COMPOSIZIONE 1

CORSO DI GAME PROGRAMMING
1° ANNO

Docente **Davide Caio**

ACCADEMIA ITALIANA VIDEOGIOCHI

GAMEOBJECT

Guardiamo come abbiamo definito fino ad ora il GameObject, in particolare a tutti i moduli che si occupano di un aspetto del nostro gameobject che possiamo dire sia separato rispetto agli altri:

- Parte visiva (sprite)
- Parte fisica
- Posizione

Quindi ogni entità nel nostro gioco che erediti da gameobject deve avere per forza una parte visiva e una parte fisica.

Da qui noi definiamo due altri template di gameobject, e cioè gli Actor (che rappresentano le nostre navicelle) e i Bullet (cioè i proiettili del nostro gioco). Ognuna di queste entità va a sommare un nuovo comportamento rispetto al nostro gameobject base.

IL PROBLEMA

Il problema con una gerarchia del genere è che stiamo utilizzando solamente l'ereditarietà per definire nuovi gameobject che hanno nuovi comportamenti, oppure una diversa combinazione di comportamenti.

Ogni entità che ad esempio non ha sia una componente fisica che visiva (come ad esempio il background) non è un gameobject.

Ma come fare se io volessi che ogni entità all'interno del mio gioco sia un gameobject, e che poi possa plasmare questo gameobject aggiungendo volta per volta il modulo che gli serve per ottenere il comportamento che gli serve? (ma ovviamente senza ereditare da GameObject ogni possibile combinazione di tutti i moduli che vogliamo implementare nel nostro gioco).

COMPOSIZIONE

La composizione è una tecnica di programmazione con cui le classi dovrebbero ottenere un comportamento polimorfico e il riutilizzo del codice in base alla loro composizione (contenendo istanze di altre classi che implementano la funzionalità desiderata) piuttosto che ereditare da una classe base ed estendere direttamente le funzionalità.

Questo significa avere una sola classe GameObject che rappresenta qualsiasi entità sarà presente nel nostro gioco, e avere una classe Component dal quale poi erediteranno tutti i componenti che andremo a creare nel nostro gioco.

COMPONENT

Cerchiamo di capire cos'è un componente. Un componente sarà un modulo "attaccato" al nostro GameObject che definirà un particolare comportamento del GameObject stesso in una determinata occasione.

Alcuni componenti dovranno essere in grado di renderizzare, altri di gestire la logica di gioco, altri di far interagire il gameobject con la fisica.

Ma tutti dovranno ereditare dalla stessa classe base Component.

GAMEOBJECT

Avendo dato la definizione di Component nella slide precedente quindi cos'è un GameObject?

Un GameObject è semplicemente un aggregatore di componenti, che definiscono un particolare comportamento del GameObject stesso, e la totalità dei suoi componenti definiscono quindi il comportamento specifico di ogni istanza di GameObject.

TRANSFORM

Ogni GameObject deve in ogni caso avere una posizione nel mondo.

Sappiamo quindi che il nostro modulo sarà la Transform, che non dovrà fare nient'altro che memorizzare la Posizione (rotazione e scale) dell'oggetto nel mondo.

Attenzione: non potrà esistere un oggetto senza una transform.

COS'È QUINDI UNA SCENA?

Una scena non è nient'altro che una raccolta di GameObject che sono presenti in quella scena (sia attivi che disattivi).

In questo modo il game loop sarà basato sulla scena, che si occuperà di chiamare le varie funzioni del modulo di rendering (DrawMgr) del modulo fisico (PhysicsMgr) e del nostro game loop (update, late update, ecc).

INTERFACCE

Iniziamo con le interfacce, che utilizzeremo per implementare il nostro GameLoop

- IStartable → per tutti i componenti che vogliono avere una Start () (cioè una funzione che viene chiamata dopo l'inizializzazione del GameObject e della Scena)
- IUpdatable → per tutti i componenti che vogliono avere una Update () e una LateUpdate ()
- ICollidable → per tutti i componenti che vogliono avere una OnCollide
- IDrawable → per tutti i componenti che vogliono renderizzare qualcosa
- IFixedUpdatable → per tutti i componenti fisici.

