UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

A.G.E.
is
a Game Engine

Helder Daniel

hdaniel@ualg.pt

v2025 1.0

Motivation

- Develop a free Game Engine
- Describe Game Engine operation

All source code snippets are presented in Java

However, the Game Engine can be implemented in any programming language

Introduction

- AGE was developed primarily for 2D games using sprites
- With some effort particularly in the GUI, it can be extended to 3D games

- We will demonstrate some features using a 2D Space Shooter

Game Objects

- Each element that is presented on the screen is an object
- with an independent behaviour

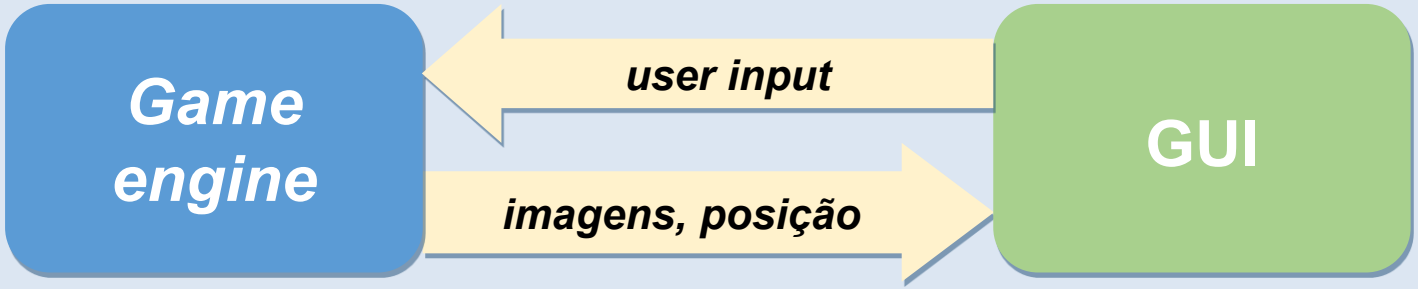
- We addressed them as **Game Objects**

Game Objects

Engine e GUI

- A operação do *engine* é independente do interface gráfico (GUI)

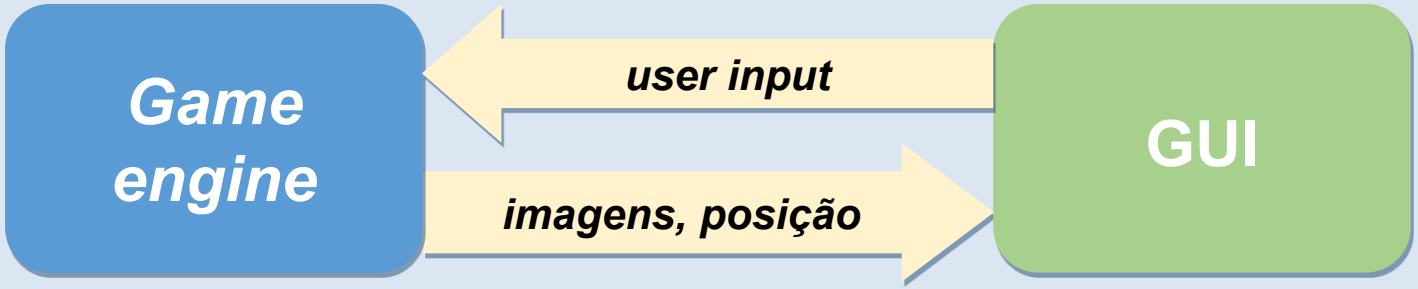
- O *engine* vê o GUI como um serviço:
  - ao qual pede o Input do jogador
  - envia-lhe imagens e posições para apresentar no ecrã



Engine e GUI

- A operação do *engine* é tipicamente efetuada
- num ciclo infinito com 3 passos

- em cada iteração gera-se uma *frame*



Engine e GUI

- A operação do *engine* é tipicamente efetuada
- num ciclo infinito com 3 passos

