Programming 2 - Assignment 1

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

April 7, 2022

1 Testing the Program

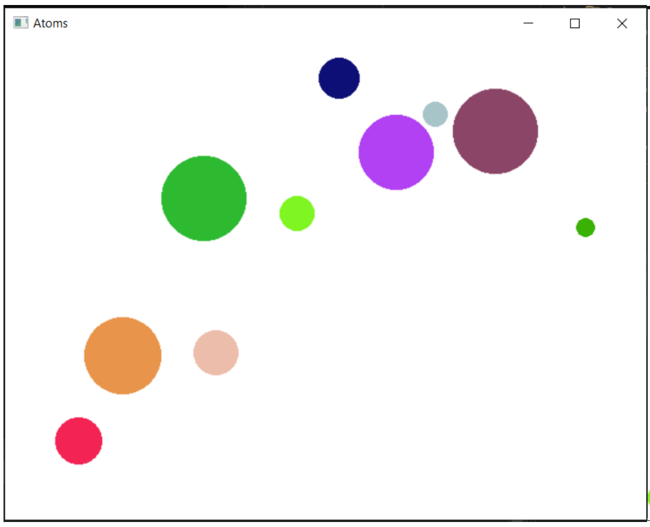
For evaluation purposes a series of tests were performed. The program was executed first without the implementation of collisions between different atoms for an fixed input saved in "Input.txt" and random generated atoms. Next the program was tested including the implementation of collision between different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

For the file "Input.txt" the exact values will not be stated in this documentation, because this file was part of the assignment. However the .txt file should be included in the submissions .zip file.

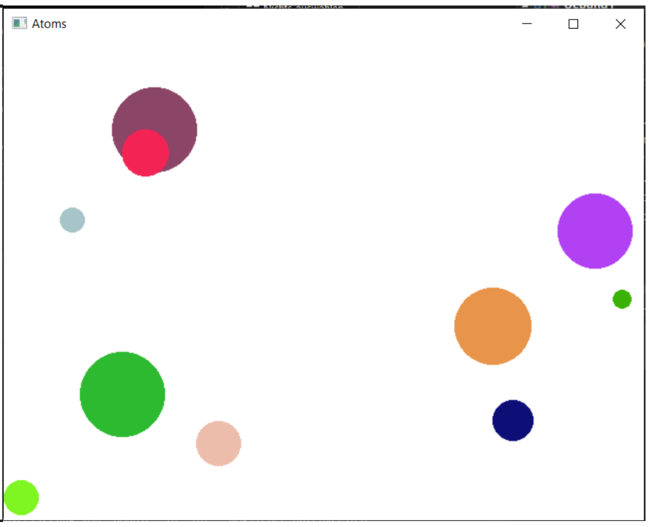
For each test, the text output of the program was recorded, stating the initial values of the atoms. Furthermore, screenshots of the initial-state and the end-state are included.

