Gymnázium, Praha 6, Arabská 14

Programování

Ročníková práce

2023 **Oliver Hurt, 2.E**

Čestné prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne 28.4. 2023 podpis: ……

Anotace

Práce obsahuje stručný popis celulárního automatu známého pod názvem Game Of Life. Dále obsahuje i popis implementace – některých metod a tříd. V závěru je ukázka programu a návrhy pro další funkce, které by bylo vhodné do programu zakomponovat. Program je napsán v programovacím jazyce Java a kód je dostupný online – https://github.com/gyarab/2022-2e-hurt-GameOfLife

The paper contains a brief description of a cellular atomaton known as Game Of Life. It also contains a description of the implementation – some methods and classes. The paper also includes preview of the program and suggestions for additional features that could be incorporated into the program. The program is written in the Java programming language and the code is available online – https://github.com/gyarab/2022-2e-hurt-GameOfLife

Zadání

Zadání mého ročníkového projektu znělo následovně.

„Conwayova hra života – 2D grid buňek. V počátečním stavu je každá buňka mrtvá. Pro každou buňku platí jednoduchá pravidla. Každá živá buňka s méně než dvěma živými sousedy zemře.
Každá živá buňka se dvěma nebo třemi živými sousedy zůstává žít.
Každá živá buňka s více než třemi živými sousedy zemře.
Každá mrtvá buňka s právě třemi živými sousedy oživne. Vždy se vypočítá stav další generace pro celou plochu a pak se najednou přemění. I s takto jednoduchým systémem lze vytvářet komplexní tvary. Koncový stav nelze předpovědět. Desktopová aplikace. Na začátku se klikne na jednotlivé buňky nastaví počáteční stav. —> spustí se simulace.”

1. Úvod

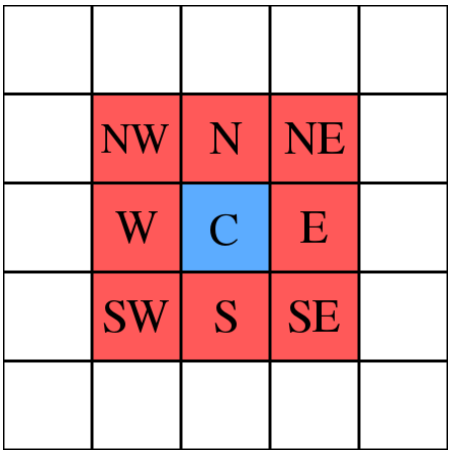
Cílem projektu bylo vytvořit základ, pro Game Of Life, který mohou použít i jiní lidé. Snažil jsem se psát program čitelně tak, aby pokud někdo bude chtít experimentovat a modifikovat Game Of Life, nebude muset začínat od nuly. Program je psán objektově, aby bylo co nejjednodušší třídy Cell upravit a rozšířit pro komplexnější celulární automaty.

2. Game Of Life

Game Of Life známá také pod jménem Conwayova Hra života je hra o nula hráčích, která nemá konec. Hrou je spíše obrazně – řadí se mezi celulární automaty (angl. celluar automata). Hra života svým chováním připomíná chování živého organismu. Odehrává na nekonečném poli buněk, které mohou být mrtvé nebo živé. Jejich stav se mění podle následujících pravidel.

1. Každá živá buňka s méně než dvěma živými sousedy zemře.
2. Každá živá buňka se dvěma nebo třemi živými sousedy zůstává žít.
3. Každá živá buňka s více než třemi živými sousedy zemře.
4. Každá mrtvá buňka s právě třemi živými sousedy oživne.

Za sousedy se považují ty buňky, které se dané buňky dotýkají hranou či vrcholem (Moorovo okolí) – viz obrázek č. 1.



Obrázek č.1. – Moorovo okolí

Nový stav buňky určuje přechodová funkce podle stavu této buňky a jejího okolí v předchozím kroku[1]. V našem případě je přechodová funkce definována právě výše uvedenými pravidly. Stav všech buněk se mění naráz[2].

