    int maxLineBoxIndex = minLineBoxIndex + TotalLinesInBox();
    int minColumnBoxIndex = GetBoxColumnIndex(index);
    int maxColumnBoxIndex = minColumnBoxIndex + TotalColumnsInBox();

    for (int i = minLineBoxIndex; i < maxLineBoxIndex; ++i) {
        for (int j = minColumnBoxIndex; j < maxColumnBoxIndex; ++j) {
            numberStore[numberStoreIndex] = GetValueOf(i, j);
            numberStoreIndex++;
        }
    }

    return !ContainsAnyDuplicates(numberStore, size);
}
```

- Vráti true ak je box validný, false ak nie je

1. Načíta index riadka ktorý chceme skontrolovať
2. Do **size** uloží zistenú veľkosť boxu
3. Zadefinuje si pomocné pole **numberStore** o veľkosti **size**
4. Vytvorí premennú **minLineBoxIndex** ktorá bude začiatočnou hodnotou v for-cykle, a naplní ju prvým riadkom z daného boxu
5. Vytvorí premennú **maxLineBoxIndex** ktorá bude konečnou hodnotou v for-cykle (prvý riadok boxu + počet všetkých riadkov v boxe -> to nám dá hodnotu o 1 väčšiu než je index posledného riadku boxu)
6. Vytvorí premennú **minColumnBoxIndex** ktorá bude začiatočnou hodnotou v for-cykle, a naplní ju prvým stĺpcom z daného boxu
7. Vytvorí premennú **maxColumnBoxIndex** ktorá bude konečnou hodnotou v for-cykle (prvý stĺpcom boxu + počet všetkých stĺpcov v boxe -> to nám dá hodnotu o 1 väčšiu než je index posledného stĺpca boxu)
8. V cykle postupne naplní pole **numberStore** hodnotami z boxu
9. Zavolá **ContainsAnyDuplicates** a jeho bool hodnotu zneguje keďže tu chceme jeho opak

1. Kontrola sudoku

```
bool SudokuPlan::IsValid() {
    bool result = true;

    for (int column = 0; column < TotalColumns(); ++column) {
        result &= IsColumnValid( &: column);
    }
    for (int line = 0; line < TotalLines(); ++line) {
        result &= IsLineValid( &: line);
    }
    for (int box = 0; box < TotalBoxes(); ++box) {
        result &= IsBoxValid( &: box);
    }

    return result;
}
```

1. .Result nastaví ako true.
2. V cykle prejde všetky stĺpce v sudoku a zisťuje, či sú valídne.
3. V cykle prejde všetky riadky v sudoku a zisťuje, či sú valídne.
4. V cykle prejde všetky boxy v sudoku a zisťuje, či sú valídne.
5. Ak čo i len raz nastane že sa result zmení na false, tak vďaka **&** funkcia vráti false.

- Vráti true ak je sudoku valídne, false ak nie je

2. Časť - Generovanie nového sudoku

```
int GetNextNumberFromCirle(int min, int current, int max);

void SwitchLines(int line1Index, int line2Index);
void SwitchLinesInBox(int boxIndex, int line1Index, int line2Index);
void SwitchColumns(int column1Index, int column2Index);
void SwitchColumnsInBox(int boxIndex, int column1Index, int column2Index);
void SwitchBoxes(int box1Index, int box2Index);

void SetDifficulty(SudokuDifficulty difficulty);
void DropNumbers(int count);
void ShakeExistingSudoku(int seed);
void CreateNew(int seed);
```

- Rozhodla som sa ho generovať tak, že na začiatku sa vytvorí vždy to isté sudoku a na základe presunov jeho stĺpcov, riadkov a boxov sa vytvorí nové sudoku

2. Generovanie nového sudoku

```
void SudokuPlan::SwitchColumns(int column1Index, int column2Index) {
    if(column1Index == column2Index){
        return;
    }

    int size = TotalLines();
    int tempMemory, newValue;

    for (int i = 0; i < size; ++i) {
        tempMemory = GetValueOf(i, column1Index);
        newValue = GetValueOf(i, column2Index);
        SetValueTo(newValue, i, column1Index);
        SetValueTo(tempMemory, i, column2Index);
    }
}
```

- Navzájom vymení zadané stĺpce:
- Ak sa budú indexy stĺpcov zhodovať nespraví nič
- Inak prejde v cykle všetky prvky pre každý stĺpec a navzájom ich prehodí cez funkciu SetValueTo

