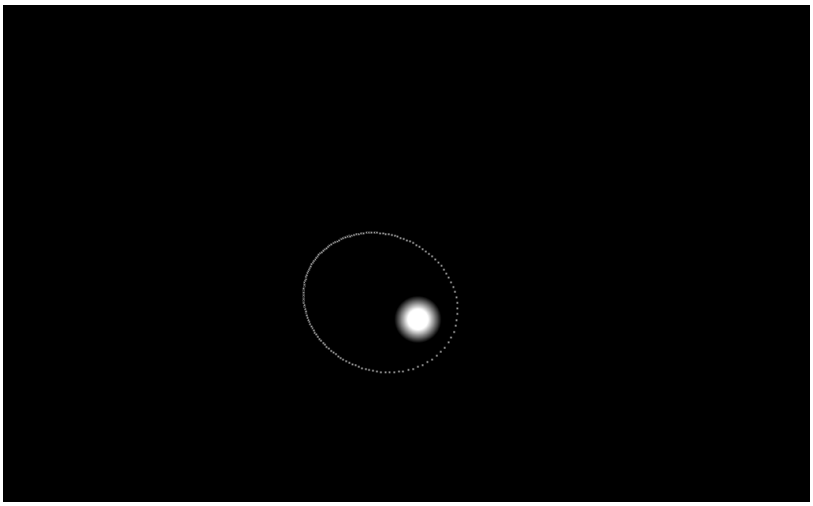
My Program: https://github.com/JackBai0914/StarSystemProgram/blob/f285cbc21096300aeda909cb0b6e834f2c43b8c5/Desktop/Solar_Binary3.0.exe

Report on the Simulation of Star Systems in C++ Program

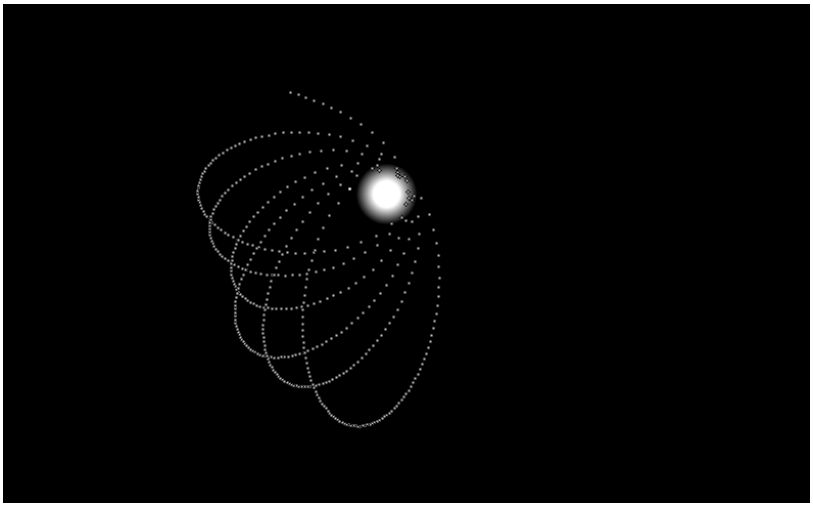
Last August, I became interested in the trajectory of the stars. I was curious as to why the trajectory of the planet is approximately elliptical, and I also wanted to explore the law of the binary star systems. Plus, I had just learned graphic programming, so I developed my own idea of simulating the movement of celestial bodies.

Part I: Single Star System

My program was very simple at the beginning. The formula for a = g = C * m(sun) / (d^2) is used to calculate the trajectory of the planet's rotation around the sun. C is the appropriate constant I set, and d is the distance between the planet and the star. Obviously, this equation is simplified by the universal gravitational formula. In detail, the initial position and initial velocity of the planet are randomly generated by the program, and the planet's acceleration is calculated at a certain time unit.