Game Of Life a celulární automaty obecně jsou zajímavé protože, je lze použít k modelování mnoha různých věcí – například fyzikálních systémů od dynamiky tekutin přes spřažené magnetické spiny až po chemické reakčně-difúzní systémy[3] nebo např. šíření virových onemocnění ve společnosti.

[1]Izhikevich, E. M., Conway, J. H., & Seth, A. (2015, June 21). *Game of Life - Scholarpedia*. Game of Life - Scholarpedia. https://doi.org/10.4249/scholarpedia.1816

[2]Lipa, C. (n.d.). *Conway's Game of Life'*. Conway's Game of Life'. http://pi.math.cornell.edu/~lipa/mec/lesson6.html

[3]Roberts, S. (2020, December 28). *The lasting lessons of John Conway's Game of Life*. The New York Times. Retrieved May 4, 2023, from https://www.nytimes.com/2020/12/28/science/math-conway-game-of-life.html

3. Seznam tříd

MainMenu

Program začíná voláním funkce main() ve Třídě MainMenu. MainMenu pomocí knihovny AWT vytvoří menu se **třemi** tlačítky. Z menu můžeme nastavit počáteční konfiguraci nebo spustit simulaci.

ButtonPanel

Třída ButtonPanel je zavolána[4] po stisknutí tlačítka „Konfigurace počátečního stavu". Třída na obrazovku vykreslí pole tlačítek. Chování po stisku tlačítka je definováno ve třídě ButtonSettings. Ve výchozím nastavení se po kliknutí tlačítko zbarví modře. Tím indikuje, že daná buňka je živá. Pokud na modrou buňku klikneme, vrátí se do původního nastavení – mrtvá buňka.

Cell

Třída Cell definuje buňku. Každá buňka zná svoje souřadnice a zda je na živu.

GameOfLife

Třída GameOfLife nastavuje scénu pro simulaci. Je spouštěna po kliknutí na tlačítko „spustit simulaci" v menu. Vykresluje Canvas pro animaci a Hbox pro tlačítka. Pomocí tlačítek můžeme animaci spustit, zastavit, resetovat nebo přejít na další krok. Třída obsahuje handler pro animaci a zahajuje simulaci voláním třídy Life.

Life

Třída Life obsahuje metody definující chování simulace a je stěžejní třídou projektu. Třída nastavuje počáteční stav – buď použije předtím v menu nastavenou konfiguraci nebo vygeneruje náhodně novou.

[4] Ve skutečnosti není volána Třída ale její konstruktor při vytváření instance dané Třídy. V programu často vytvářím pouze jednu instanci Třídy a ta má za úkol vykreslovat scénu. Používám proto zjednodušení ve formě „Třída je zavolána".

4. Třída ButtonSettings

```
boolean alive = false;

ButtonSettings(int i, int j) {
    super("");
    this.x = i;
    this.y = j;
}

@Override
public void actionPerformed(ActionEvent arg0) {
    //System.out.println("Do " + arg0.getActionCommand());

    if (arg0.getSource() instanceof JButton) {
        if(alive == false) {//na button se kliknulo po prve
            ((JButton) arg0.getSource()).setBackground(Color.BLUE);
            ((JButton) arg0.getSource()).setContentAreaFilled(true);
            ((JButton) arg0.getSource()).setOpaque(true);
            alive = true;

        }
        else if(alive == true){//na button se kliknulo po druhe
            ((JButton) arg0.getSource()).setBackground(new Color(238,238,238));
            ((JButton) arg0.getSource()).setContentAreaFilled(true);
            ((JButton) arg0.getSource()).setOpaque(true);
            alive = false;
        }
        //System.out.println(this.x + " " + this.y + " " + this.alive);
    }
}
```

Třída ButtonSettings nastavuje chování tlačítek. Objekt typu ButtonSettings má tři atributy boolean *alive*, int *x* a int *y*. Po kliknutí na dané tlačítko se kontroluje stav tlačítka. Pokud je tlačítko „mrtvé" má šedivou barvu a pokud je „živé" nastaví se barva tlačítka na modrou. Proměnné *x* a *y* reprezentují souřadnice, na kterých se tlačítko nachází.