```
void SudokuPlan::SwitchLines(int line1Index, int line2Index) {
    if(line1Index == line2Index){
        return;
    }

    int size = TotalColumns();
    int tempMemory, newValue;

    for (int i = 0; i < size; ++i) {
        tempMemory = GetValueOf(line1Index, i);
        newValue = GetValueOf(line2Index, i);
        SetValueTo(newValue, line1Index, i);
        SetValueTo(tempMemory, line2Index, i);
    }
}
```

- Navzájom vymení zadané riadky:
- Ak sa budú indexy riadkov zhodovať nespraví nič
- Inak prejde v cykle všetky prvky pre každý riadok a navzájom ich prehodí cez funkciu SetValueTo

2. Generovanie nového sudoku

```
void SudokuPlan::SwitchColumnsInBox(int boxIndex, int column1Index, int column2Index) {
    if(column1Index == column2Index){
        return;
    }

    int boxMinColumn = GetBoxColumnIndex(boxIndex);
    int boxMaxColumn = boxMinColumn + TotalColumnsInBox();

    int newColumn1Index = boxMinColumn + column1Index;
    int newColumn2Index = boxMinColumn + column2Index;

    if ((boxMinColumn <= newColumn1Index && newColumn1Index < boxMaxColumn)
        && (boxMinColumn <= newColumn2Index && newColumn2Index < boxMaxColumn)) {

        SwitchColumns(newColumn1Index, newColumn2Index);
    }
}
```

1. Ak sú indexy stĺpcov rovnaké nespraví nič
2. Inak skontroluje pre každý columnIndex či patrí do daného boxu
3. Ak áno, vymení ich cez funckciu SwitchColumns
4. Ak nie, neurobí nič

- Navzájom vymení zadané stĺpce v danom boxe

2. Generovanie nového sudoku

```
void SudokuPlan::SwitchLinesInBox(int boxIndex, int line1Index, int line2Index) {
    if(line1Index == line2Index){
        return;
    }

    int boxMinLine = GetBoxLineIndex(boxIndex);
    int boxMaxLine = boxMinLine + TotalLinesInBox();

    int newLine1Index = boxMinLine + line1Index;
    int newLine2Index = boxMinLine + line2Index;

    if ((boxMinLine <= newLine1Index && newLine1Index < boxMaxLine)
        && (boxMinLine <= newLine2Index && newLine2Index < boxMaxLine)) {

        SwitchLines(newLine1Index, newLine2Index);
    }
}
```

1. Ak sú indexy riadkov rovnaké nespraví nič
2. Inak skontroluje pre každý lineIndex či patrí do daného boxu
3. Ak áno, vymení ich cez funkciu SwitchLines
4. Ak nie, neurobí nič

- Navzájom vymení zadané riadky v danom boxe

2. Generovanie nového sudoku

```
void SudokuPlan::SwitchBoxes(int box1Index, int box2Index) {
    if(box1Index == box2Index){
        return;
    }

    int box1MinLine = GetBoxLineIndex(box1Index);
    int box1MaxLine = box1MinLine + TotalLinesInBox();
    int box1MinColumn = GetBoxColumnIndex(box1Index);
    int box1MaxColumn = box1MinColumn + TotalColumnsInBox();

    int box2MinLine = GetBoxLineIndex(box2Index);
    int box2MinColumn = GetBoxColumnIndex(box2Index);

    for (int i = box1MinLine, j = box2MinLine; i < box1MaxLine; ++i, ++j) {
        SwitchLines(i, j);
    }
    for (int i = box1MinColumn, j = box2MinColumn; i < box1MaxColumn; ++i, ++j) {
        SwitchColumns(i, j);
    }
}
```

1. Ak sú indexy boxov rovnaké nespraví nič
2. Inak si pre oba boxy nastaví začiatočné indexy pre riadky aj stĺpce a zistí konečný index len pre jeden box, keďže ich veľkosť musí byť rovnaká
3. V cykle prejde všetky prvky v ich riadkoch a navzájom ich vymení cez SwitchLines
4. V cykle prejde všetky prvky v ich stĺpcoch a navzájom ich vymení cez SwitchColumns

- Navzájom vymení zadané riadky a stĺpce v daných boxoch

2. Generovanie nového sudoku

