Programming 2 - Assignment 1

Felix Dreßler (k12105003)

April 5, 2022

1 Programming 1 - First Assignment

```
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
39
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  //*********************************************************
65  // Funtion "random"
66  //
67  // This function gives a random number inbetween given limits
68  //
69  // input: two int numbers which define the lower and the upper
70  // limits of the outputted random number
71  //
72  // output: a random number in between the given limits
73  // including the limits
74  //*********************************************************
75
76  int random(int llimit, int ulimit) {
77
78    return (rand() % (ulimit - llimit + 1)) + llimit;
79  }
80
81  //*********************************************************
82  //  Function "number"
83  //
84  //  creates Atoms with their initial Values as stated above
85  //  N Atoms will be created with random colour, random size and
86  //  random velocity at a random position.
87  //
88  // input:
89  //
90  //
91  // output:
92  //*********************************************************
93
94  double number(int argc, const char* argv[]) {
95    int n = 3;
96
97    if (argc == 2)
98    {
99      ifstream Input{ argv[1] };
100     if (!Input)
101     {
102       cout << "Error: check Input file (numbers)" << endl;
103       return -1;
104     }
105
106     Input >> n;
107     Input.close();
108   }
109
110   cout << "the number of Atoms is: " << n << endl;
111
112   return n;
113 }
114
115 //*********************************************************
116 //Function "init"
117 //
```

```
118 // text
119 //
120 // input:
121 //
122 // output:
123 //*********************************************************
124
125 void init(int n, Atom Atom[], int argc, const char* argv[]) {
126
127   if (argc == 2)
128   {
129     ifstream Input{ argv[1] };
130     if (!Input) {
131       cout << "Error:␣check␣Input␣file␣(init)" << endl;
132       return;
133     }
134
135     while (Input)
136     {
137       int n;
138       Input >> n;
139       for (int j = 0; j < n; j++)
140       {
141         Input >> Atom[j].c;
142         Input >> Atom[j].r;
143         Input >> Atom[j].x;
144         Input >> Atom[j].y;
145         Input >> Atom[j].vx;
146         Input >> Atom[j].vy;
147
148         cout << "Atom␣" << j + 1 << "␣has␣the␣following␣values␣assigned:" << endl;
149         cout << "Color" << j + 1 << "␣is␣␣␣␣␣" << Atom[j].c << endl;
150         cout << "Radius" << j + 1 << "␣is␣␣␣␣" << Atom[j].r << endl;
151         cout << "x␣Pos." << j + 1 << "␣is␣␣␣␣" << Atom[j].x << endl;
152         cout << "y␣Pos." << j + 1 << "␣is␣␣␣␣" << Atom[j].y << endl;
153         cout << "vx" << j + 1 << "␣is␣␣␣␣␣␣␣" << Atom[j].vx << endl;
154         cout << "vy" << j + 1 << "␣is␣␣␣␣␣␣␣" << Atom[j].vy << endl;
155       }
156     }
157
158     Input.close();
159   }
160   else if(argc == 1) {
161     srand(time(0));
162     for (int j = 0; j < n; j++) {
163       Atom[j].c = random(000, 0xFFFFFF);
164       Atom[j].r = random(20, 40);
165       Atom[j].vx = random(5, 25);
166       Atom[j].vy = random(5, 25);
167       Atom[j].x = random(Atom[j].r, W - Atom[j].r);
168       Atom[j].y = random(Atom[j].r, H - Atom[j].r);
169
170       //the following function should check, if Atoms were to overlap
171
172       bool valid=true;
173
174       for (int l = 0; l <= j; l++) {
175         int m = 0;
176         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)
```

```
177         {
178           Atom[j].x = random(Atom[j].r, W - Atom[j].r);
179           Atom[j].y = random(Atom[j].r, H - Atom[j].r);
180 
181           m++;
182 
183           if (m >= 2) {
184             valid = false;
185           }
186         }
187         else if(!valid){
188           cout << "Error:␣Atoms␣would␣overlap,␣please␣try␣again!" << endl;
189           exit(1);
190         }
191       }
192 
193       cout << "Atom␣" << j + 1 << "␣has␣the␣following␣values␣assigned:" << endl;
194       cout << "Color" << j + 1 << "␣is␣␣␣␣␣" << Atom[j].c << endl;
195       cout << "Radius" << j + 1 << "␣is␣␣␣␣" << Atom[j].r << endl;
196       cout << "x␣Pos." << j + 1 << "␣is␣␣␣␣" << Atom[j].x << endl;
197       cout << "y␣Pos." << j + 1 << "␣is␣␣␣␣" << Atom[j].y << endl;