5. Animation handler

```
AnimationTimer runAnimation = new AnimationTimer() {
    private long lastUpdate = 0;

    @Override
    public void handle(long now) {
        if ((now - lastUpdate) >= TimeUnit.MILLISECONDS.toNanos(100)) {
            life.tick();
            lastUpdate = now;
        }
    }
};
```

Animation handler je součástí třídy GameOfLife. Implementuje metodu *handle* třídy AnimationTimer[5]. V proměnné *now* je uložen čas od spuštění frame v nanosekundách. Od *now* odčítáme hodnotu proměnné *lastUpdate* a pokud je rozdíl větší než daný interval (v našem případě 100 ns), aktualizujeme stav buněk metodou *tick()*.

[5]*AnimationTimer (JavaFX 8)*. (n.d.). AnimationTimer (JavaFX 8). https://docs.oracle.com/javase/8/javafx/api/javafx/animation/AnimationTimer.html

6. Metoda tick()

```
public void tick() {
    //pro kazdou bunkku
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            Cell c = grid[i][j];
            int neighbors = countAliveNeighbors(c, grid);

            //definujeme pravidla - game of life
            if (c.stav == 0 && neighbors == 3) {
                Cell x = new Cell(i, j, 1);
                next[i][j] = x;
            }
            else if (c.stav == 1) {
                if (neighbors < 2) {
                    Cell x = new Cell(i, j, 0);
                    next[i][j] = x;
                }
                else if (neighbors > 3) {
                    Cell x = new Cell(i, j, 0);
                    next[i][j] = x;
                }
                else {
                    //zustane stejny stav
                    next[i][j] = grid[i][j];
                }
            }
        }
    }

    //prekopirujeme a vykreslime vysledek
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            grid[i][j] = next[i][j];
        }
    }
    draw();
}
```

Metoda tick stanovuje jak bude vypadat další generace. Na základě pravidel buňka umře, ožije, nebo zůstane její stav stejný.

Pro každou buňku zjistíme počet sousedů pomocí metody *countAliveNeighbors().* Pokud je buňka mrtvá a má přesně tři sousedy, tak v další generaci ožije. Pokud je buňka živá a má méně než dva sousedy, tak umře. Pokud je buňka živá a má více než 3 sousedy, tak také umře. Jinak zůstane ve stejném stavu, jako v předchozí generaci.

Nová generace se zapisuje do 2D pole *next*. To děláme protože procházíme pole postupně a pokud by se změna projevila rovnou v tom samém poli, ovlivnilo by to výpočet živých sousedů u dalších buněk, což by vedlo k celkově nesprávnému výpočtu.

Po tom co máme celou novou generaci správně zapsanou v poli *next,* překopírujeme ji do pole *grid*. Potom pole *grid* vykreslíme metodou *draw()*.

Aby probíhala simulace, je metoda *tick()* volána v nekonečné smyčce.

7. Metoda countAliveNeighbors()

```
private int countAliveNeighbors(Cell c, Cell maze[][]) {
    int sum = 0;
    for (int i = -1; i < 2; i++) {
        for (int j = -1; j < 2; j++) {
            int col = (c.x + i + rows) % rows;
            int row = (c.y + j + cols) % cols;
            sum += maze[col][row].stav;
        }
    }
    int u = c.x;
    int z = c.y;
    sum = sum - maze[u][z].stav;
    return sum;
}
```

Metoda countAliveNeighbors() spočítá, kolik má daná buňka sousedů. Metoda používá Moorovo sousedství. Prochází postupně čtverec 3x3 s danou buňkou uprostřed a pokud narazí na živou buňku, přičte si jedna. Jelikož se Hra života odehrává na nekonečném prostoru, musíme ošetřit případy, kdy hledáme sousedy nacházející se na kraji pole. Uvažme následující příklad.

