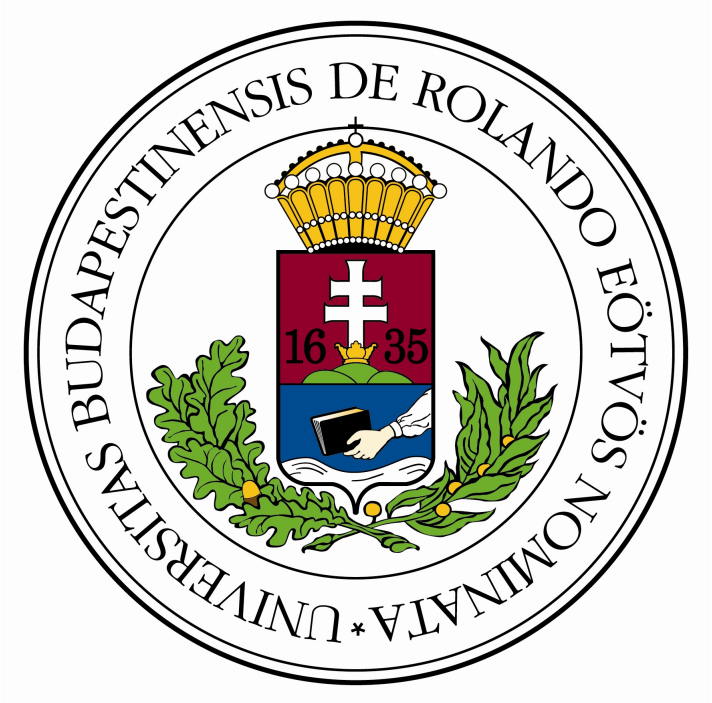
Számítógépes szimulációk labor - Sejtautomaták

Zsigmond István, IIQCC8

Május 06, 2019



1. Conway-féle élet játék

A sejtautomaták olyan matematikai/szimulációs problémák, melyben egymás környezetében szabályosan, rácsszerűen elhelyezett, élő vagy halott sejtek születési-halálozási folyamatokon mennek át diszkrét időlépésenként valamilyen szabálynak megfelelően. A Conway-féle élet játék egy ilyen sejtautomata, a sejtek születési-halálozási szabálya pedig a következők:

1.1. Szabályok

- Ha **n** db élő szomszéd van, akkor nem változik a sejt állapota.
- Ha $\mathbf{n}+\mathbf{1}$ db élő szomszéd van, akkor a sejt élő lesz, függetlenül a jelenlegi állapotától.
- Minden egyéb esetben elpusztul.

A cél ezen problémának szimulációja és **n** érték hatásának vizsgálata. További része a feladatnak olyan algoritmus írása, melyben a felhasználó megválaszthatja a kezdeti elrendezést, illetve a peremfeltételeket.

- Nyílt peremfeltétel: a peremeken halott sejtek vannak.
- Periódikus peremfeltétel: a peremeket az ellenkező oldal elemei képezik.
- Élő peremfeltétel: a peremeken mindig élő sejtek vannak.
- Véletlenszerű peremfeltétel: a peremeken véletlenszerűen sorsolt állandó állapotú sejtek vannak.

1.2. Eredmények

A szimulációs kódomat úgy írtam meg, hogy a megfelelő paraméterek megadásával egy mátrixot generáljon le, melyben 1-es és 0-k vannak. 1-esek jelentik az élő sejteket, 0-k a halottakat, majd minden lépésnél fájlba kiírja az eredmény mátrixokat. A lefordított program futtatása és működése:

- A lefordított programot úgy kell futtatni, hogy egyúttal megadjuk a kívánt mátrix dimenzióit és a kimeneti fájl nevét.
- A program megkérdezi, hogy melyik feladatrészt akarjuk szimulálni. Miután kiválasztottuk a 'Conway-féle élet játék' lehetőséget, a program bekéri az **n** értékét és a kívánt léptetések számát (ennyi fájl fog születni, +1 a kiinduló elrendezés).
- Ezután megkell adni, milyen peremfeltétellel szeretnénk indítani a szimulációt (1 - nyílt, 2 - periódikus, 3 - élő, 4 - véletlenszerű)
- Ezután lehetőségünk van kiválasztani, hogy manuálisan akarjuk megadni a kiinduló elrendezést (ezzel adva szabadságot tetszőleges kezdeti feltétel/elrendezés megadására), vagy indítsunk egy random generált elrendezésből.