MANAGER

A questo punto possiamo prendere delle classi di manager già presenti in space shooter che possiamo riutilizzare nel nostro codice:

- GfxMgr → per raggruppare e istanziare una volta sola tutte le texture che mi serviranno nel nostro gioco.
- DrawMgr → che si occupa della parte di renderizzazione del nostro engine. facciamo anche un piccolo upgrade per gestire meglio l'ordine di render delle sprite.

TRANSFORM - IMPLEMENTAZIONE

Sarà un componente che rappresenta la posizione, rotazione, e scale dell'oggetto.

Avrà quindi due vector2 per la posizione e la scale, e un float per la rotazione.

COMPONENT - IMPLEMENTAZIONE

Component (classe astratta) avrà 3 proprietà:

- gameObject → reference al suo GameObject
- transform → reference alla transform attaccata al suo GameObject
- enabled → è attivo?

GAMEOBJECT - IMPLEMENTAZIONE (PARTE 1)

Un GameObject è così un contenitore di Component più una Transform sempre presente.

Avrà almeno un’altra proprietà che rappresenta se in questo momento è attivo oppure no.

USER COMPONENT vs ENGINE COMPONENT

Potremmo dividere in due categorie i componenti di un GameObject,

- quelli che potremmo chiamare EngineComponent, che quindi si occupano più della parte di engine (renderizzazione e fisica) del GameObject (sempre presenti perché parte del core dell'engine)
- user defined component: quelli che invece si occupano più della logica di gioco e che saranno definiti gioco per gioco.

USER COMPONENT

User component non è nient'altro che una classe astratta che eredita da Component e implementa automaticamente le interfacce IStartable, IUpdatable, ICollidable.

Ogni componente che sarà creato dall'utente (che utilizzerà il nostro engine) dovrà ereditare da UserComponent.

ESERCIZIO 1

Quali sono i componenti “Engine” che possiamo riconoscere nel nostro progetto?

Quali sono invece i componenti “User”?

SPRITE RENDERER

Eredita direttamente da Component (quindi è un EngineComponent), e rappresenta un componente IDrawable che se attaccato a un GameObject si occupa di renderizzarlo.

Si aggiungerà automaticamente al DrawMgr e dovrà avere i seguenti campi:

- la Sprite che deve renderizzare
- la Texture da disegnare sulla Sprite
- un DrawLayer
- il pivot della sprite
- l'offset della sprite, cioè la distanza rispetto alla posizione del gameobject (in dubbio se tenerlo oppure no)

GAMEOBJECT - GAME LOOP

Implementiamo a questo punto all'interno di GameObject i metodi richiamati dal game loop che cicleranno su tutti i componenti e, per i componenti di tipo UserComponent, andranno a richiamare a loro volta i metodi corretti.

- Start
- Update
- LateUpdate
- OnCollide

SCENE - IMPLEMENTAZIONE

Scene sarà una classe astratta che rappresenterà una Scena, cioè un collezione di GameObjects.

Nella fase di Inizializzazione si occuperà di creare tutti i GameObject che si trovano nella scena e poi dovrà implementare le funzioni del GameLoop che andranno ad intercettare quelle di ogni GameObject presente nella scena.

Creiamo quindi la classe GameScene che eredita da Scene e nella sua inizializzazione andiamo a creare tutti i GameObject che ci servono (per ora solo i due background per il movimento, il background mover e il player).

GAME

La classe game sarà molto simile a quella vista fino ad ora. Dovrà occuparsi di Inizializzare la finestra e far partire la prima scena.

PLAY SCENE

Ereditiamo da Scene e creiamo la nostra PlayScene da dove saranno caricati tutti gli assets e saranno creati tutti i gameobject di questa scena.

Proviamo il gioco se funziona

PLAYER COMPONENT

Iniziamo a creare uno UserComponent che attaccato a un gameobject lo renderà il player.

Cosa dovrà implementare?

ESERCIZIO PER CASA

Creare un componente che attaccato a un gameobject si assicura che non esca dai bordi della finestra.

TIPS: quale funzione deve overridere per essere sicuro che lo spostamento sia già avvenuto, e possa effettivamente controllare che non sia uscito dallo schermo?