0,0									
6,0								6,8	6,9
7,0								7,8	**7,9**
8,0								8,8	8,9
									9,9

Obrázek č. 2 – sousedství buňky

Červený čtverec je daná buňka a její sousedi jsou označeni zeleně. Čísla uvnitř buňky označují souřadnice x, y na kterých se buňka nachází. Výpočet souřadnic souseda se řídí dle rovnice ve funkci. int col = (c.x + i + rows) % rows; int row = (c.y + j + cols) % cols; Výpočet od zavolání funkce bude vypadat následovně. (**7** + (-1) + 10) % 10 = 6 ; (**9** + (-1) + 10) % 10 = 8 —> první souřadnice jsou 6,8. Následně (**7** + (-1) + 10) % 10 = 6; (**9** + (0) + 10) % 10 = 9 —> druhé souřadnice jsou 6,9. Výpočet pokračuje (**7** + (-1) + 10) % 10 = 6; (**9** + 1 + 10) % 10 = 0 —> další souřadnice jsou 6,0. A takto pokračuje výpočet dál. Jako poslední krok odečteme stav té buňky, která volala funkci.

Díky operátoru modulo jsme vytvořili nekonečné 2D pole.

8. Metoda init()

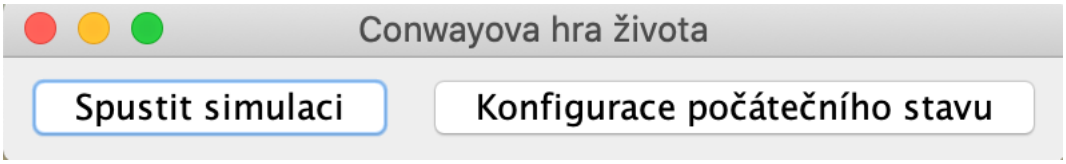
Metoda init() načte uživatelem nastavenou základní konfiguraci, nebo vygeneruje konfiguraci náhodnou, pokud uživatel žádnou nedodal.

```
public void init() {
    ButtonSettings e = new ButtonSettings(0,0);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if(MainMenu.nastaveno == false){
               int n = random.nextInt(2);
                grid[i][j] = new Cell(i, j, n);
            }
            else {
                //importneme 2D pole z nastavovače
                e = ButtonPanel.grid_arr[i][j];
                if (e.alive == true) {
                    grid[i][j] = new Cell(i, j, 1);
                } else if (e.alive == false) {
                    grid[i][j] = new Cell(i, j, 0);
                }
            }

        }
    }
    //nastavyme vsechny bunky v next na 0
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            next[i][j] = new Cell(i,j, 0);
        }
    }

    draw();
}
```

Pokud uživatel kliknul na tlačítko „Konfigurace počátečního stavu”, přepnul globální proměnnou *nastaveno* na true a spustí se větev *else {* funkce. Do proměnné *e* se uloží button na daných souřadnicích z pole *grid_arr[][]* třídy ButtonSettings. Pokud byl button zmáčknut (e.alive = true) přidáme do pole *grid[][]* novou živou buňku. Pokud button zmáčknut nebyl, přiřadíme do pole buňku mrtvou. Na konci funkce si připravíme pole *next* a vykreslíme na obrazovku pole *grid* metodou *draw()*.

