НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

КУРСОВИЙ ПРОЄКТ

з дисципліни:

“Математичне моделювання”

На тему “Modeling the Solar system”

Виконала:
студентка групи КМ-82
Арістова І.В.

Керівник:
Норкін Б. В.

КИЇВ – 2021

Зміст

ВСТУП

Математична модель - наближений опис будь-якого класу явищ зовнішнього світу, виражений за допомогою математичної символіки. Аналіз математичної моделі дозволяє проникнути в сутність явищ, це потужний метод пізнання зовнішнього світу, а також прогнозування та управління.

Метод математичного моделювання, що зводить вивчення явищ зовнішнього світу до математичного завдання, займає провідне місце серед інших методів дослідження. Він дозволяє проектувати нові технічні засоби, що працюють в оптимальних режимах, для вирішення складних завдань науки та техніки; проектувати нові явища. М. м. виявили себе як важливий засіб управління. Вони застосовуються в різних галузях знання, стали необхідним апаратом в області економічного планування та є важливим елементом автоматизованих систем управління.

Метою даної роботи є побудова моделі Сонячної системи. Розроблене програмне забезпечення дозволяє змінювати параметри об'єктів, вводити в систему нові об'єкти, тобто моделювати різні варіанти розвитку подій.

ПЗ було написано на мові Python, скомпільовано та запущено на операційній системі Windows. В програмі використовується набір модулів Pygame, призначений для написання комп'ютерних ігор та мультимедіа-додатків.

ТЕОРИТИЧНІ ВІДОМОСТІ

Типовим прикладом, що ілюструє характерні етапи у побудові М. м., є модель Сонячної системи. Спостереження зоряного неба почалися у давнину. Первинний аналіз цих спостережень дозволив виділити планети із усього різноманіття небесних світил. Отже, першим кроком було виділення об'єктів вивчення. Другим кроком стало визначення закономірностей їх рухів. Першою була модель Птоломея (2 ст. н. о.), що виходила зі становища, що планети та Сонце здійснюють рухи навколо Землі (геоцентрична модель).

М. Коперником у 1543 була запропонована принципово нова основа законів руху планет, що вважає, що планети обертаються навколо Сонця по колу (геліоцентрична система). Це була якісно нова (але нематематична) модель Сонячної системи. Однак не існувало параметрів системи (радіусів кіл і кутових швидкостей-руху), що приводять кількісні висновки теорії у відповідність до спостережень, так що М. Коперник був змушений вводити поправки в рухи планет по колу (епіцикли).

Наступним кроком у розвитку М. м. Сонячної системи були дослідження І.Кеплера (поч. 17 ст), який сформулював закони руху планет. Положення М.Коперника та І. Кеплера давали кінематичний опис руху кожної планети окремо, не торкаючись ще причин, що зумовлюють ці рухи.

Принципово новим кроком були роботи І. Ньютона, який запропонував динамічну модель Сонячної системи, що базується на законі всесвітнього тяжіння. Динамічна модель узгоджується з кінематичною моделлю, запропонованою І. Кеплером.

Закон всесвітнього тяжіння

В рамках класичної механіки гравітаційна взаємодія описується законом всесвітнього тяжіння Ньютона, який свідчить, що сила гравітаційного тяжіння F

між двома матеріальними точками маси m1 і m2, розділеними відстанню r, пропорційна обом масам і обернено пропорційна квадрату відстані - тобто:

$$F = G\frac{m_1 m_2}{r^2}$$

де G - гравітаційна постійна, рівна приблизно 6,67384×10^-11 Н×м2×кг^-2.

Рух планет

Припустимо, що у відомій точці планета розпочала свій рух і має певну швидкість. Вона рухається навколо Сонця якоюсь кривою, яку визначають за допомогою закону всесвітнього тяжіння Ньютона.

У деякий момент часу планета знаходиться в певному місці, на відстані r від Сонця; у цьому випадку відомо, що на неї діє сила, спрямована по прямій до Сонця, яка, згідно із законом тяжіння, дорівнює певній постійній, помноженій на добуток мас планети та Сонця та поділеній на квадрат відстані між ними.

