Programming 2 - Assignment 1

Felix Dreßler (k12105003)

April 5, 2022

1 Main.cpp

1.1 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 row 32. This Header-File will be discussed in the next section.

There are also four global variables defined in rows 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 rows 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
```

1.2 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   int r;
56   int vx;
57   int vy;
58   int x;
59   int y;
60 };
```

1.3 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  }
```

1.4 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
121   if (argc == 2)
122   {
123     ifstream Input{ argv[1] };
124     if (!Input) {
125       cout << "Error: check Input file (init)" << endl;
126       return;
127     }
128
129     while (Input)
130     {
131       int n;  //saves the first number in the input document to be able to
132           //access the other numbers, this variable will not be used
133       Input >> n;
134       for (int j = 0; j < n; j++)
135       {
136         //we assume that the user only gives us valid placements
137         //i.e. the atoms do not overlap
138         Input >> Atom[j].c;
139         Input >> Atom[j].r;
140         Input >> Atom[j].x;
141         Input >> Atom[j].y;
142         Input >> Atom[j].vx;
143         Input >> Atom[j].vy;
144
145         //gives ut the values of each atom
146         cout << "Atom " << j + 1 << " has the following values assigned:" << endl;
147         cout << "Color" << j + 1 << " is      " << Atom[j].c << endl;
148         cout << "Radius" << j + 1 << " is     " << Atom[j].r << endl;
149         cout << "x Pos." << j + 1 << " is     " << Atom[j].x << endl;
150         cout << "y Pos." << j + 1 << " is     " << Atom[j].y << endl;
151         cout << "vx" << j + 1 << " is         " << Atom[j].vx << endl;
```

```
152         cout << "vy" << j + 1 << " is         " << Atom[j].vy << endl;
153       }
154     }
155
156     Input.close();
157   }
158   else if(argc == 1) {
159     srand(time(0));
160     for (int j = 0; j < n; j++) {
161       Atom[j].c = random(000, 0xFFFFFF);
162       Atom[j].r = random(20, 40);
163       Atom[j].vx = random(5, 25);
164       Atom[j].vy = random(5, 25);
165       Atom[j].x = random(Atom[j].r, W - Atom[j].r);
166       Atom[j].y = random(Atom[j].r, H - Atom[j].r);
167
168       //the following function should check, if Atoms were to overlap if they overlap,
169       //it tries three times to create a new one, if it fails on the third time, it exits
170       bool valid=true;
171
172       for (int l = 0; l <= j; l++) {
173         int m = 0;
174
175         int dx = Atom[j].x - Atom[l].x; //difference between the x-coordinates of the two
               compared atoms
176         int dy = Atom[j].y - Atom[l].y; //difference between the y-coordinates of the two
               compared atoms
177         int rsum = Atom[j].r + Atom[l].r; //sum of the radi of the two atoms compared
178
179         if (dx*dx + dy*dy < rsum && j != l && valid)
180         {
181           Atom[j].x = random(Atom[j].r, W - Atom[j].r);
182           Atom[j].y = random(Atom[j].r, H - Atom[j].r);
183
184           m++;
185
186           if (m > 2) {
187             valid = false;
188           }
189         }
190         else if(!valid){
191           cout << "Error: Atoms would overlap, please try again!" << endl;
192           exit(1);
193         }
194       }
195
196       cout << "Atom " << j + 1 << " has the following values assigned:" << endl;
197       cout << "Color" << j + 1 << " is      " << Atom[j].c << endl;
198       cout << "Radius" << j + 1 << " is     " << Atom[j].r << endl;
199       cout << "x Pos." << j + 1 << " is     " << Atom[j].x << endl;
200       cout << "y Pos." << j + 1 << " is     " << Atom[j].y << endl;
201       cout << "vx" << j + 1 << " is         " << Atom[j].vx << endl;
202       cout << "vy" << j + 1 << " is         " << Atom[j].vy << endl;
203     }
204   }
205   else { cout << "Error: Please give a valid Argument!"; }
206 }
```

