Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(ФГБОУ ВО «СибГУТИ»)

*Кафедра прикладной математики и кибернетики*

Отчет по учебно-исследовательской практике

Вариант № 1

Выполнил:

студент гр. ИП-513

Майоров С.А.

Проверил:

ст. преподаватель кафедры ПМиК

Милешко А.В.

Новосибирск

2017

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. Гравитационная задача N тел

2.2. Метод Рунге-Кутты

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Диаграмма классов

5. ТЕКСТ ПРОГРАММЫ

6. РЕЗУЛЬТАТЫ ТЕСТОВ

ПОСТАНОВКА ЗАДАЧИ

Написать программу, моделирующую планетарную систему, соответствующую принципам ООП и работающую под ОС Windows или Linux.

ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

Гравитационная задача N тел.

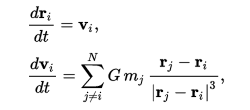
Гравитационная задача N тел является классической проблемой небесной механики и гравитационной динамики Ньютона.

Она формулируется следующим образом.

В пустоте находится N материальных точек, массы которых известны {mi}. Пусть попарное взаимодействие точек подчинено закону тяготения Ньютона, и пусть силы гравитации аддитивны. Пусть известны начальные на момент времени t=0 положения и скорости каждой точки ri|t =0 = ri0, vi|t =0 = vi0. Требуется найти положения точек для всех последующих моментов времени.

В математической формулировке:

Эволюция системы N гравитирующих тел (материальных точек) описывается следующей системой уравнений:

,

где mi, ri, vi — масса, радиус-вектор и скорость i-го тела соответственно (i изменяется от 1 до N), G — гравитационная постоянная. Массы тел, а также положения и скорости в начальный момент времени считаются известными. Необходимо найти положения и скорости всех частиц в произвольный момент времени.

С появлением компьютерной техники появилась реальная возможность изучать свойства систем гравитирующих тел путём численного решения системы уравнений движения (ранее доступны были только аналитические методы). Для этого используются, например, метод Рунге — Кутты (четвёртого или более высокого порядка).

Численные методы сталкиваются с теми же проблемами, что и аналитические — при тесных сближениях тел необходимо уменьшать шаг интегрирования, а при этом быстро растут численные ошибки. Кроме того, при «прямом» интегрировании число вычислений силы для каждого шага растёт с ростом числа тел приблизительно как N^2, что делает практически невозможным моделирование систем, состоящих из десятков и сотен тысяч тел (в этой работе встроенного технического ограничения для таких вычислений нет, но рекомендуется остановиться на 200-300 объектах).

Метод Рунге-Кутты.

Методы Рунге — Кутты - большой класс численных методов решения задачи Коши для обыкновенных дифференциальных уравнений и их систем. Первые методы данного класса были предложены около 1900 года немецкими математиками К. Рунге и М. В. Куттой.

Метод Рунге — Кутты четвёртого порядка при вычислениях с постоянным шагом интегрирования столь широко распространён, что его часто называют просто методом Рунге — Кутты.

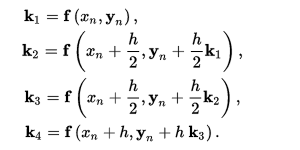
Рассмотрим задачу Коши для системы обыкновенных дифференциальных уравнений первого порядка. (Далее y, f, ki Rn, а x, h R1).



Тогда приближенное значение в последующих точках вычисляется по итерационной формуле:



Вычисление нового значения проходит в четыре стадии:



где h — величина шага сетки по x.

Этот метод имеет четвёртый порядок точности. Это значит, что ошибка на одном шаге имеет порядок O(h5), а суммарная ошибка на конечном интервале интегрирования имеет порядок O(h4) .

ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

В ходе работы постановка задачи о N телах не претерпела значительных изменений.

Метод Рунге-Кутты был применен для вычисления скоростей и координат по следующим правилам:

h = 1, dx = vx, dy = vy, dvx = ax = fx(x), dvy = ay = fy(y), где

fx(x) и fy(y) – функции расчета суммы ускорений, вызванных взаимодействием с N-1 объектом, где в роли гравитационной постоянной участвует точность моделирования.

ОПИСАНИЕ ПРОГРАММЫ

Программа представляет собой набор классов, осуществляющих различные манипуляции с объектами модели планетарной системы:

1) FlyObject – класс, представляющий объект планетарной системы (астероид, планету, звезду). В нем заключены математические манипуляции над координатами и скоростью объекта с помощью метода Рунге-Кутты.

2) AsteroidBelt – класс, являющийся генератором астероидных полей в пределах двух концентрических окружностей с фиксированным центром.

