

**Trabalho Prático 2**

**Licenciatura:**

Engenharia Informática e Multimédia

**Unidade Curricular:**

Modelação e Simulação de Sistemas Naturais

**Docente:**

Arnaldo Abrantes

**Alunos:**

Henrique Matos nº48608

João Gonçalves nº 47507

**Turma:**

33D

Contents

**No table of contents entries found.**

**INTRODUÇÃO**

**Exercício A.**

Nesta alínea foi simulado movimento em queda livre sem atrito, ou seja, no vazio, onde todos os corpos independentemente da sua massa, só estão sujeitos à ação da força gravítica.

No enunciado era dada a constante gravítica de e uma posição inicial de 1000m na vertical, sendo a velocidade incial nula (0 m/s).

Tendo as fórmulas:

Temos a seguinte resolução: Calendar

Description automatically generated

Verificando os cálculos, sabemos que em resposta às perguntas, o corpo demora 14.2 segundos a chegar ao solo (ou, aproximadamente, cerca de 14 segundos) e a velocidade com que atinge é de 137.2 m/s.

No entanto, é apresentado a força gravítica e a velocidade com valores negativos. Isto deve-se ao simples facto de serem vetores contrários à natureza ascendente do eixo de posição vertical, possuindo valores negativos. O valor absoluto é a representação desejada enquanto que o valor negativo é a representação do valor vetorial.

Assim, através da simulação em *Silico*, podemos verificar que os cálculos estão corretos já que esta representa com clareza o facto do corpo ao começar a 1000m de altura e com aceleração de , ao fim de 14.2 segundos, este atinge o solo ja que a sua posição é de 0m.

- Nota: o gráfico de relação posição-tempo extende-se até aos 15 segundos para poder demonstrar com clareza o corpo parado aos 14.2 segundos.

Chart, diagram

Description automatically generated

**Exercício C.**

Na alínea C, simulou-se o comportamento dum sistema solar constituído por 4 planetas diferentes, assemelhando-se à parte interior à cintura de asteróides do nosso Sistema Solar (Mercúrio, Vénus, Terra e Marte).

Para que tal fosse possível, começou-se por definir a classe **Mover**, definindo variáveis de posição, velocidade e aceleração do tipo Pvector, e dois floats responsáveis pela massa e pelo raio do corpo em questão, sendo que todas estas seriam definidas em construtor. Criou-se também um método de aplicação de forças, passando um argumento de força sob a forma de vetor PVector, que iria adicionar ao vetor aceleração uma divisão entre os vetores **força** e **massa**.

De seguida definimos um método de movimento passando o argumento de intervalo de tempo, em que adicionamos ao vetor de velocidade uma multiplicação entre a aceleração e o tempo, adicionamos ao vetor de posição uma multiplicação em PVector entre a velocidade e o tempo, multiplicamos também a aceleração por 0. Tais calculos servirão todos como as bases das forças a serem usadas na simulação do sistema solar em questão (e do sistema de partículas, que será explicado mais à frente).

Definimos também *getters* e *setters* de modo a poder obter sempre que necessário os vetores posição e velocidade.

Tendo a classe **Mover** pronta, começamos então a implementação da classe

**CelestialBody**, sendo este responsável pelos comportamentos dos corpos celestes representados no sistema solar, relacionado as diversas forças físicas em jogo, definindo inicialmente a constante gravitacional de valor **6.67e-11**.

Sabendo que a classe **CelestialBody** extende a classe Mover, o construtor passa o método *super* de modo a poder obter as mesmas variáveis de posição, vetor, massa e raio, adicionando a nova variável “color”, para que se possa fazer a distinção e pintura dos corpos celestes.

Começou-se então por definir um método que calcularia as forças de atração presentes entre os corpos, passando um argumento sob a forma de objeto da classe **Mover**. Neste método calcula-se a magnitude do vetor de força de atração através do método de processing, “Pvector.mag()”, define-se a fórmula de força gravitacional, como se pode ver na figura, e retornamos o vetor normalizado, multiplicado pela tal força.

Para desenhar os corpos celestes, usando o método **draw** do Processing, vai se buscar as coordenadas dos pixeis da janela e do vetor de raio e criam-se circulos com as coordenadas obtidas.

*- Antes de entrarmos em detalhe na classe responsável completamente pelo sistema solar (****SolarSystemApp****), iremos explicar a realização do sistema de partículas, já que este estará presente como corpo celeste no sistema solar*

**Sistema de Partículas**

No que toca ao sistema de partículas, começando pela explicação da classe **Particle**, esta extende a classe **Mover**, já explicada acima, assim sendo herda o super construtor da última, passando apenas como novos argumentos as variáveis cor e tempo de vida.

Nesta classe define-se um *timer* responsável pelo tempo de vida da partícula em que se o *timer* for maior que o tempo de vida, a partícula morre.

Na classe **ParticleSystem**, que também extende **Mover**, passa-se mais uma vez o super construtor desta última classe, adicionando apenas os argumentos das novas variáveis que serão a cor da partícula, o tempo de vida e a velocidade da partícula. Também se é definido uma lista de partículas, através do objeto da classe **Particle**.

O movimento das partículas é definido através dum método constituído por um *for loop* que percorre todas as partículas da lista de partículas criada acima, fazendo com que se movam de acordo com o intervalo de tempo, passado como argumento. Caso a partícula esteja morta, ou seja, o *timer* é maior que o seu tempo de vida pré definido, esta é removida da lista de partículas.

