Monte Carlo Simulations of Tetromino Fluids

Verfasser:	Philipp Deutsch
Matrikelnummer:	0348326
Studienkennzahl:	033 261
Studienrichtung:	Bachelorstudium Technische Physik
Betreuer:	Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gerhard Kahl

Wien, im Februar 2011

Contents

1 Tetromino Fluids

In the simulations detailed here, the work of Brian C. Barns, Daniel W. Siderius and Lev D. Gelb as described in their paper *Structure, Thermodynamics, and Solubility in Tetromino Fluids* [2] is reconstructed. The goal is to simulate the self-assembly properties of 2-dimensional tetrominoes in a grand-canonical ensemble depending on their chemical potential.

Tetrominoes are orthogonally connected objects that occupy four lattice sites in $\mathbb{Z}^2$. The seven possible tetrominoes are shown in figure 1.

Figure 1: The 7 possible tetrominoes, from left to right: T, O, L, J, S, Z and I

Here we study how a "fluid" made up of these "particles" behaves using grand-canonical Monte Carlo simulations. All simulations were performed using a fixed 32x32 lattice, i.e. the volume is kept constant. As a further simplification we will not consider effects due to temperature, so that the chemical potential will be the only parameter affecting the distribution of the tetrominoes. The simulation allows for four types of moves:

1. Insertion of a particle

   An unoccupied lattice point and an orientation is chosen at random, and, if the tetromino can fit on this spot, it is inserted with probability

   $$\mathbb{P}_{\text{Ins}} = \exp(\mu) \cdot \frac{V}{N+1},$$

   where V is the size of the lattice and N is the number of particles already in the volume. A detailed discussion of $\mathbb{P}_{\text{Ins}}$ and the deletion probability $\mathbb{P}_{\text{Del}}$ defined below can be found in [6]. The chemical potential, denominated by μ, is an external parameter and ranges in the simulations from -5 to 5 using intervals of 0.2.

   The probability $\mathbb{P}_{\text{Ins}}$ reflects the effect on the chemical potential on the particles and we will see in section 3 that this plays a huge role in the evential distribution and density of the tetromino fluid.

2. Deletion of a particle

   Here, similar to insertions, a tetromino in the fluid is chosen at random with a uniform distribution and erased with probability

$$\mathbb{P}_{\text{Del}} = \exp(-\mu) \cdot \frac{N}{V}.$$

   As before, the probability changes dramatically as the chemical potential is increased, and the the number of particles will increase with higher values for μ.

3. Translation of a particle

   A tetromino is selected at random (again using a uniform distribution) and is then displaced into one of the four lattice direction, each with equal probability. If this move cannot be realized, the particle remains unchanged.

4. Rotation of a particle

   A particle is selected at random and rotated into one of the other rotational directions, which is also selected at random. As before, if this move cannot be performed, the particle remains in it's place.

Figure 2: The anchor points for the 7 tetrominoes

To maximize the efficiency of the simulations, the ratio between the moves insertion:deletion:translation:rotation was assumed to be 3:3:9:2, following [2]. To decide where exactly a tetromino is inserted in the box or how to rotate or translate a particle, it is necessary to define anchor points for every tetromino. These anchor points are shown in figure 2. They represent the point which is kept fixed when a particle is rotated and they determine at what position a tetromino will be inserted.

One the one hand, the output of the simulations is the density of the system, i.e. the number of occupied lattice points divided by the total volume of the

system. On the other hand we can, from the snapshots of the configuration taken when the simulation converged, make qualitative analyses of the self-assembly properties of the fluids.

The simulation is divided into blocks of 3000 moves. At the end of each block the average density of the system is calculated and the simulation terminates when the following logarithmic convergence criterion is met:

$$\ln|\frac{\rho_n}{\rho_{n-1}}| < 0.001,$$

where ρ_n denotes the density of the system after block n. This leads to sufficiently smooth density curves while keeping the simulation time at an acceptable level (about 1 minute per simulation point on a VAIO Laptop with an i7 processor running Ubuntu 10.10).

2 Implementation of the Simulation

The simulation program was written in C++, using LaTeX and tikz to produce the snapshots of the fluids at equilibrium. The source code was split into 6 files which are reproduced with comments in chapter 4. They contain:

- `main.cpp`

  The main function which specifies the most important parameters (type of tetromino, block length, convergence criterium and chemical potential), calls the function for performing moves and for printing the output.

- `random.cpp`

  This code contains all prerequisites for the random number generator. Random numbers are needed at many points throughout the program: To determine which step to perform, find a free lattice site or to select a tetromino or its orientation. These random numbers are generated using the built- in function `rand()` and the program's starting time as a seed.

- `fit.cpp`

  This file contains all functions related to the lattice. Moving a tetromino, checking whether a piece fits at a given position or calculating the density and the number of free lattice points is performed here.

- `moves.cpp`

  Here all the functions which are related to the particle moves themselves are defined. This includes the function for choosing a move at every step but also the specific functions for inserting, deleting, translating or rotating a tetromino.

- `vec.cpp`

  This file contains the code for some vector operations which are needed in the simulation, in particular a function which calculates the mean of a given vector, to be used to calculate the average density in a simulation block.