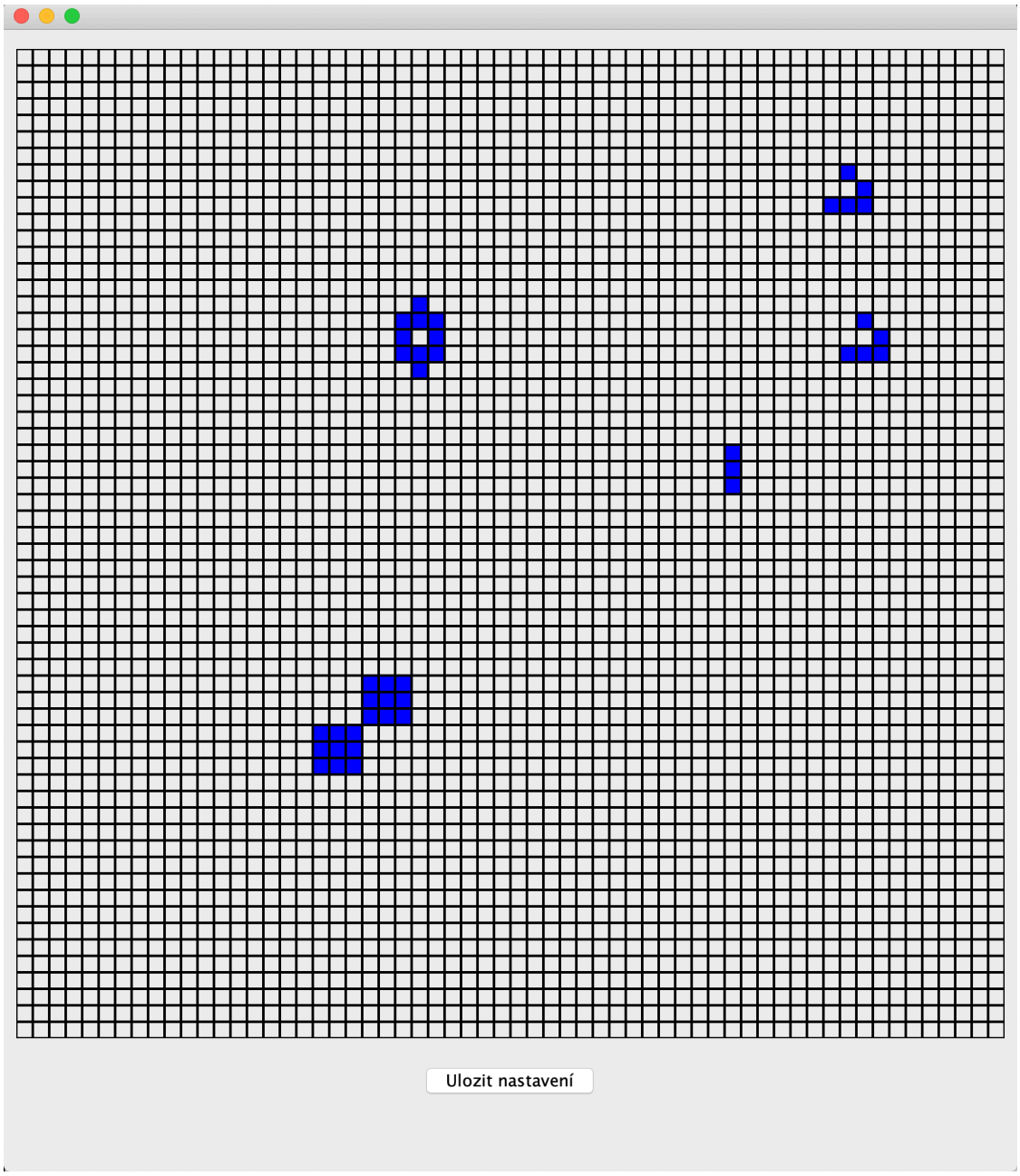
9. Ukázka programu

Na obrázku č. 3 je vidět hlavní menu aplikace.



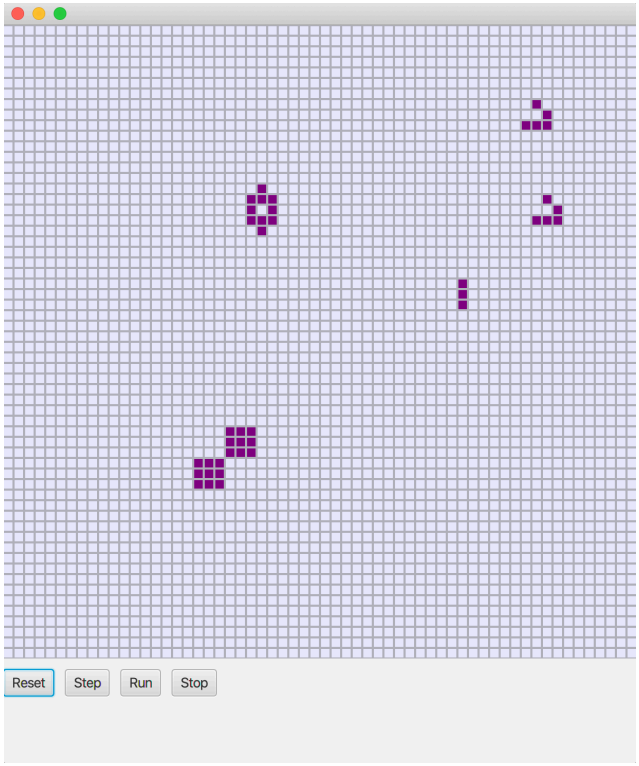
Obrázek č. 3 – menu aplikace

Na obrázku č. 4 je konfigurace počátečního stavu od uživatele. Velikost plochy je 60x60. Živé buňky jsou označeny modře.