- em cada iteração gera-se uma *frame*

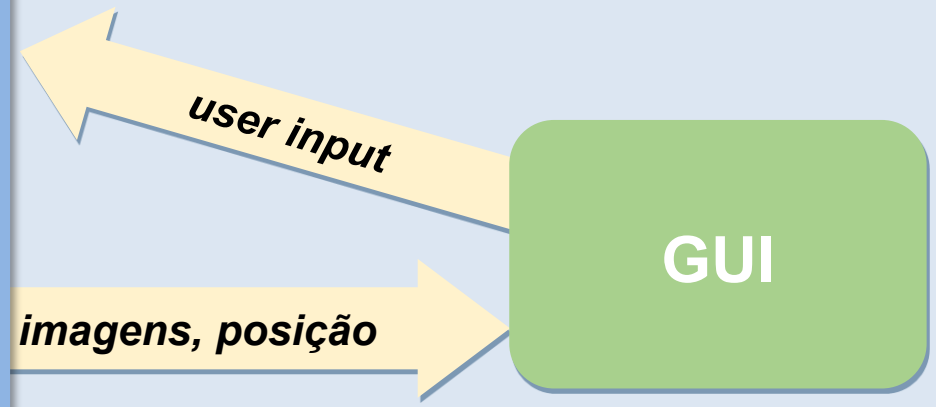
Game engine

loop:

- pede input do utilizador ao GUI

- atualiza as posições dos objetos

- envia a lista de objetos e posições para o GUI para criar uma *frame*



Engine e GUI

- Para atualizar as posições dos **GameObjects** em cada *frame*

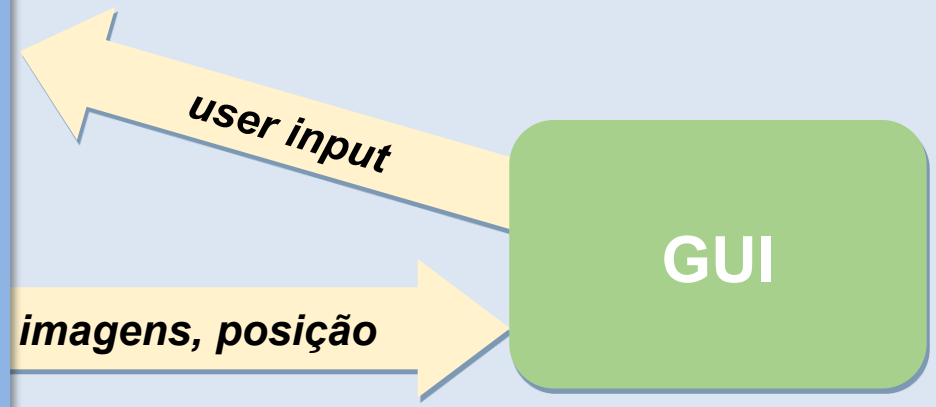
- O *engine* mantém uma lista de **GameObjetcs**
- **com todas as suas propriedades**

loop:

- pede input do utilizador ao GUI

- atualiza as posições dos objetos

- envia a lista de objetos e posições para o GUI para criar uma *frame*



GameObject

- Cada **GameObject** é uma instancia de uma classe
  - com os atributos necessários para o apresenta-lo no ecrã
  - mas também para o funcionamento do jogo

GameObject
- Transform
- Shape
- Collider
- Behaviour

GameObject

- **Transform** contém toda a informação sobre:
  - posição
  - layer
  - rotação
  - escala

GameObject
Transform
Shape
Collider
Behaviour

GameObject

GameObject
Transform
Shape
Collider
Behaviour

- **Transform** contém toda a informação sobre:
  - **posição**
  - layer
  - rotação
  - escala

A posição é representada no plano cartesiano

por um ponto com coordenadas reais com sinal

GameObject

GameObject
Transform
Shape
Collider
Behaviour

- **Transform** contém toda a informação sobre:
  - posição
  - **layer**
  - rotação
  - escala

Num jogo 2D os layers permitem modelar GameObjects em vários níveis

Podemos assim evitar colisões entre aeronaves e automóveis

é um valor inteiro com sinal

GameObject

GameObject
Transform
Shape
Collider
Behaviour

- **Transform** contém toda a informação sobre:
  - posição
  - layer
  - **rotação**
  - escala

A rotação é indicada em graus no sentido anti-horário onde zero graus representa o Norte.

