Programming 2 - Assignment 1

Felix Dreßler (k12105003)

April 7, 2022

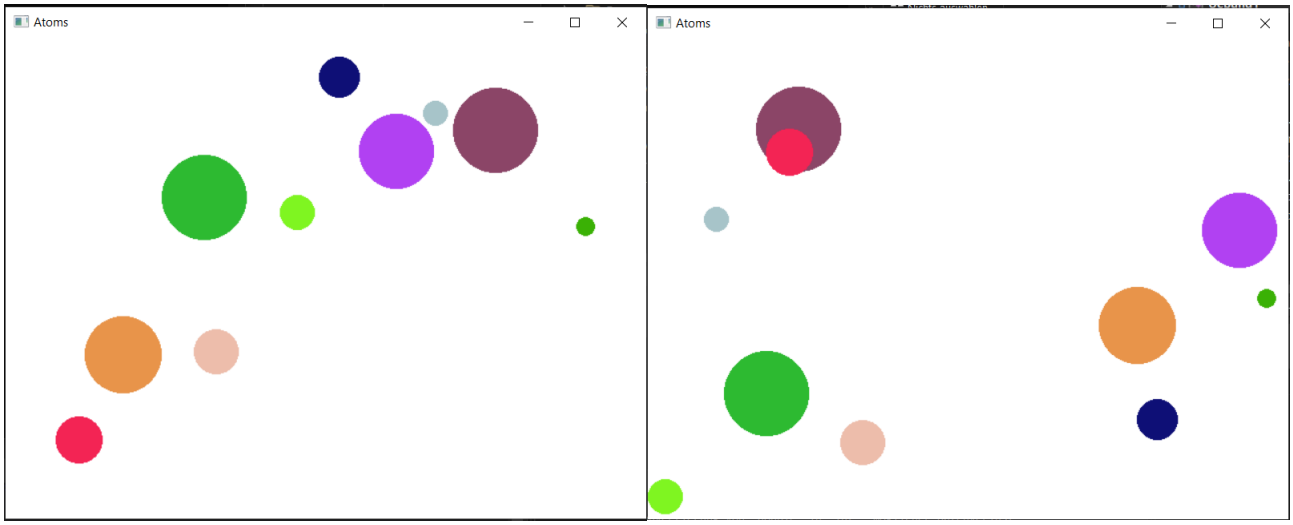
1 Testing the Program

For testing purposes a series of tests was performed. The program was tested first without the implementation of collisions between different atoms for an fixed input saved in "Input.txt" and random generated atoms. Next the program was tested including the implementation of collision between different atoms. This was executed once using the same fixed "Input.txt" and with random generated atoms.

For each test, the text output of the program was recorded, stating the initial values of the atoms. Furthermore, screenshots of the initial-state and the end-state are included.