Az így kapott eredményeket Python-ban értékeltem ki. A PIL nevezető könyvtár segítségével képpé alakítottam az eredménymátrixokat, imageio könyvtár segítségével pedig .gif formátumba rendeztem a kapott képeket. Minden peremfeltételre megnéztem 2-2 esetet ($\mathbf{n}=\mathbf{1},\mathbf{4}$),. Sajnos latex-ben nem vagy csak nagyon nehezen lehet megoldani gif-ek megjelenítését, ezért ezeknek eredményeit feltettem a weboldalamra.

A .gif-ek vagy az eredménymátrix-ok vizsgálata alapján vannak jellegzetességek a különböző peremfeltételek között.

- Nyílt peremfeltétel esetén a sejtek állapota csak a kezdeti elrendezéstől függ, a széleken kívüli sejtek halottak, tehát onnan behatás nem érkezik.

- Periódikus peremfeltétel esetén a sejtek állapota a széleken egymásba visszaforog, tehát azokon a helyeken élő vagy halott sejtek vannak a kezdeti elrendezéstől függően. Ennek köszönhetően a széleken a rendszernek nagy esélye van lépésről-lépésre kihalnia és megszületnie.

- Élő határfeltétel esetén a kezdeti elrendezéstől függetlenül a peremeken állandóan élő sejtek vannak jelenek, mely a széleken lévő sejtek folyamatos kihalását eredményezi. Ez természetesen függ **n**-től, nagyobb ($\mathbf{n} \geq \mathbf{2}$) érték esetén a széleken előfordulhat, hogy még megjelenik élő sejt, $\mathbf{n} = \mathbf{1}$ esetén azonban erre már nincs esély.

- Véletlenszerű peremfeltétel esetén pedig az eredmények jó megfigyelése alapján tudjuk csak megmondani, hogy az véletlen peremfeltétel. Mivel a peremet sose látjuk, így nehéz lehet nagy rácsoknál megállapítani ezt, de kis rácsoknál könnyen észrevehető, ahogy 1-2 sejt "szabélyellenesen" él vagy hal adott pillanatban. Ez a véletlenszerű peremfeltétel következménye, ezek a sejtek a rejtett, peremen lévő élő/halott sejtek miatt élnek vagy halnak adott pillanatban.

A rácson befele haladva azonban nincs hatása a kezdeti peremfeltételnek, csupán **n** értékének: egyre növekvő **n** esetén az élő sejtek száma nagyobb mértékben csökken lépésenként.

Ezen eredmények olyan esetben igazak, ahol az erendezés véletlenszerű vagy jól megválasztott. Könnyedén megadhatunk ugyanis olyan kezdeti feltételeket/elrendezéseket, amikoris pl. a rendszer minden sejtje a következő lépésben halott lesz adott **n**-nél, függetlenül a peremfeltételektől.

2. 2D és 3D homokdombok szomulációja

2.1. Szabályok

Ebben a szimulációban homokdombok kialakulását kellett szimulálni. Ezt 2D-ben és 3D-ben is megtettem, a rendszer időfejlődése pedig a következő szabályt követi:

- Kiválasztunk egy **n** értéket, mely a stabilitási paraméter lesz. Ez azt jelenti, hogy egymásra pakolva a homokszemeket az addig nem dől le, amíg túl nem lép a relatív magassága ezen a számon.

