Programming 2 - Assignment 1

Felix Dreßler (k12105003)

April 5, 2022

1 Programming 1 - First Assignment

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  //Header "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"
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
```

```
40 //*********************************************************
41 // struct "Atom"
42 //
43 // defines the data structure for an Atom
44 // Atoms hold the values:
45 // c ... colour
46 // r ... radius
47 // vx ... velocity in x
48 // vy ... velocity in y
```

```
49 // x ... x-value for position
50 // y ... y-value for position
51 //
52 //*********************************************************
53
54 typedef struct Atom
55 {
56   int c;
57   int r;
58   int vx;
59   int vy;
60   int x;
61   int y;
62 };
```

```
63 //*********************************************************
64 // Funtion "random"
65 //
66 // This function gives a random number inbetween given limits
67 //
68 // input: two int numbers which define the lower and the upper
69 // limits of the outputted random number
70 //
71 // output: a random number in between the given limits
72 // including the limits
73 //*********************************************************
74
75 int random(int llimit, int ulimit) {
76
77   return (rand() % (ulimit - llimit + 1)) + llimit;
78 }
```

```
79 //*********************************************************
80 //  Function "number"
81 //
82 //  creates Atoms with their initial Values as stated above
83 //  N Atoms will be created with random colour, random size and
84 //  random velocity at a random position.
85 //
86 // input:
87 //
88 //
89 // output:
90 //*********************************************************
91
92 double number(int argc, const char* argv[]) {
93   int n = 3;
94
95   if (argc == 2)
96   {
97     ifstream Input{ argv[1] };
98     if (!Input)
99     {
100      cout << "Error:␣check␣Input␣file␣(numbers)" << endl;
101      return -1;
102    }
103
104    Input >> n;
105    Input.close();
```

```
106   }
107
108   cout << "the␣number␣of␣Atoms␣is:␣" << n << endl;
109
110   return n;
111 }
```

```
112 //*********************************************************
113 //Function "init"
114 //
115 // text
116 //
117 // input:
118 //
119 // output:
120 //*********************************************************
121
122 void init(int n, Atom Atom[], int argc, const char* argv[]) {
123
124   if (argc == 2)
125   {
126     ifstream Input{ argv[1] };
127     if (!Input) {
128       cout << "Error:␣check␣Input␣file␣(init)" << endl;
129       return;
130     }
131
132     while (Input)
133     {
134       int n;
135       Input >> n;
136       for (int j = 0; j < n; j++)
137       {
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         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
168
169       bool valid=true;
170
171       for (int l = 0; l <= j; l++) {
172         int m = 0;
173         if (sqrt(pow(Atom[j].x - Atom[l].x, 2) + pow(Atom[j].y - Atom[l].y, 2)) < Atom[j
              ].r + Atom[l].r && j != l && valid)
174         {
175           Atom[j].x = random(Atom[j].r, W - Atom[j].r);
176           Atom[j].y = random(Atom[j].r, H - Atom[j].r);
177
178           m++;
179
180           if (m >= 2) {
181             valid = false;
182           }
183         }
184         else if(!valid){
185           cout << "Error:␣Atoms␣would␣overlap,␣please␣try␣again!" << endl;
186           exit(1);
187         }
188       }
189
190       cout << "Atom␣" << j + 1 << "␣has␣the␣following␣values␣assigned:" << endl;
191       cout << "Color" << j + 1 << "␣is␣␣␣␣␣" << Atom[j].c << endl;
192       cout << "Radius" << j + 1 << "␣is␣␣␣␣" << Atom[j].r << endl;
193       cout << "x␣Pos." << j + 1 << "␣is␣␣␣␣" << Atom[j].x << endl;
194       cout << "y␣Pos." << j + 1 << "␣is␣␣␣␣" << Atom[j].y << endl;
195       cout << "vx" << j + 1 << "␣is␣␣␣␣␣␣␣" << Atom[j].vx << endl;
196       cout << "vy" << j + 1 << "␣is␣␣␣␣␣␣␣" << Atom[j].vy << endl;
197     }
198   }
199   else { cout << "Error:␣Please␣give␣a␣valid␣Argument!"; }
200 }
```

