

**BIL142 Bilgisayar Proglamlama II**

**DÖNEM PROJESİ RAPORU**

**Ahmet Mete Gümüş - 231201069**

**Muhammed Halil Güler 191204082**

### ****Projenin Hedeflenen Program Özeti****

Bu proje, üç cisim problemini temel alarak, gök cisimlerinin evren içerisinde birbirlerine uyguladıkları kütle çekim kuvvetini hesaplayan ve hareketlerini simüle eden bir C++ dili ile yazılmış simülasyondur. Roketlerin dinamik hareketi, gezegenlerin yörüngesi ve asteroidlerin serbest hareketi gibi karmaşık fiziksel olaylar, nesne yönelimli programlama anlayışı çerçevesinde modellenmiştir.

Projenin başlıca hedefleri:

* Fiziksel cisimlerin kütle çekim kuvvetini hesaplama ve buna dayalı işlemler.
* Taylor serisi yaklaşımı kullanarak hareket güncellemeleri yapma.
* Roketlerin ek kuvvet oluşturma ve kütle kaybını simüle etme.
* Cisimlerin yörünge hareketlerini Canvas grafik kütüphanesi yardımıyla çizime aktarma.

Bu projede yazılan kod , bellek yönetimi açısından dikkatli hazırlanmış ve yapılan işlemlerin doğruluğu fiziksel formüllerle uyuşmaktadır.

### ****Nesne Yönelimli Programlama Yapısı****

Nesne yönelimli programlama mantığı bu projede şu çerçevede kullanılmıştır:

* Her fiziksel nesne (örneğin, gezegen, roket, asteroid) bir "sınıf" olarak tanımlanmıştır.
* Sınıflar arası kalıtım, ortak özelliklerin tekrar edilmesini önlemek için kullanılmıştır.
* Dinamik bağlama ve polimorfizm, farklı nesnelerin benzer davranışlarının tek bir arabirimle kontrol edilmesini sağlamıştır.

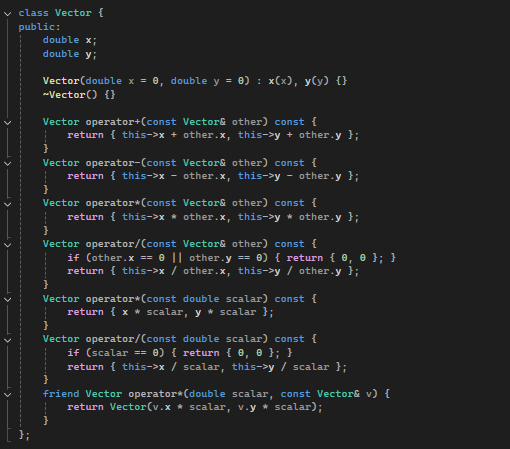
Ayrıca nesne yönelimli programlama mantığı grup içerisinde de görev dağılımı yaparken kullanılmıştır. Kodların genel işleyiş tasarımı beraber yapılmasına karşılık, Asteroid, Rocket gibi sınıflar görev dağılımı yapılarak 2 kişi tarafından paralel olarak koda entegre edilmiştir. Bu bağlamda gerektiğinde kod üzerinde canlı olarak beraber çalışabilmek için Clion IDE özelliği olan Code With Me gibi özellikler de aktif olarak kullanılmıştır.

#### ****Programda Kullanılan Sınıfların (Class) Açıklamaları****

#### ****3.1 Vector Sınıfı****

* Tanım: 2 boyutlu vektör işlemleri için kullanılır. Fiziksel kuvvetler, hızlar ve konumlar bu sınıf ile temsil edilir.
* Kullanım Amacı: Kütle çekimi, hız, ivme gibi fiziksel büyüklükler üzerinde vektör aritmetiği yapmak.