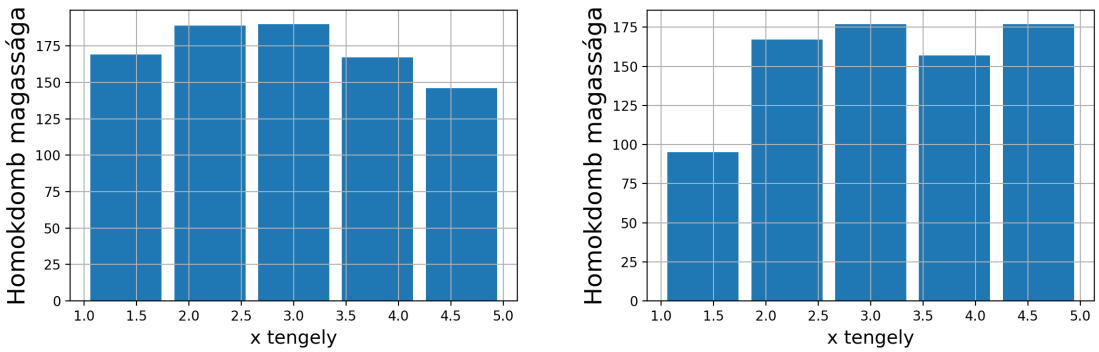
- Véletlenszerűen kiválasztunk egy pontot, ahova ledobunk egy homokszemet. Miután ledobtunk egy homokszemet, a program megvizsgálja a szomszédsági állapotokat. Ha a kiválasztott pont homokszemek és bármelyik szomszéd által képezett magasságának különbsége nagyobb, mint $\mathbf{n} + \mathbf{1}$, a pont instabil és ledől. A ledőlést úgy veszi figyelembe a program, hogy 1 homokszem leesésével a kiválasztott pont már stabil, tehát csak 1-et vesz el és ad át egy véletlenszerűen kiválasztott másik pontnak. Ez habár nem pontos, de nagy számoknál az ebből fakadó hiba eltűnik és adhat közel valós képet.

- A program véletlenszerűen sorsol egy indexet a kiválasztott pont körül egészen addig, amíg olyan szomszédot nem talál, mely kisebb a kiválasztott pontnál. A homokszem ebbe a pontba fog gurulni.

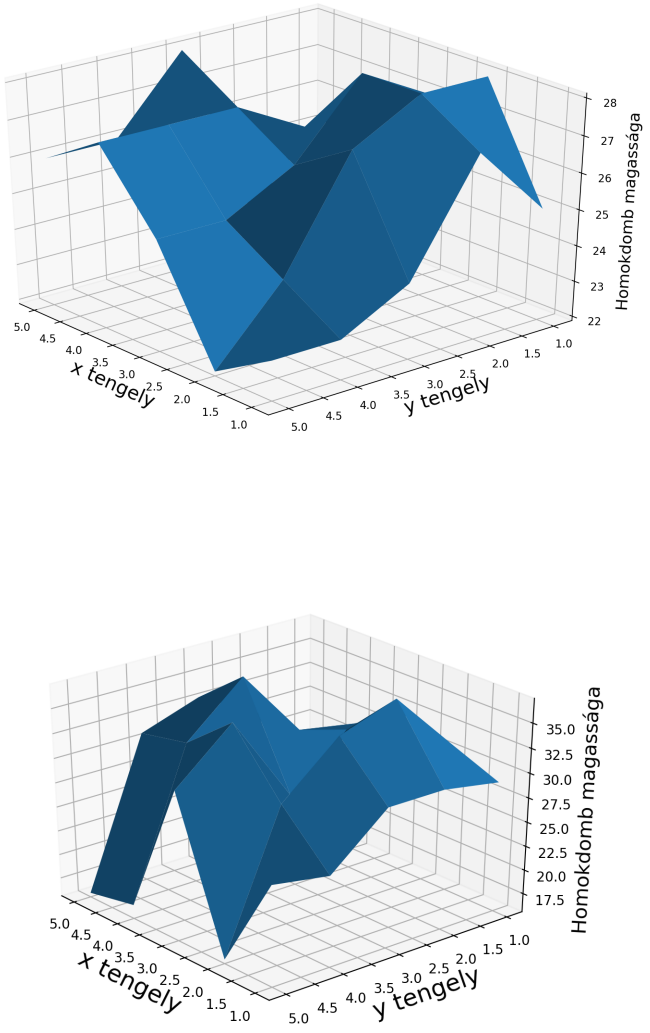
- Mindezen annyiszor megy végig a program, amennyi homokszemet ledobtunk.

2.2. Eredmények

A szimulácikat több kezdeti feltételből is meglehet közelíteni, az eredmény azonban hasonló minden esetben. Én minden esetben üres térből indítottam. Ezek eredményei láthatóak az 1. és a 2. ábrákon.



