Dokumentace projektu IMS

Model celulárního automatu

Prosinec 2022

Augustin Machyňák
xmachy02

Obsah

1 Úvod **2**
1.1 Validita 2

2 Použité technologie **3**

3 Koncepce - elementární CA a jeho úprava **4**

4 Architektura **5**
4.1 Modely 5
4.2 Simulace a vizualizace 5
4.2.1 Použití programu 5

5 Experimentování **6**
5.1 Aplikování pseudo-náhodného pravidla 7
5.2 Aplikace 2 pravidel 8
5.3 Aplikace 3 a více pravidel a zajímavé útvary 9

6 Shrnutí a závěr **11**

1 Úvod

Tato práce se zabývá implementací a experimentováním s upraveným modelem obecného elementárního celulárního automatu (dále jen CA; IMS 208). Úprava modelu spočívá v tom, že místo aplikování jednoho pravidla pro všechny generace jsou aplikována různá pravidla, ať už náhodná nebo cyklicky se opakující.

Smyslem experimentování je prozkoumat, jak se bude CA chovat a jestli pro něj platí některé z předpokladů.

1.1 Validita

Validita implementace modelu elementárního CA byla ověřena porovnáním s již existujícími a obecně známými vizualizacemi ([3]).

2 Použité technologie

Pro vizualizaci CA bylo použito grafické API OpenGL, pro které byly využity knihovny Glad (načtení OpenGL funkcí - [1]) a glfw (vytvoření OpenGL kontextu a okna závislé na operačním systému - [2]). Dále pro zajištění přenositelnosti a jednoduchosti překladu byl použit CMake. K exportování konečného stavu CA do PNG obrázku byla využita stb knihovna `stb_image_write`.

3 Koncepce - elementární CA a jeho úprava

Elementární CA je jednodimenzionální CA, ve kterém je následující stav každé *buňky* vypočítán pomocí jeho současného stavu, stavu 2 *okolních buněk* a zvoleného *pravidla* (IMS 209). Buňky se můžou nacházet v 1 ze 2 stavů (0 a 1). Pravidlem rozumíme hodnotu v rozmezí 0 až 255, která představuje pro které kombinace buňky a jejího okolí bude výsledný stav 0 nebo 1.

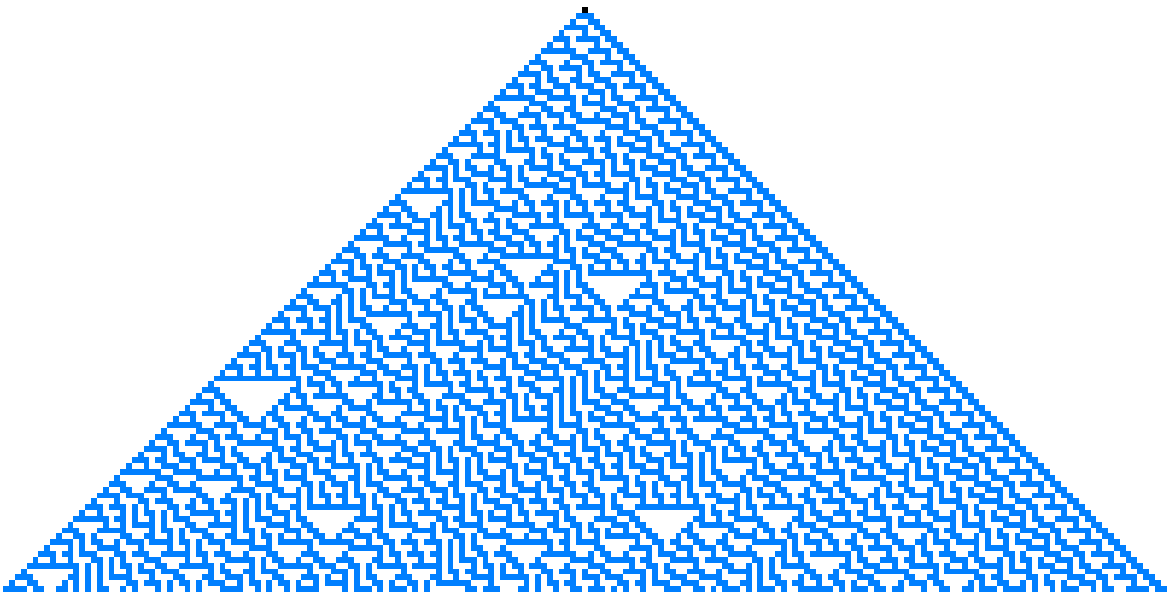
Výpočet stavů následující generace můžeme vypočítat následovně (C++ "pseudokód"):

