jvo, hdaniel, jbarateiro

Lab 6: Game Engine, parte 2

Este trabalho vai ser o segundo de uma série para efetuar em grupo e que levará ao projeto final.

Submeta o seu código ao Mooshak http://deei-mooshak.ualg.pt/~jvo/ usando o login do seu grupo, ex: **POO2425p1g1**

Uma submissão ao problema permanecerá *pending* até que seja validada pelo professor durante a aula prática, **com todos os elementos do grupo**.

Só as submissões em estado *final* serão consideradas para avaliação.
Só as submissões que respeitarem todas as restrições serão consideradas para avaliação.

Todas as submissões deverão ser feitas até: **13 de abril 2025**

A validação poderá ser feita posteriormente, se necessário, até: **24 de abril de 2025**

NB: Nos problemas seguintes, sempre que necessário, considera-se que dois double *d* e *g* são iguais se $|d-g|<1e-9$

Restrições
Em todos os problemas devem seguir os princípios e as técnicas da programação orientada por objetos.

Para validar cada problema é necessário que sejam implementados e incluídos no código os interfaces indicados.

Para cada um dos problemas seguintes, antes de escrever qualquer código, **defina um conjunto de testes unitários** que reflitam o comportamento pretendido.

Comece por **esboçar um diagrama de classes inicial (de análise) UML** com as classes necessárias para implementar os problemas descritos. Este diagrama deve ser mostrado durante a validação, não devendo ser submetido ao Mooshak.

Adicionalmente, deve comentar o código, indicando:

i) Uma linha com a descrição da responsabilidade da classe ou do que o método faz

ii) @author (apenas para classes)
iii) @version (apenas para classes; inclua uma data)
iv) @inv (apenas para classes; inclua uma descrição da invariante usada)

v) @param (apenas para métodos e construtores)
vi) @return (apenas para métodos)

vii) @see (qualquer referência bibliográfica ou sítio web consultado para o desenvolvimento do código respetivo)

Consulte agora os slides das Teóricas 8 e seguintes
antes de responder ao problema seguinte.

Problema N: Deteção de colisões entre GameObjects

Pretende-se implementar a deteção de colisões entre **GameObjects**, após serem geradas algumas *frames*.

Deve começar-se por implementar a classe **GameEngine** que tem uma lista de **GameObjects.** Após um **GameObject** ser criado é adicionado a esta lista pelo método ***add(GameObject go)*** de **GameEngine**. Deve existir também o método ***destroy(GameObject go)*** para remover o objeto desta lista.

A deteção de colisões deve ser efetuada por um objeto do tipo **ICollider** que para isso terá de estar centrado na **ITransform.position** de **GameObject**. Assim um **ICollider** deve ter uma referência para a mesma **ITransform** que o **GameObject** e movimentar-se de acordo com esta.

A movimentação, rotação e escala dos **GameObjects** deve ser efetuada através dos métodos ***move(), rotate() e scale()*** de **ITransform**, que deverão ser chamados para cada um dos **GameObjects** na lista do **GameEngine** *em cada frame*.

GameObjects em diferentes *layers* não colidem. A estratégia de deteção de colisões não deve testar colisões entre objetos em diferentes *layers*.

Entrada
A entrada do programa tem *n**4+2 linhas. A primeira tem o número de *frames* a simular: ***f***. A segunda o número de **GameObjects** a criar: ***n***. Para cada **GameObject** tem-se 4 linhas com a seguinte informação:

A primeira tem uma string com o nome do **GameObject**.
A segunda linha tem a informação para criar a **Transform** com esta ordem: dois double para as coordenadas x e y, um int para a layer, um double para a rotação em graus e um double para o fator de escala.
A terceira linha tem a informação para criar o **Collider**. Se tiver 3 doubles representa um colisor circular com centro (x, y) nos 2 primeiros valores com raio igual ao terceiro double. Se tiver 6 ou mais doubles o colisor será um polígono, e cada par de doubles representa um vértice, sendo estes ordenados no sentido horário.
A quarta linha tem 2 doubles seguidos de um int e de mais 2 doubles. Os 3 primeiros valores contém a informação sobre a velocidade constante do **GameObject** em cada *frame* nos eixos x, y e na *layer*. O quarto valor a velocidade de rotação constante no sentido anti-horário em graus, também por cada *frame*. O último valor um diferencial a ser somado à escala corrente em cada *frame*.

Saída
A saída tem um mínimo de 0 e um máximo de ***n*** linhas, uma para cada **GameObject que esteja em colisão,** ordenadas pela ordem dos objetos na entrada, cada uma com a seguinte informação:

> Nome do **GameObject** seguido dos nomes dos **GameObjects** que estão em colisão, após o número de *frames* ***f***, como indicado na primeira linha de entrada. Os objetos em colisão devem estar ordenados também pela sua ordem na entrada.

Restrições complementares
Seguir todas as indicações no enunciado.
Apresente o diagrama UML de todas as classes e interfaces desenvolvidos.
Para validar o problema é necessário que sejam implementados e incluídos no código os interfaces indicados acima.

Exemplo de Entrada 1
```
1
3
Square
2 2 0 0 1
1 1 1 3 3 3 3 1
2 1 0 0 0
Rect
5 5 0 0 1
4 3 4 7 6 7 6 3
0 0 0 0 0
Circle
10 4 0 0 1
10 4 1
-4 0 0 0 0
```

Exemplo de Saída 1
```
Square Rect
Rect Square Circle
Circle Rect
```

Exemplo de Entrada 2
```
3
3
SmallC
0 4 0 0 1
0 4 1
0 -1 1 0 0
BigC
0 0 0 0 1
0 0 2
0 1 0 0 0
Tri
5.5 2 0 0 1
6 1 5 2 5 3
-2 0 0 0 0
```

Exemplo de Saída 2
```
BigC Tri
Tri BigC
```

Exemplo de Entrada 3
```
10
2
bullet
1.5 2.5 4 0 1
1.5 2.5 1
2 0 0 0 0
```

```
target
8.5 2.5 4 0 1
8 1 8 4 9 4 9 1
1 0 0 0 0
```

Exemplo de Saída 3

Nota: O exemplo 3 não imprime nada na saída porque não há objetos em colisão.

Exemplo de Entrada 4

```
3
2
Grower
2.5 7.5 2 0 1
2 7 2 8 3 8 3 7
2 0 0 0 1
Rotor
4.5 10.5 2 0 1
4 8 4 13 5 13 5 8
0 0 0 -45 0
```

Exemplo de Saída 4

```
Grower Rotor
Rotor Grower
```