1.5 Function "Draw"

```
207 //*********************************************************
208 // Function "Draw"
209 //
210 // The draw function draws each individual "Frame" of the animation
211 // by first drawing a blank background and then drawing each individual Atom
212 // at its respective position. All of this is updated as one "Frame".
213 //
214 // input: number of Atoms and values of these Atoms
215 //
216 // output: none
217 //*********************************************************
218
219 void draw(int n, Atom Atom[]) {
220   fillRectangle(0, 0, W, H, 0xFFFFFF);
221
222   for (int j = 0; j < n; j++) {
223     fillEllipse(Atom[j].x - Atom[j].r, Atom[j].y - Atom[j].r, 2 * Atom[j].r, 2 * Atom[j].
          r, Atom[j].c);
224   }
225
226   flush();
227 }
```

1.6 Function "Update"

```
228 //*********************************************************
229 // Funtion "Update"
230 //
231 // The "Update" Function determines the position of every Atom
232 // by calculation their position through their velocities in x and y.
233 // It also handles collisions between different atoms and between
234 // atoms and walls.
235 //
236 // It first checks if an atom was to collide with a wall, if yes it
237 // changes its velocity accordingly.
238 // Next it checks if this atom was to collide with any of the other
239 // atoms, if yes it changes their velocities accordingly.
240 //
241 // Input:number of Atoms and Values of Atoms
242 //
243 // Output: none
244 //*********************************************************
245
246 void update(int n, Atom Atom[]) {
247   for (int j = 0; j < n; j++) {
248
249     Atom[j].x += Atom[j].vx;
250     Atom[j].y += Atom[j].vy;
251
252     //checks for collisions between atoms and walls
253
254     if (Atom[j].x >= W - Atom[j].r)
255     {
256       Atom[j].vx = -Atom[j].vx;
257       Atom[j].x = W - Atom[j].r;
```

```
258     }
259     if (Atom[j].x <= Atom[j].r)
260     {
261       Atom[j].vx = -Atom[j].vx;
262       Atom[j].x = Atom[j].r;
263     }
264     if (Atom[j].y >= H - Atom[j].r)
265     {
266       Atom[j].vy = -Atom[j].vy;
267       Atom[j].y = H - Atom[j].r;
268     }
269     if (Atom[j].y <= Atom[j].r)
270     {
271       Atom[j].vy = -Atom[j].vy;
272       Atom[j].y = Atom[j].r;
273     }
274
275     //checks for collisions between different atoms
276     for (int l = 0; l <= j; l++) {
277
278       int dx = Atom[j].x - Atom[l].x; //difference between the x-coordinates of the two
              compared atoms
279       int dy = Atom[j].y - Atom[l].y; //difference between the y-coordinates of the two
              compared atoms
280       int rsum = Atom[j].r + Atom[l].r; //sum of the radi of the two atoms compared
281
282       if (dx*dx + dy*dy <= rsum && j != l)
283       {
284         double alpha = atan2(dy,dx);
285         int dx1 = cos(alpha) * rsum;
286         int dy1 = sin(alpha) * rsum;
287
288         Atom[j].x += dx1 - dx;
289         Atom[j].y += dy1 - dy;
290
291         double beta = 3.1415926 - alpha;
292
293         double Vx = 0;
294         double Vy = 0;
295
296         double a; //angle of a vector as outputted from "toPolar"
297         double r; //radius of a vector as outputted from "toPolar"
298         double vx1; //new velocity in x after rotation and transformation by "toCartesian
                "
299         double vy1; //new velocity in x after rotation and transformation by "toCartesian
                "
300
301         toPolar(Atom[j].vx, Atom[j].vy, r, a);
302         a - beta;
303         toCartesian(r, a, vx1, vy1);
304
305         Atom[j].vx = vx1;
306         Atom[j].vy = vy1;
307
308         toPolar(Atom[l].vx, Atom[l].vy, r, a);
309         a - beta;
310         toCartesian(r, a, vx1, vy1);
311
312         Atom[l].vx = vx1;
313         Atom[l].vy = vy1;
```

```
314
315         //Vx and Vy as describes in the theory of elastic impact
316         Vx = (pow(Atom[l].r, 2) * Atom[l].vx + pow(Atom[j].r, 2) * Atom[j].vx) / (pow(
                Atom[j].r, 2) + pow(Atom[l].r, 2));
317         Vy = (pow(Atom[l].r, 2) * Atom[l].vy + pow(Atom[j].r, 2) * Atom[j].vy) / (pow(
                Atom[j].r, 2) + pow(Atom[l].r, 2));
318
319         Atom[j].vx = 2 * Vx - Atom[j].vx;
320         Atom[j].vy = 2 * Vy - Atom[j].vy;
321
322         Atom[l].vx = 2 * Vx - Atom[l].vx;
323         Atom[l].vy = 2 * Vy - Atom[l].vy;
324       }
325     }
326   }
327 }
```