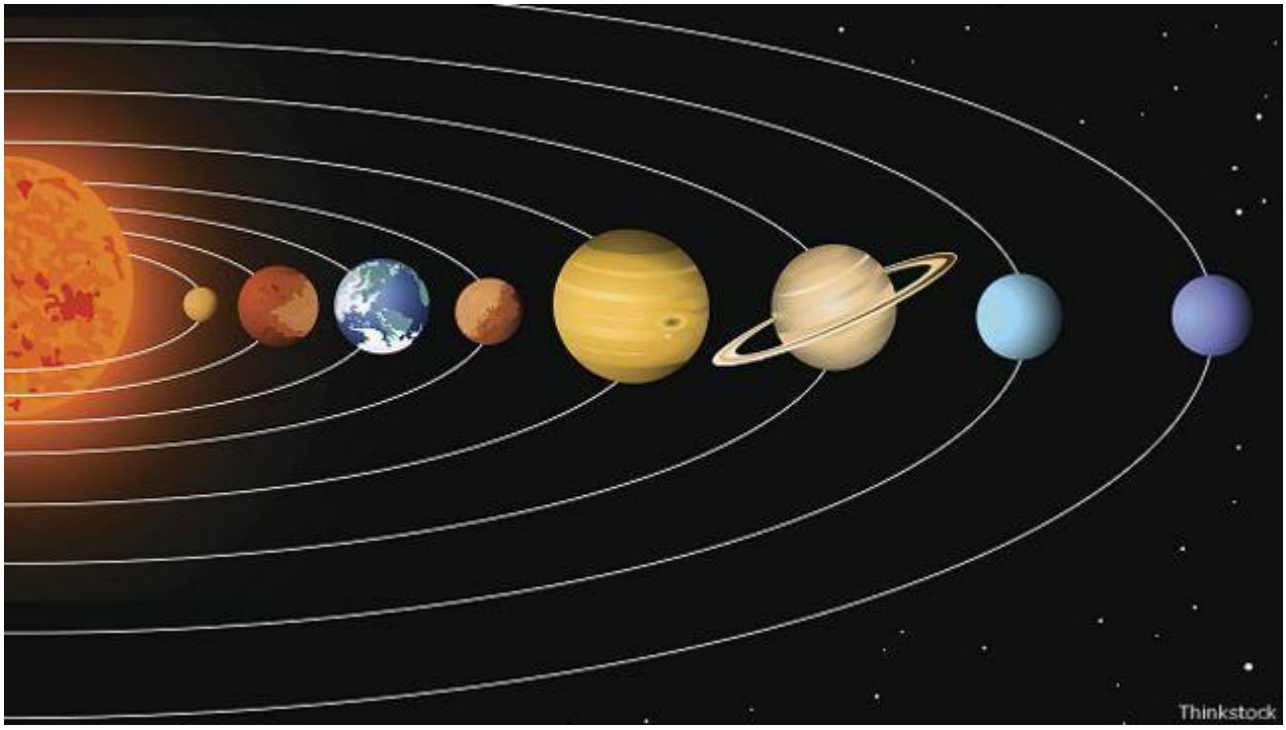
З цього закону були виведені x і y компоненти прискорення, які рівні відповідно x і y компонентам сили поділеної на масу об'єкта. Після спрощення формули вийшло, що:

ax = G * M * (x0-x) / r^3
ay = G * M * (y0-y) / r^3

де M – маса Сонця, (x0-x), (y0-y) – різниця між координатами Сонця та планети.

Для тіл Сонячної системи характерні дві ознаки:

- тіло за рахунок своєї кінетичної енергії не може подолати сили сонячного тяжіння та залишити Сонячну систему;
- тіло, що належить Сонячній системі, має постійно перебувати в області переважаючого тяжіння Сонця.



Модель Сонячної системи

ОПИС ПРОЕКТУ

Для розробки даного ПЗ було створено файли main.py, control.py, config.py, flyobj.py, де реалізується робота програми, та файли Sun_Earth_Moon.ini, System.ini та System_Asteroid.ini, де ініціалізовано параметри об’єктів.

Файл main.py: створює всі об'єкти та запускає програму.

Файл control.py: реалізує керування на екрані, проводить оновлення об'єктів та виводить на екран. За допомогою клавіш q та ESCAPE можна вийти із програми. За допомогою клавіш p та SPACE можна призупинити демонстрацію. Клавіші UP, DOWN, LEFT, RIGHT змінюють положення фокусування. Також передбачені сценарії роботи програми при зіткненні об’єктів: stop – при зіткненні програма зупиняється; remove – зіткнуті об’єкти видаляються з системи; join – зіткнуті об’єкти об’єднуються. Надалі використовується сценарій join.

