Programming 2 - Assignment 1

Felix Dreßler (k12105003)

April 4, 2022

1 Programming 1 - First Assignment

```
1  //*********************************************************
2  //
3  //*********************************************************
4
5
6  #include <iostream>
7  #include <cstdlib>
8  #include <cmath>
9
10 #include "Drawing.h"
11
12 #if defined(_WIN32) || defined (_WIN64)
13 #include <windows.h>
14 #else
15 #include <time.h>
16 static void Sleep(int ms)
17 {
18   struct timespec ts;
19   ts.tv_sec = ms / 1000;
20   ts.tv_nsec = (ms % 1000) * 1000000;
21   nanosleep(&ts, NULL);
22 }
23 #endif
24
25 using namespace std;
26 using namespace compsys;
27
28 #include  <string>  //defines the getline() function
29 #include <fstream>
30 #include "Auxiliary.h"
31
32 int W = 640;  //W,H are the width and the height of the created window
33 int H = 480;
34
35 int S = 40;   //time between the frame-updates - sleep
36 int F = 200;  //number of updates that are performed by the program
37
38 //*********************************************************
39 // struct "Atom"
40 //
41 // defines the data structure for an Atom
42 // Atoms hold the values:
43 // c ... colour
44 // r ... radius
45 // vx ... velocity in x
46 // vy ... velocity in y
47 // x ... x-value for position
48 // y ... y-value for position
49 //
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  };
61
62  //*********************************************************
63  // Funtion "random"
64  //
65  // input: two int numbers which define the lower and the upper
66  // limits of the outputted random number
67  //
68  // output: a random number in between the given limits
69  // including the limits
70  //
71  //*********************************************************
72
73  int random(int llimit, int ulimit) {
74
75    return (rand() % (ulimit - llimit + 1)) + llimit;
76  }
77
78  //*********************************************************
79  //  Function "number"
80  //
81  //  creates Atoms with their initial Values as stated above
82  //  N Atoms will be created with random colour, random size and
83  //  random velocity at a random position.
84  //
85  //*********************************************************
86
87  double number(int argc, const char* argv[]) {
88    int n = 3;
89
90    if (argc == 2)
91    {
92      ifstream Input{ argv[1] };
93      if (!Input)
94      {
95        cout << "Error:␣check␣Input␣file␣(numbers)" << endl;
96        return -1;
97      }
98
99      Input >> n;
100     Input.close();
101   }
102
103   cout << "the␣number␣of␣Atoms␣is:␣" << n << endl;
104
105   return n;
106 }
107
108 //*********************************************************
109 //
110 //*********************************************************
111
112 void init(int n, Atom Atom[], int argc, const char* argv[]) {
113
114   if (argc == 2)
115   {
116     ifstream Input{ argv[1] };
117     if (!Input) {
```

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

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

```
236     //checks for collisions between atoms and walls
237
238     if (Atom[j].x >= W - Atom[j].r)
239     {
240       Atom[j].vx = -Atom[j].vx;
241       Atom[j].x = W - Atom[j].r;
242     }
243     if (Atom[j].x <= Atom[j].r)
244     {
245       Atom[j].vx = -Atom[j].vx;
246       Atom[j].x = Atom[j].r;
247     }
248     if (Atom[j].y >= H - Atom[j].r)
249     {
250       Atom[j].vy = -Atom[j].vy;
251       Atom[j].y = H - Atom[j].r;
252     }
253     if (Atom[j].y <= Atom[j].r)
254     {
255       Atom[j].vy = -Atom[j].vy;
256       Atom[j].y = Atom[j].r;
257     }
258
259     //checks for collisions between different atoms
260     for (int l = 0; l <= j; l++) {
261
262       int dx = Atom[j].x - Atom[l].x;
263       int dy = Atom[j].y - Atom[l].y;
264       int rsum = Atom[j].r + Atom[l].r;
265
266       if (sqrt(pow(dx,2)+ pow(dy,2)) <= rsum && j != l)
267       {
268
269         double alpha = atan2(dy,dx);
270         int dx1 = cos(alpha) * rsum;
271         int dy1 = sin(alpha) * rsum;
272
273         Atom[j].x += dx1 - dx;
274         Atom[j].y += dy1 - dy;
275
276         double beta = 3.1415926 - alpha;
277
278         double a;
279         double r;
280         double vx1;
281         double vy1;
282
283         toPolar(Atom[j].vx, Atom[j].vy, r, a);
284         a - beta;
285         toCartesian(r, a, vx1, vy1);
286
287         Atom[j].vx = vx1;
288         Atom[j].vy = vy1;
289
290         toPolar(Atom[l].vx, Atom[l].vy, r, a);
291         a - beta;
292         toCartesian(r, a, vx1, vy1);
293
294         Atom[l].vx = vx1;
295         Atom[l].vy = vy1;
```

```
296
297         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));
298         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));
299
300         Atom[j].vx = 2 * Vx - Atom[j].vx;
301         Atom[j].vy = 2 * Vy - Atom[j].vy;
302
303         Atom[l].vx = 2 * Vx - Atom[l].vx;
304         Atom[l].vy = 2 * Vy - Atom[l].vy;
305
306         cout << "Kollision" << endl;//for debugging
307       }
308     }
309   }
310 }
311
312
313 int main(int argc, const char* argv[])
314 {
315   beginDrawing(W, H, "Atoms", 0xFFFFFF, false);
316   int n = number(argc, argv);
317   Atom* atoms = new Atom[n];
318   init(n, atoms, argc, argv);
319   draw(n, atoms);
320   cout << "Press␣<ENTER>␣to␣continue..." << endl;
321   string s; getline(cin, s);
322   for (int i = 0; i < F; i++)
323   {
324     update(n, atoms);
325     draw(n, atoms);
326     Sleep(S);
327   }
328   delete[] atoms;
329   cout << "Close␣window␣to␣exit..." << endl;
330   endDrawing();
331 }
```