|  |
| --- |
| Ciobanu Andra, Moretti Luca, Pagetti Gabriele, Pistillo Andrea |
| Game Objects |
| Documento di Analisi e Progettazione |

|  |
| --- |
| Gabriele Ciobanu Andra, Moretti Luca, Pagetti Gabriele, Pistillo Andrea  05/05/2021 |

Sommario

[Scopo del progetto 3](#_Toc39588124)

[Requisiti 4](#_Toc39588125)

[Progettazione 5](#_Toc39588126)

[Elenco delle classi da realizzare 5](#_Toc39588127)

[Librerie da utilizzare 5](#_Toc39588128)

# Scopo del progetto

La scopo del progetto di cui fa parte il nostro gruppo ha l’obbiettivo di realizzare un motore grafico per videogiochi 2D. Essendo un grande progetto che coinvolgeva tutta la classe siamo stati divisi in vari gruppi.

Il nostro gruppo si occupa di fare le classi che definiscono un’oggetto della scena del gioco.

# Requisiti

Elencare le funzionalità previste …

Per realizzare il nostro obbiettivo abbiamo bisogno sicuramente di realizzare una classe che rappresenti effettivamente l’oggetto della scena (“GameObject”). Per definire questa classe abbiamo bisogno di:

* Una classe che rappresenti la dimensione e la posizione (“Transform”). A sua volta questa classe necessita di una classe che definisca un vettore 2D (“Vector2”).
* Una classe che definisca le azioni dell’oggetto (“Script”).

# Progettazione

## Elenco delle classi da realizzare

1. GameObject(ArrayList<Object>, String): la classe che effettivamente rappresenta un’oggetto della scena. Prende come parametro una lista con all’interno qualsiasi tipo di oggetto che contiene i componenti legati all’oggetto, mentre la stringa rappresenta il nome dell’oggetto

* T getComponent(Class <T>): metodo getComponent restituisce un componente in base al tipo di classe inserita;
* addComponent(T): metodo addComponent può aggiungere un componente qualsiasi nella lista ma in modo controllato, infatti non tutti i componenti possono entrare;
* boolean hasComponent(Class<T> type): metodo hasComponent restituisce vero se componentList ha un componente del tipo inserito;

1. Script: definisce uno dei componenti più importanti da attaccare al GameObject, sarà poi l’utilizzatore dell’engine a modificarlo piacimento. Infatti, tramite la classe Script si decide il comportamento dell’oggetto.

* start(): il metodo che viene eseguito al caricamento della scena dal oggetto;
* update(): il metodo che viene eseguito ogni frame dal oggetto;
* onCollisionEnter(Collider, Collider): metodo che viene eseguito quando l’oggetto entra in collisione con qualcosa;

1. Transform: Classe che rappresenta la dimensione e la posizione di un oggetto. Viene per l’appunto riutilizzata in GameObject.

* Vector2 getPosition(): restituisce un vettore (vector2) che rappresenta la posizione;
* setPosition(Vector2): modifica il valore del vettore che rappresenta la posizione;
* setPosition(float, float): modifica il valore del vettore che rappresenta la posizione;
* Vector2 getScale(): restituisce un vettore che rappresenta la dimensione;
* setScale(Vector2): modifica il valore del vettore che rappresenta la dimensione;
* setScale(float, float): modifica il valore del vettore che rappresenta la dimensione;

1. Vector2(float, float): modello di un vettore bidimensionale, contiene il riferimento ai valori su due assi e tutti i metodi di utility per impostazioni e operazioni.

* boolean equals(float, float): Il metodo controlla se i valori del vettore sono uguali a quelli passati;
* boolean equals(Vector2): Il metodo controlla se il vettore è uguale al vettore passato;
* Vector2 RIGHT(): Il metodo restituisce un nuovo vettore che indica la direzione verso destra;
* Vector2 LEFT(): Il metodo restituisce un nuovo vettore che indica la direzione verso sinistra;
* Vector2 DOWN(): Il metodo restituisce un nuovo vettore che indica la direzione verso il basso;
* Vector2 UP(): Il metodo restituisce un nuovo vettore che indica la direzione verso l’alto;
* Vector2 sum(float): il metodo somma i valori dell’asse x e y con il valore passato;
* Vector2 sum(float, float): il metodo somma i valori dell’asse x e y con i valori passati;
* Vector2 sum(Vector2): il metodo somma i valori dell’asse x e y con il vettore passato;
* Vector2 multiply(float): il metodo moltiplica i valori dell’asse x e y per il valore passato;
* Vector2 multiply (float, float): il metodo moltiplica i valori dell’asse x e y per i valori passati;
* Vector2 multiply (Vector2): il metodo moltiplica i valori dell’asse x e y per il vettore passato;
* Vector2 divide(float): il metodo divide i valori dell’asse x e y per il valore passato;
* Vector2 divide (float, float): il metodo divide i valori dell’asse x e y per i valori passati;
* Vector2 divide (Vector2): il metodo divide i valori dell’asse x e y per il vettore passato;
* set(float, float): Il metodo imposta i valori del vettore;
* set(Vector2): Il metodo imposta i valori del vettore;
* setX(float): Il metodo imposta il valore sull’asse x;
* setY(float): Il metodo imposta il valore sull’asse y;
* getX(): restituisce il valore sull’asse x;
* getY(): restituisce il valore sull’asse y;

## Librerie da utilizzare

Non viene utilizzata nessuna libreria, ci limitiamo a richiamare altre classi all’interno del progetto.