```
201 //*********************************************************
202 // Function "Draw"
203 //
204 // The draw function draws each individual "Frame" of the animation
205 // by first drawing a blank background and then drawing each individual Atom
206 // at its respective position. All of this is updated as one "Frame".
207 //
208 // Input: number of Atoms and values of these Atoms
209 //
210 // Output: none
211 //*********************************************************
212
213 void draw(int n, Atom Atom[]) {
214   fillRectangle(0, 0, W, H, 0xFFFFFF);
215
216   for (int j = 0; j < n; j++) {
217     fillEllipse(Atom[j].x - Atom[j].r, Atom[j].y - Atom[j].r, 2 * Atom[j].r, 2 * Atom[j].
          r, Atom[j].c);
218   }
219
```

```
220   flush();
221 }
```

```
222 //*********************************************************
223 // Funtion "Update"
224 //
225 // The "Update" Function determines the position of every Atom
226 // by calculation their position through their velocities in x and y.
227 // It also handles Atom bouncing from Walls and later also themselves.
228 //
229 // Input:number of Atoms and Values of Atoms
230 //
231 // Output: none
232 //*********************************************************
233
234 void update(int n, Atom Atom[]) {
235
236   double Vx = 0;//maybe in for() deklarieren?
237   double Vy = 0;
238
239   for (int j = 0; j < n; j++) {
240
241     Atom[j].x += Atom[j].vx;
242     Atom[j].y += Atom[j].vy;
243
244     //checks for collisions between atoms and walls
245
246     if (Atom[j].x >= W - Atom[j].r)
247     {
248       Atom[j].vx = -Atom[j].vx;
249       Atom[j].x = W - Atom[j].r;
250     }
251     if (Atom[j].x <= Atom[j].r)
252     {
253       Atom[j].vx = -Atom[j].vx;
254       Atom[j].x = Atom[j].r;
255     }
256     if (Atom[j].y >= H - Atom[j].r)
257     {
258       Atom[j].vy = -Atom[j].vy;
259       Atom[j].y = H - Atom[j].r;
260     }
261     if (Atom[j].y <= Atom[j].r)
262     {
263       Atom[j].vy = -Atom[j].vy;
264       Atom[j].y = Atom[j].r;
265     }
266
267     //checks for collisions between different atoms
268     for (int l = 0; l <= j; l++) {
269
270       int dx = Atom[j].x - Atom[l].x;
271       int dy = Atom[j].y - Atom[l].y;
272       int rsum = Atom[j].r + Atom[l].r;
273
274       if (sqrt(pow(dx,2)+ pow(dy,2)) <= rsum && j != l)
275       {
276
277         double alpha = atan2(dy,dx);
```

```
278         int dx1 = cos(alpha) * rsum;
279         int dy1 = sin(alpha) * rsum;
280 
281         Atom[j].x += dx1 - dx;
282         Atom[j].y += dy1 - dy;
283 
284         double beta = 3.1415926 - alpha;
285 
286         double a;
287         double r;
288         double vx1;
289         double vy1;
290 
291         toPolar(Atom[j].vx, Atom[j].vy, r, a);
292         a - beta;
293         toCartesian(r, a, vx1, vy1);
294 
295         Atom[j].vx = vx1;
296         Atom[j].vy = vy1;
297 
298         toPolar(Atom[l].vx, Atom[l].vy, r, a);
299         a - beta;
300         toCartesian(r, a, vx1, vy1);
301 
302         Atom[l].vx = vx1;
303         Atom[l].vy = vy1;
304 
305         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));
306         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));
307 
308         Atom[j].vx = 2 * Vx - Atom[j].vx;
309         Atom[j].vy = 2 * Vy - Atom[j].vy;
310 
311         Atom[l].vx = 2 * Vx - Atom[l].vx;
312         Atom[l].vy = 2 * Vy - Atom[l].vy;
313       }
314     }
315   }
316 }
```

```
317 //Main as described in the Assignment
318 //further elaboration needed?
319 
320 int main(int argc, const char* argv[])
321 {
322   beginDrawing(W, H, "Atoms", 0xFFFFFF, false);
323   int n = number(argc, argv);
324   Atom* atoms = new Atom[n];
325   init(n, atoms, argc, argv);
326   draw(n, atoms);
327   cout << "Press␣<ENTER>␣to␣continue..." << endl;
328   string s; getline(cin, s);
329   for (int i = 0; i < F; i++)
330   {
331     update(n, atoms);
332     draw(n, atoms);
333     Sleep(S);
```

```
334   }
335   delete[] atoms;
336   cout << "Close window to exit..." << endl;
337   endDrawing();
338 }
```