198       cout << "vx" << j + 1 << "␣is␣␣␣␣␣␣␣" << Atom[j].vx << endl;
199       cout << "vy" << j + 1 << "␣is␣␣␣␣␣␣␣" << Atom[j].vy << endl;
200     }
201   }
202   else { cout << "Error:␣Please␣give␣a␣valid␣Argument!"; }
203 }
204 
205 //*********************************************************
206 // Function "Draw"
207 //
208 // The draw function draws each individual "Frame" of the animation
209 // by first drawing a blank background and then drawing each individual Atom
210 // at its respective position. All of this is updated as one "Frame".
211 //
212 // Input: number of Atoms and values of these Atoms
213 //
214 // Output: none
215 //*********************************************************
216 
217 void draw(int n, Atom Atom[]) {
218   fillRectangle(0, 0, W, H, 0xFFFFFF);
219 
220   for (int j = 0; j < n; j++) {
221     fillEllipse(Atom[j].x - Atom[j].r, Atom[j].y - Atom[j].r, 2 * Atom[j].r, 2 * Atom[j].
          r, Atom[j].c);
222   }
223 
224   flush();
225 }
226 
227 //*********************************************************
228 // Funtion "Update"
229 //
230 // The "Update" Function determines the position of every Atom
231 // by calculation their position through their velocities in x and y.
232 // It also handles Atom bouncing from Walls and later also themselves.
233 //
234 // Input:number of Atoms and Values of Atoms
235 //
```

```
236 // Output: none
237 //*********************************************************
238
239 void update(int n, Atom Atom[]) {
240
241   double Vx = 0;//maybe in for() deklarieren?
242   double Vy = 0;
243
244   for (int j = 0; j < n; j++) {
245
246     Atom[j].x += Atom[j].vx;
247     Atom[j].y += Atom[j].vy;
248
249     //checks for collisions between atoms and walls
250
251     if (Atom[j].x >= W - Atom[j].r)
252     {
253       Atom[j].vx = -Atom[j].vx;
254       Atom[j].x = W - Atom[j].r;
255     }
256     if (Atom[j].x <= Atom[j].r)
257     {
258       Atom[j].vx = -Atom[j].vx;
259       Atom[j].x = Atom[j].r;
260     }
261     if (Atom[j].y >= H - Atom[j].r)
262     {
263       Atom[j].vy = -Atom[j].vy;
264       Atom[j].y = H - Atom[j].r;
265     }
266     if (Atom[j].y <= Atom[j].r)
267     {
268       Atom[j].vy = -Atom[j].vy;
269       Atom[j].y = Atom[j].r;
270     }
271
272     //checks for collisions between different atoms
273     for (int l = 0; l <= j; l++) {
274
275       int dx = Atom[j].x - Atom[l].x;
276       int dy = Atom[j].y - Atom[l].y;
277       int rsum = Atom[j].r + Atom[l].r;
278
279       if (sqrt(pow(dx,2)+ pow(dy,2)) <= rsum && j != l)
280       {
281
282         double alpha = atan2(dy,dx);
283         int dx1 = cos(alpha) * rsum;
284         int dy1 = sin(alpha) * rsum;
285
286         Atom[j].x += dx1 - dx;
287         Atom[j].y += dy1 - dy;
288
289         double beta = 3.1415926 - alpha;
290
291         double a;
292         double r;
293         double vx1;
294         double vy1;
295
```

```
296         toPolar(Atom[j].vx, Atom[j].vy, r, a);
297         a - beta;
298         toCartesian(r, a, vx1, vy1);
299 
300         Atom[j].vx = vx1;
301         Atom[j].vy = vy1;
302 
303         toPolar(Atom[l].vx, Atom[l].vy, r, a);
304         a - beta;
305         toCartesian(r, a, vx1, vy1);
306 
307         Atom[l].vx = vx1;
308         Atom[l].vy = vy1;
309 
310         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));
311         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));
312 
313         Atom[j].vx = 2 * Vx - Atom[j].vx;
314         Atom[j].vy = 2 * Vy - Atom[j].vy;
315 
316         Atom[l].vx = 2 * Vx - Atom[l].vx;
317         Atom[l].vy = 2 * Vy - Atom[l].vy;
318       }
319     }
320   }
321 }
322 
323 //Main as described in the Assignment
324 //further elaboration needed?
325 
326 int main(int argc, const char* argv[])
327 {
328   beginDrawing(W, H, "Atoms", 0xFFFFFF, false);
329   int n = number(argc, argv);
330   Atom* atoms = new Atom[n];
331   init(n, atoms, argc, argv);
332   draw(n, atoms);
333   cout << "Press␣<ENTER>␣to␣continue..." << endl;
334   string s; getline(cin, s);
335   for (int i = 0; i < F; i++)
336   {
337     update(n, atoms);
338     draw(n, atoms);
339     Sleep(S);
340   }
341   delete[] atoms;
342   cout << "Close␣window␣to␣exit..." << endl;
343   endDrawing();
344 }
```