code/eca.cpp

```
Cell[Width] NextGeneration(Cell[Width] cells, int rule) {
  Cell[Width] nextGeneration;
  for (int i = 0; i < Width; i++) {
    int state =
        (cell[i-1 % Width] == 1) << 0 // levá buňka
      | (cell[i] == 1)           << 1 // současná buňka
      | (cell[i+1 % Width] == 1) << 2 // pravá buňka
    nextGeneration[i] = ((1 << state) & rule) ? 1 : 0;
  }
  return nextGeneration;
}
```

Pro obvyklý elementární CA je použito pouze 1 pravidlo a to je aplikováno pro každou *generaci*. Tato práce zkoumá, co se stane při aplikování různých pravidel (například aplikování N střídajících se pravidel) pro každou generaci a v tom také spočívá úprava elementárního CA. Je možné tedy argumentovat tím, že se již nejedná o elementární CA, avšak jejich princip je podobný a pro jednoduchost pochopení je tento název použit.

Mimo jiné je důležitou částí CA jeho vizualizace. K tomuto je nejvhodnější a nejjednodušší použít 2D pole (matici), kde barva buněk představuje jejich stav, souřadnice X jednotlivé buňky a souřadnice Y generaci.



Obr. 1: Prvních 100 generací pravidla 30 (shora dolů - 1. generace má černou barvu)

4 Architektura

4.1 Modely

Implementace modelu elementárního CA je triviální a již byla zmíněna v předchozí části. Náročnější je vytvoření prostředí, ve kterém je možné efektivně experimentovat s různými parametry. Proto byly přidány některé vhodné metody:

- Zadávání pravidel jako seznam čísel přes CLI (*Command Line Interface*)
- Zadávání počátečního stavu jako seznam čísel přes CLI s možností zadání náhodného stavu aj.
- Způsob iterování pravidel ("RuleStrategy"):
  - Náhodný - pro každou generaci je vybráno náhodné pravidlo
  - Náhodný z datasetu - je vybráno náhodné pravidlo ze seznamu pravidel zadaných uživatelem
  - Cyklický - je vybráno následující pravidlo ze seznamu pravidel zadaných uživatelem, přičemž u posledního pravidla je proveden návrat k prvnímu pravidlu
  - Cyklický bez návratu - místo návratu k prvnímu pravidlu se opakovaně aplikuje poslední pravidlo

Kód je dostupný ve třídě CA::Elementary (`src/model/elementary.*`).

4.2 Simulace a vizualizace

Podstatnou částí je vizualizace CA. Tu zajišťuje velmi malý "framework", který byl vytvořen pro potřeby zobrazení CA ve 2D. Jedná se pouze o okno, které obsahuje plochu o velikosti zadané uživatelem. Jednotlivé pixely je možné na této ploše poté upravovat. Použití tohoto frameworku vypadá následovně:

1. Vytvoření nového okna o zadané velikosti (*Gui::PixelWindow*)
2. Nastavení časového kroku
3. Implementace rozhraní *Gui::IStepper*:
   (a) Inicializace - voláno pouze jednou na začátku
   (b) Kroková funkce - voláno každý časový krok
   (c) Podmínka ukončení - kontrolováno po každém časovém kroku
   (d) Reset - voláno po zmáčknutí tlačítka; sémantika může být teoreticky různá, ale očekává se návrat do počátečního stavu
4. Spuštění simulace s parametrem instance implementovaného rozhraní (v případě elementárního CA *ModelWrapper::ElementaryWrapper*)

Tento framework je možné jednoduše použít obecně pro vizualizaci jakéhokoliv 1D/2D CA (teoreticky i více, ale pro tyto potřeby nebyl vytvořen a bylo by to tedy komplikované).

Kód je dostupný ve třídě Gui::PixelWindow (`visualization/gui/gui.*`) a ModelWrapper::ElementaryWrapper (`visualization/models_wrap/elementary_ca_wrapper.*`).