- `print.cpp`

Here the output files are generated to print a lattice configuration. The file includes functions for printing the lattice and the tetrominoes with their orientation to the terminal (for debugging purposes). More importantly, it contains a function which generates an output file with the tikz code which can be easily imported into LaTeXto generate the figures presented in this document.

The whole program can be compiled using the gnu C++ compiler via the command

```
g++ -Wall main.cpp random.cpp fit.cpp moves.cpp vec.cpp print.cpp
-o tetromino
```

and then run with `./tetromino`.

3 Results

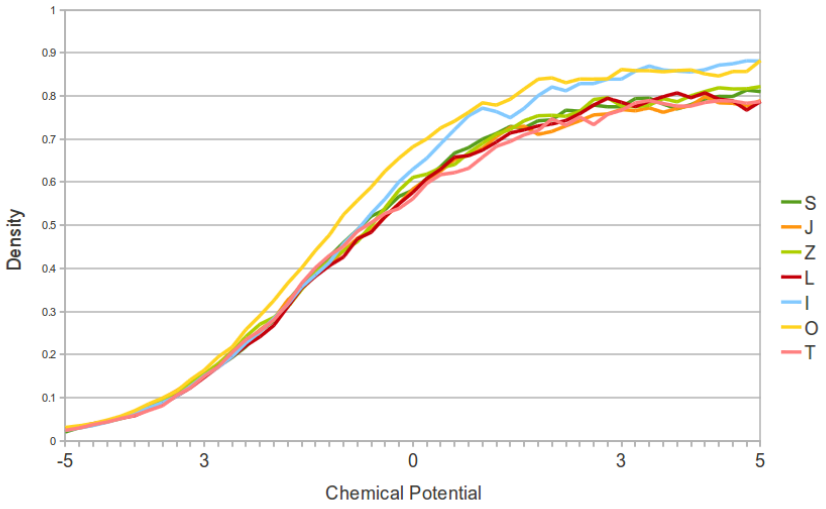

Figure 3: The change in densities of tetromino fluids for the seven different tetromino particles as a function of the chemical potential.

Pure fluids consisting of a single type of tetromino were simulated for chemical potentials ranging between -5 and 5, in intervalls of 0.2, i.e. Simulations for 420 state points were required in total. For every simulation, the density of the tetrominoes (i.e. the fraction of lattice sites covered by a tetromino) at equilibrium was calculated.

The resulting plot can be found in figure 3. The curves converge to a single curve at low chemical potential: This behaviour was expected, since with a very low chemical potential the pieces behave like identical gas particles of equal size.

At higher chemical potentials a difference between types of tetrominoes becomes apparent. Especially noticeable is the difference in density of squares (O shapes) and rods (I shapes) compared to the other types. At all densities, these two shapes form denser fluids than the other tetromino types.

In figure 4 snapshots of the O-fluid and I-fluid are shown at different chemical potentials. For squares, at low chemical potential no pattern emerges and the distribution seems to be completely guided by the condition that particles

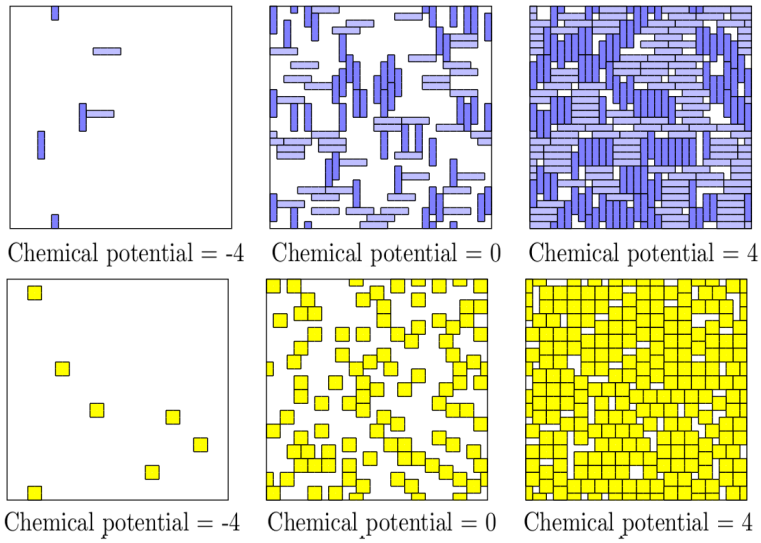

Figure 4: Configuration at different densities for I- and O-fluids

cannot overlap. For rods, on the other hand, we can observe that even at low chemical potential a significant correlation in the orientation of neighbouring tetrominoes occurs.

For higher potentials the clustering of rods is more pronounced and can also be observed for squares, which tend to form groups of about four pieces. This behaviour becomes more common at even higher densities, where a significant portion of the pieces are neatly aligned in larger clusters.

The density curves for S and Z fluids as functions of the chemical potential are nearly identical, as are the ones for J and L shapes. This is due to the fact that these pieces are mirror images of each other. Finally, the T shaped tetromino is characterized by the smallest density for all values of the chemical potential.

Similarly to rods and squares, S and Z tetrominoes tend to associate rather with particles of the same orientation, as can be seen in the top row of figure 5, especially at high chemical potentials. Neighboring particles are either offset horizontally or vertically or in one of the diagonal directions. Curiously, this second effect becomes more prevalent at higher chemical potentials, until at very high densities mostly diagonal ordering can be observed.