É efetuada em relação ao centroide do **GameObject**, isto é a posição no plano indicado na Transform.

é um valor real com sinal

GameObject

GameObject
Transform
Shape
Collider
Behaviour

- **Transform** contém toda a informação sobre:
  - posição
  - layer
  - rotação
  - **escala**

Modificando a escala enquanto um **Game Object** muda de layer pode modelar-se a aproximação ou afastamento deste

A escala é um valor real sem sinal

GameObject

GameObject
- Transform
- **Shape**
- Collider
- Behaviour

- **Shape**

  contém a imagem a apresentar no GUI
  é dependente do próprio GUI

Nesta apresentação não vamos abordar o GUI
*apenas o **engine***

GameObject

GameObject
- Transform
- Shape
- **Collider**
- Behaviour

- **Collider**

é usado para detetar colisões entre objetos

Para tornar a deteção de colisões mais simples
modelos 3D ou imagens 2D complexas são envolvidas
em figuras geométricas mais simples

GameObject

GameObject
- Transform
- Shape
- **Collider**
- Behaviour

- **Collider**

neste exemplo 2D os colisores são figuras geométricas

GameObject

GameObject
- Transform
- Shape
- Collider
- **Behaviour**

- **Behaviour**

classe que implementa os métodos de controlo de cada GameObject

mover

rodar

disparar

ou mais complexos como:

patrulhar

atacar

IGameObject

- **GameObject**

implementa a interface IGameObject

```
public interface IGameObject {

    String name();

    ITransform transform();
    IShape shape();
    ICollider[] colliders();
    IBehaviour[] behaviours();
}
```

ITransform

- **Transform**

implementa a interface ITransform

```
public interface ITransform  {

    public void move(Point dPos, int dlayer);
    public void rotate(double dTheta);
    public void scale(double dScale);

    public Point position();
    public int layer();
    public double angle();
    public double scale();

}
```

ICollider

- **Collider**

implementa a interface **ICollider**

```
public interface ICollider {
    String name();
    Point centroid();
    void onUpdate();

    boolean isColliding(ICollider other);
    boolean isColliding(CollPoly other);
    boolean isColliding(CollCircle other);

    }
```

Mover um GameObject

Para mover o **GameObject**

somar o diferencial **dPos** à posição corrente na **Transform**
O GUI irá mover a imagem de acordo com esta informação

Para mover o colisor

se for um círculo basta somar **dPos** ao centro

Se for um polígono soma-se **dPos** a todos os vértices.

Rodar um GameObject

Para rodar o **GameObject**

atualizar o ângulo na **Transform**

O GUI irá rodar a imagem de acordo com esta informação

Para rodar o colisor

se for um círculo não é necessário efetuar a operação

se for um polígono, rodam-se todos os vértices em torno do centroide

Escalar um GameObject

Para escalar um **GameObject**

a posição original mantém-se
o fator de escala na **Transform** é atualizado
O GUI irá escalar a imagem de acordo com esta informação

Para escalar o colisor

se for um círculo basta multiplicar o raio pelo fator de escala.

Se for um polígono
deslocar o polígono de forma que o seu centroide seja (0,0)
multiplicar as coordenadas de todos os vértices pelo fator de escala
voltar a deslocar para a posição original

Comportamento dos GameObjects

Comportamento dos GameObjects

- Cada **GameObject** tem comportamento independente

- Comunicação entre **GameObjects** é efetuada
- utilizando o padrão de projeto **Observer**

