|  |
| --- |
| Gabas Jyuan Christian, Quispe Cardenas Stefano Davide, Di Bitonto Alessandro |
| Scene Management |
| Documento di Analisi e Progettazione |





|  |
| --- |
| Gabas Jyuan Christian, Quispe Cardenas Stefano Davide, Di Bitonto Alessandro |

|  |
| --- |
| 25/05/2021 |

**Scopo del progetto**

Realizzare un sistema di gestione di tutti i livelli esistenti, che organizza i loro contenuti tramite classi create appositamente, in modo tale da rendere ordinato ogni elemento presente nel gioco.

# **Requisiti**

* La camera punta sempre a un determinato riferimento.
* Gestione dell’esecuzione di tutti gli oggetti, renderizzando qualsiasi renderer attaccato.
* Gestione dell’UI e della sua telecamera.
* Calcolo delle collisioni.
* Esecuzione della scena attiva.
* Gestione della creazione e distruzione dinamica degli oggetti della scena.

# **Progettazione**

Elenco delle classi da realizzare.

## **SceneManager**

Il gestore di tutte le scene del gioco, esegue la scena attiva e gestisce la creazione e distruzione dinamica di oggetti della scena.

### **Attributi**

* private static ArrayList<SceneBlueprint> sceneList: la lista delle scene esistenti
* private static int activeSceneIndex: l’index che indica quale elemento della lista delle scene deve essere la scena attiva

### private static PApplet app: il riferimento all’app principale

* private static float deltaTime: la durata dell'ultimo frame in secondi, utilizzato per fare gli adattamenti al framerate
* private static float time: il valore del tempo di esecuzione dell'app che viene utilizzato per calcolare il deltaTime

### **Metodi**

* public static void initialize(PApplet app): inizializza la lista delle scene e l’index della scena attiva
* public static void executeScene(): esegue la scena attualmente selezionata
* public static void loadScene(int index): carica la scena con l’index specificato
* public static void addScene(SceneBlueprint blueprint): aggiunge la scena specificata alla lista delle scene esistenti
* public static SceneBlueprint getActiveScene(): restituisce la scena attiva
* public static int getActiveSceneIndex(): restituisce l’index della scena attiva
* public static void instantiate(GameObject object): aggiunge un oggetto alla scena attualmente attiva
* public static void destroy(GameObject object): rimuove un oggetto dalla scena attualmente attiva
* public static PApplet getApp(): restituisce l'app principale
* public static float deltaTime(): restituisce il valore del deltaTime

## **SceneBlueprint**

Il modello di una scena che gestisce l’esecuzione di tutti gli oggetti, renderizzando qualsiasi renderer attaccato, calcolando le collisioni, gestendo l’UI e la sua telecamera.

### **Attributi**

* private ArrayList<GameObject> objectList : la lista degli oggetti presenti nella scena
* private ArrayList<Script> scriptList : la lista di tuti gli script
* private ArrayList<Renderable> renderableElements : tutti gli oggetti renderizzabili
* private ArrayList<UIElement> uiElements : gli elementi dell’UI
* private ArrayList<Collider> dynamicColliders : la lista dei collider dinamici
* private ArrayList<Collider> collidersList : la lista dei collider esistenti
* private Camera sceneCamera : la telecamera della scena
* private boolean isInitialized : indica se la scena è già stata inizializzata

### **Metodi**

* public void Initialize(ArrayList<GameObject> objectList): se è il primo caricamento inizializza la scena con i vari GameObject, crea la telecamera, inizializza la lista degli script, oggetti renderizzabili, dell’ui e dei collider attaccati ai GameObject, specifica in una lista separata i collider dinamici e poi indica che è stato inizializzato, in qualsiasi caso dopo tutto quanto chiama il metodo Start
* public void Start(): eseguito al caricamento di una scena, deve eseguire lo start di tutti gli script attaccati
* public void Update(): tutti i metodi ed elementi da eseguire ogni frame, deve eseguire l’update di tutti gli script attaccati
* public GameObject GetObject(int objId): restituisce l’oggetto equivalente all’id selezionato
* public GameObject GetObject(String name): restituisce l’oggetto che corrisponde al nome specificato, se ci sono più oggetti con lo stesso nome restituisce il primo
* public ArrayList<GameObject> GetObjects(): restituisce la lista degli oggetti
* public ArrayList<UIElement> GetUIElements(): resituisce la lista degli elementi dell’UI
* public Camera GetCamera(): restituisce la telecamera della scena
* public void RenderObjects(): esegue tutti i render necessari, fra renderer e animator
* public void RenderUI(): disegna tutti gli elementi dell’UI
* public void CalculateCollisions(): controlla per ogni oggetto dinamico le collisioni con gli altri collider, se c’è collisione chiama OnCollisionEnter per ogni script attaccato all’oggetto dinamico controllato, se il collider confrontato con l’oggetto dinamico è un collider statico allora va eseguito il calcolo della collisione con SNAP, altrimenti va eseguito il calcolo della collisione e basta
* public void setObjectList(ArrayList<GameObject> objectList): valorizza l’oggetto della lista di GameObject con l’oggetto passato come parametro
* public void AddObject(GameObject obj): aggiunge un oggetto alla scena
* public void RemoveObject(GameObject obj): rimuove un oggetto dalla scena
* public void initializeObject(GameObject object): aggiunge l’oggetto corrente nella lista corrispondente alla tipologia
* public void deinitializeObject(GameObject object): toglie l’oggetto corrente dalla lista corrispondente alla tipologia

## **Camera**

Gestione della telecamera di gioco e impostazione di un sistema per seguire un target(per esempio un giocatore).

### **Attributi**

* private GameObject target: il target da seguire, se non c’è un target allora rimane ferma
* private Vector2 position: la posizione della telecamera
* private Vector2 offset: l’offset della telecamera dalla sua posizione
* private PApplet sketch: app principale

### **Metodi**

* Camera(): costruttore
* public GameObject getTarget(): restituisce l'oggetto che la camera deve seguire e alla quale è legata
* public Vector2 getPosition(): restituisce la posizione della camera
* public Vector2 getOffset(): restituisce l’offset
* public void SetTarget(GameObject target): imposta il target da seguire
* public void SetPosition(Vector2 position): imposta la posizione della telecamera
* public void SetOffset(Vector2 offset): imposta il valore dell’offset
* public Vector2 GetOffsetPosition(): restituisce un nuovo vettore con la posizione adattata all’offset
* public void CalculateCamera(): imposta la posizione della telecamera(con i metodi di processing) utilizzando la posizione definita, se ha un target la posizione è quella del target, altrimenti è ferma nello schermo, applicare anche l’offset

## **Librerie da utilizzare**

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