1. ábra. 2D-s homokdomb szimulációja. Mind a két esetben a stabilitás $\mathbf{n} = \mathbf{20}$, ledobott részecskék száma $\mathbf{N} = \mathbf{1000}$. A 2D-s szimuláción nem sok minden látszik, de az érezhető, hogy minél nagyobb a stabilitás értéke, annál nagyobb különbségek lehetnek a szintek között.



2. ábra. 3D-s homokdomb szimulációja. A bal oldali ábrán (a) a stabilitás $\mathbf{n} = \mathbf{4}$, a jobb oldali ábrán (b) a stabilitás $\mathbf{n} = \mathbf{10}$. A ledobott homokszemek száma mindkét esetben $\mathbf{N} = \mathbf{1000}$. A 3D-s szimuláció már sokkal szemléletesebb, és egész szépen visszaadja, hogyan nézne ki egy homokdomb random leszórt szemekből képezve.

3. Függelék

A szimulációkhoz írt kód

```
#include <fstream>
#include <algorithm>
#include <iostream>
#include <cmath>
#include <string>
```

```
#include <sstream>
#include <cstdlib>
#include <time.h>

using namespace std;

int main(int argc, char **argv) {

   if (argc<4) {
       cout << "Usage: " << argv[0] << " <number of rowes> <number of columns> <output
           file name>" << endl;
       return -1;
      }

   string matrix_file;
   unsigned int N = atoi(argv[1]);
   unsigned int M = atoi(argv[2]);
   matrix_file = argv[3];
   unsigned int n;
   signed int nn;
   unsigned int t, t_max, var, var2, var3;

   cout << "Enter 1 to simulate Conway's game of life or 2 to simulate a 2D sand mass, 3
       to simulate a 3D sand mass: ";
   cin >> var3;


   if (var3 == 1) {
       cout<< "Enter the value of n, and the number of iterations to be done t: ";
       cin >> n >> t_max;
   }

   if (var3 == 2 || var3 == 3) {
       cout << "Enter the value of n, and the number of dropped sand: ";
       cin >> nn >> t_max;
   }

   if (var3 == 1){
   cout << "Enter 1 for opened bounds, 2 for periodic bounds, 3 for more options: ";
   cin >> var;
   }

   if (var == 3 && var3 == 1) {
       cout << "Enter 3 for living bounds, 4 for random bounds: ";
       cin >> var;
   }

   int matrix[M+1][N+1];
   int temp_matrix[M+1][N+1];

   if (var3 == 1 && var >= 1) {
   cout << "Enter 1 to fill the starting sctructure manually or 2 to generate random
       values: ";
   cin >> var2;
   }

   if (var2 == 1 && var3 == 1) {
   cout << "Enter elements of starting structure:" << endl;
```

```
for (unsigned int j = 1; j < (M+1); j++) {
    for (unsigned int i = 1; i < (N+1); i++) {
        cout << "Enter element start" << j << i << ": ";
        cin >> matrix[j][i];
        }
    }
  }

if (var2 == 2) {
    srand(time(NULL));
    for (unsigned int j = 1; j < (N+1); j++) {
        for (unsigned int i = 1; i < (N+1); i++) {
            matrix[j][i] = rand() % 2;
        }
    }
}

if (var3 == 2 || var3 == 3) {
    for (unsigned int j = 0; j < (M+2); j++) {
        for (unsigned int i = 0; i < (N+2); i++) {
            matrix[j][i] = 0;
        }
    }
}

clock_t tStart = clock();

ostringstream m_file;

if (var3 == 2 || var3 == 3) {
for (unsigned int m = 0; m < (M+2); m++) {
    for (unsigned int n = 0; n < (N+2); n++) {
        temp_matrix[m][n] = matrix[m][n];
        }
    }
m_file << matrix_file << ".data";
ofstream start(m_file.str().c_str());
for (unsigned int m = 1; m < (M+1); m++) {
         for (unsigned int n = 1; n < (N+1); n++) {
             start << temp_matrix[m][n] << "\t";
             matrix[m][n] = temp_matrix[m][n];
             }
             start << endl;
          }
}

if (var3 == 1) {
    for (unsigned int m = 1; m < (M+1); m++) {
        for (unsigned int n = 1; n < (N+1); n++) {