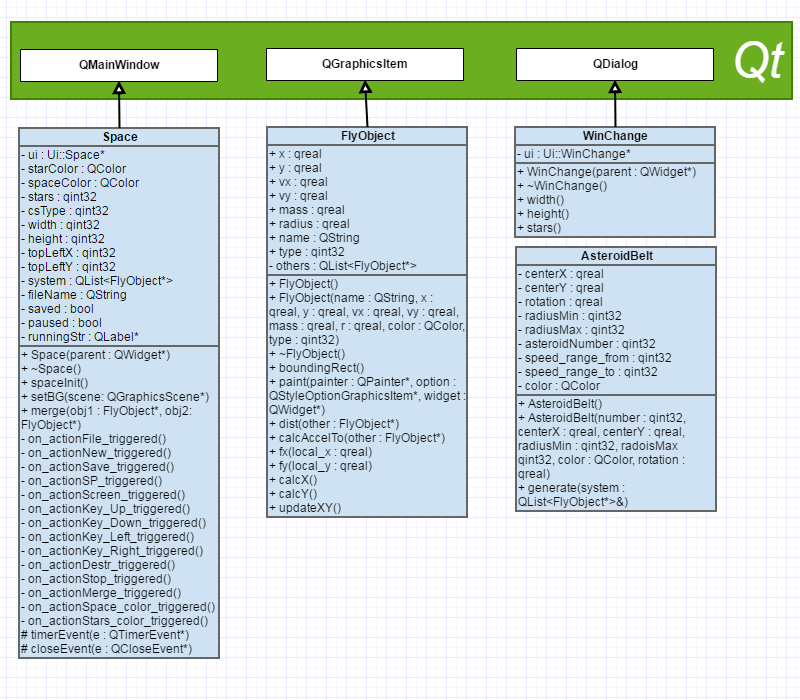
3) Space – класс-окно, реализующий «общение» пользователя с системой, доступ ко всему функционалу, а также отображающий результаты моделирования на экране. Вывод модели осуществляется графически с помощью библиотеки Qt.

Также есть побочный класс диалогового окна выбора размеров основного окна и количества статичных звезд на фоне - WinChange.

В функционал программы входит:

* Моделирование случайной планетарной системы («жизнеспособность» не гарантируется);
* Сохранение результатов в виде файла конфигурации;
* Загрузка ранее созданной системы с помощью файла конфигурации;
* Остановка моделирования в реальном времени с возможностью его дальнейшего продолжения;
* Установка размеров окна, цвета космоса и фоновых звезд;
* Определение способа коллизий в случае столкновения двух объектов;
* Перемещение по экрану.