4.2.1 Použití programu

Příklady použití (CLI):

```
$ ./CA_Visualization -h
```

Pravidla se perodicky střídají, pro 1. generaci zvolen 1 náhodný bod, zvolená pravidla 105 a 106, vypnutí opengl (obrázek představující výsledky generací je uložen do souboru `rule_105_106.png` po dokončení simulace):

```
$ ./CA_Visualization -out rule_105_106 -gloff -caargs dring rng1 105,106
```

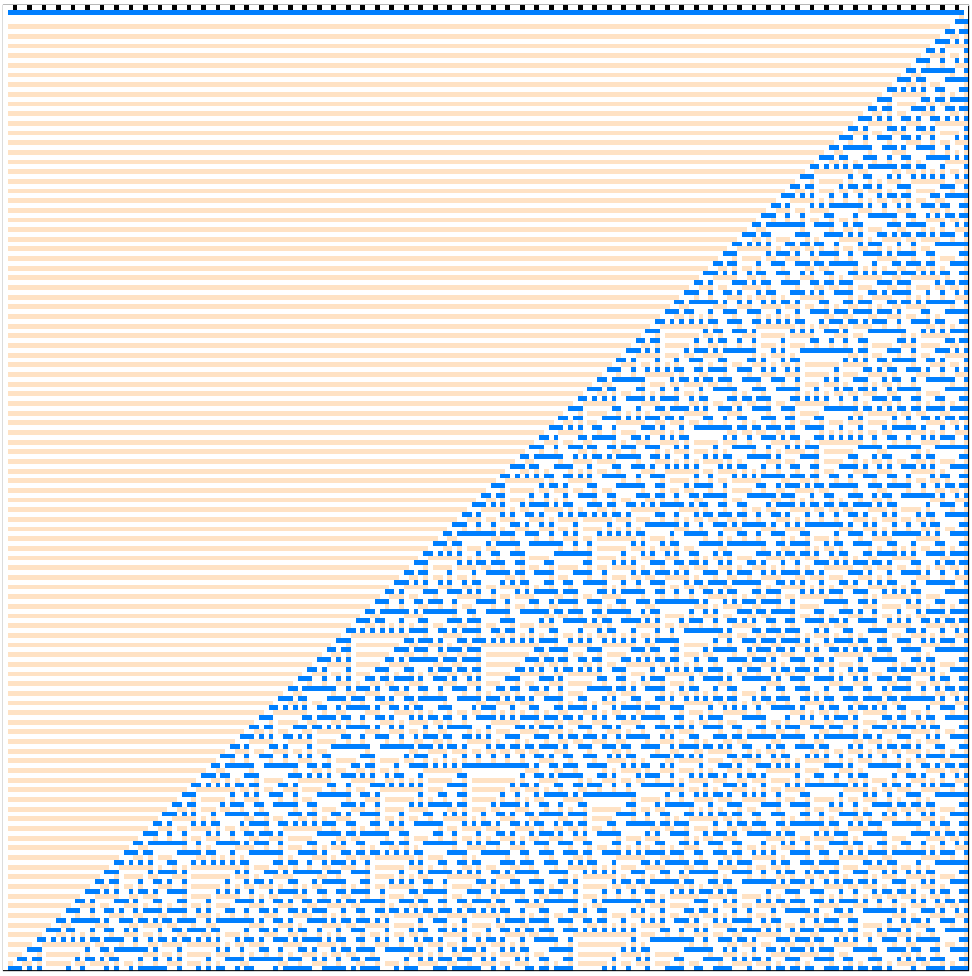
Pravidla se náhodně střídají, počáteční stav (0, 0, 1, 0, 0, 1, ...), zvolená pravidla 15, 22, 30, 60.

```
$ ./CA_Visualization -caargs drandom 0,0,1 15,22,30,60
```

5 Experimentování

Cílem experimentování je zjistit, jak se CA chová a jestli splňuje některé z předpokladů, pokud nějaké jsou.

Pro přehlednost jsou pravidla rozdělená barvy (každému pravidlu v rozmezí 0 až 255 je přiřazena jiná barva), aby bylo zjevné, kdy dojde k aplikaci jiného pravidla. Černá barva znázorňuje stav 1 buňky 1. generace, bílá barva značí stav 0 buňky, ostatní barvy značí stav 1 buňky.