            temp_matrix[m][n] = matrix[m][n];
        }
    }

m_file << matrix_file << ".data";
ofstream start(m_file.str().c_str());
for (unsigned int m = 1; m < (M+1); m++) {
         for (unsigned int n = 1; n < (N+1); n++) {
             start << temp_matrix[m][n] << "\t";
```

```
            matrix[m][n] = temp_matrix[m][n];
            }
            start << endl;
        }
 }

unsigned int neighbours;
//periodic bounds
if (var == 2 && var3 == 1) {
while (t < t_max) {
    ostringstream file;
    file << matrix_file << t << ".data";
    ofstream result(file.str().c_str());
    for (unsigned int k=1; k<(M+1); k++) {
        for (unsigned int j=1; j<(N+1); j++) {

            if (k == 1 && j == 1) {
                neighbours = matrix[k][j+1]+matrix[k+1][j+1]+matrix[k+1][j] +
                    matrix[M][2]+matrix[M][1]+matrix[M][N]+matrix[1][N]+matrix[2][N];
            }

            else if (k == M && j == 1) {
                neighbours = matrix[k-1][j]+matrix[k-1][j+1]+matrix[k][j+1] +
                    matrix[M-1][N]+matrix[M][N]+matrix[1][N]+matrix[1][1]+matrix[1][2];
            }

            else if (k == 1 && j == N) {
                neighbours = matrix[k+1][j]+matrix[k+1][j-1]+matrix[k][j-1] +
                    matrix[M][N-1]+matrix[M][N]+matrix[M][1]+matrix[1][1]+matrix[2][1];
            }

            else if (k == M && j == N) {
                neighbours = matrix[k-1][j]+matrix[k-1][j-1]+matrix[k][j-1] +
                    matrix[M-1][1]+matrix[M][1]+matrix[1][1]+matrix[1][N]+matrix[1][N-1];
            }

            else if (j == 1 && (k != 1 || k != M)) {
                neighbours =
                    matrix[k-1][j]+matrix[k-1][j+1]+matrix[k][j+1]+matrix[k+1][j+1]+matrix[k+1][j]
                    + matrix[k-1][N]+matrix[k][N]+matrix[k+1][N];
            }

            else if (k == 1 && (j != 1 || j != N)) {
                neighbours =
                    matrix[k][j+1]+matrix[k+1][j+1]+matrix[k+1][j]+matrix[k+1][j-1]+matrix[k][j-1]
                    + matrix[M][j-1]+matrix[M][j]+matrix[M][j+1];
            }

            else if (j == N && (k != 1 || k != M)) {
                neighbours =
                    matrix[k-1][j]+matrix[k-1][j-1]+matrix[k][j-1]+matrix[k+1][j-1]+matrix[k+1][j]
                    + matrix[k-1][1]+matrix[k][1]+matrix[k+1][1];
            }

            else if (k == M && (j != 1 || j != N)) {
                neighbours =
                    matrix[k][j+1]+matrix[k-1][j+1]+matrix[k-1][j]+matrix[k-1][j-1]+matrix[k][j-1]
                    + matrix[1][j-1]+matrix[1][j]+matrix[1][j+1];
```

```
            }
            else {
                neighbours =
                    matrix[k-1][j-1]+matrix[k][j-1]+matrix[k+1][j-1]+matrix[k-1][j]+matrix[k+1][j]
                +matrix[k-1][j+1]+matrix[k][j+1]+matrix[k+1][j+1];
            }

               if(neighbours == (n+1) && matrix[k][j] == 0)
                  temp_matrix[k][j] = 1;
               if((neighbours == (n+1) || neighbours == n) && matrix[k][j] == 1)
                  temp_matrix[k][j] = matrix[k][j];
               if(neighbours > (n+1) || neighbours < n)
                  temp_matrix[k][j] = 0;
             }
          }
          for (unsigned int m = 1; m < (M+1); m++) {
             for (unsigned int n = 1; n < (N+1); n++) {
                result << temp_matrix[m][n] << "\t";
                matrix[m][n] = temp_matrix[m][n];
                }
                result << endl;
             }
          t++;
       }
}
//random bounds
if (var == 4) {