Диаграмма классов



ТЕКСТ ПРОГРАММЫ

|  |
| --- |
| *main.cpp* |
| #include "space.h"  #include <QApplication>  int main(int argc, char \*argv[])  {  QApplication a(argc, argv);  Space w;  w.show();  return a.exec();  } |
| *space.cpp* |
| #include "space.h"  #include "ui\_space.h"  Space::Space(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::Space)  {  ui->setupUi(this);  topLeftX = topLeftY = 0;  width = 250;  height = 250;  stars = 0;  system = QList<FlyObject \*>();  fileName = QDir::currentPath();  saved = true;  paused = false;  ui->graphicsView->setRenderHint(QPainter::Antialiasing);  runningStr = new QLabel(" "  "New system: Ctrl+N "  "| Open file: Ctrl+O "  "| Save file: Ctrl+S "  "| Start/Pause: Space "  "| Set screen size and star number: Ctrl+F "  "| Set collision (Merge, Destruction, Stop): F1, F2, F3 "  "| Set color (Space, Stars): F5, F6 "  "| Movement: Arrows "  "| Exit: Ctrl+Q "  " ");  statusBar()->addWidget(runningStr);  this->setFixedSize(width,height);  startTimer(33);  }  Space::~Space()  {  delete ui;  delete runningStr;  system.clear();  }  void Space::setBG(QGraphicsScene \*scene)  {  if (scene != NULL){  QPixmap bg(width, height);  bg.fill(spaceColor);  QPainter painter(&bg);  painter.setBrush(starColor);  for (qint32 i = 0; i < stars; ++i){  painter.drawEllipse(qrand()%width,qrand()%height,2,2);  }  scene->setBackgroundBrush(bg);  }  }  void Space::spaceInit()  {  this->setFixedSize(width,height);  if (ui->graphicsView->scene() != NULL)  delete ui->graphicsView->scene();  QGraphicsScene\* scene = new QGraphicsScene(this);  setBG(scene);  ui->graphicsView->setScene(scene);  for (qint32 i = 0; i < system.size(); ++i){  scene->addItem(system.at(i));  }  topLeftX = topLeftY = 0;  }  FlyObject\* Space::merge(FlyObject \*obj1, FlyObject \*obj2)  {  QString name = obj1->name + "+" + obj2->name;  qreal mass = obj1->mass + obj2->mass;  qreal x = (obj1->x \* obj1->mass + obj2->x \* obj2->mass) / mass;  qreal y = (obj1->y \* obj1->mass + obj2->y \* obj2->mass) / mass;  qreal vx = (obj1->vx \* obj1->mass + obj2->vx \* obj2->mass) / mass;  qreal vy = (obj1->vy \* obj1->mass + obj2->vy \* obj2->mass) / mass;  QColor new\_color = obj1->color;  if (obj2->mass > obj1->mass)  new\_color = obj2->color;  qint32 type = ASTEROID;  if (obj1->type == STAR || obj2->type == STAR) {  type = STAR;  } else {  if (obj1->type == PLANET || obj2->type == PLANET)  type = PLANET;  }  qreal radius = obj1->radius \* sqrt(mass / obj1->mass);  if (obj1->radius < obj2->radius)  radius = obj2->radius \* sqrt(mass / obj2->mass);  FlyObject \*obj3 = new FlyObject(name, x, y, vx, vy, mass, radius, new\_color, type);  return obj3;  }  void Space::timerEvent(QTimerEvent\* e)  {  Q\_UNUSED(e);  if (!paused && ui->graphicsView->scene()){  for (qint32 i = 0; i < system.size(); ++i){  FlyObject \*obj1 = system.at(i);  for (qint32 j = 0; j < system.size(); ++j){  if (obj1 != system.at(j)){  FlyObject \*obj2 = system.at(j);  obj1->calcAccelTo(obj2);  if (obj1->dist(obj2) <= 0) {  FlyObject \*mobj = merge(obj1,obj2);  switch(csType){  case 1:  system.append(mobj);  ui->graphicsView->scene()->addItem(mobj);  system.removeAll(obj1);  delete obj1;  system.removeAll(obj2);  delete obj2;  break;  case 2:  system.removeAll(obj1);  delete obj1;  system.removeAll(obj2);  delete obj2;  break;  case 3:  paused = true;  break;  }  break;  }  }  }  }  for (qint32 i = 0; i < system.size(); ++i)  system.at(i)->updateXY();  ui->graphicsView->scene()->update();  saved = false;  }  if (ui->graphicsView->scene())  ui->graphicsView->setSceneRect(topLeftX,topLeftY,width,height);  QString rsBefore = runningStr->text().remove(runningStr->text().length()-1,1);  runningStr->setText(runningStr->text().remove(0,runningStr->text().length()-1)+rsBefore);  }  void Space::closeEvent(QCloseEvent \*e)  {  paused = true;  if (ui->graphicsView->scene() && !saved) {  QMessageBox question(QMessageBox::Question, "Save file?", "Do you want to save system before exit?",  QMessageBox::Ok, this);  question.addButton(QMessageBox::No);  question.addButton(QMessageBox::Cancel);  question.exec();  if (question.clickedButton() == question.button(QMessageBox::Ok)){  if(on\_actionSave\_triggered()){  e->accept();  } else {  e->ignore();  }  } else {  if (question.clickedButton() == question.button(QMessageBox::Cancel)){  e->ignore();  }  }  }  }  void Space::on\_actionNew\_triggered()  {  paused = true;  if (!ui->graphicsView->scene() || saved) {  spaceColor = QColor("black");  starColor = QColor("white");  qsrand(QTime::currentTime().msec());  stars = qrand() % 50 + 150;  on\_actionMerge\_triggered();  topLeftX = 0;  topLeftY = 0;  width = 800;  height = 600;  qint32 planetLine = height/2;  FlyObject \*star = new FlyObject("Star",  width/2,  planetLine,  0,  0,  qrand()%1000+4000,  qrand()%5+10,  starColor,  STAR);  system.append(star);  qint32 planetCount = qrand()%4+1;  for (qint32 i = 1; i <= planetCount; ++i){  FlyObject \*planet = new FlyObject(QString("Planet %1").arg(i),  qrand()%width,  planetLine,  0,  qrand()%10-5,  qrand()%90+10,  qrand()%10,  QColor(qrand()%256,  qrand()%256,  qrand()%256),  PLANET);  system.append(planet);  }  saved = false;  spaceInit();  } else {  QMessageBox question(QMessageBox::Question, "Save file?", "Do you want to save system before creating something new?",  QMessageBox::Ok, this);  question.addButton(QMessageBox::No);  question.addButton(QMessageBox::Cancel);  question.exec();  if (question.clickedButton() == question.button(QMessageBox::Ok)){  