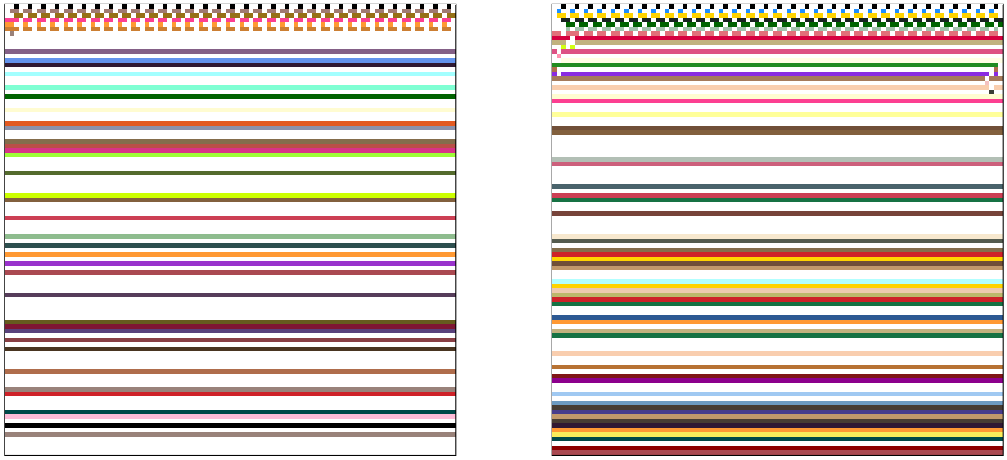


Obr. 2: 200 generací pravidla 30 a 45 (střídající se), počáteční stav (0, 0, 1, 0, 0, 1, ...)

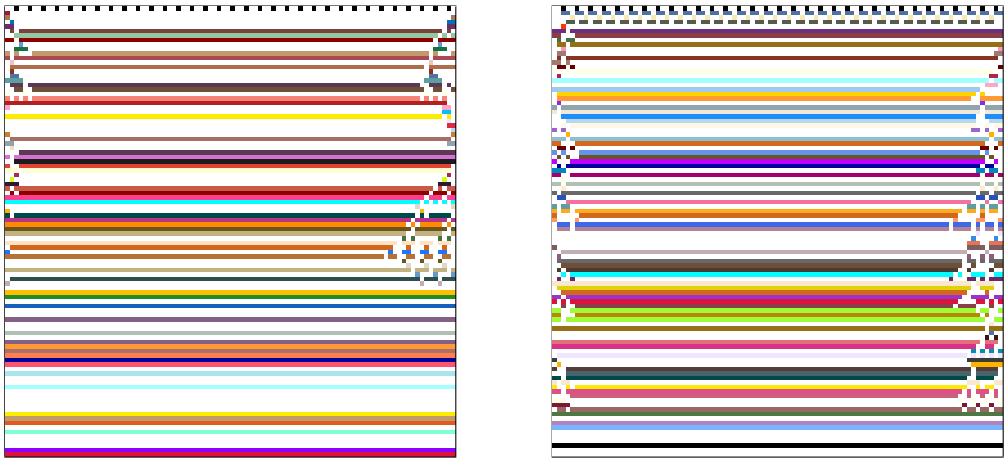
5.1 Aplikování pseudo-náhodného pravidla

Aplikováním pseudo-náhodného (dále jen náhodného) pravidla rozumíme to, že pro každou generaci bude aplikováno jiné náhodně vygenerované pravidlo. Předpokladem pro chování tohoto CA bylo, že bude vykazovat náhodné chování alespoň v některých případech. Výsledek experimentu však ukázal opak - ve většině případů byl výsledek téměř stejný a ten je, že po pár generacích se pouze střídají generace, kdy jsou všechny buňky ve stavu 0 s generací, kdy jsou všechny buňky ve stavu 1.

Aplikace náhodného pravidla, 100 generací, počáteční stav (0, 0, 1, 0, 0, 1, ...):



Obr. 3: Výsledný vzor pro většinu případů.