(Figure 1)

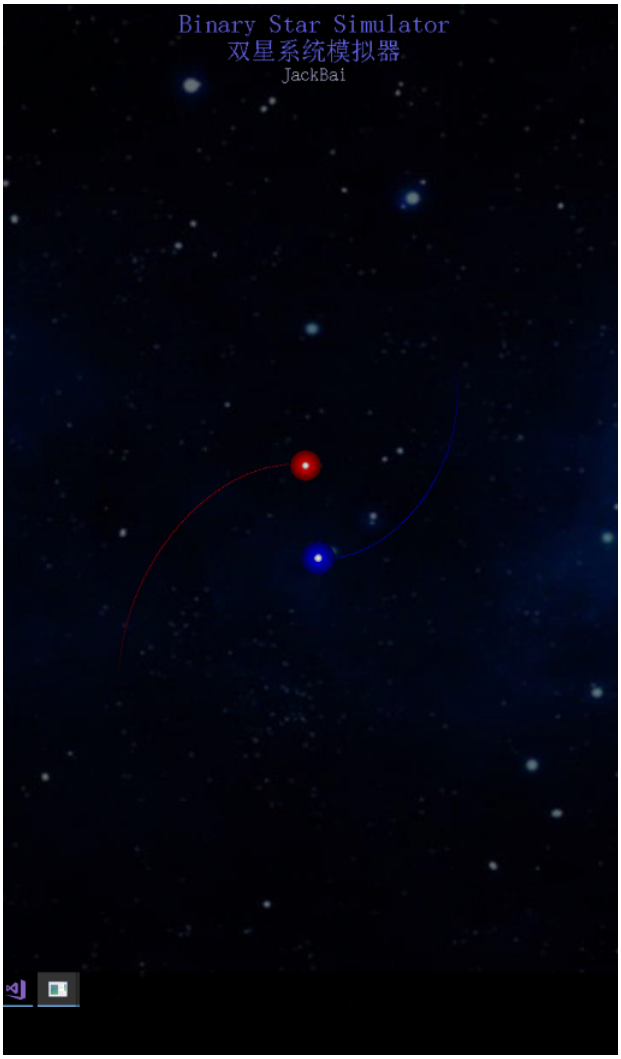


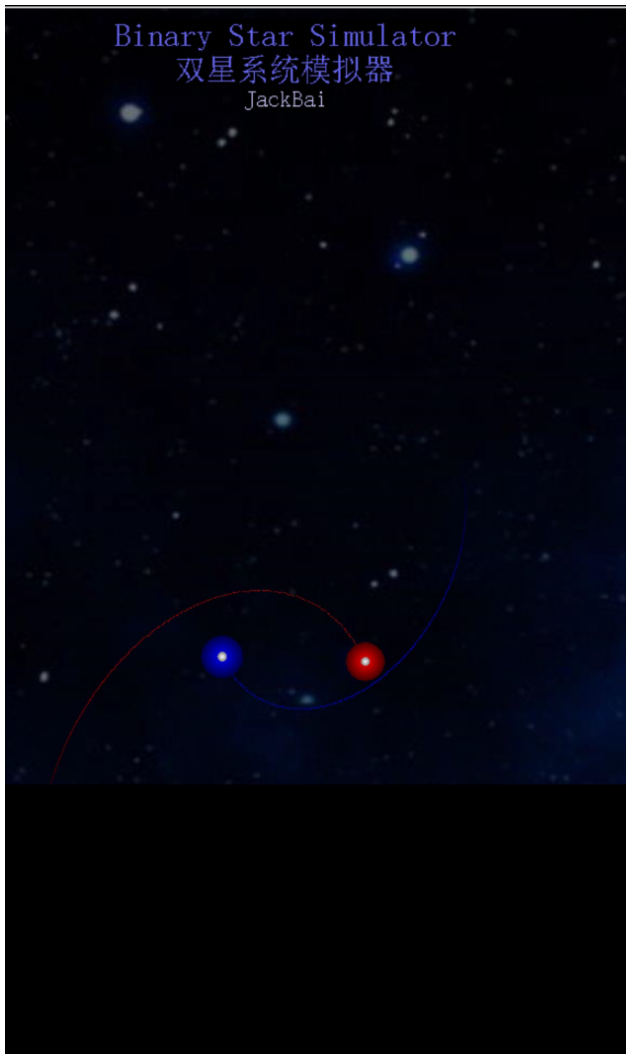
(Figure 2)

During the simulation, I found that most of the planet's trajectory is a stable ellipse (as shown in Figure 1). Observing the motion of these planets and their trails, I easily get The Elliptical Law and Kepler's Second Law of Planetary Motion. But in the process of simulation, I found that if the planet's perihelion is close to the star, its trajectory will be very special (as shown in Figure 2). I guess this may be a matter of precision.