on\_actionSave\_triggered();  } else {  if (question.clickedButton() == question.button(QMessageBox::No)){  spaceColor = QColor("black");  starColor = QColor("white");  qsrand(QTime::currentTime().msec());  stars = qrand() % 50 + 150;  on\_actionMerge\_triggered();  width = 800;  height = 600;  ui->graphicsView->scene()->clear();  system.clear();  qint32 planetLine = height/2;  FlyObject \*star = new FlyObject("Star",  width/2,  planetLine,  0,  0,  qrand()%1000+4000,  qrand()%5+10,  starColor,  STAR);  system.append(star);  qint32 planetCount = qrand()%4+1;  for (qint32 i = 1; i <= planetCount; ++i){  FlyObject \*planet = new FlyObject(QString("Planet %1").arg(i),  qrand()%width,  planetLine,  0,  qrand()%10-5,  qrand()%90+10,  qrand()%10,  QColor(qrand()%256,  qrand()%256,  qrand()%256),  PLANET);  system.append(planet);  }  saved = false;  spaceInit();  }  }  }  }  bool Space::on\_actionSave\_triggered()  {  paused = true;  if (!ui->graphicsView->scene()) {  return false;  }  fileName = QFileDialog::getSaveFileName(this, "Save file...",  fileName,"Configuration files (\*.ini);; All files (\*.\*)");  if(fileName.isEmpty())  {  return false;  }  QFile file(fileName);  if (file.open(QIODevice::WriteOnly | QIODevice::Text))  {  QByteArray data;  data += QString("WIN\_WIDTH:%1\n").arg(width);  data += QString("WIN\_HEIGHT:%1\n").arg(height);  data += QString("STAR\_NUM:%1\n").arg(stars);  data += QString("STAR\_COLOR:%1\n").arg(starColor.name());  data += QString("SPACE\_COLOR:%1\n").arg(spaceColor.name());  data += QString("ON\_COLLISION:%1\n\n").arg(csType);  for (qint32 i = 0; i < system.size(); ++i){  FlyObject \*obj = system.at(i);  switch(obj->type){  case STAR:  data += QString("STAR:%1,%2,%3,%4,%5,%6,%7,%8\n\n")  .arg(obj->name)  .arg(obj->x)  .arg(obj->y)  .arg(obj->vx)  .arg(obj->vy)  .arg(obj->mass)  .arg(obj->radius)  .arg(obj->color.name());  break;  case PLANET:  data += QString("PLANET:%1,%2,%3,%4,%5,%6,%7,%8\n\n")  .arg(obj->name)  .arg(obj->x)  .arg(obj->y)  .arg(obj->vx)  .arg(obj->vy)  .arg(obj->mass)  .arg(obj->radius)  .arg(obj->color.name());  break;  case ASTEROID:  data += QString("ASTEROID:%1,%2,%3,%4,%5,%6,%7,%8\n\n")  .arg(obj->name)  .arg(obj->x)  .arg(obj->y)  .arg(obj->vx)  .arg(obj->vy)  .arg(obj->mass)  .arg(obj->radius)  .arg(obj->color.name());  break;  }  }  file.write(data);  file.close();  saved = true;  }  else  {  QMessageBox::warning(this, "Error",  QString("Could not open file %1 for writing").arg(file.fileName()),  QMessageBox::Ok);  saved = false;  }  return saved;  }  void Space::on\_actionFile\_triggered()  {  paused = true;  bool flag;  if (!ui->graphicsView->scene() || saved) {  flag = true;  } else {  QMessageBox question(QMessageBox::Question, "Save file?", "Do you want to save system before creating something new?",  QMessageBox::Ok, this);  question.addButton(QMessageBox::No);  question.addButton(QMessageBox::Cancel);  question.exec();  if (question.clickedButton() == question.button(QMessageBox::Ok)){  flag = on\_actionSave\_triggered();  } else {  if (question.clickedButton() == question.button(QMessageBox::No)){  flag = true;  } else {  flag = false;  }  }  }  if (flag){  fileName = QFileDialog::getOpenFileName(this, "Open file...",  fileName,"Configuration files (\*.ini);; All files (\*.\*)");  if(fileName.isEmpty())  {  return;  }  topLeftX = 0;  topLeftY = 0;  spaceColor = QColor("black");  starColor = QColor("white");  qsrand(QTime::currentTime().msec());  stars = qrand() % 50 + 150;  csType = MERGE;  width = 800;  height = 600;  if(ui->graphicsView->scene()){  ui->graphicsView->scene()->clear();  system.clear();  }  QFile file(fileName);  if (file.open(QIODevice::ReadOnly | QIODevice::Text))  {  QTextStream in(&file);  while (!in.atEnd()) {  QString line = in.readLine();  if (line.startsWith("WIN\_WIDTH:")){  width = line.section(':',1,1).toInt();  }  if (line.startsWith("WIN\_HEIGHT:")){  height = line.section(':',1,1).toInt();  }  if (line.startsWith("SPACE\_COLOR:")){  spaceColor = QColor(line.section(':',1,1));  }  if (line.startsWith("STAR\_COLOR:")){  starColor = QColor(line.section(':',1,1));  }  if (line.startsWith("STAR\_NUM:")){  stars = line.section(':',1,1).toInt();  }  if (line.startsWith("ON\_COLLISION:")){  csType = line.section(':',1,1).toInt();  }  if (line.startsWith("ASTEROID\_BELT:")){  QString beltInfo = line.section(':',1,1);  AsteroidBelt \*belt = new AsteroidBelt(beltInfo.section(',',0,0).toDouble(),  beltInfo.section(',',1,1).toDouble(),  beltInfo.section(',',2,2).toInt(),  beltInfo.section(',',3,3).toInt(),  beltInfo.section(',',4,4).toInt(),  QColor(beltInfo.section(',',5,5)),  beltInfo.section(',',6,6).toDouble());  belt->generate(system);  }  if (line.startsWith("PLANET:")){  QString objInfo = line.section(':',1,1);  FlyObject \*obj = new FlyObject(objInfo.section(',',0,0),  objInfo.section(',',1,1).toDouble(),  objInfo.section(',',2,2).toDouble(),  objInfo.section(',',3,3).toDouble(),  objInfo.section(',',4,4).toDouble(),  objInfo.section(',',5,5).toDouble(),  objInfo.section(',',6,6).toDouble(),  QColor(objInfo.section(',',7,7)),  PLANET);  system.append(obj);  }  if (line.startsWith("STAR:")){  QString objInfo = line.section(':',1,1);  FlyObject \*obj = new FlyObject(objInfo.section(',',0,0),  objInfo.section(',',1,1).toDouble(),  objInfo.section(',',2,2).toDouble(),  objInfo.section(',',3,3).toDouble(),  objInfo.section(',',4,4).toDouble(),  objInfo.section(',',5,5).toDouble(),  objInfo.section(',',6,6).toDouble(),  