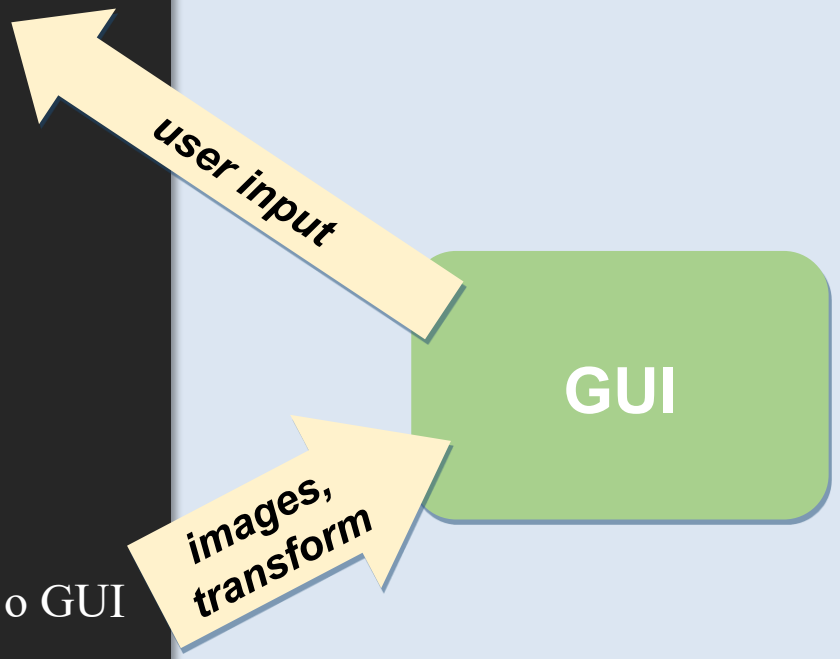
- O comportamento é modelado
- por uma instancia da classe **IBehaviour**

Comportamento dos GameObjects

- O *engine* em cada *frame*
- executa o método **onUpdate()** de **IBehaviour**

GameEngine

```
public void run() {

 for (;;) {

     ie = getUserInput()

    for (IGameObject go : enabled) {
          go.behaviour().onUpdate(dt, ie);
          go.collisor().onUpdate();
    }

     // envia lista de colisões para todos os
     // IGameObject em enabled

    // envia a lista de IGameObjects em enabled para o GUI
    }
}
```