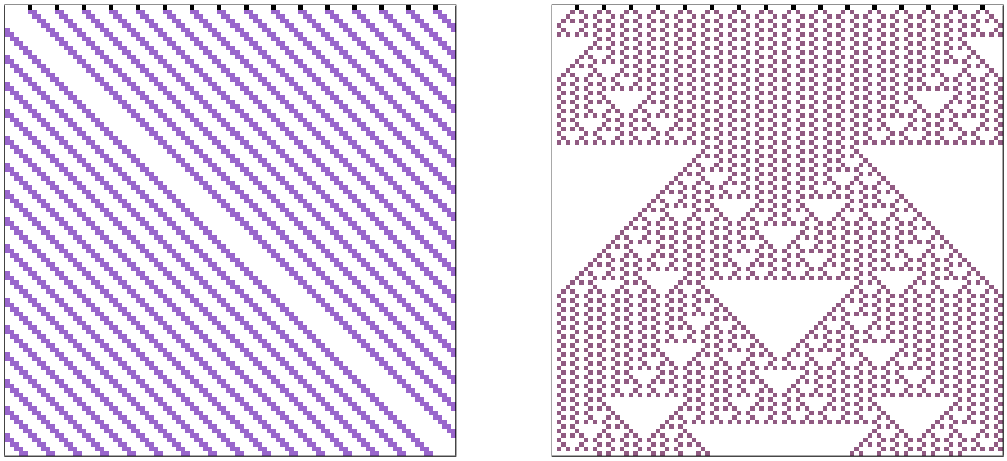
Obr. 4: V minimálním množství případů po několik generací vykazovaly CA chování, jehož výsledný vzor můžeme přirovnat k růstu kořenů. Nakonec ale vždy konvergují do stejného stavu.

Při zvolení náhodného pravidla je totiž velká šance, že buňky nepřežijí po dostatečné množství generací - v určitý moment dojde k tomu, že všechny buňky v dané generaci budou mít stejný stav a jediné pravidlo, které tento stav změní, je pravidlo, které je všechny nastaví do opačného stavu.

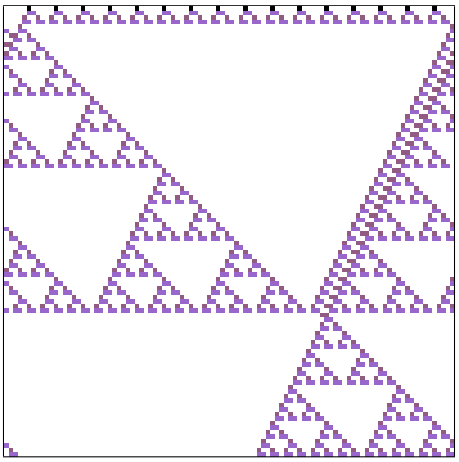
5.2 Aplikace 2 pravidel

Předpoklad pro chování tohoto CA není jednoduché určit. Bylo předpokládáno, že nejspíše dojde k tomu, že výsledný vzor bude připomínat kombinace jevů těchto vzorů - pokud jeden vzor vytváří diagonální čáry a druhý trojúhelníky, tak jejich kombinace bude vytvářet trojúhelníky s menším náklonem.

Tento předpoklad v některých případech platil:

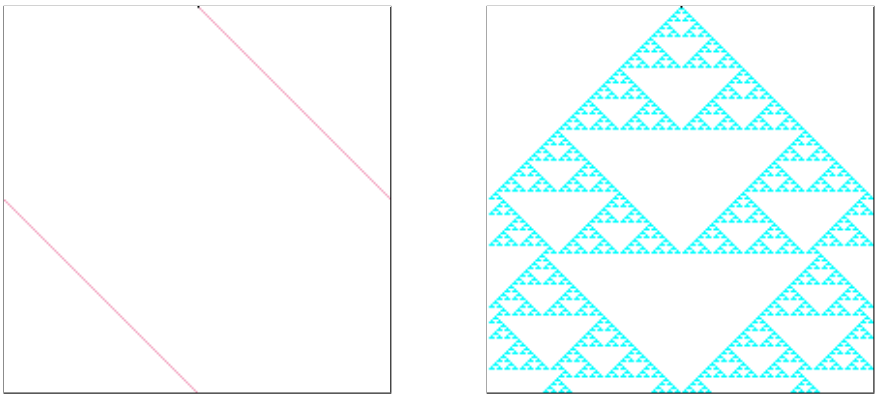


Obr. 5: Pravidla 14 a 18 s počátečním stavem (0, 0, 0, 0, 0, 1, ...).