    srand(time(NULL));
    for (unsigned int k = 0; k < (M+2); k++) {
        matrix[k][0] == rand() % 2;
        matrix[k][M+1] == rand() % 2;
    }
    for (unsigned int j = 0; j < (N+2); j++) {
        matrix[0][j] == rand() % 2;
        matrix[N+1][j] == rand() %2;
    }
while (t < t_max) {
    ostringstream file;
    file << matrix_file << t << ".data";
    ofstream result(file.str().c_str());
    for (unsigned int k = 1; k < (M+1); k++) {
        for (unsigned int j = 1; j < (N+1); j++) {
              neighbours =
                  matrix[k-1][j-1]+matrix[k][j-1]+matrix[k+1][j-1]+matrix[k-1][j]+matrix[k+1][j]
              +matrix[k-1][j+1]+matrix[k][j+1]+matrix[k+1][j+1];

              if(neighbours == (n+1) && matrix[k][j] == 0)
                 temp_matrix[k][j] = 1;
              if((neighbours == (n+1) || neighbours == n) && matrix[k][j] == 1)
                 temp_matrix[k][j] = matrix[k][j];
              if(neighbours > (n+1) || neighbours < n)
                 temp_matrix[k][j] = 0;
            }
         }
        for (unsigned int m = 1; m < (M+1); m++) {
           for (unsigned int n = 1; n < (N+1); n++) {
              result << temp_matrix[m][n] << "\t";
              matrix[m][n] = temp_matrix[m][n];
```

```
            }
            result << endl;
          }
       t++;
     }
}
//living bounds
if (var == 3) {
    for (unsigned int k = 0; k < (M+2); k++) {
        matrix[k][0] = 1;
        matrix[k][M+1] = 1;
    }
    for (unsigned int j = 0; j < (N+2); j++) {
        matrix[0][j] = 1;
        matrix[N+1][j] = 1;
    }
while (t < t_max) {
    ostringstream file;
    file << matrix_file << t << ".data";
    ofstream result(file.str().c_str());
    for (unsigned int k = 1; k < (M+1); k++) {
        for (unsigned int j = 1; j < (N+1); j++) {
              neighbours =
                  matrix[k-1][j-1]+matrix[k][j-1]+matrix[k+1][j-1]+matrix[k-1][j]+matrix[k+1][j]
              +matrix[k-1][j+1]+matrix[k][j+1]+matrix[k+1][j+1];

              if(neighbours == (n+1) && matrix[k][j] == 0)
                 temp_matrix[k][j] = 1;
              if((neighbours == (n+1) || neighbours == n) && matrix[k][j] == 1)
                 temp_matrix[k][j] = matrix[k][j];
              if(neighbours > (n+1) || neighbours < n)
                 temp_matrix[k][j] = 0;
           }
        }
        for (unsigned int m = 1; m < (M+1); m++) {
          for (unsigned int n = 1; n < (N+1); n++) {
              result << temp_matrix[m][n] << "\t";
              matrix[m][n] = temp_matrix[m][n];
              }
              result << endl;
           }
        t++;
     }
}

//open bounds
if (var == 1 && var3 == 1) {
    for (unsigned int k = 0; k < (M+2); k++) {
        matrix[k][0] = 0;
        matrix[k][M+1] = 0;
    }
    for (unsigned int j = 0; j < (N+2); j++) {
        matrix[0][j] = 0;
        matrix[N+1][j] = 0;
    }
while (t < t_max) {
    ostringstream file;
    file << matrix_file << t << ".data";
```

```
    ofstream result(file.str().c_str());
    for (unsigned int k = 1; k < (M+1); k++) {
        for (unsigned int j = 1; j < (N+1); j++) {
              neighbours =
                  matrix[k-1][j-1]+matrix[k][j-1]+matrix[k+1][j-1]+matrix[k-1][j]+matrix[k+1][j]
              +matrix[k-1][j+1]+matrix[k][j+1]+matrix[k+1][j+1];

              if(neighbours == (n+1) && matrix[k][j] == 0)
                 temp_matrix[k][j] = 1;
              if((neighbours == (n+1) || neighbours == n) && matrix[k][j] == 1)
                 temp_matrix[k][j] = matrix[k][j];