Файл config.py: завантажує дані з файлів ini, які можна змінювати.

Файл flyobj.py: реалізує клас об'єктів і розраховує положення тіл за законом всесвітнього тяжіння, визначає вплив об'єктів один на одне.

Файл System.ini: ініціалізовано параметри повної Сонячної системи.

Файл System_Asteroid.ini: ініціалізовано параметри повної Сонячної системи та параметри пролітаючого астероїду.

Файл Sun_Earth_Moon.ini: ініціалізовано параметри системи Сонце – Земля – Місяць.

Алгоритм переміщення планети у полі гравітації зірки:

1. Перед початком задаємо початкове положення планети (x, y) та початкову швидкість (vx, vy)

2. На кожному кроці обчислюємо нове прискорення за формулою:

ax = G * M * (x0-x) / r^3

ay = G * M * (y0-y) / r^3

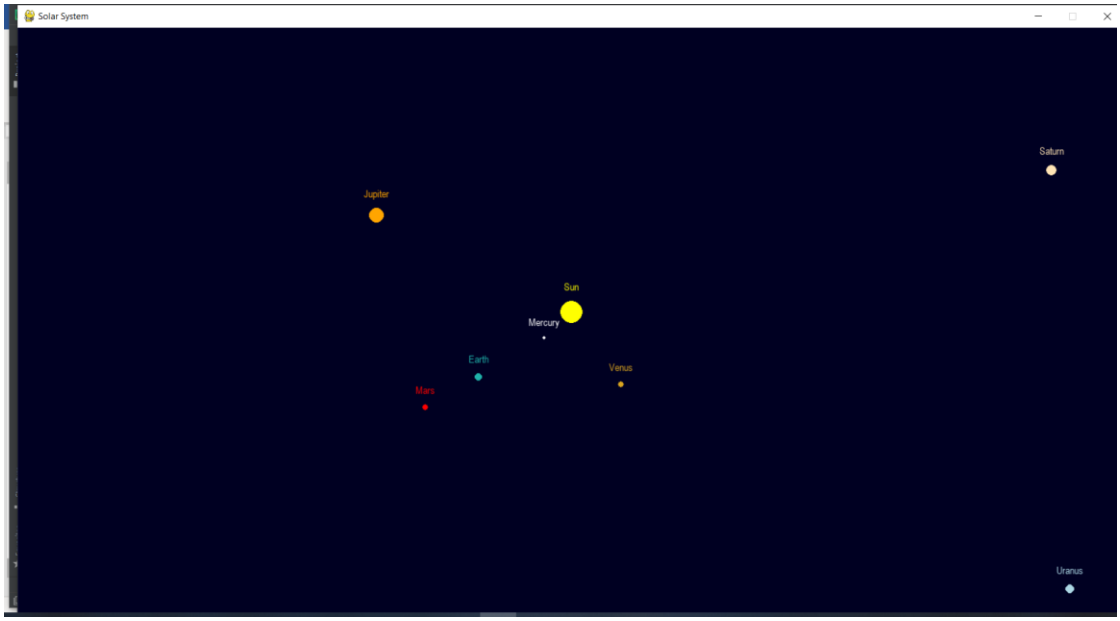
Перераховуємо швидкість та координати, використовуючи метод Рунге-Кутти:

k1 = T * fx	k1 = T * fy
q1 = T * vx	q1 = T * vy
k2 = T * fx(x + q1 / 2)	k2 = T * fy(y + q1 / 2)
q2 = T * (vx + k1 / 2)	q2 = T * (vy + k1 / 2)
k3 = T * fx(x + q2 / 2)	k3 = T * fy(y + q2 / 2)
q3 = T * (vx + k2 / 2)	q3 = T * (vy + k2 / 2)
k4 = T * fx(x + q3)	k4 = T * fy(y + q3)
q4 = T * (vx + k3)	q4 = T * (vy + k3)
vx := (k1 + 2 * k2 + 2 * k3 + k4) / 6	vy := (k1 + 2 * k2 + 2 * k3 + k4) / 6
x := (q1 + 2 * q2 + 2 * q3 + q4) / 6	y := (q1 + 2 * q2 + 2 * q3 + q4) / 6

Параметр T впливає на точність та швидкість обчислень.