**Kod Bloğu:**

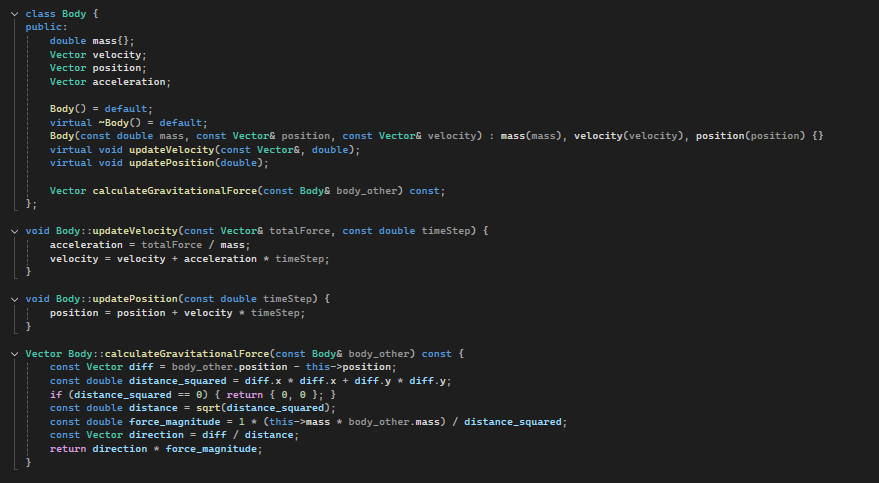


**Açıklama:**

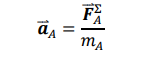
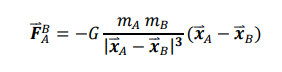
* **Toplama:** İki vektörün toplamını hesaplar ve döndürür.
* **Çıkarma:** İki vektör arasındaki farkı bulur ve döndürür.
* **Skaler Çarpma ve Bölme:** Vektörün bileşenlerini bir sabit ile çarpar ya da böler ve döndürür.

#### ****3.2 Body Sınıfı****

**Tanım:** Temel fiziksel cisimleri (örneğin, asteroid, gezegen , roket) temsil eden bir sınıf.  
Body sınıfı simülasyona eklenecek her bir cismin bağlı olduğu sınıftır.

**Kod Bloğu:** ****

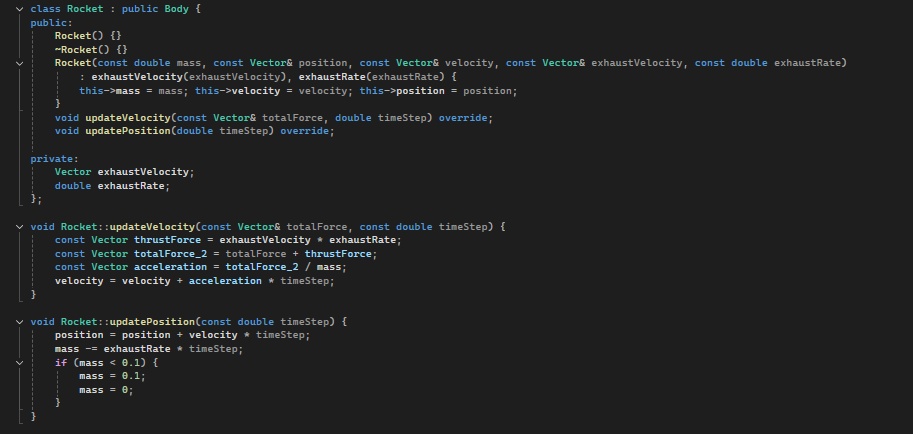
**Açıklama:**

* **updateVelocity:** Newton’un ikinci hareket yasasına dayanarak hızı günceller.
* 
* **updatePosition:** Yeni hızı kullanarak cismin konumunu hesaplar.
* **calculateGravitationalForce:** İki cisim arasındaki kütleçekim kuvvetini hesaplamak için kullanılır. Fonksiyon , Newton’un Evrensel Çekim Yasası’na dayanarak çalışır :
* ****
  1. Konum vektörü farkını hesaplar.
  2. Uzaklığın karesini hesaplar.
  3. Sıfıra bölünme kontrolü yapar.
  4. Uzaklığı hesaplar.
  5. Gravitasyonel kuvvetin büyüklüğünü hesaplar.
  6. Yön vektörünü bulur ve sonuç kuvveti döndürür.

#### ****3.3 Rocket Sınıfı****

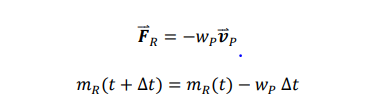
**Tanım:** Body sınıfından türetilen ve roketleri temsil eden sınıf. Roketlerin dinamik kütle kaybı ve itki kuvvetleri gibi ek özellikleri ve bu özellikleri bünyesinde barındırarak işlem yapan fonksiyonları bulunur.