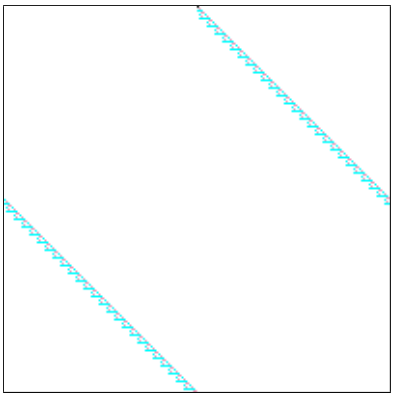


Obr. 6: Kombinace pravidel 14 a 18 se stejným počátečním stavem; pravidla se periodicky střídají

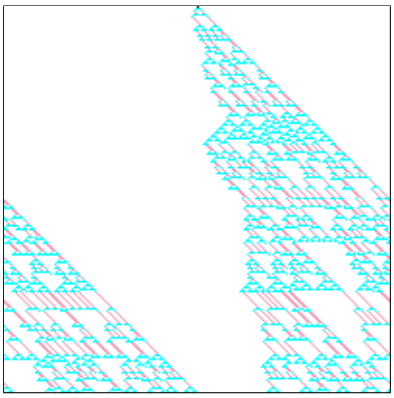
V jiných případech předpoklad neplatil:



Obr. 7: Pravidla 10 a 22



Obr. 8: Kombinace pravidel 10 a 22; pravidla se periodicky střídají

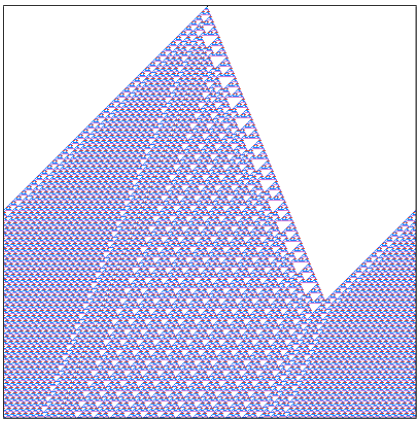


Obr. 9: Kombinace pravidel 10 a 22; pravidla se náhodně střídají

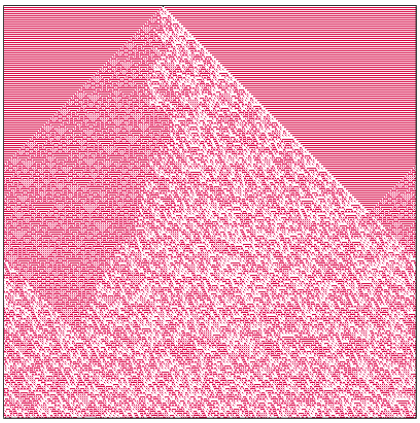
5.3 Aplikace 3 a více pravidel a zajímavé útvary

Pro aplikaci více pravidel nebyl již určen žádný předpoklad pro chování takového CA. Cílem tohoto experimentu je dosáhnout nalezení zajímavých útvarů, které je možné získat kombinací více pravidel.

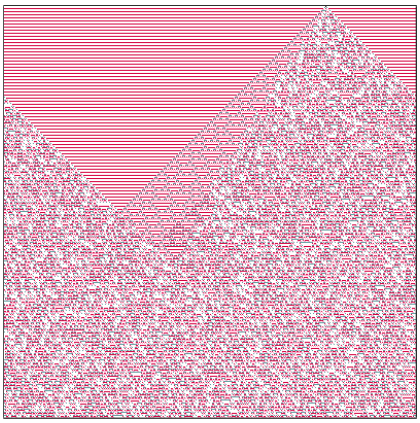
Na obrázcích níže je možné vidět některé zajímavé útvary nalezené kombinací pravidel. U popisu obrázků je napsané, které pravidla jsou aplikována. Všechny pravidla se periodicky střídají.



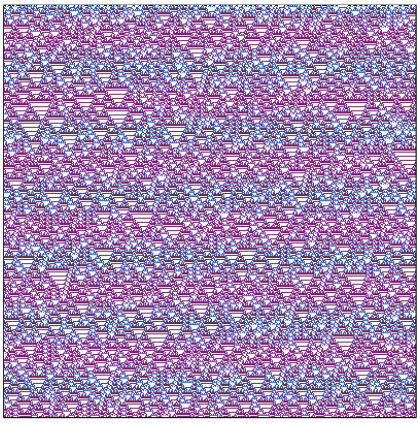
Obr. 10: (208, 30, 250, 60, 126)