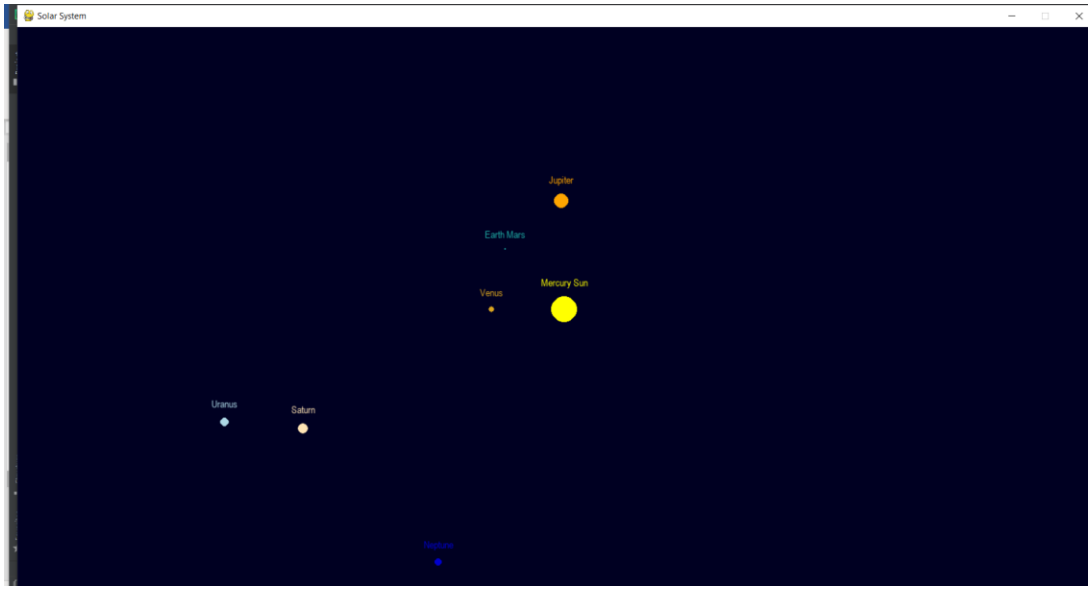
РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

При запуску програми з’являється вікно, де демонструється рух планет по орбітам. Зауважимо, що пропорції розміру об’єктів у відношенні один до одного не співпадають з реальними (для наглядності). Для побудови траєкторій руху використовуються початкові параметри об’єктів:



Класична модель Сонячної системи

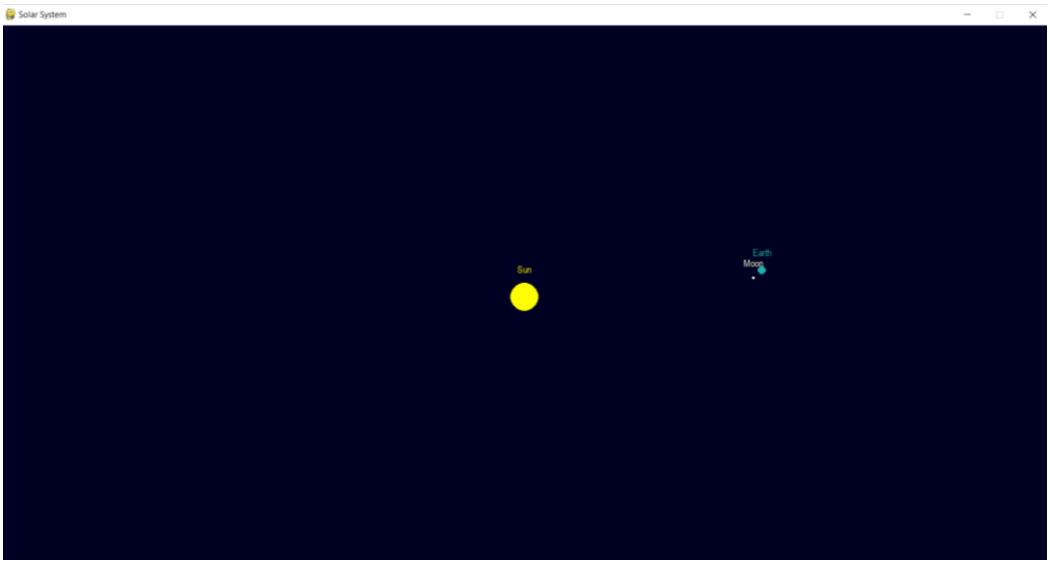
Для цієї моделі збільшимо масу Сонця:



Класична модель Сонячної системи зі збільшенням маси Сонця

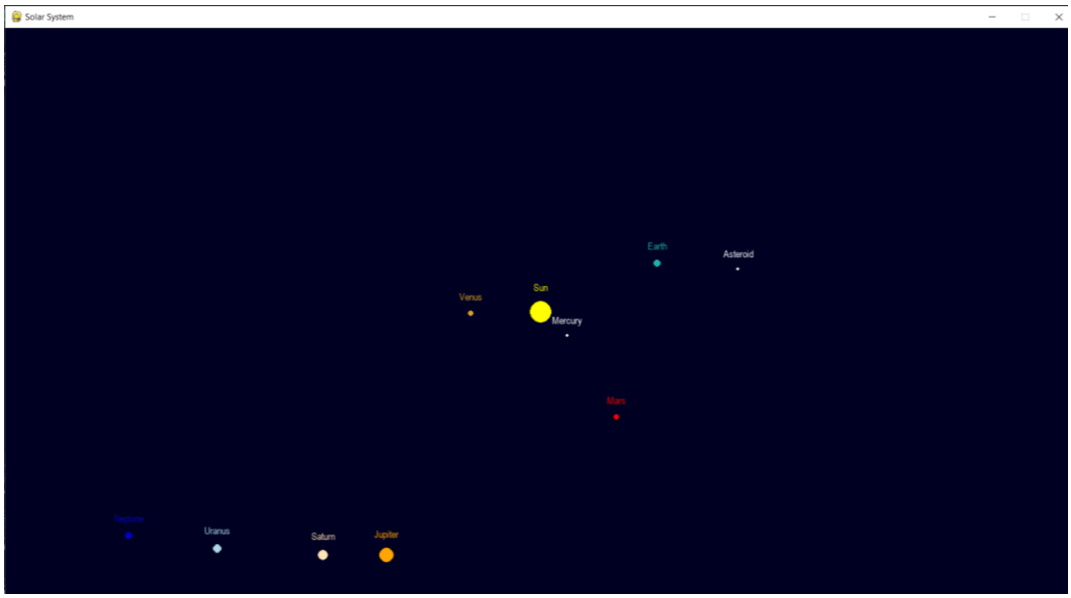
Можна побачити, що це призвело до зміни траєкторій руху об’єктів та подальшого знищення декількох планет.

Також розглянемо роботу мат. моделі на частині Сонячної системи, а саме Сонця, Землі та Місяця:



Модель Сонце – Земля – Місяць

Введемо пролітаючий астероїд в Сонячну систему та подивимось, як інші об’єкти впливають на його траєкторію:



Класична модель Сонячної системи та астероїд

Під час реалізації кожної моделі було зроблено запис екрану та викладено разом з проектом.

ВИСНОВКИ

У даній роботі було розроблено модель Сонячної системи, яка описує закон всесвітнього тяжіння, реалізовано можливість змінювати параметри об'єктів та вводити в систему нові об'єкти для моделювання різних варіантів розвитку подій. Цю математичну модель було опробувано на різних наборах даних, а саме: повна Сонячна система, система Сонце – Земля – Місяць та Сонячна система – Астероїд.

Дана програма надалі може використовуватися в навчально-освітньому процесі, а саме, на уроках фізики для ілюстрації роботи закону всесвітнього тяжіння. Згодом дана комп'ютерна програма може бути вдосконалена та покладена в основу майбутніх проектних робіт.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Лаплас П.С. Виклад системи світу. - Л.: Наука, 1982. - 376с.
2. Смульський І.І. Траєкторії при взаємодії двох тіл, що залежить від відносного відстані та швидкості// Математичне моделювання. – 1995
3. Newcomb S. The elements of the four inner planets and the fundamental constants of astronomy. Washington: Government printing office. 1895
4. Смульський І.І., Сєченов К.Є. Рівняння обертального руху Землі та їх вирішення при наявності впливу Сонця та планет / Інститут кріосфери Землі СО РАН. - Тюмень, 2007
5. PyGame Documentation: https://www.pygame.org/docs/
6. Solar: http://www.sai.msu.su/ng/solar/solar_about.html