**Kod Bloğu:**



**Açıklama:**

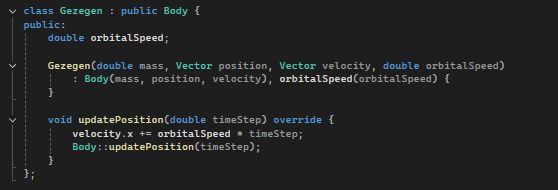
* **updateVelocity:** Roketin motor çıkışından oluşan itki kuvvetini ekler.
* **updatePosition:** Roketin kütlesini ve konumunu günceller.



#### ****3.4 Gezegen Sınıfı****

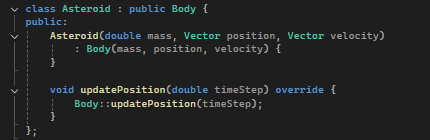
**Tanım:** Gezegenleri temsil eder, Body sınıfından türetilmiştir. Body sınıfına ek olarak gezegenlerin yörünge hızını da bünyesinde barındırmaktadır.

Not: Gezegen sınıfı proje tanımına uygun olarak , sonradan eklenmiş bir gökcismidir.



#### ****3.5 Asteroid Sınıfı****

* **Tanım:** Asteroidleri temsil eder, Body sınıfından türetilmiştir.
* Not: Asteroid sınıfı proje tanımına uygun olarak , sonradan eklenmiş bir gökcismidir.

**Kullanım Amacı:** Yörüngesiz hareket eden asteroidleri modellemek 

#### ****3.6 Universe Sınıfı****

**Tanım:** Sistemin tamamını temsil eden sınıf , simülasyona eklenen cisimlerin bulunduğu evreni ifade eder. Cisimlerin eklenmesini, kuvvet hesaplamasını ve hareket güncellemelerini kontrol eder , zaman adımlarını kullanarak simülasyonun her bir adımını hesaplar.

**Kod Bloğu:**



**Açıklama:**

* **addBody:** Sisteme yeni cisim ekler.
* **calculateForces:** Tüm cisimler arasındaki kuvvetleri hesaplar.
* **updatePositions:** Cisimlerin konumlarını günceller.
* **step:** Tüm simülasyon adımlarını tek bir fonksiyonla yürütür.

Aşağıda sınıflardaki üye değişkenler ve fonksiyonların kullanım amaçları özetlenmiştir

| **Sınıf** | **Üye Değişken/Üye Fonksiyon** | **Kullanım Amacı** |
| --- | --- | --- |
| Vector | x, y | Vektör bileşenlerini saklamak. |
|  | operator+, operator-, operator\* | Vektör işlemleri. |
|  |  |  |
| Body | mass, velocity, position, acceleration | Cismin fiziksel özelliklerini temsil eder. |
|  | updateVelocity, updatePosition | Hız ve konumu günceller. |
|  | calculateGravitationalForce | Kütle çekim kuvvetini hesaplar. |
|  |  |  |
| Rocket | exhaustVelocity, exhaustRate | Roketin püskürtme hızı ve oranı. |
|  | updateVelocity, updatePosition | Rokete özgü hız ve konum güncellemeleri. |
|  |  |  |
| Universe | timeStep, bodies | Zaman adımı ve evrendeki cisimler. |
|  | addBody, calculateForces, step | Simülasyonu yönetmek. |

### ****Program Akışı****

**1. Başlangıç**

* Program çalıştırıldığında, kullanıcıdan bir simülasyon ismi istenir. Bu isim, simülasyon çıktısını kaydetmek için kullanılacak dosyanın adıdır.
* Kullanıcıdan, simülasyona eklemek istediği cisimlerin ve roketlerin sayısı gibi temel bilgiler istenir.