Part II: Simulation of Binary Star System

The dull and boring elliptical orbit of the solar system did not satisfy me. After learning that most of the galaxies in the universe are multi-star systems, I began the process of simulating multiple stars. I generalized my program and replaced the original acceleration calculation method with the real gravitational formula. At the same time, I also carefully researched the mass of the sun, the mass of the earth, and the distance of the astronomical unit, in an effort to enhance the connection between the program and reality.





For better observation, I used the conservation of momentum formula to generate the initial position and acceleration of the planet, so that the two stars' center of mass can be always at the center of the screen. Of course, this can also be seen as a process of transforming the coordinate system. So far, I succeeded in the simulation of binary star systems. In the process, I intuitively discovered that the trajectories of the two stars are all elliptical, and one focus of the ellipse is their center of mass.

Part III: Extended Research

In order to increase the fun and simulate more common situations, I added the "Meteor" function to the program. In the program, a certain mass of celestial bodies can be generated in a variety of ways, interfering with the relatively stable system. Of course, if the generated celestial mass is small, there is a very small probability that it can become a planet in the system. After a lot of simulations, I came to two conclusions. First, the robustness of the binary system is relatively strong. As long as the meteor does not pull them apart, they can always find a new elliptical orbit (seen as changing the coordinate system). Second, the probability that a comet becomes a stable planet in a binary system is extremely small.

You can try my program by referring to the instructions.

Part IV: Future Research Directions

If I continue this project in the future, I plan to have the following three points.

1) I wrote this program for better observation, so many quantitative calculations were omitted from the process. If I continue this project, I will try to describe some rules quantitatively, such as finding the critical point when the binary star systems are “pulled apart”.
2) As I wrote in Part I, I have a huge error based on the “basic time unit” simulation method. After researching the data, I learned that calculus can be introduced into the formula to converge the error when simulating gravity. I hope that I can introduce that in my program in the future.
3) I am very fascinated by the three-body system. I hope that I can use my program to find the metastable solution of the three-body system (that is, the initial condition that can run stably for a period of time)

Attachment:
https://github.com/JackBai0914/StarSystemProgram/blob/f285cbc21096300aeda909cb0b6e834f2c43b8c5/Desktop/Solar_Binary3.0.exe