QColor(objInfo.section(',',7,7)),  STAR);  system.append(obj);  }  if (line.startsWith("ASTEROID:")){  QString objInfo = line.section(':',1,1);  FlyObject \*obj = new FlyObject(objInfo.section(',',0,0),  objInfo.section(',',1,1).toDouble(),  objInfo.section(',',2,2).toDouble(),  objInfo.section(',',3,3).toDouble(),  objInfo.section(',',4,4).toDouble(),  objInfo.section(',',5,5).toDouble(),  objInfo.section(',',6,6).toDouble(),  QColor(objInfo.section(',',7,7)),  ASTEROID);  system.append(obj);  }  }  switch(csType){  case DESTR:  on\_actionDestr\_triggered();  break;  case STOP:  on\_actionStop\_triggered();  break;  case MERGE:  on\_actionMerge\_triggered();  break;  }  spaceInit();  file.close();  saved = true;  }  else  {  QMessageBox::warning(this, "Error",  QString("Could not open file %1 for reading").arg(file.fileName()),  QMessageBox::Ok);  }  }  }  void Space::on\_actionSP\_triggered()  {  paused = !paused;  }  void Space::on\_actionScreen\_triggered()  {  paused = true;  WinChange\* dlg = new WinChange(this);  if (dlg->exec() == QDialog::Accepted)  {  qint32 new\_width = dlg->width();  if (new\_width == 0)  new\_width = width;  qint32 new\_height = dlg->height();  if (new\_height == 0)  new\_height = height;  qint32 new\_stars = dlg->stars();  if (new\_stars == 0)  new\_stars = stars;  this->setFixedSize(new\_width,new\_height);  stars = new\_stars;  setBG(ui->graphicsView->scene());  }  }  void Space::on\_actionKey\_Up\_triggered()  {  topLeftY -= STEP\_SIZE;  }  void Space::on\_actionKey\_Down\_triggered()  {  topLeftY += STEP\_SIZE;  }  void Space::on\_actionKey\_Left\_triggered()  {  topLeftX -= STEP\_SIZE;  }  void Space::on\_actionKey\_Right\_triggered()  {  topLeftX += STEP\_SIZE;  }  void Space::on\_actionDestr\_triggered()  {  csType = DESTR;  ui->actionDestr->setChecked(true);  ui->actionMerge->setChecked(false);  ui->actionStop->setChecked(false);  }  void Space::on\_actionStop\_triggered()  {  csType = STOP;  ui->actionDestr->setChecked(false);  ui->actionMerge->setChecked(false);  ui->actionStop->setChecked(true);  }  void Space::on\_actionMerge\_triggered()  {  csType = MERGE;  ui->actionDestr->setChecked(false);  ui->actionMerge->setChecked(true);  ui->actionStop->setChecked(false);  }  void Space::on\_actionSpace\_color\_triggered()  {  paused = true;  QColor clr = QColorDialog::getColor(spaceColor,  this,  "Change space color");  if (clr.isValid()){  spaceColor = clr;  setBG(ui->graphicsView->scene());  }  }  void Space::on\_actionStars\_color\_triggered()  {  paused = true;  QColor clr = QColorDialog::getColor(starColor,  this,  "Change stars color");  if (clr.isValid()){  starColor = clr;  setBG(ui->graphicsView->scene());  }  } |
| *flyobject.cpp* |
| #include "flyobject.h"  FlyObject::FlyObject()  {  mass = 0.0;  x = y = vx = vy = 0.0;  radius = 0.0;  color = QColor("black");  others = QList<FlyObject \*>();  name = "";  }  FlyObject::FlyObject(QString name,  qreal x,  qreal y,  qreal vx,  qreal vy,  qreal mass,  qreal r,  QColor color,  qint32 type)  {  this->mass = mass;  this->x = x;  this->y = y;  this->vx = vx;  this->vy = vy;  this->name = name;  this->radius = r;  this->color = color;  this->type = type;  others = QList<FlyObject \*>();  }  FlyObject::~FlyObject()  {  others.clear();  }  QRectF FlyObject::boundingRect() const  {  return QRect(0,0,radius\*2,radius\*2);  }  void FlyObject::paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget)  {  Q\_UNUSED(widget);  Q\_UNUSED(option);  this->setPos(x-radius,y-radius);  QRectF rect = boundingRect();  QPen pen(color, 1);  QBrush brush(color);  painter->setPen(pen);  painter->setBrush(brush);  painter->drawEllipse(rect);  }  void FlyObject::calcAccelTo(FlyObject \*other)  {  others.append(other);  }  qreal FlyObject::fx(qreal local\_x)  {  qreal a = 0;  for (int i = 0; i < others.size(); ++i) {  qreal x = others.at(i)->x;  qreal y = others.at(i)->y;  qreal mass = others.at(i)->mass;  qreal r = std::hypot(x - local\_x, y - this->y);  a += mass \* (x - local\_x) / (r\*r\*r);  }  return a;  }  qreal FlyObject::fy(qreal local\_y)  {  qreal a = 0;  for (int i = 0; i < others.size(); ++i) {  qreal x = others.at(i)->x;  qreal y = others.at(i)->y;  qreal mass = others.at(i)->mass;  qreal r = std::hypot(x - this->x, y - local\_y);  a += mass \* (y - local\_y) / (r\*r\*r);  }  return a;  }  void FlyObject::calcX()  {  qreal k1 = SIM\_PRECISION \* fx(x);  qreal q1 = SIM\_PRECISION \* vx;  qreal k2 = SIM\_PRECISION \* fx(x + q1 / 2);  qreal q2 = SIM\_PRECISION \* (vx + k1 / 2);  qreal k3 = SIM\_PRECISION \* fx(x + q2 / 2);  qreal q3 = SIM\_PRECISION \* (vx + k2 / 2);  qreal k4 = SIM\_PRECISION \* fx(x + q3);  qreal q4 = SIM\_PRECISION \* (vx + k3);  vx += (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;  x += (q1 + 2 \* q2 + 2 \* q3 + q4) / 6;  }  void FlyObject::calcY()  {  qreal k1 = SIM\_PRECISION \* fy(y);  qreal q1 = SIM\_PRECISION \* vy;  qreal k2 = SIM\_PRECISION \* fy(y + q1 / 2);  qreal q2 = SIM\_PRECISION \* (vy + k1 / 2);  qreal k3 = SIM\_PRECISION \* fy(y + q2 / 2);  qreal q3 = SIM\_PRECISION \* (vy + k2 / 2);  qreal k4 = SIM\_PRECISION \* fy(y + q3);  qreal q4 = SIM\_PRECISION \* (vy + k3);  vy += (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;  y += (q1 + 2 \* q2 + 2 \* q3 + q4) / 6;  }  void FlyObject::updateXY()  {  calcX();  calcY();  others.clear();  }  qreal FlyObject::dist(FlyObject \*other)  {  return std::hypot((x - other->x),(y - other->y)) - (radius + other->radius);  } |