ДОДАТОК А

Файл main.py:

```
import pygame
from control import Control
from flyobj import *
from config import *


def main():
    cfg = Config()

    # PyGame init
    pygame.init()
    screen = pygame.display.set_mode(cfg.getDisplay())
    pygame.display.set_caption("Solar System")

    # Space init
    bg = Surface(cfg.getDisplay())
    bg.fill(Color(cfg.getSpaceColor()))


    # Timer init
    timer = pygame.time.Clock()

    control = Control(timer, screen, bg, cfg)
    control.run()


if __name__ == "__main__":
    main()
```

Файл config.py:

```
import configparser
import argparse
import textwrap
from flyobj import *



class Config:
    width = 0
    height = 0
    starts = 0
    display = (0, 0)
    stopOnCollision = True
    star_colors = []
    generators = []

    def __init__(self):
        parser = argparse.ArgumentParser(description='Solar system simulator',
            formatter_class=argparse.RawDescriptionHelpFormatter,
            epilog=textwrap.dedent('''\
                Keys used:
                    q or ESC     = Exit
                    p or SPACE   = Pause
                    f            = Toggle fullscreen (if supported)
            '''))

        parser.add_argument('-f', '--file',
            dest='file',
```

```
            default='system.ini',
            help='configuration file')

        args = parser.parse_args()

        self.config = configparser.ConfigParser()
        self.config.read(args.file)

        sys = self.config['System']
        self.width = int(sys.get("WIN_WIDTH", 1500))
        self.height = int(sys.get("WIN_HEIGHT", 800))

        self.display = (self.width, self.height)


        self.space_color = sys.get("SPACE_COLOR")

        self.onCollision = sys.get("ON_COLLISION", "join")

        gens = sys.get("GENERATORS")
        if gens != None:
            self.generators = gens.split(',')



    def getSystem(self):
        s = []

        for i in self.config.sections():
            if i != "System" and not(i in set(self.generators)):
                obj = FlyObject(i,
                    int (self.config[i]["Mass"]),
                    float (self.config[i]["X"]),
                    float (self.config[i]["Y"]),
                    float (self.config[i]["VX"]),
                    float (self.config[i]["VY"]))

                obj.initSurface(int (self.config[i]["R"]),
                    self.config[i]["color"],
                    self.space_color)

                s.append(obj)

        return s

    def getDisplay(self):
        return self.display

    def getWidth(self):
        return self.width

    def getHeight(self):
        return self.height

    def getSpaceColor(self):
        return self.space_color
```

Файл control.py:

```
import pygame
import os
from pygame import *
from flyobj import *

CRASH_DIST = .5
OUT_DIST = 10000
```

```
STEPS = 5
KEY_MOVE_PX = 20


class Control:
    timer = ""
    cfg = ""
    screen = ""
    bg = ""
    screen_number = 0

    def __init__(self, timer, screen, background, cfg):
        self.timer = timer
        self.cfg = cfg
        self.screen = screen
        self.bg = background

    def run(self):
        r_min = R_MIN_MAX
        r_max = 0.0

        offset_x = 0
        offset_y = 0

        collapsed_object1 = ""
        collapsed_object2 = ""

        #Solar system init
        system = self.cfg.getSystem()

        done = False
        paused = False
        while not done:
            self.timer.tick(60)
            for e in pygame.event.get():
                if e.type == QUIT:
                    done = True
                    break
                if e.type == KEYDOWN:
                    if e.key == K_q or e.key == K_ESCAPE:
                        done = True
                        break
                    if e.key == K_p or e.key == K_SPACE:
                        paused = not paused
                    if e.key == K_f:
                        pygame.display.toggle_fullscreen()
                    if e.key == K_UP:
                        offset_y += KEY_MOVE_PX
                    if e.key == K_DOWN:
                        offset_y -= KEY_MOVE_PX
                    if e.key == K_LEFT:
                        offset_x += KEY_MOVE_PX
                    if e.key == K_RIGHT:
                        offset_x -= KEY_MOVE_PX

            if not paused:
                for st in range(STEPS):
                    for i in system:
                        if i.name == 'Sun':
                            continue
                        for j in system:
                            if i.name != j.name:
                                dist = i.dist(j)
                                dist -= (i.radius + j.radius)
                                i.calcAccelTo(j)
```

```
                                r_max = max(r_min, dist)

                                if dist < r_min:
                                    r_min = dist
                                    collapsed_object1 = i
                                    collapsed_object2 = j

                    #Join, crash or stop!
                    if r_min < CRASH_DIST:
                        if self.cfg.onCollision == "stop":
                            done = True
                            print("Collision detected")
                        elif self.cfg.onCollision == "remove":
                            print("Collision detected {0},
{1}".format(collapsed_object1.name, collapsed_object2.name))
                            system.remove(collapsed_object1)
                            system.remove(collapsed_object2)
                            r_min = R_MIN_MAX
                        elif self.cfg.onCollision == "join":
                            print("Join detected {0},
{1}".format(collapsed_object1.name, collapsed_object2.name))
                            system.append(join(collapsed_object1,
collapsed_object2))
                            system.remove(collapsed_object1)
                            system.remove(collapsed_object2)
                            r_min = R_MIN_MAX

                    for i in system:
                        i.update()

                #Put space to screen
                self.screen.blit(self.bg, (0, 0))

                #Put each object to screen
                for i in system:
                    i.draw(self.screen, offset_x, offset_y)

                #update screen
                pygame.display.update()


                if r_max > OUT_DIST:
                    print("Out of system")
                    done = True
                    break
```

