|  |
| --- |
| Davide Furfaro, Samuele Verga |
| 4GraphiCs |
| Il motore grafico della 4C  Documento di Analisi e Progettazione |

|  |
| --- |
| Davide Furfaro, Samuele Verga  14/04/2021 |

**Sommario**

[Scopo del progetto](#_gjdgxs) **5**

[Requisiti](#_30j0zll) **6**

[Progettazione](#_1fob9te) **7**

[Elenco dei gruppi ed i loro membri](#_3znysh7) 7

[Gruppo Scene Management](#_3e8sep16mp4h) 7

[Gruppo Collisions](#_v42054i9athe) 7

[Gruppo UI](#_av6rg9izldfn) 7

[Gruppo GameObject](#_rzlmxn2p8dw6) 7

[Gruppo Graphic System](#_anbh6q99bb4r) 7

[Elenco delle classi, interfacce ed enumerativi da realizzare](#_rxs9p7h6kcgj) 8

[Gruppo Scene Management](#_bbusifa1ci8u) 8

[Gruppo Collisions](#_jawv3wfqjtez) 8

[Gruppo UI](#_gr4kfw7uohps) 8

[Gruppo GameObject](#_85wfa9kpeq0e) 8

[Gruppo Graphic System](#_2giifmftzvy3) 8

[Librerie da utilizzare](#_2et92p0) 8

[**Gruppo Scene Management**](#_vaj4wxj0hqqe) **9**

[SceneManager](#_g4nr2ihvo5b5) 9

[Attributi](#_3hrojexg0ept) 9

[Metodi](#_p31so6fxb8hu) 9

[SceneBlueprint](#_3u5rb5i9kzd9) 10

[Attributi](#_vf7levklrnwk) 10

[Metodi](#_r7d2b17mw4f5) 10

[Camera](#_aqv4whv8t1g3) 11

[Attributi](#_9vthglau06t9) 11

[Metodi](#_rruil3t56d8d) 11

[**Gruppo Collisions**](#_mx5gyn4d064i) **12**

[Collider](#_xdjk98qw82z5) 12

[Attributi](#_enh1c4gqqu19) 12

[Metodi](#_71z4h24g8r6f) 12

[RectCollider extends Collider](#_wzir245ivvf9) 13

[Metodi](#_hj41uhlxdeht) 13

[CircleCollider extends Collider](#_sq5p9qawdbl2) 13

[Metodi](#_ovsqukx7wtkn) 13

[CollisionDirection](#_cb0yu3pgmxc) 13

[**Gruppo UI**](#_w9wqgms6gt56) **14**

[UIElement](#_d5ujhmry1mu4) 14

[Attributi](#_tt7x1yssgsig) 14

[Metodi](#_s0smtjd47d9c) 14

[Button](#_z5hj0zl62w1o) 14

[Attributi](#_pd1qriu41l2v) 14

[Metodi](#_l85rz0h67zt6) 14

[Panel](#_ig45aoxh8ycx) 14

[Metodi](#_lc88fulwus5e) 14

[**Gruppo GameObject**](#_kpsyfw9fm2a) **15**

[GameObject](#_nyt9ek7k5v7h) 15

[Attributi](#_yk7398dluhf1) 15

[Metodi](#_v0egwrf7cp1u) 15

[Script](#_c4u0mt8guqwm) 15

[Attributi](#_1x7fp4cqsr84) 15

[Metodi](#_cptjdeh7cptx) 15

[Transform](#_rmznifess7wp) 16

[Attributi](#_32abty7g83yp) 16

[Metodi](#_mcmfa5xk7t54) 16

[Vector2](#_zc2p53b951r5) 17

[Attributi](#_sdzkm8yi1zoe) 17

[Metodi](#_jxbtus4wggbh) 17

[**Gruppo Graphic System**](#_sin643rhat5k) **18**

[Renderable](#_qjch6hhug6bh) 18

[Attributi](#_poogl4ee7px) 18

[Metodi](#_se8p92pl8dex) 18

[Renderer extends Renderable](#_6cs0n8cwyehp) 18

[Attributi](#_gnngbo3etht2) 18

[Metodi](#_jvtsc3d44v0v) 18

[Animator extends Renderable](#_xx44dlnd2j4p) 18

[Attributi](#_wwzoq565ikap) 18

[Metodi](#_j7mazbxjq5xb) 18

[Animation](#_sb6acvl7dseb) 19

[Attributi](#_bsl91wnxnhuv) 19

[Metodi](#_cd2ol7mc2wai) 19

[DrawType](#_6sqa7x7ssv00) 19

# Scopo del progetto

Lo scopo di questo progetto è quello di realizzare un **motore di gioco** che mira ad applicare le competenze ottenute nel corso dell’anno dalla classe.

Uno degli obiettivi principali è quello di imparare a lavorare in un team di sviluppo completo con compiti diversi per portare a termine un progetto di dimensioni medie/grandi.

È importante inoltre imparare a documentare correttamente il codice scritto in modo tale che chi deve lavorare con il lavoro fornito sia in grado di capire ed utilizzare il codice scritto.