1.1 Tests without atom-collison

1.1.1 Input.txt

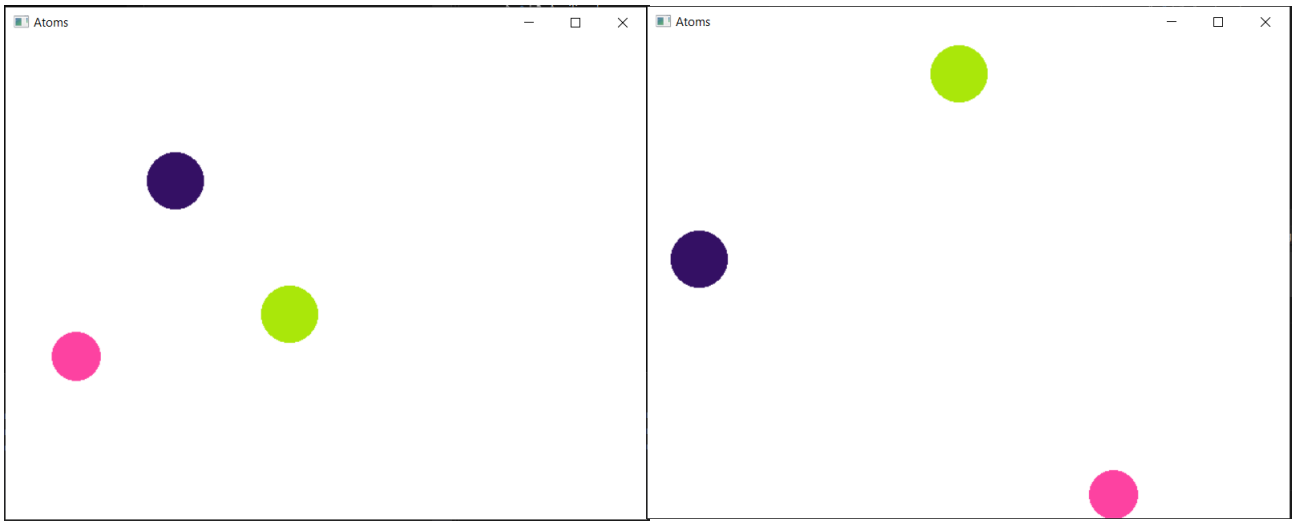


(a) start state (b) end state

1.1.2 randomly created atoms

```
the number of Atoms is: 3
Atom 1 has the following values assigned:
Color1 is      11200266
Radius1 is     28
x Pos.1 is     283
y Pos.1 is     275
vx1 is         6
vy1 is         16
Atom 2 has the following values assigned:
Color2 is      3412068
Radius2 is     28
x Pos.2 is     169
y Pos.2 is     142
vx2 is         17
vy2 is         7
Atom 3 has the following values assigned:
Color3 is      16597665
```

```
Radius3 is      24
x Pos.3 is      70
y Pos.3 is      317
vx3 is          20
vy3 is          18
```



(a) start state (b) end state

1.2 Tests with atom-collsion

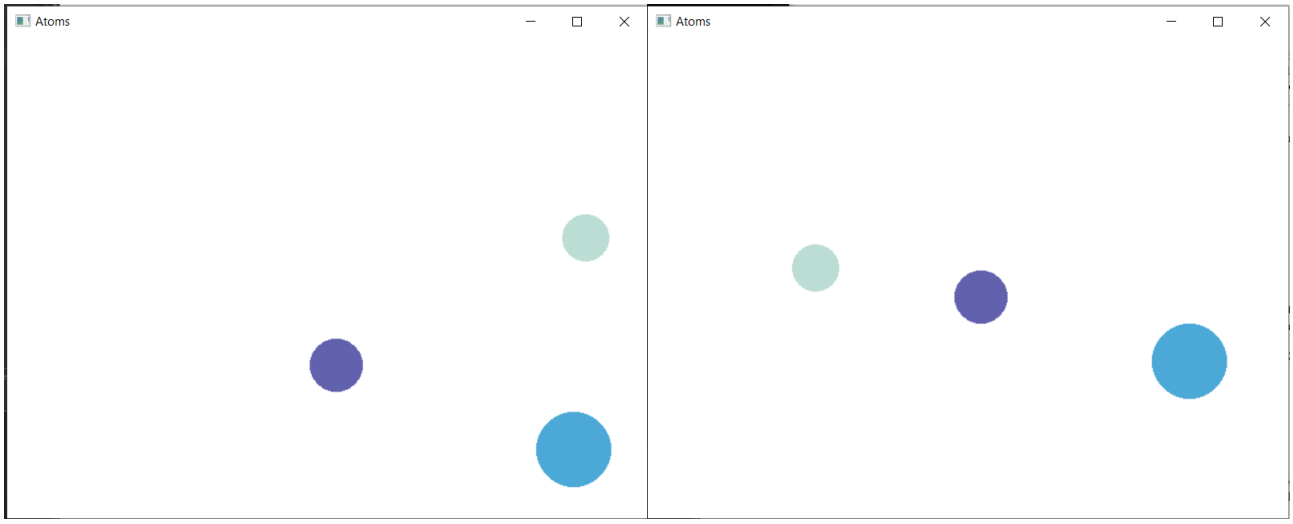
1.2.1 Input.txt

(a) start state (b) end state

1.2.2 randomly created atoms

Input Data:

```
the number of Atoms is: 3
Atom 1 has the following values assigned:
Color1 is       12312020
Radius1 is      23
x Pos.1 is      577
y Pos.1 is      200
vx1 is          5
vy1 is          8
Atom 2 has the following values assigned:
Color2 is       5024215
Radius2 is      37
x Pos.2 is      565
y Pos.2 is      411
vx2 is          22
vy2 is          18
Atom 3 has the following values assigned:
Color3 is       6381998
Radius3 is      26
x Pos.3 is      328
y Pos.3 is      327
vx3 is          7
vy3 is          17
```