**2. Cisimlerin ve Roketlerin Sisteme Eklenmesi**

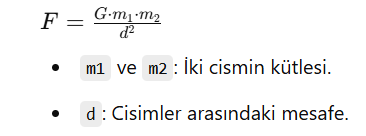
* **Kullanıcıdan bilgi alınır:** Program, her bir cismin veya roketin kütlesi, başlangıç konumu ve hızı gibi fiziksel özelliklerini terminalden alır.
* **Nesneler oluşturulur:** Alınan bilgilerle Body, Rocket, Gezegen veya Asteroid sınıflarından bir nesne oluşturulur.
* **Evrene eklenir:** Bu nesneler, Universe sınıfına eklenir ve cisimler dinamik olarak bir vektörde tutulur.

**3. Simülasyonun Başlatılması**

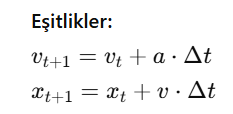
* Simülasyon, belirli bir zaman adımı (örneğin, 1 saniye) ile başlatılır.
* Zaman adımı (timeStep) programın doğruluğu ve performansı arasında bir denge kurar.
* Her simülasyon adımında:
  1. **Kütle Çekim Kuvvetlerinin Hesaplanması:** Evrendeki tüm cisimlerin birbirlerine uyguladıkları kütle çekim kuvvetleri hesaplanır.
  2. **Hız ve Konum Güncellemeleri:** Her bir cismin hızı ve konumu, hesaplanan kuvvetlere göre güncellenir.

**4. Fiziksel Hesaplamalar**

* **Kütle Çekim Kuvvetleri:** Her bir cismin diğer cisimlere uyguladığı kütle çekim kuvvetleri, Newton'ın Kütle Çekim Yasası'na göre hesaplanır:

****

* + m1 ve m2: İki cismin kütlesi.
  + d: Cisimler arasındaki mesafe.
* **Hız ve Konum Güncellemeleri:** Taylor serisi yaklaşımlarıyla hız ve konum güncellenir:

****

**5. Roketlerin Özel Hesaplamaları**

* Roketler, Newton yasalarına ek olarak Tsiolkovsky Roket Denklemleri ile modellenir.
* Roketlerin itki kuvveti, püskürtme hızı ve kütle azalması gibi faktörler hesaplamalara eklenir:
  + Püskürtme hızı (exhaustVelocity) ve püskürtme oranı (exhaustRate) roketin hızını etkiler.
  + Kütle azalması süreklidir ve her adımda güncellenir.

**6. Grafiksel Çıktı**

* Simülasyon boyunca her cismin konumu, canvas sınıfı kullanılarak bir grafik dosyasına işlenir.
* Farklı türdeki cisimler için farklı renkler atanır (örneğin, kırmızı roket için, yeşil gezegen için).

**7. Simülasyon Sonu**

* Simülasyon tamamlandığında program kullanıcıya bilgi verir.
* Sonuçlar, grafik dosyası olarak kaydedilir ve tüm dinamik olarak oluşturulan nesneler bellekten temizlenir.

### ****Performans ve Bellek Yönetimi****

* **Dinamik Bellek Yönetimi:**

Kodda dinamik bellek yönetimi kullanılmıştır, yani program çalışırken bellek ayrımı yapılır ve gerektiğinde bu bellek serbest bırakılır.

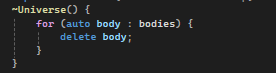
**a) Bellek Tahsisi:**

Cisimler (Body sınıfı nesneleri) new anahtar kelimesi kullanılarak oluşturulur.

Heap üzerinde bellek tahsis edilmesi, belleğin uzun ömürlü olmasını sağlar.

**b) Bellek Serbest Bırakma:**

Universe sınıfında dinamik olarak oluşturulan Body nesnelerinin bellekten silinmesi, ~Universe destructor’ında yapılır.



delete: delete operatörü, new ile tahsis edilen belleği serbest bırakır ve bu belleği tekrar kullanılabilir hale getirir.

Döngü, bodies vektöründe bulunan tüm Body\* işaretçilerini tek tek siler.

**Performans Optimizasyonu**

* Kuvvet hesaplamaları için döngü yapısı optimize edilmiştir.