Файл flyobj.py:

```
from pygame import *
import math

# Simulation precision.
# Lower value is better precision but lower simulation speed
T = 0.05


class FlyObject:
    mass = 0.0
    x, y, vx, vy, ax, ay = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
    radius = 0.0
    surfaceColor = "black"
    spaceColor = "black"
    others = []
    image = ""
```

```
    name = ''

    # Creates new flying object like planet or star
    def __init__(self, name, mass, x, y, vx, vy):
        self.mass = mass
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
        self.name = name
        self.others = []
        self.font = None
        self.text = None

        print("{0}, ({1}, {2}) v=({3}, {4}), mass={5}".format(name, x, y, vx,
vy, mass))

    def initSurface(self, radius, surfaceColor, spaceColor):
        self.radius = radius
        self.spaceColor = spaceColor
        self.surfaceColor = surfaceColor
        self.image = Surface((self.radius * 2, self.radius * 2))
        self.image.fill(Color(self.spaceColor))
        draw.circle(self.image, Color(surfaceColor), (self.radius, self.radius),
self.radius)
        self.font = font.Font(font.match_font('Arial'), 14)
        self.text = self.font.render(self.name, True, self.surfaceColor)

    # Distance to other F.O.
    def dist(self, other):
        return math.hypot((self.x - other.x),
                          (self.y - other.y))

    # Calculates acceleration to other object
    def calcAccelTo(self, other):
        self.others.append((other.mass, other.x, other.y))

    def fx(self, local_x):
        a = 0
        for (mass, x, y) in self.others:
            r = math.hypot(x - local_x, y - self.y)
            a += mass * (x - local_x) / r ** 3

        return a

    def fy(self, local_y):
        a = 0
        for (mass, x, y) in self.others:
            r = math.hypot(x - self.x, y - local_y)
            a += mass * (y - local_y) / r ** 3

        return a

    def calcX(self):
        k1 = T * self.fx(self.x)
        q1 = T * self.vx

        k2 = T * self.fx(self.x + q1 / 2)
        q2 = T * (self.vx + k1 / 2)

        k3 = T * self.fx(self.x + q2 / 2)
        q3 = T * (self.vx + k2 / 2)

        k4 = T * self.fx(self.x + q3)
        q4 = T * (self.vx + k3)
```

```
        self.vx += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        self.x += (q1 + 2 * q2 + 2 * q3 + q4) / 6

    def calcY(self):
        k1 = T * self.fy(self.y)
        q1 = T * self.vy

        k2 = T * self.fy(self.y + q1 / 2)
        q2 = T * (self.vy + k1 / 2)

        k3 = T * self.fy(self.y + q2 / 2)
        q3 = T * (self.vy + k2 / 2)

        k4 = T * self.fy(self.y + q3)
        q4 = T * (self.vy + k3)

        self.vy += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        self.y += (q1 + 2 * q2 + 2 * q3 + q4) / 6

    def update(self):
        self.calcX()
        self.calcY()
        self.others.clear()

    # Draw to screen
    def draw(self, screen, offset_x=0, offset_y=0):
        screen.blit(self.image, (int(self.x - self.radius) + offset_x,
int(self.y - self.radius) + offset_y))
        r = self.text.get_rect()
        r.centerx = int(self.x) + offset_x
        r.centery = int(self.y - self.radius) + offset_y - 20
        screen.blit(self.text, r)


def join(object1, object2):
    name = object1.name + " " + object2.name
    mass = object1.mass + object2.mass

    x = (object1.x * object1.mass + object2.x * object2.mass) / mass
    y = (object1.y * object1.mass + object2.y * object2.mass) / mass

    vx = (object1.vx * object1.mass + object2.vx * object2.mass) / mass
    vy = (object1.vy * object1.mass + object2.vy * object2.mass) / mass

    new_color = object1.surfaceColor
    if object2.mass >= object1.mass:
        new_color = object2.surfaceColor

    object3 = FlyObject(name, mass, x, y, vx, vy)
    radius = int((mass ** (1 / 3.0)) / 2)
    object3.initSurface(radius, new_color, object1.spaceColor)

    return object3
```