The remaining shapes, J, L and T, do not display a pronounced self-assembly

strategy but seem to be distributed more randomly. In fact, they associate best with neighboring particles of differing orientation. This is also the reason why these tetromino fluids do not reach the same densities as the others, especially rods and squares: The interlocking of the pieces with types of different orientation leads to a more compact structure where gaps cannot be filled as easily via rotation or translation.

The fluids do not display sharp (first-order) phase transitions. As detailed in [2], there remains the possibility of continouous phase transition, although this remains to date an open question. There are a lot of open questions regarding the self-assembly of polynomies, especially in 3 dimensions. But at the moment, even with a more efficient implementation of the simulation, similar analyses to these for more complex shapes and higher dimensions are almost impossible by current technological standards.

Relatively simple simulations with tetromino fluids do however provide an insight into the complex arrangements of simple shapes and how they display intricate structures and behaviours accross the studied range of chemical potentials, such as clustering and like-orientation. These simulations make it evident that the behaviour of even as primitive shapes as tetrominoes depend heavily on the interaction with other pieces and on their orientation, and show how difficult their behaviour is to predict.

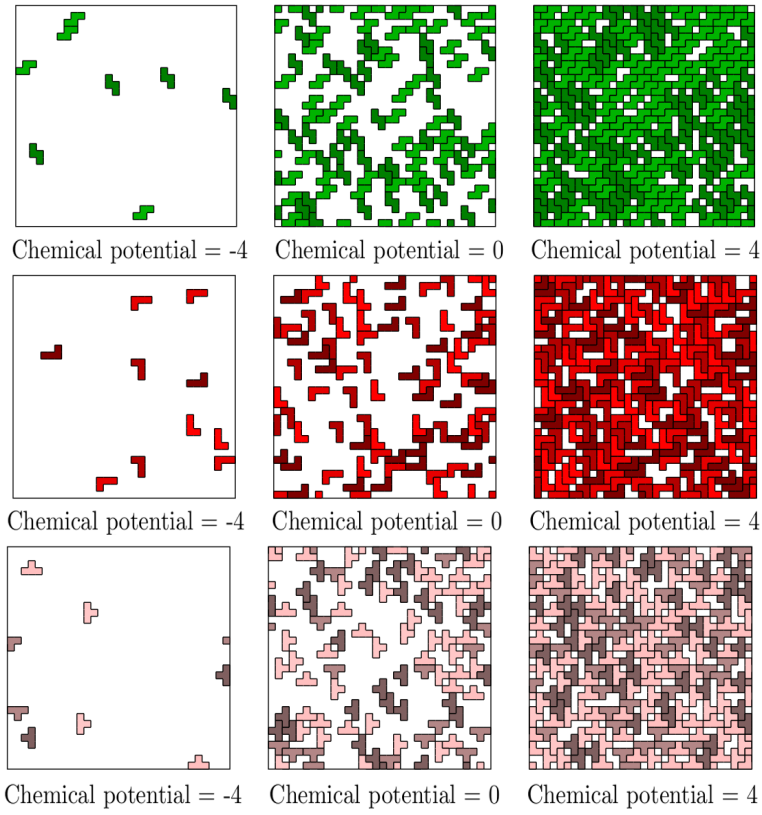

Figure 5: Configurations at different chemical potenatial for S-, L- and T-fluids

4 Sourcecode

4.1 main.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <cmath>
#include <vector>
#include <sstream>

using namespace std;

#include "defs.h"
#include "fit.h"
#include "random.h"
#include "moves.h"
#include "print.h"
#include "vec.h"

int main(){
    int lattice[SIZE*SIZE];
    string tetro[SIZE*SIZE];
    vector<double> vecDensity;
    double lastMean;
    ofstream dataFile;

/*  Possible tetromino types: T, O, I, L, Z, J, S */
    string type = "I";
    dataFile.open("tetromino.txt");

    srand(time(0));

    double pot = 10.0;
    dataFile << "Block, Mean, DeltaM" << endl;

/*  Initialize everything */
    int steps = 0;
    int blocks = 0;
    for (int i=0; i<SIZE*SIZE; i++){
        lattice[i] = 0;
        tetro[i] = "";
    }

/*  Do the first 3000 steps no matter what */
```

```
    vecDensity.clear();
    for (steps = 0; steps < 3000; steps++) {
        performStep(lattice, tetro, pot, type);
        vecDensity.push_back(density(lattice));
    }

/*  Continue until the density converges */
    do {
        lastMean = vecMean(vecDensity);
        vecDensity.clear();
        for (steps = 0; steps < 3000; steps++) {
            performStep(lattice, tetro, pot, type);
            vecDensity.push_back(density(lattice));
        }
        blocks++;
    //  Display some information and write to file
        cout << blocks << " " << vecMean(vecDensity) << endl;
        dataFile << blocks << ", ";
        dataFile << vecMean(vecDensity) << ", ";
        dataFile << abs(vecMean(vecDensity)/lastMean);
        dataFile << endl;

        } while (log(abs(vecMean(vecDensity)/lastMean)) > 0.001);

    dataFile.close();