È altrettanto Importante comprendere come sviluppare del codice efficiente, modulare, leggibile, aggiornabile e riutilizzabile. Nel nostro caso la modifica di un elemento del motore grafico non deve influenzare il codice degli altri moduli. Capire questo concetto migliora la qualità del codice (e onestamente anche la salute mentale) di tutti coloro che partecipano al progetto, sia ora che in futuro.

Al completamento del motore grafico verrà creato un gioco che dimostra il funzionamento dell’engine grafico.

# Requisiti

* Sviluppo di un sistema per la gestione delle varie schermate e dei vari livelli in modo da poter creare e controllare tutto in modo comodo e semplice
* Gestione della telecamera di ogni schermata e livello in modo da tener traccia di tutte le informazioni relative e adattare elementi come l’interfaccia grafica (**Screen Space1**)
* Sviluppo di un sistema di collisioni funzionante che renda possibile l’interazione di vari elementi del gioco, principalmente rettangoli, quadrati e cerchi
* Sviluppo dei vari elementi per l’interfaccia grafica in modo da creare menu facilmente comprensibili e rendere più immersivo il gioco, va oltretutto resa possibile la decisione di creare un elemento dell’interfaccia in **Screen Space1** oppure **World Space2**
* Gestione degli oggetti che popolano un livello, devo poter creare terreni, personaggi ecc. comodamente e aggiungergli qualsiasi elemento di codice necessario tramite un **Sistema di Scripting3**
* Gestione di un sistema per la visualizzazione grafica del motore grafico, dotato di rendering sia con che senza **Texture4**, deve essere inoltre possibile la creazione di un sistema di animazioni
* A motore grafico concluso andrà realizzato un gioco che dimostra le capacità dell’engine grafico
* Necessario l’utilizzo dei modelli di programmazione esplorati in classe (MVC)
* Ogni modulo dell’engine avrà della documentazione compresa di UML (necessario un UML rifatto una volta concluso il modulo per aiutare la comprensione del funzionamento interno del progetto).
* Utilizzo attivo della piattaforma GitHub per caricare i progressi del proprio progetto allo scopo di migliorare la collaborazione nel team e cross-team (utilizzare GitHub Desktop per agevolare l’operazione)
  + Ogni gruppo avrà una **Feature Branch5** dedicata dove potrà lavorare indipendentemente da ogni gruppo
  + Eventuali problemi irrisolti all’interno di un modulo (o del progetto) dovranno essere segnalati nella sezione apposita (Issues)
* All’interno dei moduli, creare e utilizzare la cartella test per inserire il codice di testing (esempio: /4GraphiCs/src/test/java/package\_name\_modulo/TestFunzionalità.class)

**1 - Screen Space significa che la posizione definita è relativa alla posizione della telecamera in modo tale che sia sempre all’interno dello schermo**

**2 - World Space significa che la posizione definita è fissa all’interno del mondo**

**3 - Per Sistema di Scripting si intende un sistema che ci da la possibilità di interagire con un oggetto del motore grafico e aggiungergli comodamente un elemento facilmente programmabile**

**4 - Una Texture è un’immagine che definisce l’aspetto di un oggetto**

**5 - Una Feature Branch è un ramo all’interno del progetto che introduce una funzione; immaginiamo di dividere il nostro progetto in un albero dove ogni ramo è una funzionalità che, quando sarà pronta, verrà aggiunta all’albero principale.**

# Progettazione

## Elenco dei gruppi ed i loro membri

### Gruppo Scene Management

* Gabas : Capogruppo
* Di Bitonto
* Quispe

### Gruppo Collisions

* Mercurio : Capogruppo
* Funes
* Bottari
* Umbaca

### Gruppo UI

* Elnar : Capogruppo
* Mareghello
* Biccirè

### Gruppo GameObject

* Pagetti : Capogruppo
* Ciobanu
* Moretti
* Pistillo

### Gruppo Graphic System

* Furriolo : Capogruppo
* Dareisa
* Iurcea

## Elenco delle classi, interfacce ed enumerativi da realizzare

### Gruppo Scene Management

* SceneManager
* SceneBlueprint
* Camera

### Gruppo Collisions

* Collider
  + RectCollider
  + CircleCollider
* CollisionDirection

### Gruppo UI

* UIElement
  + Button
  + Panel
  + CheckBox

### Gruppo GameObject

* GameObject
* Script
* Transform
* Vector2

### Gruppo Graphic System

* Renderable
  + Renderer
  + Animator
* Animation
* DrawType

## Librerie da utilizzare

* Minim - Audio player
* DeltaTime - Calcolo del DeltaTime per il gioco, il DeltaTime è la durata in secondi dell’ultimo frame

# Gruppo Scene Management

Il gruppo si occuperà del sistema per la gestione dei livelli, tutti i livelli esistenti ed i loro contenuto sarà gestito tramite queste classi

## 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

* static PApplet app : il riferimento all’app principale
* static ArrayList<SceneBlueprint> sceneList : la lista delle scene esistenti
* static int activeSceneIndex : l’index che indica quale elemento della lista delle scene deve essere la scena attiva