Source Code:

```
#include "stdafx.h"
#include <graphics.h>
#include <tchar.h>
#define TIME (ld)clock()/CLOCKS_PER_SEC
using namespace std;
typedef long long ll;
typedef long double ld;
typedef double db;
//LENTH DEFINITION m
#define X (1880)
#define Y (1200)
#define SP 1
#define SCALE 1.5e9
#define AU 1.5e11


//SPEED DEFINITION m/s
#define EARTH_SPEED 3e4

//MASS DEFINITION kg
#define MASS_SUN 2e30

//CONSTANT DEFINITION
#define T (3600 * 12)
#define GAP (1) //record gap
#define G 6.67e-11


int sleep_time = 0;
ld Rand() { return (ld)rand() / RAND_MAX; }


namespace Graph {
 COLORREF rec[(int)(X * SP) + 10][(int)(Y * SP) + 10];
 void record() {
  for (int i = 0; i < X * SP; i++)
   for (int j = 0; j < Y * SP; j++) {
    rec[i][j] = getpixel(i, j);
    /*
    float H, S, L;
    RGBtoHSL(rec[i][j], &H, &S, &L);
    L *= 0.66;
```

```
    rec[i][j] = HSLtoRGB(H, S, L);
    putpixel(i, j, rec[i][j]);
    */
   }
//  saveimage(_T("BG.jpg"));
 }
 void fillin() {
  for (int i = 0; i < X * SP; i++)
   for (int j = 0; j < Y * SP; j++)
    putpixel(i, j, rec[i][j]);
 }
 void circle(ld x, ld y, ld r, ld dec = 30, int corn = 255, int
addr = 0, int addg = 0, int addb = 0) {
  if (corn <= 0) {
   corn = dec = 0;
  }
  x *= SP;
  y *= SP;
  r *= SP;
  dec /= SP;
  for (int i = x - r; i <= x + r; i++)
   for (int j = y - r; j <= y + r; j++) {
    if (i < 0 || j < 0 || i >= X || j >= Y)
     continue;
    ld dis = hypot(i - x, j - y);
    if (dis <= r / 6) {
     ld L = min(corn, max(0.0, 185 - dis / (r * 185 / 255) * 185 *
3) + 70);
     putpixel(i, j, RGB(max(0.0, L), max(0.0, L), max(0.0, L)));
    }
    else if (dis <= r) {
     ld L = min(corn, max(0.0, 185 - dis / (r * 185 / 255) * 185)
+ 70);
     if (L <= 0)   putpixel(i, j, rec[i][j]);
       else if (L > 70)putpixel(i, j, RGB(max(0.0, L - addr),
max(0.0, L - addg), max(0.0, L - addb)));
    }
   }
 }
}using namespace Graph;
```

```
namespace MassPoint {
 struct massPoint {
  ld x, vx, ax;
  ld y, vy, ay;
  ld mass, size, dec, corn;
  int life;
  massPoint() { x = vx = ax = y = vy = ay = 0, mass = size = dec =
corn = life = 0; }
  massPoint(string s) {
   if (s == "comet") {
    ld ivx, ivy, imass;
    cout << "please input 5 numbers separated by space, ";
      cout << "representing the comet's x-axis position(1900 in
total), y-axis position(1200 in total), "
     << "x-axis speed(relative to the earth average speed), y-axis
speed(relative to the earth average speed), "
     << "and mass(relative to the sun) : " << endl;
    cin >> x >> y >> ivx >> ivy >> imass;
    vx = ivx * 30000 / SCALE;
    vy = ivy * 30000 / SCALE;
    corn = 255;
    mass = MASS_SUN * imass;
    size = pow(161 * mass / MASS_SUN / 10 * 100, 0.333);
    dec = 15;
    life = 0;
    return;
   }
   if (s == "comet2") {
    x = 20;
    y = 20;
    vx = 1 * 30000 / SCALE;
    vy = 1 * 30000 / SCALE;
    corn = 255;
    mass = MASS_SUN * 1;
    size = pow(161 * mass / MASS_SUN / 10 * 100, 0.333);
    dec = 15;
    life = 0;
    return;
   }
   if (s == "comet_sm") {
    int dir = rand() % 4;
    vx = (rand() % 40000 + 10000) / SCALE;
    vy = (rand() % 40000 + 10000) / SCALE;
```

```
    if (dir == 0) x = 0, y = Y * Rand();
    if (dir == 1) x = X - 1, y = Y * Rand();
    if (dir == 2) x = X * Rand(), y = 0;
    if (dir == 3) x = X * Rand(), y = Y - 1;
    if (x >= X / 2) vx = -vx;
    if (y >= Y / 2) vy = -vy;