1.1 Tests without atom-collison

1.1.1 Input.txt



(a) start state

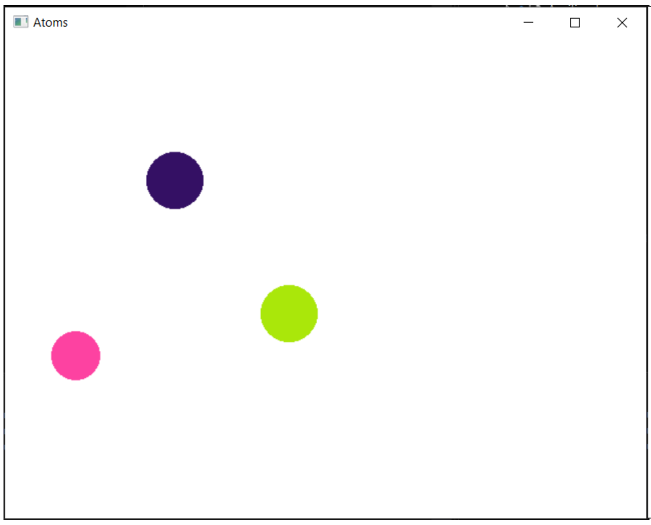


(b) end state

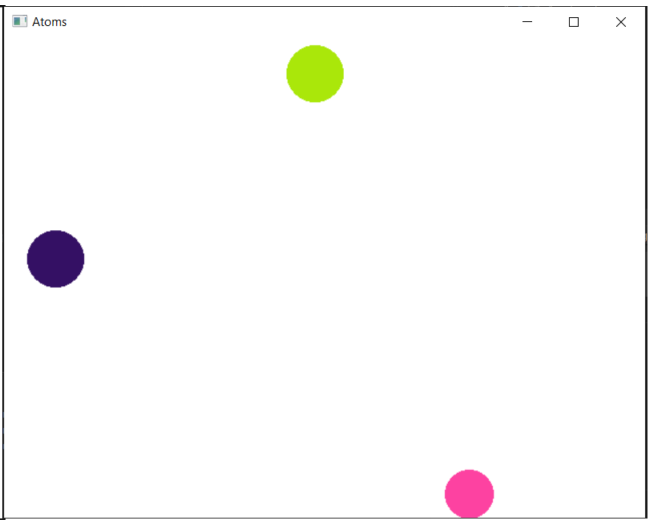
1.1.2 randomly created atoms

```
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
```