(a) start state (b) end state

2 The Program - Main.cpp

2.1 Header of Main.cpp

The following paragraph shows the beginning of the File "Main.cpp" of the "Atoms" Project. We see, that in comparison to the given Header in the assignment, there is an "Auxiliary.h" included in line 32. This Header-File will be discussed in the next section.

There are also four global variables defined in lines 34 to 38. W and H are width and height of the created window, in which the atoms will be simulated. S describes the time that will pass between each frame. It will be passed to the Sleep function that is describes in lines 14 to 25.

```
1  //*********************************************************
2  //"Main.cpp"
3  //
4  // is the Main cpp file of the "Atoms" project
5  //
6  // created by Felix Dressler, 04.04.2022
7  //*********************************************************
8  #include <iostream>
9  #include <cstdlib>
10 #include <cmath>
11
12 #include "Drawing.h"
13
14 #if defined(_WIN32) || defined (_WIN64)
15 #include <windows.h>
16 #else
17 #include <time.h>
18 static void Sleep(int ms)
19 {
20   struct timespec ts;
21   ts.tv_sec = ms / 1000;
22   ts.tv_nsec = (ms % 1000) * 1000000;
23   nanosleep(&ts, NULL);
24 }
25 #endif
```

```
26
27 using namespace std;
28 using namespace compsys;
29
30 #include  <string>  //defines the getline() function
31 #include <fstream>
32 #include "Auxiliary.h" //includes auxiliary functions such as "toPolar", "random", ...
33
34 int W = 640;  //W,H are the width and the height of the created window
35 int H = 480;
36
37 int S = 40;   //time between the frame-updates - sleep
38 int F = 200;  //number of updates that are performed by the program
```

2.2 structure "Atom"

```
39 //*********************************************************
40 // struct "Atom"
41 //
42 // defines the data structure for an Atom
43 // Atoms hold the values:
44 // c ... colour
45 // r ... radius
46 // vx ... velocity in x
47 // vy ... velocity in y
48 // x ... x-value for position
49 // y ... y-value for position
50 //*********************************************************
51
52 typedef struct Atom
53 {
54   int c;
55   double r;
56   double vx;
57   double vy;
58   double x;
59   double y;
60 };
```

2.3 Function "number"

```
61 //*********************************************************
62 //  Function "number"
63 //
64 //  This function determines the number of atoms that should be
65 // created. It checks if there was an input file given. If yes, it
66 // takes the number of atoms from this file. If not it gives the back
67 // the number 3.
68 //
69 // input:
70 // argc ... number of arguments given when programm is called
71 // argv[] ... argument that was given when programm is called
72 // (should be the directory to a .txt file that holds start values
73 // for the simulation.
74 //
```

```
75 //
76 // output:
77 // n ... number of atoms that shoudl be created
78 //*********************************************************
79
80 double number(int argc, const char* argv[]) {
81   int N = 3;
82
83   if (argc == 2)
84   {
85     ifstream Input{ argv[1] };
86     if (!Input)
87     {
88       cout << "Error: check Input file (numbers)" << endl;
89       return -1;
90     }
91
92     Input >> N;
93     Input.close();
94   }
95
96   cout << "the number of Atoms is: " << N << endl;
97
98   return N;
99 }
```