*Приведен текст наиболее важных исходников*

Результаты тестов

С программой поставляются конфигурационные файлы – тесты. Ниже приведены результаты работы некоторых из них.

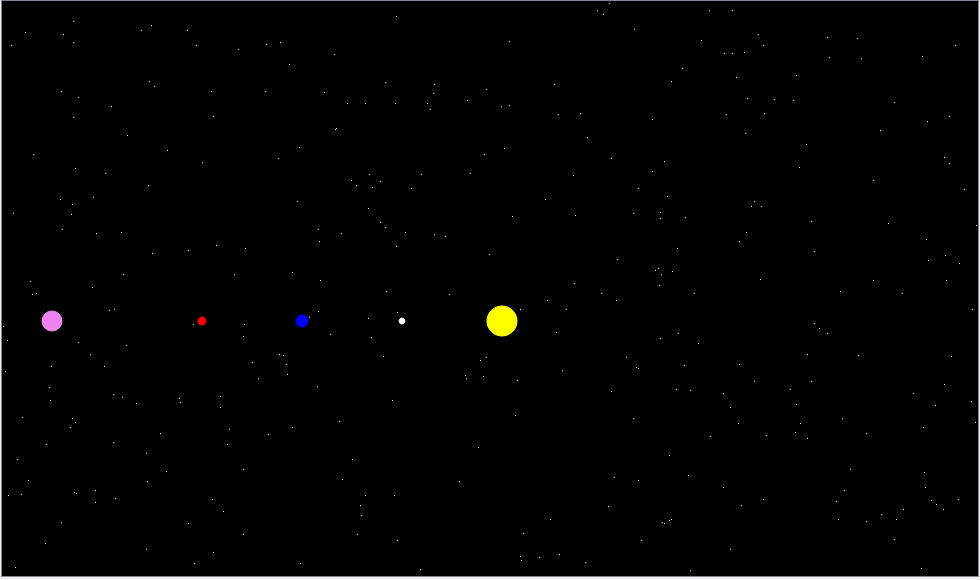


Рис. 1. main.ini (в момент запуска)