**Kuvvet Hesaplamalarının Optimizasyonu**

* **Açıklama:**
  + Kütle çekim kuvvetlerinin hesaplanmasında kullanılan calculateGravitationalForce fonksiyonu, Newton’ın Kütle Çekim Yasası’na uygun bir şekilde optimize edilmiştir.
  + Hesaplamalarda kullanılan mesafe karesinin (distance\_squared) sıfır kontrolü ve karekökün yalnızca gerektiğinde alınması gibi önlemler performansı artırır.
* **Performans Etkisi:**
  + Gereksiz karekök hesaplamaları önlenerek işlem süresi kısaltılmıştır.
  + Mesafe karesi sıfır olduğunda hesaplama atlanır, bu da bölme işleminden kaynaklanan hataları ve ek hesaplama yükünü azaltır.

**Polymorphism ile Tek Döngüde İşlem**

* **Açıklama:**
  + Polymorphism sayesinde, farklı türdeki cisimlerin (Body, Rocket, Gezegen, vb.) aynı döngüde işlenmesi sağlanmıştır.
  + Bu, cisim türüne göre ayrı döngüler yazılmasının önüne geçerek kodun hem okunabilirliğini hem de işlem verimliliğini artırır.
* **Performans Etkisi:**
  + Tüm cisimler, aynı işlem sırasına dahil edilerek döngülerin toplam çalışma süresi düşürülmüştür.

**Sanal Fonksiyonlar ile Davranış Yönetimi**

* **Açıklama:**
  + updateVelocity ve updatePosition gibi fonksiyonlar sanal olarak tanımlanmıştır. Bu sayede her sınıf, kendine özgü bir şekilde davranış sergiler.
  + Sanal fonksiyonlar, çalışma zamanında yalnızca ihtiyaç duyulan davranışın çağrılmasını sağlar.
* **Performans Etkisi:**
  + Farklı türdeki cisimler için gereksiz işlem yapılmaz, çünkü sadece ilgili sınıfın davranışı uygulanır.
  + Dinamik bir yapıda çalışırken esneklik sağlar.

**Sadeleştirilmiş Matematiksel İşlemler**

* **Açıklama:**
  + Fiziksel hesaplamalar sadeleştirilerek daha hızlı bir algoritma elde edilmiştir:
    - Vector sınıfındaki operatör aşırı yüklemeleri (operator+, operator-, vb.) sayesinde işlemler kısa ve okunabilir bir şekilde yazılmıştır.
    - Mesafe karelerinin doğrudan hesaplanması gibi önlemler işlem yükünü azaltır.
* **Performans Etkisi:**
  + İşlem sayısını azaltarak hesaplama sürelerini kısaltır.
  + Daha temiz ve okunabilir kod, hata oranını düşürür.

**Gereksiz İşlemlerin Atlanması**

* **Açıklama:**
  + Simülasyonda mesafe sıfır olduğunda kuvvet hesaplamaları atlanmıştır.
  + Roket kütlesi sıfıra yaklaştığında (mass < 0.1), kütle güncellemesi durdurulur.
* **Performans Etkisi:**
  + Gereksiz işlem yapılmasının önüne geçilerek simülasyon daha hızlı tamamlanır.
  + Bölme gibi maliyetli işlemler sıfır kontrolü ile optimize edilmiştir.

### ****Kalıtım ve Polimorfizm****

**Kalıtım (Inheritance)**

**1. Kalıtımın Kullanımı**

* Programdaki temel sınıf olan Body, tüm fiziksel cisimleri temsil eder ve Rocket, Gezegen, Asteroid gibi özel türler bu sınıftan türetilmiştir.
* **Amaç:** Ortak özellikleri ve davranışları bir temel sınıfta tanımlayarak kod tekrarını azaltmak. Örneğin, mass, position, velocity gibi değişkenler ve updateVelocity, updatePosition gibi fonksiyonlar tüm cisimlerde ortaktır.

**2. Kalıtılan Sınıflar**

**a) Body Sınıfı**

* **Temel Sınıf:** Tüm cisimlerin temel özelliklerini içerir.
* **Özellikler:**
  + **Değişkenler:** mass, position, velocity, acceleration.
  + **Fonksiyonlar:** updateVelocity, updatePosition, calculateGravitationalForce.

**b) Rocket Sınıfı**

* **Kalıtım:** Body sınıfından türetilmiştir.
* **Özellikler:**
  + Ek özellikler: exhaustVelocity (püskürtme hızı), exhaustRate (yakıt tüketim oranı).
  + Davranışların özelleştirilmesi: updateVelocity ve updatePosition fonksiyonları, roketlerin itki kuvvetini ve kütle azalmasını hesaba katmak için yeniden tanımlanmıştır.