```
void SudokuPlan::ShakeExistingSudoku(int seed) {
    int min = 0;
    int boxIndex = min;

    int max = TotalBoxes() - 1;
    for (int i = 0; i < seed; ++i) {
        SwitchLinesInBox( boxIndex: 0, line1Index: 0, line2Index: 1);

        SwitchBoxes(boxIndex,  box2Index: GetNextNumberFromCirle(min, boxIndex, max));
        boxIndex = GetNextNumberFromCirle(min, boxIndex, max);

        SwitchColumnsInBox( boxIndex: 0,  column1Index: 0, column2Index: 1);
    }
}
```

1. V for-cykle prehodí riadky boxov cez funkciu SwitchLinesInBox
2. Prehodí celé boxy cez funkciu SwitchBoxes
3. Prehodí stĺpce v boxe cez funkciu SwitchColumnsInBox
4. Zväčší index boxu a zopakuje cyklus

- Vytvorí nové sudoku, prehodením jeho boxov častí boxov, toľkokrát koľko je seed

2. Generovanie nového sudoku

```
int SudokuPlan::GetNextNumberFromCirle(int min, int current, int max) {
    if (min <= current && current < max) {
        return ++current;
    }
    return min;
}
```

- Zväčší nám index

2. Generovanie nového sudoku

```
void SudokuPlan::CreateNew(int seed) {
    if (seed == 0) {
        int min = 1;
        int max = TotalLines();
        int sizeOfBox = TotalBoxesInLine();

        int initialValue = 0;
        int newValue = 0;
        for (int i = 0; i < max; ++i) {
            if ((i % sizeOfBox) == 0) {
                newValue = ++initialValue;
            } else {
                newValue += TotalBoxesInColumn();
            }

            for (int j = 0; j < max; ++j) {
                SetValueTo(newValue, i, j);
                newValue = GetNextNumberFromCirle(min, newValue, max);
            }
        }
    } else {
        CreateNew( seed: 0);
        ShakeExistingSudoku(seed);
        IsValid();
        SudokuPlan::SetDifficulty( difficulty: sudokuDifficulty);
    }
}
```

- Vytvorí nové sudoku, podľa zadaného čísla

1. Ak sú indexy boxov rovnaké nespraví nič
2. Inak si pre oba boxy nastaví začiatočné indexy pre riadky aj stĺpce a zistí konečný index len pre jeden box, keďže ich veľkosť musí byť rovnaká
3. V cykle prejde všetky prvky v ich riadkoch a navzájom ich vymení cez SwitchLines
4. V cykle prejde všetky prvky v ich stĺpcoch a navzájom ich vymení cez SwitchColumns

3. Časť - Príprava na hru

```
void SudokuPlan::SetDifficulty(SudokuDifficulty difficulty) {
    int count = TotalLines() *  TotalColumns();
    float coefficient = 0;
    switch (difficulty){
        case SudokuDifficulty::Easy:{
            coefficient = 0.25;
            break;
        }
        case SudokuDifficulty::Medium:{
            coefficient = 0.35;
            break;
        }
        case SudokuDifficulty::Hard:{
            coefficient = 0.45;
            break;
        }
        default:{
            coefficient = 0;
            break;
        }
    }
    int numberOfDropped = count * coefficient;
    DropNumbers(numberOfDropped);
}
```

- Nastavuje typ náročnosti

1. Zistí ktorý typ náročnosti bol zvolený
2. Vyberie podľa neho koeficient
3. Zavolá funkciu na vytvorenie prázdnych políčok DropNumbers

3. Príprava na hru