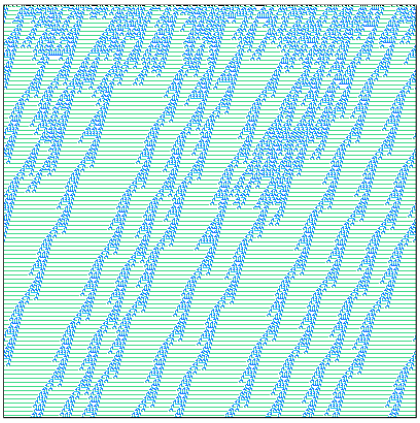
Obr. 11: (105, 106)



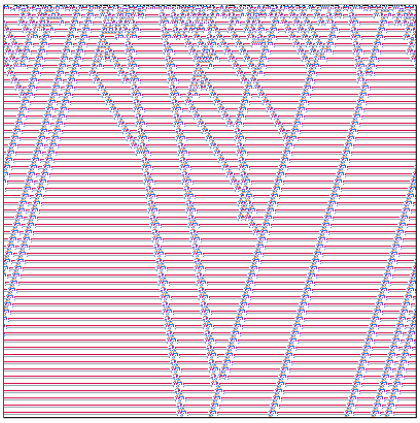
Obr. 12: (89, 105, 106)



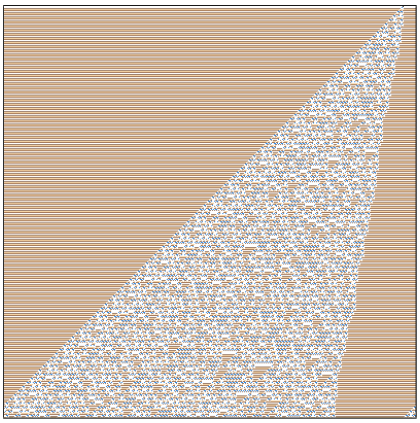
Obr. 13: (60, 89, 110, 178)



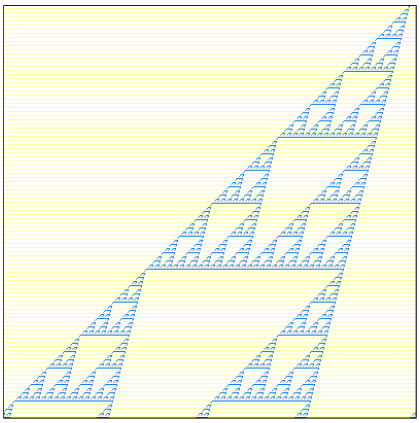
Obr. 14: (15, 22, 30, 60)



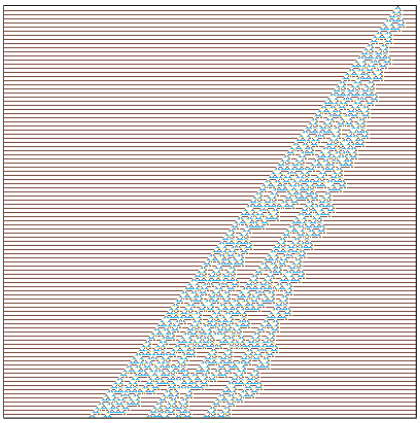
Obr. 15: (30, 105, 69, 91, 60, 232, 86)



Obr. 16: (64, 41, 195, 244)



Obr. 17: (16, 90, 30, 60)



Obr. 18: (18, 22, 26, 73)

6 Shrnutí a závěr

Z výsledků experimentů vyplývá, že žádný z prvotně očekávaných předpokladů nebyl splněn - aplikováním náhodného pravidla se CA nechová náhodně a aplikování 2 pravidel nutně neznamená, že výsledný vzor bude obsahovat jevy vyskytující se ve vzorech při aplikování těchto pravidel samostatně. Experimentováním s kombinacemi více pravidel byly nalezeny kombinace pravidel se zajímavými výslednými vzory, které není možné získat aplikováním pouze jednoho pravidla.

Mj. v rámci projektu vznikl nástroj pro vizualizaci libovolného elementárního CA a jejich kombinací.

References

[1] Glad - multi-language vulkan/gl/gles/egl/glx/wgl loader-generator based on the official specs. https://github.com/Dav1dde/glad.

[2] Glfw - open source, multi-platform library for opengl, opengl es and vulkan application development. https://github.com/glfw/glfw.

[3] Eric W. Weisstein. "elementary cellular automaton." from mathworld - a wolfram web resource. https://mathworld.wolfram.com/ElementaryCellularAutomaton.html.