     cout << "comet info : " << x << " " << y << " " << vx << " "
<< vy << " " << endl;
    corn = 255;
    mass = MASS_SUN * Rand() * 0.05;
    size = 3;
    dec = 15;
    life = 0;
    return;
   }
   if (s == "cin") {
    cin >> x >> vx >> y >> vy;
    corn = 255;
    mass = 10 * MASS_SUN * Rand();
    size = pow(161 * mass / MASS_SUN / 10 * 100, 0.333);
    dec = 15;
    life = 0;
    return;
   }
   if (s == "sun") {
    x = X * 0.5;
    y = Y * 0.5;
    vx = vy = 0;
    corn = 255;
    mass = 10 * MASS_SUN * Rand();
    size = pow(2 * 256 * mass / MASS_SUN / 10 * 100, 0.333);
    dec = 15;
    life = 0;
    return;
   }
   if (s == "earth") {
    x = Rand() * X * 0.5 + X * 0.25;
    y = Rand() * Y * 0.5 + Y * 0.25;
     vx = (rand() % 40000 + 10000) / SCALE; if (x > X / 2) vx = -
vx;
     vy = (rand() % 40000 + 10000) / SCALE; if (y > Y / 2) vy = -
vy;
    corn = 255;
```

```
    mass = 0.0001 * MASS_SUN * Rand();
    size = pow(25000 * 256 * mass / MASS_SUN / 10 * 100, 0.333);
    dec = 15;
    life = 0;
    return;
   }
   x = Rand() * X * 0.5 + X * 0.25;
   y = Rand() * Y * 0.5 + Y * 0.25;
   vx = (rand() % 40000 + 10000) / SCALE; if (x > X / 2) vx = -vx;
   vy = (rand() % 40000 + 10000) / SCALE; if (y > Y / 2) vy = -vy;
   corn = 255;
   mass = 8 * MASS_SUN * Rand() + 0.2 * MASS_SUN;
   size = pow(256 * mass / MASS_SUN / 10 * 100, 0.333);
   dec = 15;
   life = 0;
  }
  void Move() { life++, vx += ax * T, vy += ay * T, x += vx * T, y
+= vy * T; }
  void Paint(int r = 0, int g = 0, int b = 0) { circle(x, y, size,
dec, corn, r, g, b); }
  void Erase() { circle(x, y, size + 1, 0, 0); }
  bool Die() { return x < 0 || x >= X || y < 0 || y >= Y; }
  void clear_G() { ax = ay = 0; }
  void calc_G(massPoint b) {
   ld r = hypot(b.x - x, b.y - y) * SCALE;
   ld A = G * b.mass / r / r; //F = GMm/r^2, a = F/m
            //   cout << A << endl;
   A /= SCALE;
   ld alp = atan2(b.y - y, b.x - x);
   ax += A * cos(alp);
   ay += A * sin(alp);
  }
 };
 vector <massPoint*> seeds;
 massPoint sun;
 massPoint comet[1010];
 int cst = 0;

 void timePass(int sleeptime = sleep_time) {
  for (int i = 0; i < seeds.size(); i++) {
   seeds[i]->clear_G();
  }
  for (int i = 0; i < seeds.size(); i++)
```

```
   for (int j = 0; j < seeds.size(); j++)
    if (i != j)
     seeds[i]->calc_G(*seeds[j]);
  for (int i = 0; i < seeds.size(); i++) {
   seeds[i]->Move();
  }
  Sleep(sleeptime);
 }
 void catastrophe() {
  comet[++ cst] = (massPoint("comet"));
  seeds.push_back(&comet[cst]);
 }
 void daily_comet() {
  comet[++cst] = (massPoint("comet_sm"));
  seeds.push_back(&comet[cst]);
 }
 void catastrophe2() {
  comet[++cst] = (massPoint("comet2"));
  seeds.push_back(&comet[cst]);
 }
}using namespace MassPoint;
vector <massPoint> fath_rec;
vector <massPoint> moth_rec;




void Record(string s, massPoint sd) {
 FILE *outp;  fopen_s(&outp, s.c_str(), "w");
 fprintf(outp, "x = %.13lf, vx = %.13lf, y = %.13lf, vy = %.13lf
\n", (db)sd.x, (db)sd.vx, (db)sd.y, (db)sd.vy);
 fclose(outp);
}
int pic_num = 0;
int pic_str;
void takePic() {
 stringstream pic_name;
  pic_name  <<  "copy  planet.jpg  planet"  <<  pic_str  <<  "_"  <<
(pic_num++) << ".jpg";
 saveimage(_T("planet.jpg"));
 system(pic_name.str().c_str());
}
const int Take = -1;
```

```
bool operation(massPoint c) {
 if (_kbhit()) {
  int info = _getch();
//  _getch();