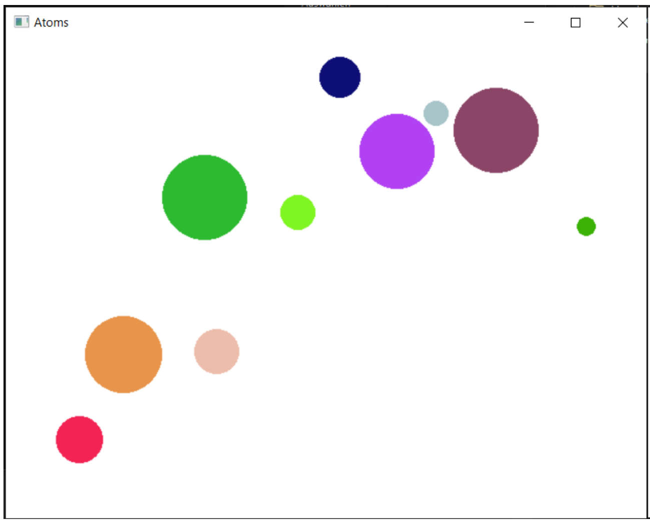
(a) start state



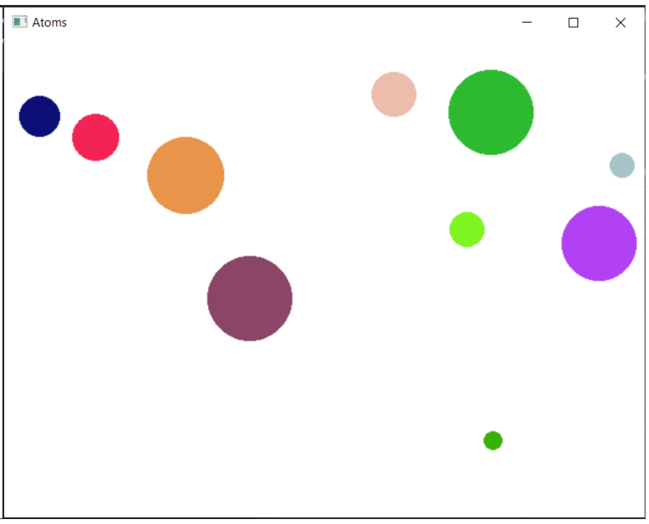
(b) end state

1.2 Tests with atom-collsion

1.2.1 Input.txt



(a) start state

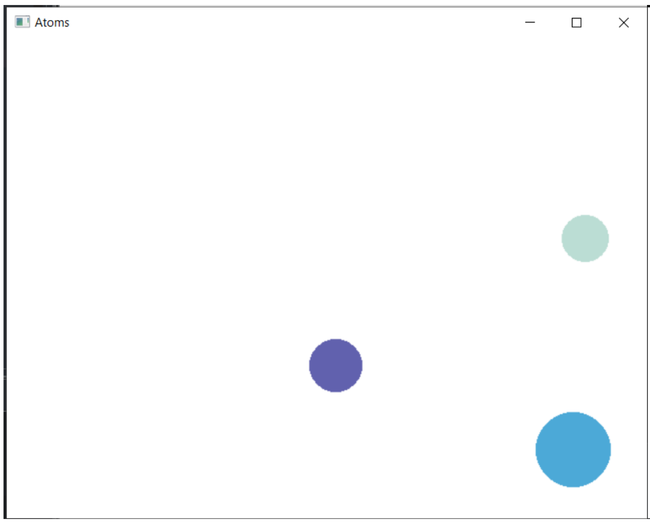


(b) end state

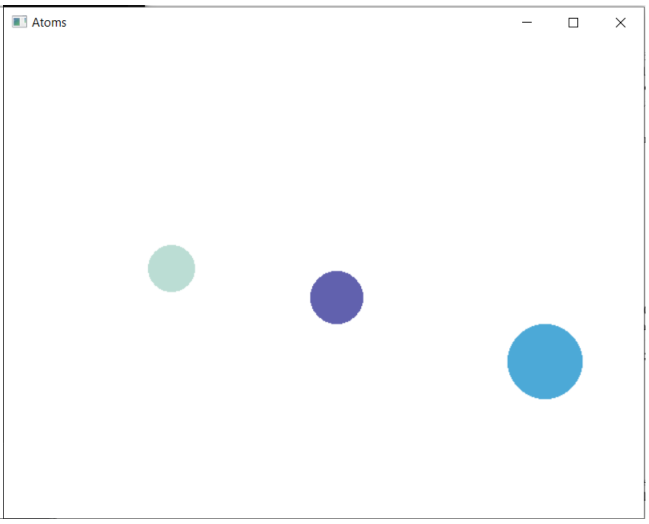
1.2.2 randomly created atoms

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
```

```

using namespace std;
using namespace compsys;

#include  <string>  //defines the getline() function
#include <fstream>
#include "Auxiliary.h" //includes auxiliary functions such as "toPolar", "random", ...

int W = 640;  //W,H are the width and the height of the created window
int H = 480;

int S = 40;   //time between the frame-updates - sleep
int F = 200;  //number of updates that are performed by the program
```

2.2 structure "Atom"

```
//*********************************************************
// struct "Atom"
//
// defines the data structure for an Atom
// Atoms hold the values:
// c ... colour
// r ... radius
// vx ... velocity in x
// vy ... velocity in y
// x ... x-value for position
// y ... y-value for position
//*********************************************************

typedef struct Atom
{
  int c;
  double r;
  double vx;
  double vy;
  double x;
  double y;
};
```

2.3 Function "number"

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

```
//
// output:
// n ... number of atoms that shoudl be created
//*******************************************************

double number(int argc, const char* argv[]) {
  int N = 3;

  if (argc == 2)
  {
    ifstream Input{ argv[1] };
    if (!Input)
    {
      cout << "Error: check Input file (numbers)" << endl;
      return -1;
    }

    Input >> N;
    Input.close();
  }

  cout << "the number of Atoms is: " << N << endl;

  return N;
}
```

2.4 Function "init"

```
//*******************************************************
//Function "init"
//
// This function initializes n Atoms with their respective values.
// If an input file was given, it uses the values that are stated there.
// If no input file was given, it defines the values of the atoms at random
// within a given Interval.
//
// input:
// n ... number of atoms that should be created
// Atom[] ... gives an array of Atoms (struct defined above)
// argc ... number of arguments given when programm is called
// argv[] ... argument that was given when programm is called
// (should be the directory to a .txt file that holds start values
// for the simulation.
//
// output: none
//*******************************************************

void init(int n, Atom Atom[], int argc, const char* argv[]) {
  if (argc == 2)
  {
    ifstream Input{ argv[1] };
    if (!Input) {
      cout << "Error: check Input file (init)" << endl;
      return;
    }

    while (Input)
    {
      int n;  //saves the first number in the input document to be able to
          //access the other numbers, this variable will not be used
      Input >> n;
      for (int j = 0; j < n; j++)
      {
        //we assume that the user only gives us valid placements
        //i.e. the atoms do not overlap
        Input >> Atom[j].c;
        Input >> Atom[j].r;
        Input >> Atom[j].x;
        Input >> Atom[j].y;
        Input >> Atom[j].vx;
        Input >> Atom[j].vy;

        //gives ut the values of each atom
        cout << "Atom " << j + 1 << " has the following values assigned:" << endl;
        cout << "Color" << j + 1 << " is     " << Atom[j].c << endl;
        cout << "Radius" << j + 1 << " is    " << Atom[j].r << endl;
        cout << "x Pos." << j + 1 << " is    " << Atom[j].x << endl;
        cout << "y Pos." << j + 1 << " is    " << Atom[j].y << endl;
        cout << "vx" << j + 1 << " is         " << Atom[j].vx << endl;
        cout << "vy" << j + 1 << " is         " << Atom[j].vy << endl;
      }
    }