2.4 Function "init"

```
100 //*********************************************************
101 //Function "init"
102 //
103 // This function initializes n Atoms with their respective values.
104 // If an input file was given, it uses the values that are stated there.
105 // If no input file was given, it defines the values of the atoms at random
106 // within a given Interval.
107 //
108 // input:
109 // n ... number of atoms that should be created
110 // Atom[] ... gives an array of Atoms (struct defined above)
111 // argc ... number of arguments given when programm is called
112 // argv[] ... argument that was given when programm is called
113 // (should be the directory to a .txt file that holds start values
114 // for the simulation.
115 //
116 // output: none
117 //*********************************************************
118
119 void init(int n, Atom Atom[], int argc, const char* argv[]) {
120   if (argc == 2)
121   {
122     ifstream Input{ argv[1] };
123     if (!Input) {
124       cout << "Error: check Input file (init)" << endl;
125       return;
126     }
127
128     while (Input)
129     {
```

```
130       int n;  //saves the first number in the input document to be able to
131           //access the other numbers, this variable will not be used
132       Input >> n;
133       for (int j = 0; j < n; j++)
134       {
135         //we assume that the user only gives us valid placements
136         //i.e. the atoms do not overlap
137         Input >> Atom[j].c;
138         Input >> Atom[j].r;
139         Input >> Atom[j].x;
140         Input >> Atom[j].y;
141         Input >> Atom[j].vx;
142         Input >> Atom[j].vy;
143
144         //gives ut the values of each atom
145         cout << "Atom " << j + 1 << " has the following values assigned:" << endl;
146         cout << "Color" << j + 1 << " is      " << Atom[j].c << endl;
147         cout << "Radius" << j + 1 << " is     " << Atom[j].r << endl;
148         cout << "x Pos." << j + 1 << " is     " << Atom[j].x << endl;
149         cout << "y Pos." << j + 1 << " is     " << Atom[j].y << endl;
150         cout << "vx" << j + 1 << " is         " << Atom[j].vx << endl;
151         cout << "vy" << j + 1 << " is         " << Atom[j].vy << endl;
152       }
153     }
154
155     Input.close();
156   }
157   else if(argc == 1) {
158     srand(time(0));
159     for (int j = 0; j < n; j++) {
160       Atom[j].c = random(000, 0xFFFFFF);
161       Atom[j].r = random(20, 40);
162       Atom[j].vx = random(5, 25);
163       Atom[j].vy = random(5, 25);
164       Atom[j].x = random(Atom[j].r, W - Atom[j].r);
165       Atom[j].y = random(Atom[j].r, H - Atom[j].r);
166
167       //the following function should check, if Atoms were to overlap if they overlap,
168       //it tries three times to create a new one, if it fails on the third time, it exits
169       bool valid=true;
170
171       for (int l = 0; l < j; l++) {
172         int m = 0;
173
174         double dx = Atom[j].x - Atom[l].x;
175         //difference between the x-coordinates of the two compared atoms
176         double dy = Atom[j].y - Atom[l].y;
177         //difference between the y-coordinates of the two compared atoms
178         double rsum = Atom[j].r + Atom[l].r; //sum of the radi of the two atoms compared
179
180         for (int k=0; dx * dx + dy * dy < rsum*rsum && valid && k<=m; k++)
181         {
182           Atom[j].x = random(Atom[j].r, W - Atom[j].r);
183           Atom[j].y = random(Atom[j].r, H - Atom[j].r);
184
185           m++;
186
187           if (m > 2) {
188             valid = false;
189           }
```

```
190         }
191         if(!valid){
192           cout << "Error: Atoms would overlap, please try again!" << endl;
193           exit(1);
194         }
195       }
196
197       cout << "Atom " << j + 1 << " has the following values assigned:" << endl;
198       cout << "Color" << j + 1 << " is      " << Atom[j].c << endl;
199       cout << "Radius" << j + 1 << " is     " << Atom[j].r << endl;
200       cout << "x Pos." << j + 1 << " is     " << Atom[j].x << endl;
201       cout << "y Pos." << j + 1 << " is     " << Atom[j].y << endl;
202       cout << "vx" << j + 1 << " is         " << Atom[j].vx << endl;
203       cout << "vy" << j + 1 << " is         " << Atom[j].vy << endl;
204     }
205   }
206   else { cout << "Error: Please give a valid Argument!"; }
207 }
```