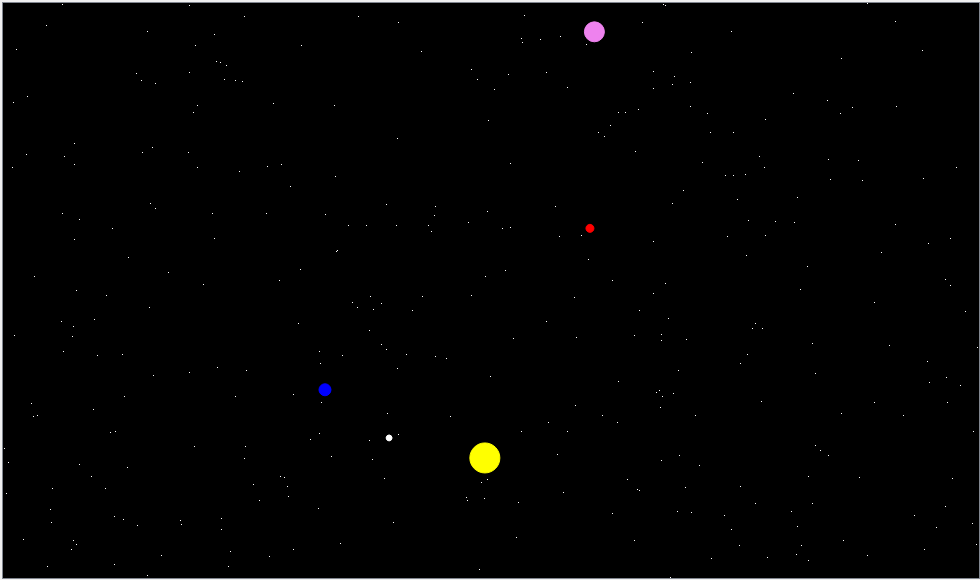


Рис. 2. main.ini (во время работы программы)

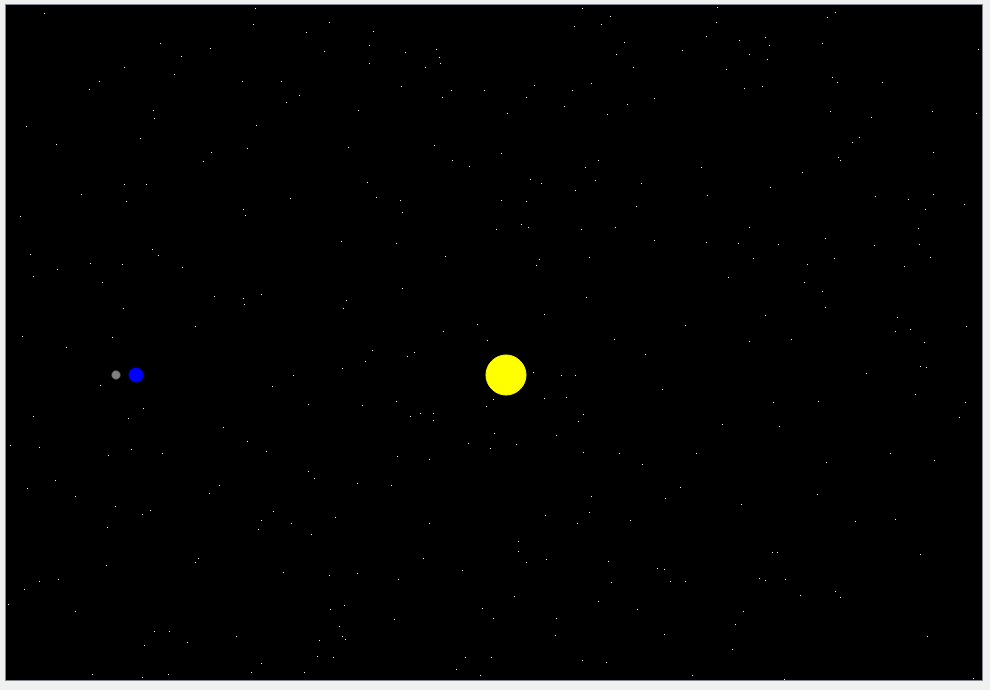


Рис. 3. satellite.ini (в момент запуска)