    Input.close();
  }
  else if(argc == 1) {
    srand(time(0));
    for (int j = 0; j < n; j++) {
      Atom[j].c = random(000, 0xFFFFFF);
      Atom[j].r = random(20, 40);
      Atom[j].vx = random(5, 25);
      Atom[j].vy = random(5, 25);
      Atom[j].x = random(Atom[j].r, W - Atom[j].r);
      Atom[j].y = random(Atom[j].r, H - Atom[j].r);

      //the following function should check, if Atoms were to overlap if they overlap,
      //it tries three times to create a new one, if it fails on the third time, it exits
      bool valid=true;

      for (int l = 0; l < j; l++) {
        int m = 0;

        double dx = Atom[j].x - Atom[l].x;
        //difference between the x-coordinates of the two compared atoms
        double dy = Atom[j].y - Atom[l].y;
        //difference between the y-coordinates of the two compared atoms
        double rsum = Atom[j].r + Atom[l].r; //sum of the radi of the two atoms compared

        for (int k=0; dx * dx + dy * dy < rsum*rsum && valid && k<=m; k++)
        {
          Atom[j].x = random(Atom[j].r, W - Atom[j].r);
          Atom[j].y = random(Atom[j].r, H - Atom[j].r);

          m++;

          if (m > 2) {
            valid = false;
          }
        }
        if(!valid){
          cout << "Error: Atoms would overlap, please try again!" << endl;
          exit(1);
        }
      }

      cout << "Atom " << j + 1 << " has the following values assigned:" << endl;
      cout << "Color" << j + 1 << " is      " << Atom[j].c << endl;
      cout << "Radius" << j + 1 << " is     " << Atom[j].r << endl;
      cout << "x Pos." << j + 1 << " is     " << Atom[j].x << endl;
      cout << "y Pos." << j + 1 << " is     " << Atom[j].y << endl;
      cout << "vx" << j + 1 << " is         " << Atom[j].vx << endl;
      cout << "vy" << j + 1 << " is         " << Atom[j].vy << endl;
    }
  }
  else { cout << "Error: Please give a valid Argument!"; }
}
```

2.5 Function "Draw"

```
//**********************************************************
// Function "Draw"
//
// The draw function draws each individual "Frame" of the animation
// by first drawing a blank background and then drawing each individual Atom
// at its respective position. All of this is updated as one "Frame".
//
// input: number of Atoms and values of these Atoms
//
// output: none
//**********************************************************

void draw(int n, Atom Atom[]) {
  fillRectangle(0, 0, W, H, 0xFFFFFF);

  for (int j = 0; j < n; j++) {
    fillEllipse(Atom[j].x - Atom[j].r, Atom[j].y - Atom[j].r,
      2 * Atom[j].r, 2 * Atom[j].r, Atom[j].c);
  }

  flush();
}
```

2.6 Function "Update"

```
//*******************************************************
// Funtion "Update"
//
// The "Update" Function determines the position of every Atom
// by calculation their position through their velocities in x and y.
// It also handles collisions between different atoms and between
// atoms and walls.
//
// It first checks if an atom was to collide with a wall, if yes it
// changes its velocity accordingly.
// Next it checks if this atom was to collide with any of the other
// atoms, if yes it changes their velocities accordingly.
//
// Input:number of Atoms and Values of Atoms
//
// Output: none
//*******************************************************

void update(int n, Atom Atom[]) {
  for (int j = 0; j < n; j++) {

    Atom[j].x += Atom[j].vx;
    Atom[j].y += Atom[j].vy;

    //checks for collisions between atoms and walls

    if (Atom[j].x >= W - Atom[j].r)
    {
      Atom[j].vx = -Atom[j].vx;
      Atom[j].x = W - Atom[j].r;
    }
    if (Atom[j].x <= Atom[j].r)
    {
      Atom[j].vx = -Atom[j].vx;
      Atom[j].x = Atom[j].r;
    }
    if (Atom[j].y >= H - Atom[j].r)
    {
      Atom[j].vy = -Atom[j].vy;
      Atom[j].y = H - Atom[j].r;
    }
    if (Atom[j].y <= Atom[j].r)
    {
      Atom[j].vy = -Atom[j].vy;
      Atom[j].y = Atom[j].r;
    }