  cout << "you typed: '" << (char)(info) << "'" << endl;
  if (info == 'a' || info == 'b' || info == 'c') {
   stringstream name;
   name << (char)info;
   name << rand() % 1000 + 1000;
   Record(name.str(), *seeds[seeds.size() - 1]);
  }
  if (info == 'n')
   return 1;
  if (info == 'p') {
   takePic();
   cout << "A picture was taken, check your file." << endl;
  }
  if (info == 'u' && sleep_time > 0)
   sleep_time--, cout << "speed up" << endl;
  if (info == 'd')
   sleep_time++, cout << "speed down" << endl;
  if (info == 'e') {
   closegraph();
   exit(0);
  }
  if (info == 't') {
   cout << c.life << endl;
  }
  if (info == ':') {
   catastrophe();
  }
  if (info == ';') {
   daily_comet();
   cout << "A small comet was generated." << endl;
  }
  if (info == 'C') {
   catastrophe2();
   cout << "A small comet was generated." << endl;
  }
 }
 if (c.life == Take)
  takePic();
 return 0;
```

```
}

int predict() {
 int dist_fit = 0;
 vector <massPoint> storage;
   for    (int    i    =    0;    i    <    seeds.size();    i++)
storage.push_back(*seeds[i]);
 for (int i = 1; i <= 2000; i++) {
  timePass(0);
  for (int j = 0; j < seeds.size(); j++) {
   if (seeds[j]->Die()) {
    for (int k = 0; k < seeds.size(); k++)
     *seeds[k] = storage[k];
    return i;
   }
//   if (i % 13 == 0)
//    putpixel(seeds[j]->x, seeds[j]->y, RGB(100, 100, 100));
  }
   if (hypot(seeds[0]->x - seeds[1]->x, seeds[0]->y - seeds[1]->y)
>= Y * 0.66)  dist_fit = -1;
//   if (hypot(seeds[0]->x - seeds[1]->x, seeds[0]->y - seeds[1]-
>y) >= Y / 3 && dist_fit >= 0) dist_fit = 1;
 }
 for (int i = 0; i < seeds.size(); i++)
  *seeds[i] = storage[i];
 if (dist_fit == -1)
  return -1;
 return 0;
}

int main()
{
 srand(time(0));
 pic_str = rand() % 1000;
 if (pic_str < 1000) pic_str += 1000;

 initgraph(X * SP, Y * SP, SHOWCONSOLE);

  loadimage(NULL,    _T("C:\\Users\\Jack    Bai\\Desktop\\Projects\
\TwoStar\\image\\BG.jpg"), X * SP, Y * SP);
 settextcolor(RGB(100, 100, 240));
 settextstyle(29 * SP, 0, _T("宋体"));
```

```
 outtextxy(X * SP / 2 - 155 * SP, 10 * SP, _T("Binary Star Simulator"));
 outtextxy(X * SP / 2 - 100 * SP, 42 * SP, _T("双星系统模拟器"));
 settextcolor(RGB(200, 200, 250));
 settextstyle(21 * SP, 0 * SP, _T("宋体"));
 outtextxy(X * SP / 2 - 36 * SP, 73 * SP, _T("JackBai"));
 settextcolor(RGB(255, 255, 255));
 settextstyle(25 * SP, 0, _T("Consolas"));
 outtextxy(20 * SP, 20 * SP, _T("check the terminal for more info"));
 outtextxy(20 * SP, 50 * SP, _T("'n': new world"));
 outtextxy(20 * SP, 80 * SP, _T("'u': speed up"));
 outtextxy(20 * SP, 110 * SP, _T("'d': speed down"));
 outtextxy(20 * SP, 140 * SP, _T("'p': screenshot"));
 outtextxy(20 * SP, 170 * SP, _T("'e': exit"));
 outtextxy(20 * SP, 200 * SP, _T("'a'/'b'/'c': record seeds"));
 settextcolor(RGB(135, 193, 252));
 outtextxy(20 * SP, 230 * SP, _T("';': generate random comet"));
 outtextxy(20 * SP, 260 * SP, _T("':': generate handmade comet"));
 outtextxy(20 * SP, 290 * SP, _T("'C': Catastrophe!"));
 settextcolor(RGB(255, 205, 205));
 settextstyle(25 * SP, 0, _T("Consolas"));
 outtextxy(20 * SP, 330 * SP, _T("after typed 'n',"));
 outtextxy(20 * SP, 360 * SP, _T("enter a character: "));
 outtextxy(20 * SP, 390 * SP, _T("  's': solar system"));
 outtextxy(20 * SP, 420 * SP, _T("  'b': binary system"));

 record();
 BeginBatchDraw();

 while (1) {
  seeds.clear();
  fath_rec.clear();