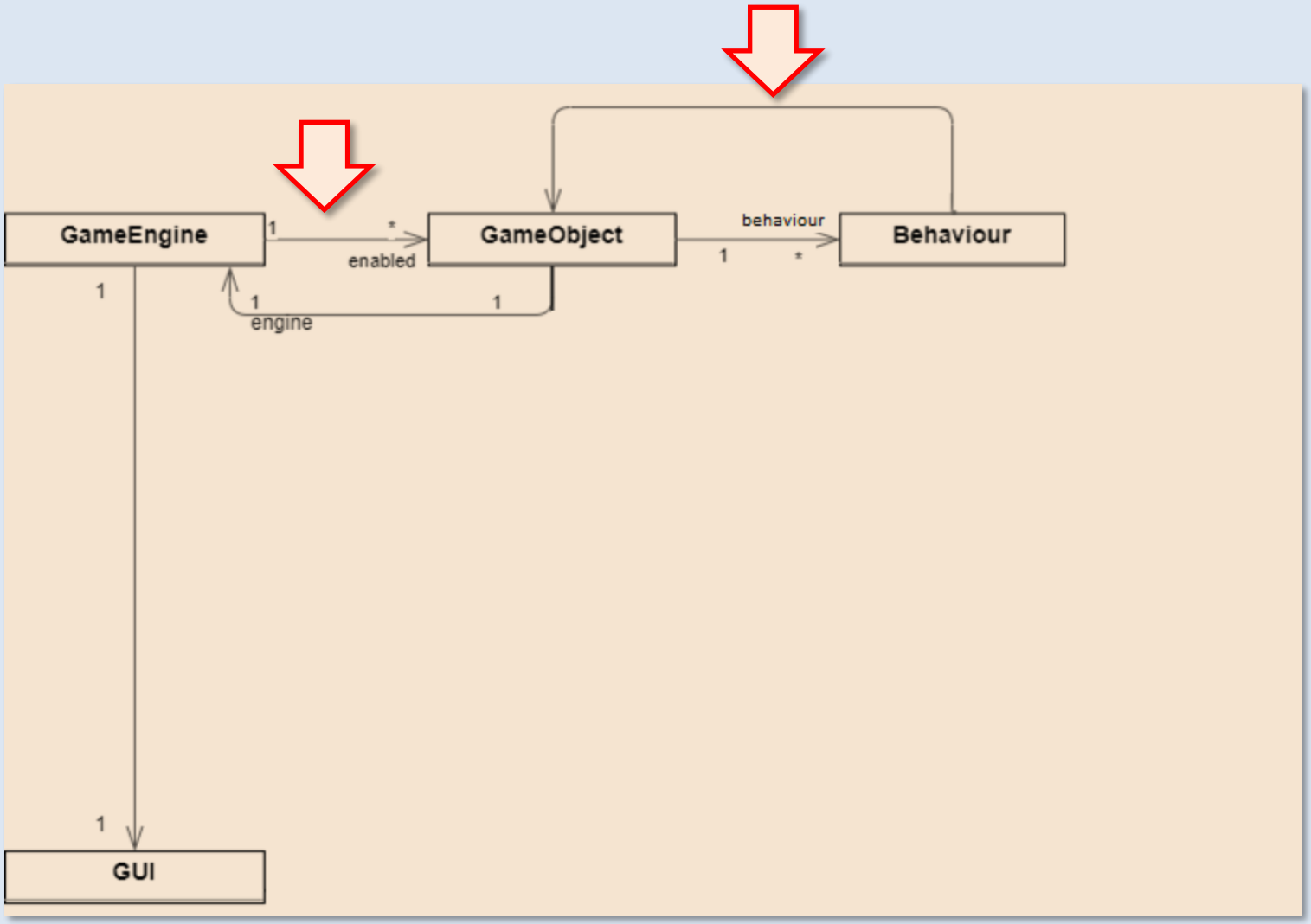
IBehaviour

- **IBehaviour**

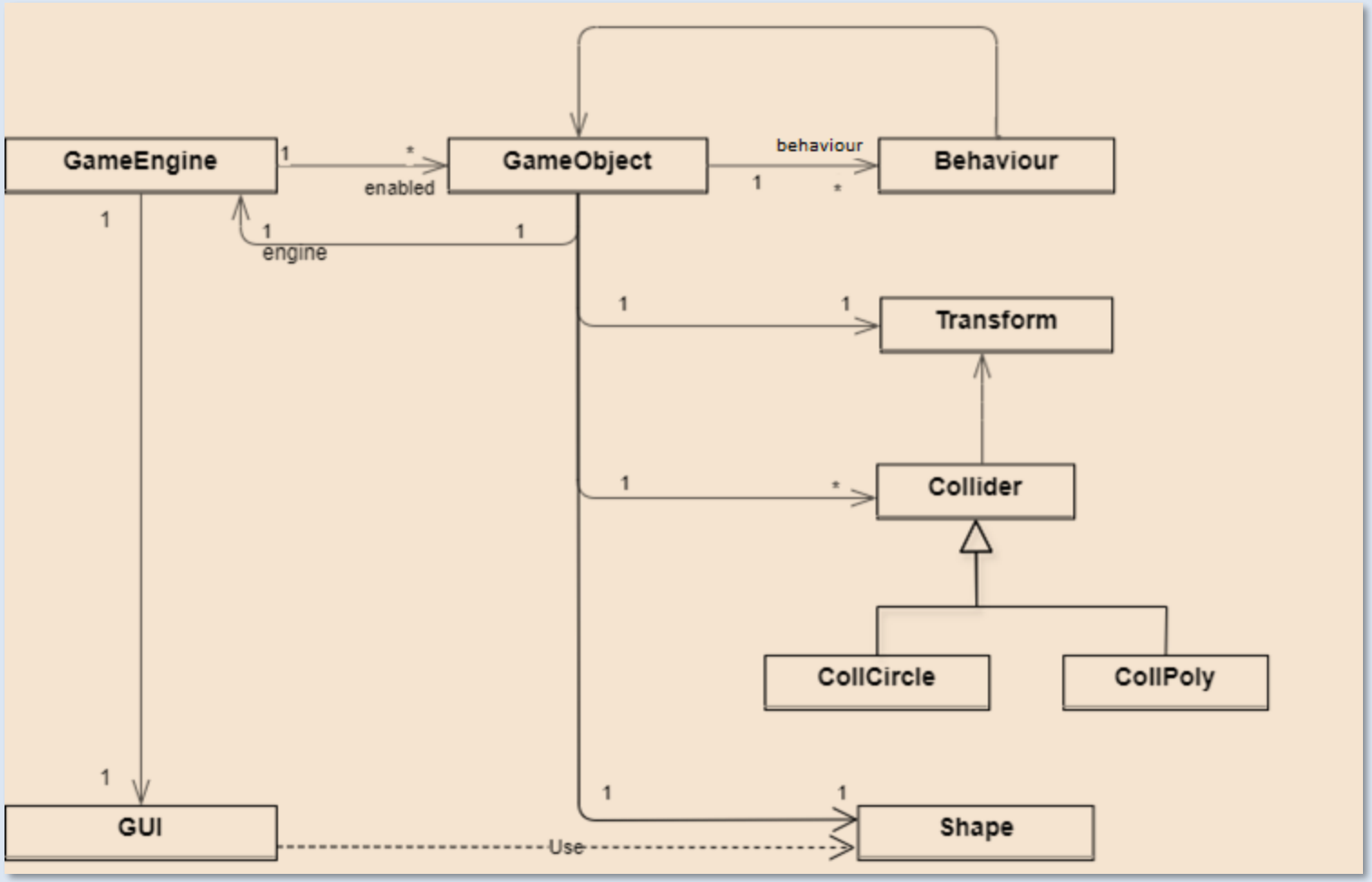
para acesso ao **GameObject** a que está associado

```
public interface IBehaviour {

    public IGameObject gameObject();

    public void gameObject(IGameObject go);









}
```

Engine e GameObjects



Engine e GameObjects



IBehaviour

- **IBehaviour**

```
public interface IBehaviour {

    public IGameObject gameObject();

    public void gameObject(IGameObject go);




    void onUpdate(double dT, IInputEvent ie);
    void onCollision(List<IGameObject> gol);

}
```

chamado pelo game engine em cada frame

IBehaviour

- **IBehaviour**

```
public interface IBehaviour {

    public IGameObject gameObject();

    public void gameObject(IGameObject go);

    void onInit();
    void onEnabled();
    void onDisabled();
    void onDestroy();
    void onUpdate(double dT, IInputEvent ie);
    void onCollision(List<ICollider> cols);

}
```

GameEngine

```
public interface IBehaviour {

    void onInit();



}
```

```
public interface IGameEngine {

    public void addEnabled(IGameObject go);
    public void addDisabled(IGameObject go);
```

GameEngine

```
public interface IBehaviour {

    void onInit();
    void onEnabled();
    void onDisabled();


}
```

```
public interface IGameEngine {

    public void addEnabled(IGameObject go);
    public void addDisabled(IGameObject go);

    public void enable(IGameObject go);
    public void disable(IGameObject go);
```

GameEngine

```
public interface IBehaviour {

    void onInit();
    void onEnabled();
    void onDisabled();


}
```

```
public interface IGameEngine {

    public void addEnabled(IGameObject go);
    public void addDisabled(IGameObject go);

    public void enable(IGameObject go);
    public void disable(IGameObject go);

    public boolean isEnabled(IGameObject go);
    public boolean isDisabled(IGameObject go);
```