1.7 Main

```
328 //Main as described in the Assignment
329
330 int main(int argc, const char* argv[])
331 {
332   beginDrawing(W, H, "Atoms", 0xFFFFFF, false);
333   int n = number(argc, argv);
334   Atom* atoms = new Atom[n];
335   init(n, atoms, argc, argv);
336   draw(n, atoms);
337   cout << "Press <ENTER> to continue..." << endl;
338   string s; getline(cin, s);
339   for (int i = 0; i < F; i++)
340   {
341     update(n, atoms);
342     draw(n, atoms);
343     Sleep(S);
344   }
345   delete[] atoms;
346   cout << "Close window to exit..." << endl;
347   endDrawing();
348 }
```

2 Auxiliary

2.1 Auciliary.h

```
349 //*********************************************************
350 //Header "Auxiliary.h"
351 //
352 // declares auxiliary functions for use in the "Atoms" project
353 // for further elaboration of functions, see "Auxiliary.cpp"
354 //
355 // created by Felix Dressler, 04.04.2022
356 //*********************************************************
357 #pragma once
```

```
358
359 //calculates radius r and angle a of a vector using its Cartesian coordinates
360 void toPolar(double x, double y, double& r, double& a);
361
362
363 //calculates the Cartesian coordinates of a vector using its Polar-form
364 void toCartesian(double r, double a, double& x, double& y);
365
366 //creates a random number inbetween two limtis
367 int random(int llimit, int ulimit);
```

2.2 Auxiliary.cpp

```
369 //*********************************************************
370 //File "Auxiliary.cpp"
371 //
372 // defines auxiliary functions for use in the "Atoms" project
373 // that are declared in "Auxiliary.h"
374 //
375 // created by Felix Dressler, 04.04.2022
376 //*********************************************************
377 #include <iostream>
378 #include <cstdlib>
379 #include <cmath>
380
381 #include "Auxiliary.h"
382
383 //*********************************************************
384 // Funtion "toPolar"
385 //
386 //calculates radius rand angle a of a vector using its Cartesian coordinates
387 //*********************************************************
388
389 void toPolar(double x, double y, double& r, double& a)
390 {
391   a = atan2(y, x);
392   r = sqrt(x * x + y * y);
393 }
394
395 //*********************************************************
396 // Funtion "toCaresian"
397 //
398 //calculates the Cartesian coordinates of a vector using its Polar-form
399 //*********************************************************
400
401 void toCartesian(double r, double a, double& x, double& y)
402 {
403   x = r * cos(a);
404   y = r * sin(a);
405 }
406
407 //*********************************************************
408 // Funtion "random"
409 //
410 // This function gives a random number inbetween given limits
411 //
412 // input: two int numbers which define the lower and the upper
413 // limits of the outputted random number
```

```
414 //
415 // output: a random number in between the given limits
416 // including the limits
417 //*******************************************************
418
419 int random(int llimit, int ulimit) {
420
421   return (rand() % (ulimit - llimit + 1)) + llimit;
422 }
```