**c) Gezegen Sınıfı**

* **Kalıtım:** Body sınıfından türetilmiştir.
* **Özellikler:**
  + Ek özellik: orbitalSpeed (yörünge hızı).
  + Davranış özelleştirmesi: updatePosition fonksiyonu, gezegenin yörünge hızını hesaba katar.

**d) Asteroid Sınıfı**

* **Kalıtım:** Body sınıfından türetilmiştir.
* **Özellikler:**
  + Hiçbir ek özellik içermez.
  + updatePosition fonksiyonu, temel sınıfın davranışını kullanır.

**3. Kalıtımın Avantajları**

* **Kod Tekrarını Azaltır:** Tüm ortak özellikler ve davranışlar Body sınıfında tanımlanmış, böylece türetilen sınıflarda tekrar yazılmamıştır.
* **Genel Yapı Sağlar:** Body sınıfı, diğer tüm cisimler için genel bir çatı sunar. Örneğin, tüm cisimlerin kütlesi, hızı ve konumu bu sınıfta tutulur.
* **Esneklik:** Her türetilmiş sınıf (Rocket, Gezegen, Asteroid), yalnızca kendine özgü davranışları tanımlayarak temel yapıdan ayrılabilir.

**Polymorphism (Çokbiçimlilik)**

**1. Polymorphism'in Kullanımı**

* **Amaç:** Body türündeki göstericiler (Body\*) aracılığıyla farklı türetilmiş sınıfların (örneğin, Rocket, Gezegen, Asteroid) özel davranışlarının dinamik olarak çağrılmasını sağlamak.
* **Uygulama:** updateVelocity ve updatePosition gibi fonksiyonlar, türetilen sınıflarda override edilmiştir. Bu, her bir sınıfın kendi özel davranışlarını sergilemesine olanak tanır.

**2. Dinamik Fonksiyon Çağrıları**

* **Sanallık:** Body sınıfındaki updateVelocity ve updatePosition fonksiyonları virtual olarak tanımlanmıştır.
* **Çalışma Zamanında Davranış:** Body\* türündeki göstericiler, Rocket, Gezegen veya Asteroid nesnelerine işaret ettiğinde, ilgili sınıfın override edilmiş fonksiyonu çağrılır.

**3. Örnek: Polymorphism ile Farklı Davranışlar**

* **Scenario:** Bir Body\* gösterici aynı döngü içinde Rocket, Gezegen ve Asteroid nesnelerine işaret edebilir. Ancak her sınıfın kendi davranışı uygulanır.

**Rocket için:**

* updateVelocity: İtki kuvveti ve kütle azaltma hesaplanır.
* updatePosition: Roket kütlesi azaltılarak yeni konumu hesaplanır.

**Gezegen için:**

* updatePosition: Gezegenin yörünge hızı hesaba katılır.

**4. Polymorphism'in Avantajları**

* **Dinamik Davranış Yönetimi:** Tüm nesneler Body\* göstericisi üzerinden aynı döngüde işlenebilir. Her nesne kendi davranışını sergiler.
* **Genişletilebilirlik:** Yeni bir tür (örneğin, Star sınıfı) eklenmek istendiğinde, sadece Body sınıfından türetilip ilgili fonksiyonlar override edilir.
* **Kodun Sadeleştirilmesi:** Kullanıcı, farklı türdeki nesneler için ayrı ayrı işlemler yazmak zorunda kalmaz.

**Sonuç**

Bu program, **kalıtım** ve **polymorphism** kavramlarını şu yönlerden etkin bir şekilde kullanmaktadır:

1. **Kalıtım:**
   * Ortak özellikler ve davranışlar Body sınıfında tanımlanmıştır.
   * Özel davranışlar türetilmiş sınıflarda (Rocket, Gezegen, Asteroid) implement edilmiştir.
2. **Polymorphism:**
   * Body\* türündeki göstericiler, tüm cisimlerin uygun davranışlarını dinamik olarak çağırır.
   * Simülasyonun esnekliği ve genişletilebilirliği artırılmıştır.