```
void SudokuPlan::DropNumbers(int count) {
    int counter = 0;
    for (int lineIndex = 0; lineIndex < TotalLines(); ++lineIndex) {
        for (int columnIndex = 0; columnIndex < TotalColumns(); ++columnIndex) {
            if(counter % 2 == 0){
                if(columnIndex != 0 && columnIndex % 2 ==0 ){
                    SetValueTo( value: -1, columnIndex, lineIndex);
                    counter++;
                }
            }
            else if(counter % 2 != 0){
                if(lineIndex != 0 && columnIndex != 0 && columnIndex % 2 != 0){
                    SetValueTo( value: -1, lineIndex, columnIndex);
                    counter++;
                }
            }
            if(counter == count){
                return;
            }
        }
    }
}
```

- Zruší určitý počet prvkov v sudoku

1. V for-cykle prejde všetky prvky sudoku
2. A cca toľkokrát aké číslo je v count nahradí vybraný prvok -1 (pretože tá sa pri vykreslovaní objaví ako x)

   (poznámka: pri každom sudoku budú chýbať políčka na rovnakých indexoch

4. Časť - Class Game

```
#ifndef CVICENIE5_GAME_H
#define CVICENIE5_GAME_H

#include "SudokuPlan.h"

class Game {
private:
    SudokuPlan sudokuPlan;
public:
    Game(SudokuType type);
    void Show();
    void NextGame(int seed);
    void SetDifficulty(SudokuDifficulty difficulty);
    int FillNumber(int newNumber, int lineIndex, int columnIndex);
};

#endif //CVICENIE5_GAME_H
```

- sudokuPlan sú naše dáta z **class SudokuType**.

4. Class Game

```
Game::Game(SudokuType type) {
    this->sudokuPlan = SudokuPlan(type);
    Game::SetDifficulty( difficulty: SudokuDifficulty::Easy);
}
```

Game je konštruktor, ktorý vytvorí sudoku podľa typu aký zadáme. Automaticky mu nastaví obtiažnosť easy.

```
void Game::SetDifficulty(SudokuDifficulty difficulty){
    sudokuPlan.SetDifficulty(difficulty);
}
```

SetDifficulty nastaví obtiažnosť podľa výberu hráča.

```
void Game::NextGame(int seed) {
    sudokuPlan.CreateNew(seed);
}
```

Vytvorí nové sudoku

4. Class Game

```
int Game::FillNumber(int newNumber, int lineIndex, int columnIndex) {
    int previousValue = sudokuPlan.GetValueOf(lineIndex, columnIndex);

    sudokuPlan.SetValueTo(newNumber, lineIndex, columnIndex);
    int boxIndex = sudokuPlan.GetBoxIndex(lineIndex, columnIndex);

    if (sudokuPlan.IsColumnValid( &: columnIndex)
        && sudokuPlan.IsLineValid( &: lineIndex)
        && sudokuPlan.IsBoxValid( &: boxIndex)) {

        return sudokuPlan.GetValueOf(lineIndex, columnIndex);
    } else {

        sudokuPlan.SetValueTo(previousValue, lineIndex, columnIndex);
        return sudokuPlan.GetValueOf(lineIndex, columnIndex);
    }
}
```

- Ak môžeme, vložíme do sudoku nový prvok a vrátime jeho hodnotu

1. Do previousValue si uložíme pôvodnú hodnotu prvku ktorý ideme zmeniť
2. Prvok prepíšeme
3. Zistíme index jeho boxu
4. Skontrolujeme si či je sudoku po doplnení valídne
   - Ak áno, vrátime doplnenú hodnotu
   - Ak nie, prepíšeme hodnotu naspäť na pôvodnú a vrátime tú

4. Class Game

- Vypíše sudoku, voľné políčka nahradí za x

```
void Game::Show() {
    for (int line = 0; line < sudokuPlan.TotalLines(); line++) {
        if (line > 0 && line % sudokuPlan.TotalLinesInBox() == 0) {
            for (int column = 0; column < sudokuPlan.TotalColumns() + sudokuPlan.TotalBoxesInLine() - 1; column++) {
                cout << "-" << "  ";
            }
            cout << endl;
        }

        for (int column = 0; column < sudokuPlan.TotalColumns(); column++) {
            if (column > 0 && column % sudokuPlan.TotalBoxesInColumn() == 0) {
                cout << "|" << "  ";
            }
            int value = sudokuPlan.GetValueOf(line, column);
            if (value > 0) {
                cout << value << "  ";
            } else {
                cout << "X" << "  ";
            }
        }

        cout << endl;
    }
}
```