/*  Print lattice and tetrominos in the terminal and write to file */
    cout << endl;
    printLattice(lattice);
    printTetros(tetro);
    printToFile("tetromino.tex",lattice,tetro);

    cout << "Final density: " << vecMean(vecDensity) << endl;

    return 0;
}
```

4.2 fit.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <vector>
#include <sstream>

#include "defs.h"
```

```
using namespace std;

int move_one(int position, int x, int y){
/*  Return the position when translated one step into direction x or y */
    int newpos;
    if (x == 1 && y == 0){
        newpos = position + 1;
        if (newpos%SIZE == 0) newpos = newpos - SIZE;
        return newpos;
    }
    else if (x == -1 && y == 0){
        newpos = position - 1;
        if (newpos%SIZE == SIZE-1 || newpos < 0) newpos = newpos + SIZE;
        return newpos;
    }
    else if (x == 0 && y == 1){
        newpos = position - SIZE;
        if (newpos < 0) newpos = newpos + SIZE*SIZE;
        return newpos;
    }
    else if (x == 0 && y == -1){
        newpos = position + SIZE;
        if (newpos > SIZE*SIZE-1) newpos = newpos - SIZE*SIZE;
        return newpos;
    }
    return -1;
}

vector<int> points(int position, int orientation, string type){
/*  Return the points on the lattice occupied by a tetromino of a given type
    at a given position */
    vector<int> output;
    stringstream type_or;
    type_or << type << orientation;

    int a, b, c;

    if (type_or.str() == "T1") {
        a = move_one(position, 1, 0);
        b = move_one(position, -1, 0);
        c = move_one(position, 0, 1);
    }
    else if  (type_or.str() == "T2") {
        a = move_one(position, 1, 0);
        b = move_one(position, 0, 1);
        c = move_one(position, 0, -1);
    }
    else if  (type_or.str() == "T3") {
        a = move_one(position, 1, 0);
```

```
    b = move_one(position, -1, 0);
    c = move_one(position, 0, -1);
}
else if (type_or.str() == "T4") {
    a = move_one(position, 0, 1);
    b = move_one(position, 0, -1);
    c = move_one(position, -1, 0);
}
else if (type_or.str() == "O1") {
    a = move_one(position, 0, 1);
    b = move_one(position, 1, 0);
    c = move_one(move_one(position,0,1), 1, 0);
}
else if (type_or.str() == "O2") {
    a = move_one(position, 0, -1);
    b = move_one(position, 1, 0);
    c = move_one(move_one(position,0,-1), 1, 0);
}
else if (type_or.str() == "O3") {
    a = move_one(position, 0, -1);
    b = move_one(position, -1, 0);
    c = move_one(move_one(position,0,-1), -1, 0);
}
else if (type_or.str() == "O4") {
    a = move_one(position, 0, 1);
    b = move_one(position, -1, 0);
    c = move_one(move_one(position,0,1), -1, 0);
}
else if (type_or.str() == "L1") {
    a = move_one(position, 0, 1);
    b = move_one(position, 0, -1);
    c = move_one(move_one(position,0,-1), 1, 0);
}
else if (type_or.str() == "L2") {
    a = move_one(position, 1, 0);
    b = move_one(position, -1, 0);
    c = move_one(move_one(position,0,-1), -1, 0);
}
else if (type_or.str() == "L3") {
    a = move_one(position, 0, 1);
    b = move_one(position, 0, -1);
    c = move_one(move_one(position,0,1), -1, 0);
}
else if (type_or.str() == "L4") {
    a = move_one(position, 1, 0);
    b = move_one(position, -1, 0);
    c = move_one(move_one(position,0,1), 1, 0);
}
else if (type_or.str() == "I1") {
    a = move_one(position, 1, 0);
```

```
    b = move_one(position, -1, 0);
    c = move_one(move_one(position,1,0), 1, 0);
}
else if (type_or.str() == "I2") {
    a = move_one(position, 0, -1);
    b = move_one(position, 0, 1);
    c = move_one(move_one(position,0,-1), 0, -1);
}
else if (type_or.str() == "I3") {
    a = move_one(position, -1, 0);
    b = move_one(position, 1, 0);
    c = move_one(move_one(position,-1,0), -1, 0);
}
else if (type_or.str() == "I4") {
    a = move_one(position, 0, 1);
    b = move_one(position, 0, -1);
    c = move_one(move_one(position,0,1), 0, 1);
}
else if (type_or.str() == "S1") {
    a = move_one(position, 0, 1);
    b = move_one(position, -1, 0);
    c = move_one(move_one(position,0,1), 1, 0);
}
else if (type_or.str() == "S2") {
    a = move_one(position, 0, 1);
    b = move_one(position, 1, 0);
    c = move_one(move_one(position,0,-1), 1, 0);
}
else if (type_or.str() == "S3") {
    a = move_one(position, 1, 0);
    b = move_one(position, 0, -1);
    c = move_one(move_one(position,0,-1), -1, 0);
}
else if (type_or.str() == "S4") {
    a = move_one(position, -1, 0);
    b = move_one(position, 0, -1);
    c = move_one(move_one(position,0,1), -1, 0);
}
else if (type_or.str() == "Z1") {
    a = move_one(position, 1, 0);
    b = move_one(position, 0, 1);
    c = move_one(move_one(position,0,1), -1, 0);
}
else if (type_or.str() == "Z2") {
    a = move_one(position, 1, 0);
    b = move_one(position, 0, -1);
    c = move_one(move_one(position,0,1), 1, 0);
}
else if (type_or.str() == "Z3") {
    a = move_one(position, -1, 0);
```

```
        b = move_one(position, 0, -1);
        c = move_one(move_one(position,0,-1), 1, 0);
    }
    else if (type_or.str() == "Z4") {
        a = move_one(position, -1, 0);
        b = move_one(position, 0, 1);
        c = move_one(move_one(position,-1,0), 0, -1);
    }
    else if (type_or.str() == "J1") {
        a = move_one(position, 0, 1);
        b = move_one(position, 0, -1);
        c = move_one(move_one(position,0,-1), -1, 0);
    }
    else if (type_or.str() == "J2") {
        a = move_one(position, 1, 0);
        b = move_one(position, -1, 0);
        c = move_one(move_one(position,0,1), -1, 0);
    }
    else if (type_or.str() == "J3") {
        a = move_one(position, 0, 1);
        b = move_one(position, 0, -1);
        c = move_one(move_one(position,0,1), 1, 0);
    }
    else if (type_or.str() == "J4") {
        a = move_one(position, 1, 0);
        b = move_one(position, -1, 0);
        c = move_one(move_one(position,0,-1), 1, 0);
    }

    output.push_back(position);
    output.push_back(a);
    output.push_back(b);
    output.push_back(c);
    return output;
}

int checkFit(int lattice[], int position, int orientation, string type){
/*  Return 0 if the lattice has space for a tetromino of a certain type
    at given position and orientation otherwise return 1 */
    int sum = 0;
    vector<int> vec_points = points(position, orientation, type);
    for(int i=0; i<4; i++){
        sum += lattice[vec_points[i]];
    }
    if (sum == 0) return 0;
    else return 1;
}

double density(int lattice[]){
/*  Calculate the density of the tetrominoes in the lattice */
```

```
    double sum = 0.;
    for (int i=0;i<SIZE*SIZE;i++){
        if (lattice[i] != 0) sum += 1.;
    }
    return sum/(SIZE*SIZE);
}

int freePoints(int lattice[]) {
/*  Calculate the number of free points in the lattice */
    int sum = 0;
    for (int i=0; i<SIZE*SIZE; i++) {
        if (lattice[i] == 0) sum +=1;
    }
    return sum;
}

int count(string tetro[], string type) {
/*  Count the number of tetrominoes of a given type in the lattice */
    int sum = 0;
    for (int i=0; i<SIZE*SIZE; i++) {
        if (tetro[i] == type) sum += 1;
    }
    return sum;
}
```