              if(neighbours > (n+1) || neighbours < n)
                 temp_matrix[k][j] = 0;
            }
         }
        for (unsigned int m = 1; m < (M+1); m++) {
           for (unsigned int n = 1; n < (N+1); n++) {
              result << temp_matrix[m][n] << "\t";
              matrix[m][n] = temp_matrix[m][n];
              }
              result << endl;
           }
        t++;
     }
}
//3D sand mass
if (var3 == 3) {
    while(t < t_max) {
        unsigned int rv1 = rand() % N + 1;
        unsigned int rv2 = rand() % M + 1;

        temp_matrix[rv1][rv2] += 1;
        if (((temp_matrix[rv1][rv2] - temp_matrix[rv1-1][rv2-1]) >= (nn + 1)) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1-1][rv2]) >= (nn + 1)) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1-1][rv2+1]) >= (nn + 1)) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1][rv2-1]) >= (nn + 1)) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1][rv2+1]) >= (nn + 1)) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1+1][rv2-1]) >= (nn + 1)) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1+1][rv2]) >= (nn + 1)) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1+1][rv2+1]) >= (nn + 1))) {
           while (matrix[rv1][rv2] != temp_matrix[rv1][rv2]) {
               srand(time(NULL));
               signed int rv3 = rand() % 3 - 1;
               signed int rv4 = rand() % 3 - 1;

               if (temp_matrix[rv1-rv3][rv2-rv4] < temp_matrix[rv1][rv2]) {
                   temp_matrix[rv1][rv2] -= 1;
                   if (rv3 == 0 && rv4 == 0) {
                       temp_matrix[rv1-1][rv2-1] += 1;
                   }
                   else {
                       temp_matrix[rv1-rv3][rv2-rv4] += 1;
                   }

                   matrix[rv1][rv2] = temp_matrix[rv1][rv2];
                   matrix[rv1-rv3][rv2-rv4] = temp_matrix[rv1-rv3][rv2-rv4];
               }
            }
```

```
        }

        matrix[rv1][rv2] = temp_matrix[rv1][rv2];
    t++;
    }
 ofstream start(m_file.str().c_str());
 for (unsigned int m=1; m<(M+1); m++) {
          for (unsigned int n=1; n<(N+1); n++) {
              start << temp_matrix[m][n] << "\t";
              matrix[m][n]=temp_matrix[m][n];
              }
              start << endl;
           }
}
//2D sand mass
if (var3 == 2) {
    while(t < t_max) {
        unsigned int rv1 = rand() % (N-1) + 1;
        unsigned int rv2 = 1;

        temp_matrix[rv1][rv2] += 1;
        if (((temp_matrix[rv1][rv2] - temp_matrix[rv1-1][rv2]) >= nn + 1) ||
            ((temp_matrix[rv1][rv2] - temp_matrix[rv1+1][rv2]) >= (nn + 1))) {
            while (matrix[rv1][rv2] != temp_matrix[rv1][rv2]) {
                srand(time(NULL));
                signed int rv3 = rand() % 3 - 1;

                if (temp_matrix[rv1-rv3][rv2] < temp_matrix[rv1][rv2]) {
                    temp_matrix[rv1][rv2] -= 1;
                    temp_matrix[rv1-rv3][rv2] += 1;

                    matrix[rv1][rv2] = temp_matrix[rv1][rv2];
                    matrix[rv1-rv3][rv2] = temp_matrix[rv1-rv3][rv2];
                }
             }
        }
        matrix[rv1][rv2] = temp_matrix[rv1][rv2];
    t++;
    }

 ofstream start(m_file.str().c_str());
 for (unsigned int m=1; m<(M+1); m++) {
          for (unsigned int n=1; n<(N+1); n++) {
              start << temp_matrix[m][n] << "\t";
              matrix[m][n]=temp_matrix[m][n];
              }
              start << endl;
           }
}



cout << "Time elapsed: " << (double)(clock() - tStart)/CLOCKS_PER_SEC << endl;

return 0;
}
```