2.5 Function "Draw"

```
208 //*********************************************************
209 // Function "Draw"
210 //
211 // The draw function draws each individual "Frame" of the animation
212 // by first drawing a blank background and then drawing each individual Atom
213 // at its respective position. All of this is updated as one "Frame".
214 //
215 // input: number of Atoms and values of these Atoms
216 //
217 // output: none
218 //*********************************************************
219
220 void draw(int n, Atom Atom[]) {
221   fillRectangle(0, 0, W, H, 0xFFFFFF);
222
223   for (int j = 0; j < n; j++) {
224     fillEllipse(Atom[j].x - Atom[j].r, Atom[j].y - Atom[j].r,
225       2 * Atom[j].r, 2 * Atom[j].r, Atom[j].c);
226   }
227
228   flush();
229 }
```

2.6 Function "Update"

```
230 //*********************************************************
231 // Funtion "Update"
232 //
233 // The "Update" Function determines the position of every Atom
234 // by calculation their position through their velocities in x and y.
235 // It also handles collisions between different atoms and between
236 // atoms and walls.
237 //
238 // It first checks if an atom was to collide with a wall, if yes it
```

```
239 // changes its velocity accordingly.
240 // Next it checks if this atom was to collide with any of the other
241 // atoms, if yes it changes their velocities accordingly.
242 //
243 // Input:number of Atoms and Values of Atoms
244 //
245 // Output: none
246 //*********************************************************
247
248 void update(int n, Atom Atom[]) {
249   for (int j = 0; j < n; j++) {
250
251     Atom[j].x += Atom[j].vx;
252     Atom[j].y += Atom[j].vy;
253
254     //checks for collisions between atoms and walls
255
256     if (Atom[j].x >= W - Atom[j].r)
257     {
258       Atom[j].vx = -Atom[j].vx;
259       Atom[j].x = W - Atom[j].r;
260     }
261     if (Atom[j].x <= Atom[j].r)
262     {
263       Atom[j].vx = -Atom[j].vx;
264       Atom[j].x = Atom[j].r;
265     }
266     if (Atom[j].y >= H - Atom[j].r)
267     {
268       Atom[j].vy = -Atom[j].vy;
269       Atom[j].y = H - Atom[j].r;
270     }
271     if (Atom[j].y <= Atom[j].r)
272     {
273       Atom[j].vy = -Atom[j].vy;
274       Atom[j].y = Atom[j].r;
275     }
276
277     //checks for collisions between different atoms
278     for (int l = 0; l <= j; l++) {
279
280       double dx = Atom[j].x - Atom[l].x;
281       //difference between the x-coordinates of the two compared atoms
282       double dy = Atom[j].y - Atom[l].y;
283       //difference between the y-coordinates of the two compared atoms
284       double rsum = Atom[j].r + Atom[l].r; //sum of the radi of the two atoms compared
285
286       if (dx*dx + dy*dy <= rsum*rsum && j != l)
287       {
288         double alpha = atan2(dy,dx);
289         double dx1 = cos(alpha) * rsum;
290         double dy1 = sin(alpha) * rsum;
291
292         Atom[j].x += dx1 - dx;
293         Atom[j].y += dy1 - dy;
294
295         double beta = 3.1415926 - alpha;
296
297         double Vx = 0;
298         double Vy = 0;
```

```
299
300         double a;//angle of a vector as outputted from "toPolar"
301         double r;//radius of a vector as outputted from "toPolar"
302         double vx1;//new velocity in x after rotation and transformation by "toCartesian"
303         double vy1;//new velocity in x after rotation and transformation by "toCartesian"
304
305         toPolar(Atom[j].vx, Atom[j].vy, r, a);
306         a - beta;
307         toCartesian(r, a, vx1, vy1);
308
309         Atom[j].vx = vx1;
310         Atom[j].vy = vy1;
311
312         toPolar(Atom[l].vx, Atom[l].vy, r, a);
313         a - beta;
314         toCartesian(r, a, vx1, vy1);
315
316         Atom[l].vx = vx1;
317         Atom[l].vy = vy1;
318
319         //Vx and Vy as describes in the theory of elastic impact
320         Vx = (pow(Atom[l].r, 2) * Atom[l].vx + pow(Atom[j].r, 2) * Atom[j].vx)
321           / (pow(Atom[j].r, 2) + pow(Atom[l].r, 2));
322         Vy = (pow(Atom[l].r, 2) * Atom[l].vy + pow(Atom[j].r, 2) * Atom[j].vy)
323           / (pow(Atom[j].r, 2) + pow(Atom[l].r, 2));
324
325         Atom[j].vx = 2 * Vx - Atom[j].vx;
326         Atom[j].vy = 2 * Vy - Atom[j].vy;
327
328         Atom[l].vx = 2 * Vx - Atom[l].vx;
329         Atom[l].vy = 2 * Vy - Atom[l].vy;
330       }
331     }
332   }
333 }
```