  moth_rec.clear();
  clearrectangle(0, 0, X * SP, Y * SP);
  fillin();
  FlushBatchDraw();

  while (!_kbhit()) {}
  int info = _getch();

  massPoint fath("suibianlai");
  massPoint moth("suibianlai");
```

```
  cout << info << " " << (int)'b' << endl;
  if (info == 'b') {
   fath = massPoint("suibianlai");
   moth = massPoint("suibianlai");
   moth.x = (X / 2 - fath.x) * fath.mass / moth.mass + X / 2;
   moth.y = (Y / 2 - fath.y) * fath.mass / moth.mass + Y / 2;
   moth.vx = -fath.vx * fath.mass / moth.mass;
   moth.vy = -fath.vy * fath.mass / moth.mass;
   seeds.push_back(&fath);
   seeds.push_back(&moth);
  }
  else {
   fath = massPoint("sun");
   moth = massPoint("earth");
   seeds.push_back(&fath);
   seeds.push_back(&moth);
  }


  while (predict()) {
   seeds.clear();
//   cout << "Generated a broken world" << endl;
   //   cout << "predicting " << endl;
   if (info == 'b') {
    fath = massPoint("suibianlai");
    moth = massPoint("suibianlai");
    moth.x = (X / 2 - fath.x) * fath.mass / moth.mass + X / 2;
    moth.y = (Y / 2 - fath.y) * fath.mass / moth.mass + Y / 2;
    moth.vx = -fath.vx * fath.mass / moth.mass;
    moth.vy = -fath.vy * fath.mass / moth.mass;
    seeds.push_back(&fath);
    seeds.push_back(&moth);
   }
   else {
    fath = massPoint("sun");
    moth = massPoint("earth");
    seeds.push_back(&fath);
    seeds.push_back(&moth);
   }
  }
  cout << "Generated a stable world. " << endl;
```

```
  int cut = predict();
//  cout << "result : " << cut << endl;
  while (true) {
   if (operation(fath)) break;
//       if  (fath.Die()  ||  moth.Die())  {cout  <<  "END"  <<
endl;continue;}
   for (int i = 0; i < seeds.size(); i++) {
    seeds[i]->Erase();
    if (seeds[i]->Die()) {
     cout << "One star died." << endl;
     for (int j = i + 1; j < seeds.size(); j++)
      seeds[j - 1] = seeds[j];
     seeds.pop_back();
    }
   }
   if (!seeds.size()) {
    cout << "Destroyed." << endl;
    Sleep(300);
    break;
   }
//   cout << "size :" << seeds.size() << endl;
   timePass();
//     if  (fath.Die()  ||  moth.Die())  {cout  <<  "END"  <<  endl;
continue;}
   if (fath.life % GAP == 0) {
    for (int i = 0; i < seeds.size(); i++)
     seeds[i]->Paint(255, 0, 255);
    fath_rec.push_back(fath);
    moth_rec.push_back(moth);
       for  (int  i  =  max(0,  (int)fath_rec.size()  -  500);  i  <
fath_rec.size(); i++) {
     fath_rec[i].corn -= 0.7;
      if (fath_rec[i].x < 0 || fath_rec[i].y < 0 || fath_rec[i].x
>= X || fath_rec[i].y >= Y) continue;
     int L = max(fath_rec[i].corn, 0);
      if (L <= 30)putpixel(fath_rec[i].x * SP, fath_rec[i].y * SP,
rec[(int)fath_rec[i].x][(int)fath_rec[i].y]);
     else  putpixel(fath_rec[i].x * SP, fath_rec[i].y * SP, RGB(L,
0, 0));

    }
       for  (int  i  =  max(0,  (int)moth_rec.size()  -  700);  i  <
moth_rec.size(); i++) {
```

```
     moth_rec[i].corn -= 0.5;
      if (moth_rec[i].x < 0 || moth_rec[i].y < 0 || moth_rec[i].x
>= X || moth_rec[i].y >= Y) continue;
     int L = max(moth_rec[i].corn, 0);
      if (L <= 30)putpixel(moth_rec[i].x * SP, moth_rec[i].y * SP,
rec[(int)moth_rec[i].x][(int)moth_rec[i].y]);
     else  putpixel(moth_rec[i].x * SP, moth_rec[i].y * SP, RGB(0,
0, L));
    }
    fath.Paint(0, 255, 255);
    moth.Paint(255, 255, 0);
    FlushBatchDraw();
   }

  }
//  Sleep(100);
 }
 EndBatchDraw();
 closegraph();
 return 0;
}
```