4.3 random.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <vector>

using namespace std;

#include "defs.h"
#include "fit.h"

double randomUni() {
/* Return a uniform random number in the unit interval */
    return rand() / double(RAND_MAX);
}


int randomStep() {
/* Return a random integer between 1 and 17 */
    return rand() % 17 + 1;
```

```
}

int randomDirection() {
/* Return a random integer between 1 and 4 */
    return rand() % 4 + 1;
}

int randomPoint() {
/* Return a random integer between 0 and SIZE*SIZE */
    return rand() % (SIZE*SIZE);
}

int randomFree(int lattice[]) {
/* Return a random free lattice point, chosen uniformly at random */
    int test = randomPoint();
    if (lattice[test] == 0) return test;
    else return randomFree(lattice);
}

int randomTetro(string tetro[], string type) {
/* Return the position of a random Tetromino of a given type.
    If the lattice is empty return -1 */
    int num_tetro = count(tetro, type);
    int rand_tetro = 0;
    if (num_tetro == 0) return -1;
    else {
        rand_tetro = rand() % num_tetro + 1;
        int count_tetro = 0;
        for (int i=0; i<SIZE*SIZE; i++) {
            if (tetro[i] != "") count_tetro++;
            if (count_tetro == rand_tetro) {
               return i;
            }
        }
    }
    return -1;
}
```

4.4 moves.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <cmath>
#include <vector>
```

```
using namespace std;

#include "defs.h"
#include "fit.h"
#include "random.h"

int insert_tetro(int lattice[], string tetro[], int position,
    int orientation, string type){
/*  Insert a piece at the given position and orientation */
    if(checkFit(lattice, position, orientation, type)==1) return 1;
    else {
        vector<int> vec_points = points(position, orientation, type);
        lattice[position] = orientation;
        for(int i=1; i<4; i++){
            lattice[vec_points[i]] = 8;
        }
        tetro[position] = type;
        return position;
    }
}

/*  Delete a piece at the given position */
int delete_tetro(int lattice[], string tetro[], int position, string type){
    int orientation = lattice[position];
    vector<int> vec_points = points(position, orientation, type);
    for(int i=0; i<4; i++){
            lattice[vec_points[i]] = 0;
    }
    tetro[position] = "";
    return position;
}

int translate_tetro(int lattice[], string tetro[], string type){
/*  Translate a pice */
    int position = randomTetro(tetro, type);
    if(position < 0) return 1;
    int orientation = lattice[position];
    int direction = randomDirection();
    int newposition;
    delete_tetro(lattice, tetro, position, type);

    switch (direction) {
    case 1:
        newposition = position + 1;
        break;
    case 2:
        newposition = position + SIZE;
        break;
    case 3:
        newposition = position - 1;
```

```
        break;
    case 4:
        newposition = position - SIZE;
        break;
    }
    if(newposition > SIZE*SIZE-1) newposition = newposition - SIZE*SIZE;
    if(newposition < 0) newposition = newposition + SIZE*SIZE;

/*  Check if the piece fits at the new position */
    if (checkFit(lattice, newposition, orientation, type) == 0) {
        insert_tetro(lattice, tetro, newposition, orientation, type);
    }
/*  If not, insert it at the original spot */
    else {
        insert_tetro(lattice, tetro, position, orientation, type);
    }
    return 0;
}

int rotate_tetro(int lattice[], string tetro[], string type) {
/*  Rotate a tetromino at the given position */
    int position = randomTetro(tetro, type);
    if (position < 0 ) return 1;
    int orientation = lattice[position];
    delete_tetro(lattice, tetro, position, type);

    int neworientation = randomDirection();
/*  Check if the piece fits at the new orientation */
    if (checkFit(lattice, position, neworientation, type) == 0) {
        insert_tetro(lattice, tetro, position, neworientation, type);
    }