Файл System.ini:

[System]
WIN_WIDTH = 1500
WIN_HEIGHT = 800
SPACE_COLOR = #000022

```
[Sun]
X=750
Y=400
VX=0.0
VY=-0.05
Mass=20000
R=15
color=yellow


[Mercury]
X=700
Y=400
VX=0.0
VY=20
Mass=10
R=2
color=white


[Venus]
X=650
Y=400
VX=0.0
VY=15
Mass=15
R=4
color=goldenrod


[Earth]
X=600
Y=400
VX=0.0
VY=12
Mass=40
R=5
color=lightseagreen


[Mars]
X=500
Y=400
VX=0.0
VY=8
```

```
Mass=20
R=4
color=red


[Jupiter]
X=370
Y=400
VX=0.0
VY=7.5
Mass=60
R=10
color=orange

[Saturn]
X=300
Y=400
VX=0.0
VY=7
Mass=50
R=7
color=moccasin

[Uranus]
X=200
Y=400
VX=0.0
VY=6.5
Mass=30
R=6
color=lightblue

[Neptune]
X=100
Y=400
VX=0.0
VY=6
Mass=20
R=5
color=mediumblue
```

Файл Sun_Earth_Moon.ini:

```
[System]
WIN_WIDTH = 1500
```

```
WIN_HEIGHT = 800
SPACE_COLOR = #000022

[Sun]
X=750
Y=370
VX=0.0
VY=0.1
Mass=10000
R=20
color=yellow


[Earth]
X=450
Y=350
VX=0.0
VY=6.0
Mass=250
R=6
color=lightseagreen


[Moon]
X=435
Y=350
VX=0.0
VY=10
Mass=2
R=2
color=lightyellow
```

Файл System_Asteroid.ini:

```
[System]
WIN_WIDTH = 1500
WIN_HEIGHT = 800
SPACE_COLOR = #000022


[Sun]
X=750
Y=400
VX=0.0
VY=-0.05
Mass=20000
```

```
R=15
color=yellow


[Mercury]
X=700
Y=400
VX=0.0
VY=20
Mass=10
R=2
color=white

[Asteroid]
X=600
Y=0
VX=0.0
VY=10
Mass=5
R=2
color=white

[Venus]
X=650
Y=400
VX=0.0
VY=15
Mass=15
R=4
color=goldenrod


[Earth]
X=600
Y=400
VX=0.0
VY=12
Mass=40
R=5
color=lightseagreen


[Mars]
X=500
Y=400
VX=0.0
```

```
VY=8
Mass=20
R=4
color=red


[Jupiter]
X=370
Y=400
VX=0.0
VY=7.5
Mass=60
R=10
color=orange

[Saturn]
X=300
Y=400
VX=0.0
VY=7
Mass=50
R=7
color=moccasin

[Uranus]
X=200
Y=400
VX=0.0
VY=6.5
Mass=30
R=6
color=lightblue

[Neptune]
X=100
Y=400
VX=0.0
VY=6
Mass=20
R=5
color=mediumblue
```