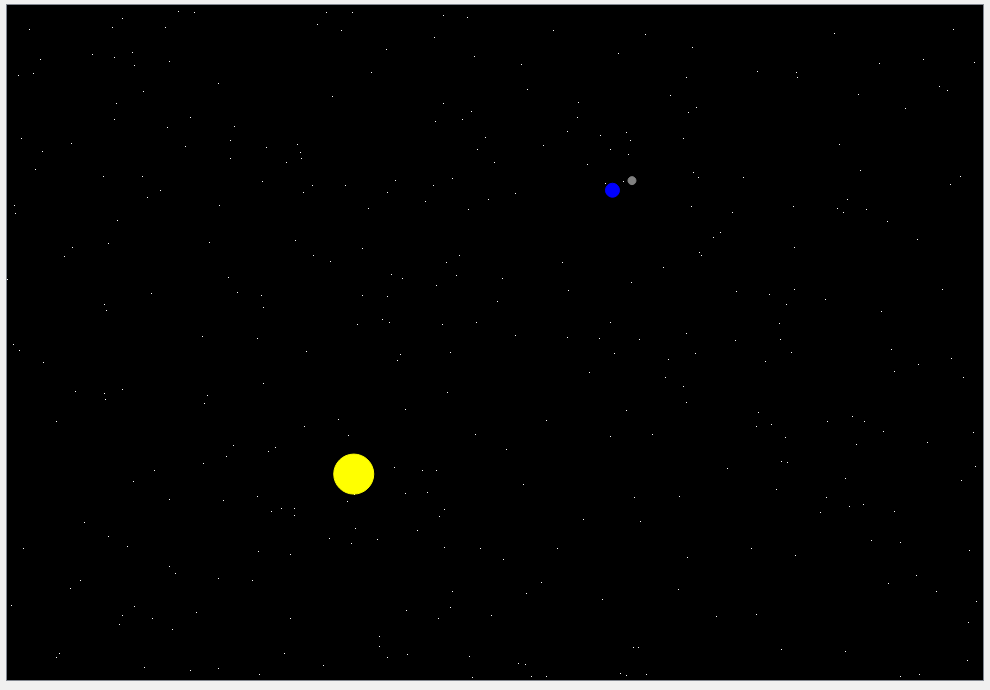


Рис. 4. satellite.ini (во время работы программы)

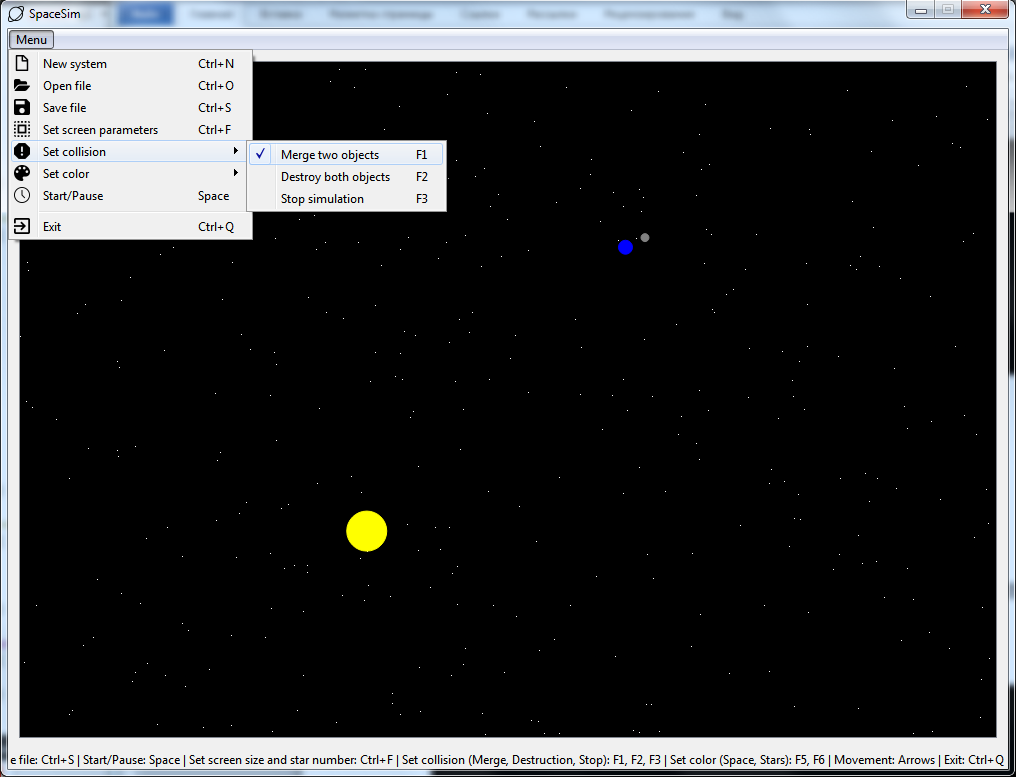


Рис. 5. satellite.ini (общий вид работы программы)