/*  If not, insert it at the original spot */
    else {
        insert_tetro(lattice, tetro, position, orientation, type);
    }
    return 0;
}

int performStep(int lattice[], string tetro[], double pot, string type) {
/*  Perform a step at the ratios 3:3:6:2 */
    int step = randomStep();
    int occupied = randomTetro(tetro, type);
    int free = randomFree(lattice);
    int orientation = randomDirection();
    double probability;

    switch(step) {
    case 1:
    case 2:
    case 3:
```

```
        probability = exp(pot) * (SIZE*SIZE) / (count(tetro,type) + 1);
        if (randomUni() < probability) {
              insert_tetro(lattice, tetro, free, orientation, type);
          }
          break;
      case 4:
      case 5:
      case 6:
          if (occupied != -1) {
              probability = exp(-pot) * count(tetro, type) / (SIZE*SIZE);
            if (randomUni() < probability) {
                  delete_tetro(lattice, tetro, occupied, type);
              }
          }

          break;
      case 7:
      case 8:
      case 9:
      case 10:
      case 11:
      case 12:
      case 13:
      case 14:
      case 15:
          translate_tetro(lattice, tetro, type);
          break;
      case 16:
      case 17:
          rotate_tetro(lattice, tetro, type);
          break;
      }
      return 0;
}
```

4.5 print.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <vector>
#include <sstream>

using namespace std;

#include "defs.h"
```

```
#include "fit.h"
#include "random.h"
#include "moves.h"

int print_rectangle(ofstream &tetrofile, string shade, double x, double y,
    int l, int r, int u, int d){
/*  print a rectangle at the specified point (x,y), using a color given in shade
    Left, right, upper and lower Borders are drawn only when l,r,u or d equal 1,
    respectively. */

    tetrofile << "\\path[fill=" << shade << "](" << x << "," << y <<")";
    tetrofile << " rectangle (" << x + 0.1 << "," << y + 0.1 << ");\n";
    if (l==1) {
        tetrofile << "\\draw (" << x << "," << y << ") -- ("
        << x << "," << y + 0.1 << ");\n";
    }
    else {
        tetrofile << "\\draw[color=" << shade << "] (" << x << ","
        << y + 0.01 << ") -- (" << x << "," << y + 0.09 << ");\n";
    }
    if (r==1) {
        tetrofile << "\\draw (" << x + 0.1 << "," << y << ") -- ("
        << x + 0.1 << "," << y + 0.1 << ");\n";
    }
    else {
        tetrofile << "\\draw[color=" << shade << "] ("  << x + 0.1 << ","
        << y + 0.01 << ") -- (" << x + 0.1 << "," << y + 0.09 << ");\n";
    }
    if (u==1) {
        tetrofile << "\\draw (" << x << "," << y + 0.1 << ") -- ("
        << x + 0.1 << "," << y + 0.1 << ");\n";
    }
    else {
        tetrofile << "\\draw[color=" << shade << "] (" << x + 0.01 << ","
        << y + 0.1 << ") -- (" << x + 0.09 << "," << y + 0.1 << ");\n";
    }
    if (d==1) {
        tetrofile << "\\draw (" << x << "," << y << ") -- ("
        << x + 0.1 << "," << y << ");\n";
    }
    else {
        tetrofile << "\\draw[color=" << shade << "] (" << x + 0.01 << ","
        << y << ") -- (" << x + 0.09 << "," << y << ");\n";
    }
    return 0;
}


int print_tetro(ofstream &tetrofile, int lattice[], int position,
    string type){
```

```
/*    Prints the tetromino at the specified position and orientation
*/
    double x,y;
    double x1, y1, x2, y2, x3, y3;
    string color;
    stringstream shade;

    int orientation = lattice[position];

/*  The lattice points are stored in a vector */
    vector<int> vec_points = points(position, orientation, type);

/*  The points in the vector a transformed into cartesian coordinates for
    plotting */
    x = position%SIZE / 10.;
    y = (SIZE - position/SIZE - 1) / 10.;

    x1 = vec_points[1]%SIZE / 10.;
    y1 = (SIZE - vec_points[1]/SIZE - 1) / 10.;
    x2 = vec_points[2]%SIZE / 10.;
    y2 = (SIZE - vec_points[2]/SIZE - 1) / 10.;
    x3 = vec_points[3]%SIZE / 10.;
    y3 = (SIZE - vec_points[3]/SIZE - 1) / 10.;