2.7 Main

```
334 //Main as described in the Assignment
335
336 int main(int argc, const char* argv[])
337 {
338   beginDrawing(W, H, "Atoms", 0xFFFFFF, false);
339   int n = number(argc, argv);
340   Atom* atoms = new Atom[n];
341   init(n, atoms, argc, argv);
342   draw(n, atoms);
343   cout << "Press <ENTER> to continue..." << endl;
344   string s; getline(cin, s);
345   for (int i = 0; i < F; i++)
346   {
347     update(n, atoms);
348     draw(n, atoms);
349     Sleep(S);
350   }
351   delete[] atoms;
352   cout << "Close window to exit..." << endl;
353   endDrawing();
```

```
354 }
```

3 The Program - Auxiliary

For better clarity, auxiliary functions were outsourced to the files "Auxiliary.h" and "Auxiliary.cpp".

3.1 Auciliary.h

In the file "Auxiliary.h", the auxiliary functions are declared.

```
1  //*********************************************************
2  //Header "Auxiliary.h"
3  //
4  // declares auxiliary functions for use in the "Atoms" project
5  // for further elaboration of functions, see "Auxiliary.cpp"
6  //
7  // created by Felix Dressler, 04.04.2022
8  //*********************************************************
9  #pragma once
10
11 //calculates radius r and angle a of a vector using its Cartesian coordinates
12 void toPolar(double x, double y, double& r, double& a);
13
14
15 //calculates the Cartesian coordinates of a vector using its Polar-form
16 void toCartesian(double r, double a, double& x, double& y);
17
18 //creates a random number inbetween two limtis
19 int random(int llimit, int ulimit);
```

3.2 Auxiliary.cpp

In the file "Auxiliary.cpp" the auxiliary functions are defined.

"random" creates a random value in between two limits. It may seem unintuitive to implement this function this way, but in order to be able to reach more possible values with bigger given limtits (for example the limits 0 and 0xFFFFFF which are used to represent colours) we would not be able to get any red. This is because the RAND_MAX is too low on some common C++ compilers.

```
1  //*********************************************************
2  //File "Auxiliary.cpp"
3  //
4  // defines auxiliary functions for use in the "Atoms" project
5  // that are declared in "Auxiliary.h"
6  //
7  // created by Felix Dressler, 04.04.2022
8  //*********************************************************
9  #include <iostream>
10 #include <cstdlib>
11 #include <cmath>
12
13 #include "Auxiliary.h"
14
15 //*********************************************************
16 // Funtion "toPolar"
17 //
18 //calculates radius rand angle a of a vector using its Cartesian coordinates
19 //*********************************************************
```

```
20
21 void toPolar(double x, double y, double& r, double& a)
22 {
23   a = atan2(y, x);
24   r = sqrt(x * x + y * y);
25 }
26
27 //*******************************************************
28 // Funtion "toCaresian"
29 //
30 //calculates the Cartesian coordinates of a vector using its Polar-form
31 //*******************************************************
32
33 void toCartesian(double r, double a, double& x, double& y)
34 {
35   x = r * cos(a);
36   y = r * sin(a);
37 }
38
39 //*******************************************************
40 // Funtion "random"
41 //
42 // This function gives a random number inbetween given limits
43 //
44 // input: two int numbers which define the lower and the upper
45 // limits of the outputted random number
46 //
47 // output: a random number in between the given limits
48 // including the limits
49 //*******************************************************
50
51 int random(int llimit, int ulimit) {
52
53   return ((rand()+rand()*(RAND_MAX+1)) % (ulimit - llimit + 1)) + llimit;
54 }
```