### Farklı sayıda ve başlangıç değerleri ile sisteme eklenecek cisim ve roketlerin izledikleri yörüngelere ait çizimler:

Farklı sayılarda ve başlangıç değerleri ile sisteme eklenen cisim ve roketlerin izledikleri yörüngelerin grafik çıktıları(10adet) ekte(Ek.1) sunulmuştur.

### Programa eklenen yeni özellikler

1) Programa 2 yeni cisim class’ı eklendi. (Gezegen ve Asteroid)

2) Kullanıcı dostu , göz yormayan temiz bir arayüz geliştirildi.

3) Simülasyona eklenen cisimlerin verilerini terminal ekranından girdi olarak alma yeniliği getirildi.

4) Terminal ekranından girilen verilerin istenilen şekilde olup olmadığını kontrol eden kod bloğu eklendi. (Örneğin: Cisim verilerini girerken double tipinde bir değişken girilmemesi durumunda sistem buna izin vermeyerek kullanıcıdan tekrardan örneğe uygun bir şekilde verileri girmesini ister.)

### Benzetimden elde edilebilecek fiziksel gözlemler

Bu kodun simülasyonu Newton’un Evrensel Kütleçekim Yasası’nı temel alınarak geliştirildiği için, bir çok fiziksel gözlemler elde edilebilir. Bu gözlemler, cisimler arasındaki etkileşimlerin ve hareketlerin incelenmesini sağlar.

Aşağıda bu kodun fiziksel gözlemler açısından neler sunabileceği detaylı olarak açıklanmıştır:

**1. Ikili ve Çoklu Cisim Hareketleri**

**•İki Cisim Problemi:**

•İki kütle arasındaki etkileşim Evrensel Kütleçekim Yasasına göre modellenir:

•Bu yasa kullanılarak, iki cismin birbirleri etrafındaki yörünge hareketi (örneğin, Dünya-Güneş sistemi gibi) gözlemlenebilir.

**•Örnek Gözlemler:**

•Yörünge şekilleri (dairesel, eliptik, parabolik vb.).

•Yörünge periyotları ve hızları.

•Cisimlerin zaman içindeki konum değişimleri.

**•Üç veya Daha Fazla Cisim Problemi:**

•İki veya daha fazla cisim arasında etkileşim olduğunda çoklu cisim hareketleri gözlemlenir.

•Üç veya daha fazla cisim arasındaki etkileşimler, sistemde kaotik ve karmaşık davranışlar oluşturabilir.

**•Örnek Gözlemler:**

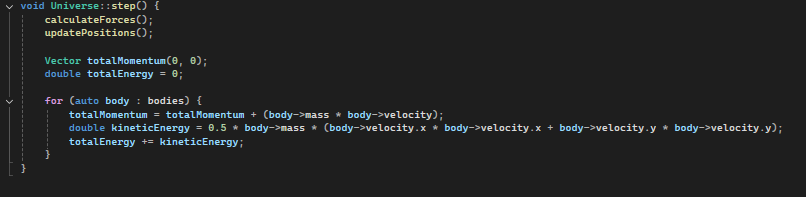
•Cisimlerin birbirleriyle etkileşiminden kaynaklanan dengesiz yörüngeler.

•Sistemin zamana bağlı kararsızlığı (örneğin, bir cismin sistemden fırlatılması).

•Kaotik davranışların ortaya çıkışı.

**2. Momentum ve Enerji Korunumu**

•Kodda sistemin toplam momentumu ve kinetik enerjisi hesaplanmaktadır.

**** •Momentum Korunumu:**

•Kütleçekim kuvvetleri iç kuvvetlerdir. Sistemde harici bir kuvvet olmadığında, toplam momentum korunur.

•Bu gözlem, simülasyonun doğruluğunu test etmek için kullanılabilir.

**•Enerji Korunumu:**

•Sistemde toplam enerji, kinetik enerji ve potansiyel enerji toplamından oluşur:

•Kütleçekim kuvvetleri nedeniyle potansiyel enerji değişse bile toplam enerji korunmalıdır (sayısal yöntem hataları haricinde).

**•Gözlemler:**

•Zaman içinde toplam momentumun sabit kalıp kalmadığı.

•Toplam enerjide bir sapma olup olmadığı.

•Sistemin kararlılığına bağlı olarak enerji transferleri.