/*  Print some debugging information in case something goes wrong */
    if (x1 < 0 || y1 < 0 ){
            cout << "This should not have happened" << endl;
            cout << position << ", " << orientation << endl;
    }
/*  Print the four rectangles for each tetromino */
    if (type == "T"){ color = "pink";
        if (orientation == 1) {
            shade << color;
            print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 1, 0);
        }
        else if (orientation == 2) {
            shade << color << "!90!white";
            print_rectangle(tetrofile, shade.str(), x, y, 1, 0, 0, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 0, 1);
        }
        else if (orientation == 3) {
            shade << color << "!70!black";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 1);
```

```
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 0, 1);
        }
        else if (orientation == 4) {
            shade << color << "!50!black";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 1, 0, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 1, 1);
        }
    }
    else if (type == "S"){
        color = "green";
        if (orientation == 1) {
            shade << color << "!70!black";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 0, 1, 1, 1);
        }
        else if (orientation == 2) {
            shade << color << "!50!black";
            print_rectangle(tetrofile, shade.str(), x, y, 1, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 0, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 0, 1);
        }
        else if (orientation == 3) {
            shade << color << "!70!black";
            print_rectangle(tetrofile, shade.str(), x, y, 1, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 0, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 1, 1);
        }
        else if (orientation == 4) {
            shade << color << "!50!black";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 1, 0);
        }
    }
    else if (type == "L"){ color = "red";
        if (orientation == 1) {
            shade << color;
            print_rectangle(tetrofile, shade.str(), x, y, 1, 1, 0, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 0, 1, 1, 1);
        }
        else if (orientation == 2) {
```

```
            shade << color << "!90!black";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 0, 1);
        }
        else if (orientation == 3) {
            shade << color << "!70!black";
            print_rectangle(tetrofile, shade.str(), x, y, 1, 1, 0, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 1, 1);
        }
        else if (orientation == 4) {
            shade << color << "!50!black";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 1, 0);
        }
    }
    else if (type == "I") {
        color = "blue";
        if (orientation == 1) {
            shade << color << "!25!white";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 0, 1, 1, 1);
        }
        else if (orientation == 2) {
            shade << color << "!50!white";
            print_rectangle(tetrofile, shade.str(), x, y, 1, 1, 0, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 1, 0, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 0, 1);
        }
        else if (orientation == 3) {
            shade << color << "!25!white";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 0, 1, 1, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 1, 1);
        }
        else if (orientation == 4) {
            shade << color << "!50!white";
            print_rectangle(tetrofile, shade.str(), x, y, 1, 1, 0, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 1, 0, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 1, 0);
```

```
    }
}
else if (type == "J") {
    color = "violet";
    if (orientation == 1) {
        shade << color;
        print_rectangle(tetrofile, shade.str(), x, y, 1, 1, 0, 0);
        print_rectangle(tetrofile, shade.str(), x1, y1, 1, 1, 1, 0);
        print_rectangle(tetrofile, shade.str(), x2, y2, 0, 1, 0, 1);
        print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 1, 1);
    }
    else if (orientation == 2) {
        shade << color << "!90!black";
        print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 1, 1);
        print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 1);
        print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 0, 1);
        print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 1, 0);
    }
    else if (orientation == 3) {
        shade << color << "!50!white";
        print_rectangle(tetrofile, shade.str(), x, y, 1, 1, 0, 0);
        print_rectangle(tetrofile, shade.str(), x1, y1, 1, 0, 1, 0);
        print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 0, 1);
        print_rectangle(tetrofile, shade.str(), x3, y3, 0, 1, 1, 1);
    }
    else if (orientation == 4) {
        shade << color << "!20!white";
        print_rectangle(tetrofile, shade.str(), x, y, 0, 0, 1, 1);
        print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 0);
        print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 1);
        print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 0, 1);
    }
}
else if (type == "Z") {
    color = "orange";
    if (orientation == 1) {
        shade << color;
        print_rectangle(tetrofile, shade.str(), x, y, 1, 0, 0, 1);
        print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 1);
        print_rectangle(tetrofile, shade.str(), x2, y2, 0, 1, 1, 0);
        print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 1, 1);
    }
    else if (orientation == 2) {
        shade << color << "!70!black";
        print_rectangle(tetrofile, shade.str(), x, y, 1, 0, 1, 0);
        print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 0, 1);
        print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 0, 1);
        print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 1, 0);
    }
    else if (orientation == 3) {
```