GameEngine

```
public interface IBehaviour {

    void onInit();
    void onEnabled();
    void onDisabled();


}
```

```
public interface IGameEngine {

    public void addEnabled(IGameObject go);
    public void addDisabled(IGameObject go);

    public void enable(IGameObject go);
    public void disable(IGameObject go);

    public boolean isEnabled(IGameObject go);
    public boolean isDisabled(IGameObject go);

    public Iterable<IGameObject> getEnabled();
    public Iterable <IGameObject> getDisabled();
```

para GUI

GameEngine

```
public interface IBehaviour {

    void onInit();
    void onEnabled();
    void onDisabled();
    void onDestroy();


}
```

```
public interface IGameEngine {

/**
     * Destroy IGameObject go whether it is enabled or disabled
     * pre: go != null
     * pos: go.onDestroy()
     */
    public void destroy(IGameObject go);

    /**
     * Destroy all IGameObjects
     * pos: calls onDestroy() for each IGameObject
     */
    public void destroyAll();
```

GameEngine

```
public interface IBehaviour {

    void onInit();
    void onEnabled();
    void onDisabled();
    void onDestroy();
    void onUpdate(double dT, IInputEvent ie);

}
```

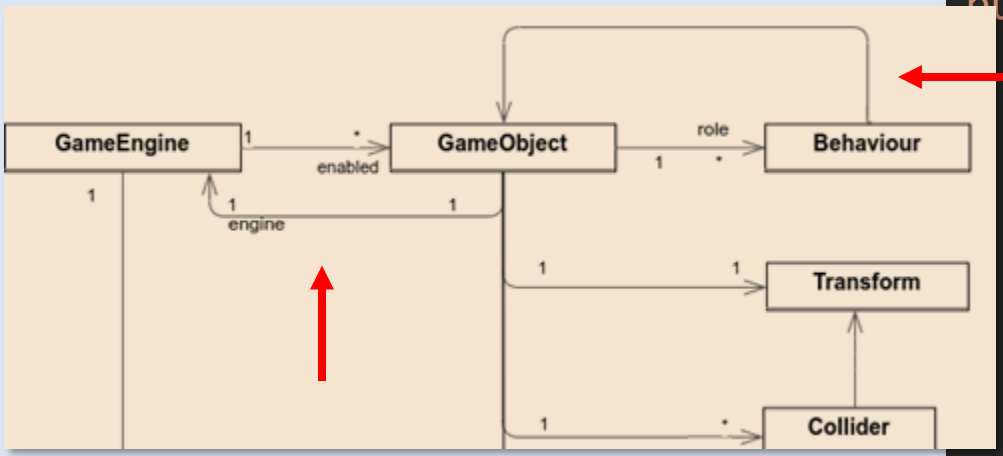
```
public interface IGameEngine {

    /**
     * Generates a new frame:
     * Get user input from UI
     * update all the enabled GameObjects
     * check collisions and send info to GameObjects
     * update UI
     * pos: UI.input() &&
     *      calls Behaviour.onUpdate() for all enabled objects &&
     *      Behaviour.checkCollisions() &&
     *      UI.draw()
     */
    public void run();
```

manipula a transform

GameEngine

```
public interface IBehaviour {

    void onInit();
    void onEnabled();
    void onDisabled();
    void onDestroy();
    void onUpdate(double dT, IInputEvent ie);
    void onCollision(List<List<ICollider> cols);
}
```

```
public interface IGameEngine {


    /**
     * Check collisions for all the enabled objects
     * pos: calls Behaviour.onCollision(gol) for all enabled GameObjects
     *       passing in the list of all the objects that collided with each IGameObject
     */
    public void checkCollisions();
```




```
public interface aBehaviour : Behaviour {

  //...
  void onUpdate(double dT, IInputEvent ie) {

   }

  void onCollision(List< List<ICollider> gol) {

  }

  void onDestroy() {

  }
```

```
public interface IGameEngine {

   public void enable(IGameObject go);
   //...

   public void destroy(IGameObject go);

   //...
```

Comunicação entre GameObjects

- A comunicação entre **GameObjects**
- pode ser efetuada utilizando o padrão de desenho **Observer**
- para isso **IBehaviour** deverá ser estendido de acordo