Para realizar a adição de partículas, o método responsável primeiro define a velocidade de movimento das partículas em ambos os eixos bi-dimensionais, cria a particula passando o construtor do objeto da classe **Particle** e adiciona por fim à lista de partículas.

Para demonstrar as partículas é usado o método **display** de modo a percorrer a lista de partículas e demonstrando-a toda.

Por fim, criou-se a classe responsável pelo sistema de partículas completo, **ParticleSystemApp**, implementando a interface *IProcessingApp*.

Nesta classe define-se uma lista de particulas usando o objeto da classe anterior, a janela e o viewport, e a variável relativa à classe **SubPlot** que é responsável pela gestão de viewport.

Esta classe começa por configurar o comportamento da janela através do objeto relativo à classe **SubPlot** e por definir a nova lista de partículas.

Seguidamente, define-se a maneira que serão desenhadas as partículas, aplicando as forças gravíticas da classe **Mover** e o seu movimento na simulação, sendo que neste caso caem em direção à zona inferior da janela.

Para demonstrar as partículas em ecrã, é necessário recorrer a um clique no rato, definindo a sua cor, extensão em eixos bi-dimensionais, tempo de vida, posição do rato e vetores de força, adicionando por fim à lista de partículas a partícula criada.

Background pattern

Description automatically generated

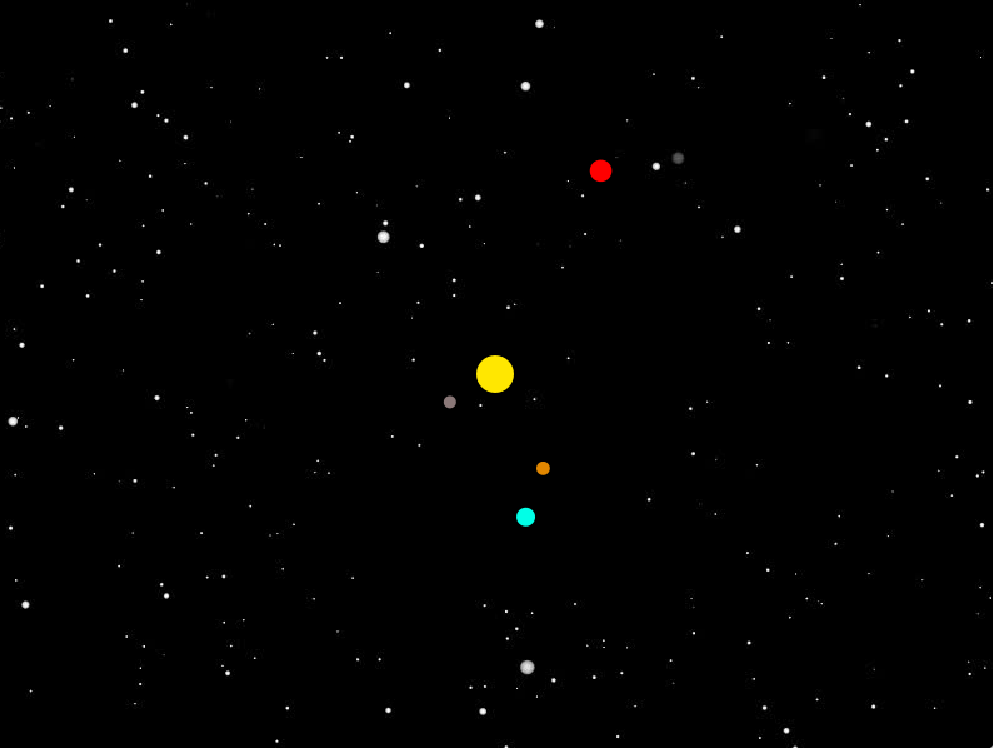
1 - sistema de partículas

Assim sendo, passando para a parte final da alínea, concatenou-se ambos o sistema solar e o sistema de partículas de modo a criar algo único.

Para criar o sistema solar, na classe **SolarSystemApp**, implementou-se a interface *IprocessingApp*, definindo as diversas características dos planetas como a massa, velocidade de órbita e distância ao Sol. Para se observar a simulação também se definiu as variáveis de escala de janela/viewport.

Criaram-se objetos da classe **CelestialBody** sendo estes os corpos celestes presentes na simulação, passando como parâmetros todas as características dos planetas.

Para efetuar o desenho da simulação, aplicou-se então as forças responsáveis pelas interações dos corpos, presentes nas classes **CelestialBody** e **Mover**, sendo estes métodos relacionados à atração de corpos e aplicação de forças, por fim dando display dos desenhos.



2 - sistema solar

Nesta simulação, fez-se com que o sistema de partículas fosse responsável pelo *spawn* de cometas, a cada clique do rato. Assim sendo, aplicou-se o mesmo método de **mousePressed** usado na classe **ParticleSystemApp** já que o comportamento é idêntico, mudando apenas a direção de queda, sendo que os cometas atravessam o sistema solar na diagonal, deixando uma cauda/rasto mais uniforme. Foi necessário definir também as variaveis responsaveis pelas particulas e também o comportamento de desenho juntamente com os outros corpos celestes.

A picture containing light, night, night sky

Description automatically generated

3 - sistema solar com os cometas

* Logo, icon

  Description automatically generatedUma nota interessante a tirar do sistema solar – para que se possa observar os planetas sem rasto, foi necessário usar pushStyle e popStyle, métodos do processing, para que após a criação da caixa de simulação se pudesse manipular a transparência do rasto, que neste caso é nula já que a intenção é usar uma imagem. Sem qualquer manipulação, o seu estado original é o seguinte:

4 - sistema solar com rasto no seu estado inicial