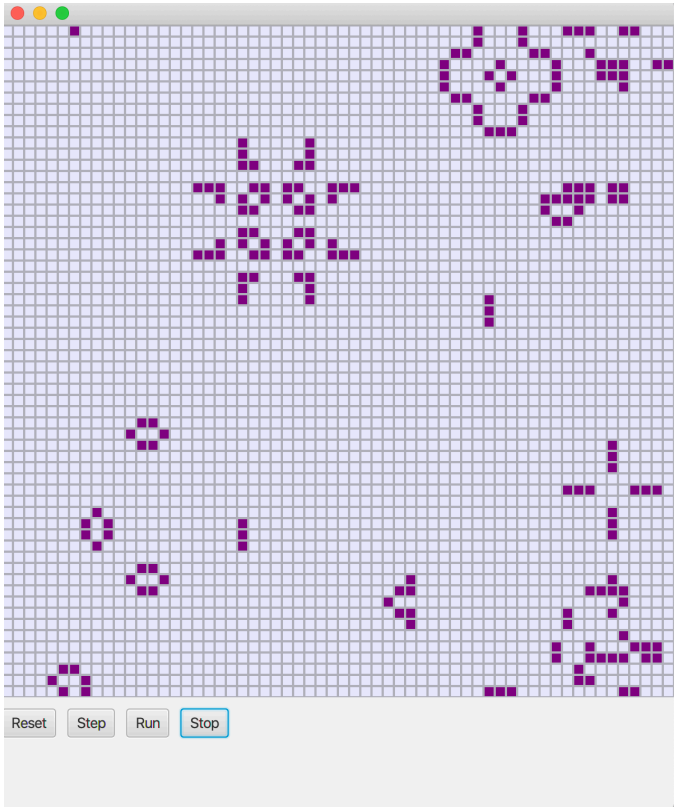
Obrázek č. 4 – konfigurace počátečního stavu

Na obrázku č. 5 je vidět začátek Hry.



Obrázek č. 5 – spuštění programu

Na obrázku č. 6 je několikátá generace po spuštění simulace.



Obrázek č. 6 – vývoj po spuštěni simulace

10. Závěr

Projekt splnil zadání a jsem s ním spokojen. Program je napsán tak, aby ostatní uživatelé mohli kód sami používat a rozšiřovat. Pro lepší využití programu by bylo vhodné doplnit funkci načítaní a ukládání počáteční konfigurace např. ze csv souboru. Uživatel by tak mohl jednoduše znovu používat a zkoumat zajímavé počáteční konfigurace. Další užitečnou funkcí by byla možnost upravovat rychlost animace. Zatím je možnost upravit rychlost animace jednoduše pomocí konstanty k kódu, ale pokud uživatel spouští program jako exe soubor, nemá možnost rychlost animace ovlivnit.

Program byl napsán s objektově s cílem a důrazem na rozšířitelnost kódu. V budoucnu můžeme stavět na tomto programu jako základu a např. rozšířit třídu Cell. Buňky v simulaci nemusí mít pouze stav mrtvá a živá, ale i zdravá, nakažená, nemocná, mrtvá atd. Pomocí Game Of LIfe lze např. simulovat vývoj virového onemocnění v obyvatelstvu apod.

Z práce jsem si odnesl zkušenosti s javaFX a schopnost psát čitelný kód.

11. Zdroje obrázků

File:Moore neighborhood with cardinal directions.svg – Wikimedia Commons. (2015, March 5). File:Moore Neighborhood With Cardinal directions.svg - Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Moore_neighborhood_with_cardinal_directions.svg