```
            shade << color;
            print_rectangle(tetrofile, shade.str(), x, y, 0, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 0, 1, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 0, 1, 1, 1);
        }
        else if (orientation == 4) {
            shade << color << "!70!black";
            print_rectangle(tetrofile, shade.str(), x, y, 0, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 1, 0, 1);
        }
    }
    else if (type == "O"){ color = "yellow";
        shade << color;
        if (orientation == 1) {
            print_rectangle(tetrofile, shade.str(), x, y, 1, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 0, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 0, 1, 1, 0);
        }
        if (orientation == 2) {
            print_rectangle(tetrofile, shade.str(), x, y, 1, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 1, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 0, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x3, y3, 0, 1, 0, 1);
        }
        if (orientation == 3) {
            print_rectangle(tetrofile, shade.str(), x, y, 0, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 1, 0);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 0, 1);
        }
        if (orientation == 4) {
            print_rectangle(tetrofile, shade.str(), x, y, 0, 1, 0, 1);
            print_rectangle(tetrofile, shade.str(), x1, y1, 0, 1, 1, 0);
            print_rectangle(tetrofile, shade.str(), x2, y2, 1, 0, 0, 1);
            print_rectangle(tetrofile, shade.str(), x3, y3, 1, 0, 1, 0);
        }
    }

    return 0;
}

/*  Print the whole lattice to the terminal */
int printLattice(int lattice[]) {
    for (int i=0;i<SIZE*SIZE;i++) {
        if (lattice[i] == 0) cout << ".";
        else cout << lattice[i];
```

```
        if ((i+1)%SIZE ==  0) cout << endl;
    }
    cout << endl;
    return 0;
}

/*  Print tetromino pieces and their orientation to the terminal */
int printTetros(string tetro[]) {
    for (int i=0;i<SIZE*SIZE;i++) {
        if (tetro[i] == "") cout << ".";
        else cout << tetro[i];
        if ((i+1)%SIZE == 0) cout << endl;
    }
    cout << endl;
    return 0;
}

/*  Print the configuration to a file.
    Can be shown in latex using \usepackage{tikz} in the preamble and then
    just \input{filename} in the document body.
*/
int printToFile(string filename, int lattice[], string tetro[]){

    ofstream tetrofile;
    tetrofile.open(filename.c_str());
    tetrofile << "\\begin{tikzpicture}\n";

    for (int i=0; i<SIZE*SIZE; i++) {
        if (tetro[i] != "") {
            print_tetro(tetrofile, lattice, i, tetro[i]);
        }
    }
    tetrofile << "\\draw(0,0) rectangle(" << SIZE/10. << ","
        << SIZE/10. << ");\n";
    tetrofile << "\\end{tikzpicture}";
    tetrofile.close();

    return 0;
}
```

4.6 vec.cpp

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;
```

```
double vecMean(vector<double> vec){
/* Caclulate the mean of a vector */
    double sum = 0;
    int length = 0;

    for(vector<double>::const_iterator i = vec.begin(); i < vec.end(); i++){
        sum += *i;
        length = i - vec.begin() + 1;
    }

    return sum/length;
}

double vecStdDev(vector<double> vec){
/* Calculate the standart deviation of a vector */
    double mean = vecMean(vec);
    double sum = 0;
    int length = 0;

    for(vector<double>::const_iterator i = vec.begin(); i < vec.end(); i++){
        sum += (*i - mean) * (*i - mean);
        length = i - vec.begin() + 1;
    }
    return sqrt(sum/(length-1));
}
```

4.7 fit.h

```
int move_one(int position, int steps_x, int steps_y);

vector<int> points(int position, int orientation, string type);

int checkFit(int lattice[], int position, int orientation, string type);

double density(int lattice[]);

int freePoints(int lattice[]);

int count(string tetro[], string type);
```

4.8 random.h

```
double randomUni();

int randomStep();
```

```
int randomDirection();

int randomPoint();

int randomFree(int lattice[]);

int randomTetro(string tetro[], string type);
```

4.9 moves.h

```
int insert_tetro(int lattice[], string tetro[], int position,
    int orientation, string type);

int delete_tetro(int lattice[], string tetro[], int position,
    string type);

int translate_tetro(int lattice[], string tetro[], string type);

int rotate_tetro(int lattice[], string tetro[]);

int switch_tetro(int lattice[], string tetro[]);

int change_tetro(int lattice[], string tetro[]);

int performStep(int lattice[], string tetro[], double pot,
    string type);
```

4.10 print.h

```
int print_rectangle(ofstream &tetrofile, string shade, double x, double y);

int print_tetro(ofstream &tetrofile, int lattice[], int position, string type);

int printLattice(int lattice[]);

int printTetros(string tetro[]);

int printToFile(string filename, int lattice[], string tetro[]);

int printXY(int position);
```

4.11 vec.h

```
double vecMean(vector<double> vec);
```

```
double vecStdDev(vector<double> vec);
```

4.12 defs.h

```
#define SIZE 37
```

References

[1] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, 1987.

[2] B. C. Barnes, D. W. Siderius, and L. D. Gelb. Structure, thermodynamics, and solubility in tetromino fluids. *Langmuir : the ACS journal of surfaces and colloids*, 25(12):6702–16, June 2009.

[3] E. D. Demaine and S. Hohenberger. Tetris is Hard , Even to Approximate. *Technology*, pages 1–56, 2008.

[4] D. Frenkel and B. Smit. *Understanding Molecular Simulation. From Algorithms to Applications*. Academic Press, second edi edition, 2002.

[5] J. L. Jacobsen. Tetromino tilings and the Tutte polynomial. *Science*, 2008.

[6] M. Mezei. A cavity-biased (T, V, μ) Monte Carlo method for the computer simulation of fluids. *Molecular Physics*, 40(4):901–906, 1980.