    //checks for collisions between different atoms
    for (int l = 0; l <= j; l++) {

      double dx = Atom[j].x - Atom[l].x;
      //difference between the x-coordinates of the two compared atoms
      double dy = Atom[j].y - Atom[l].y;
      //difference between the y-coordinates of the two compared atoms
      double rsum = Atom[j].r + Atom[l].r; //sum of the radi of the two atoms compared

      if (dx*dx + dy*dy <= rsum*rsum && j != l)
      {
        double alpha = atan2(dy,dx);
        double dx1 = cos(alpha) * rsum;
        double dy1 = sin(alpha) * rsum;

        Atom[j].x += dx1 - dx;
        Atom[j].y += dy1 - dy;

        double beta = 3.1415926 - alpha;

        double Vx = 0;
        double Vy = 0;

        double a;//angle of a vector as outputted from "toPolar"
        double r;//radius of a vector as outputted from "toPolar"
        double vx1;//new velocity in x after rotation and transformation by "toCartesian"
        double vy1;//new velocity in x after rotation and transformation by "toCartesian"

        toPolar(Atom[j].vx, Atom[j].vy, r, a);
        a - beta;
        toCartesian(r, a, vx1, vy1);

        Atom[j].vx = vx1;
        Atom[j].vy = vy1;

        toPolar(Atom[l].vx, Atom[l].vy, r, a);
        a - beta;
        toCartesian(r, a, vx1, vy1);

        Atom[l].vx = vx1;
        Atom[l].vy = vy1;

        //Vx and Vy as describes in the theory of elastic impact
        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));
        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));

        Atom[j].vx = 2 * Vx - Atom[j].vx;
        Atom[j].vy = 2 * Vy - Atom[j].vy;

        Atom[l].vx = 2 * Vx - Atom[l].vx;
        Atom[l].vy = 2 * Vy - Atom[l].vy;
      }
    }
  }
}
```

2.7 Main

```
//Main as described in the Assignment

int main(int argc, const char* argv[])
{
  beginDrawing(W, H, "Atoms", 0xFFFFFF, false);
  int n = number(argc, argv);
  Atom* atoms = new Atom[n];
  init(n, atoms, argc, argv);
  draw(n, atoms);
  cout << "Press <ENTER> to continue..." << endl;
  string s; getline(cin, s);
  for (int i = 0; i < F; i++)
  {
    update(n, atoms);
    draw(n, atoms);
    Sleep(S);
  }
  delete[] atoms;
  cout << "Close window to exit..." << endl;
  endDrawing();
}
```

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.

```
//*******************************************************
//Header "Auxiliary.h"
//
// declares auxiliary functions for use in the "Atoms" project
// for further elaboration of functions, see "Auxiliary.cpp"
//
// created by Felix Dressler, 04.04.2022
//*******************************************************
#pragma once

//calculates radius r and angle a of a vector using its Cartesian coordinates
void toPolar(double x, double y, double& r, double& a);


//calculates the Cartesian coordinates of a vector using its Polar-form
void toCartesian(double r, double a, double& x, double& y);

//creates a random number inbetween two limtis
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.

```
//*******************************************************
//File "Auxiliary.cpp"
//
// defines auxiliary functions for use in the "Atoms" project
// that are declared in "Auxiliary.h"
//
// created by Felix Dressler, 04.04.2022
//*******************************************************
#include <iostream>
#include <cstdlib>
#include <cmath>

#include "Auxiliary.h"

//*******************************************************
// Funtion "toPolar"
//
//calculates radius rand angle a of a vector using its Cartesian coordinates
//*******************************************************
```

```

void toPolar(double x, double y, double& r, double& a)
{
  a = atan2(y, x);
  r = sqrt(x * x + y * y);
}

//*******************************************************
// Funtion "toCaresian"
//
//calculates the Cartesian coordinates of a vector using its Polar-form
//*******************************************************

void toCartesian(double r, double a, double& x, double& y)
{
  x = r * cos(a);
  y = r * sin(a);
}

//*******************************************************
// Funtion "random"
//
// This function gives a random number inbetween given limits
//
// input: two int numbers which define the lower and the upper
// limits of the outputted random number
//
// output: a random number in between the given limits
// including the limits
//*******************************************************

int random(int llimit, int ulimit) {

  return ((rand()+rand()*(RAND_MAX+1)) % (ulimit - llimit + 1)) + llimit;
}
```