### Metodi

* static void Initialize(PApplet app) : inizializza l’app, la lista delle scene e l’index della scena attiva
* static void ExecuteScene() : esegue la scena attualmente selezionata
* static void LoadScene(int index) : carica la scena con l’index specificato
* static void AddScene(SceneBlueprint blueprint) : aggiunge la scena specificata alla lista delle scene esistenti
* static SceneBlueprint GetActiveScene() : restituisce la scena attiva
* static int GetActiveSceneIndex() : restituisce l’index della scena attiva
* static void Instantiate(GameObject obj) : aggiunge un oggetto alla scena attualmente attiva, chiama il metodo start degli script attaccati
* static void Destroy(GameObject obj) : rimuove un oggetto dalla scena attualmente attiva

## 

## 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

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

### Metodi

* 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
* void Start(): eseguito al caricamento di una scena, deve eseguire lo start di tutti gli script attaccati
* void Update() : tutti i metodi ed elementi da eseguire ogni frame, deve eseguire l’update di tutti gli script attaccati
* GameObject GetObject(int objId) : restituisce l’oggetto equivalente all’id selezionato
* GameObject GetObject(String name) : restituisce l’oggetto che corrisponde al nome specificato, se ci sono più oggetti con lo stesso nome restituisce il primo
* ArrayList<GameObject> GetObjects() : restituisce la lista degli oggetti
* ArrayList<UIElement> GetUIElements() : resituisce la lista degli elementi dell’UI
* Camera GetCamera() : restituisce la telecamera della scena
* void RenderObjects() : esegue tutti i render necessari, fra renderer e animator
* void RenderUI() : disegna tutti gli elementi dell’UI
* 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
* void AddObject(GameObject obj) : aggiunge un oggetto alla scena
* void RemoveObject(GameObject obj) : rimuove un oggetto dalla scena

## 

## Camera

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

### Attributi

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

### Metodi

* void SetTarget(GameObject target) : imposta il target da seguire
* 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
* void SetOffset(Vector2 offset) : imposta il valore dell’offset
* void SetPosition(Vector2 position) : imposta la posizione della telecamera
* Vector2 GetOffsettedPosition() : restituisce un nuovo vettore con la posizione adattata all’offset

# Gruppo Collisions

Il sistema delle collisioni del gioco viene gestito da questo gruppo, verranno scritti i vari algoritmi per capire quando avviene una collisione in modo tale da poter popolare il mondo di oggetti sia statici che dinamici con cui interagire

## Collider

LA POSIZIONE È IL CENTRO DELL’OGGETTO

Classe di base per le collisioni, ha un riferimento all’oggetto che lo contiene e alla sua posizione e dimensione(per comodità anche se si potrebbe accedere tramite il gameObject), si può indicare se un collider dovrebbe muoversi oppure no in modo tale da calcolare in modo ottimale le collisioni.

### Attributi

* GameObject gameObject : l’oggetto a cui è attaccato questo componente
* Transform transform : la posizione e la dimensione dell’oggetto a cui è attaccato questo componente
* Vector2 previousPosition : la posizione precedente del collider
* boolean dynamicObject : indica se l’oggetto dovrebbe muoversi oppure no

### Metodi

**ATTENZIONE: PER OGNI METODO DI CONTROLLO DELLE CONDIZIONI BISOGNA RICORDARSI CHE UN ELEMENTO TROPPO VELOCE OPPURE A FRAMERATE BASSO SI RISCHIA DI NON RILEVARE LA COLLISIONE, BISOGNA CONFRONTARE LA POSIZIONE ATTUALE CON QUELLA PRECEDENTE E QUELLA DEL COLLIDER CONFRONTATO IN MODO TALE DA CAPIRE SE UNA COLLISIONE SAREBBE DOVUTA AVVENIRE**

**Esempio: una palla cade dal cielo, il framerate è basso quindi “salta” la collisione con una piattaforma, confronto le due posizioni e capisco se è passato dall’altro collider indicato, a questo punto indico se avviene una collisione oppure no**

**CollisionDirection restituita deve essere RIGHT se la collisione arriva da destra, LEFT da sinistra, UP da sopra, DOWN da sotto, NONE se non c’è collisione**

* abstract CollisionDirection CheckCollision(Collider other) : controlla se c’è collisione tra il collider e un secondo collider indicato e ne restituisce la direzione, i calcoli devono essere diversi in base al tipo di collider rilevato
* abstract CollisionDirection CheckCollisionSnap(Collider other) : controlla se c’è collisione tra il collider e un secondo collider indicato, restituendone la direzione e bloccando il collider al collider confrontato, se per esempio sta arrivando una palla dal cielo e arriva ad una piattaforma allora la palla si deve fermare sopra alla piattaforma, lo stesso vale per le altre dimensioni, i calcoli devono essere diversi in base al tipo di collider rilevato

## 

## RectCollider extends Collider

Indica un collider rettangolare

### Metodi

* CollisionDirection CheckCollision(Collider other) : controlla se c’è collisione tra il collider e un secondo collider indicato e ne restituisce la direzione, i calcoli devono essere diversi in base al tipo di collider rilevato
* CollisionDirection CheckCollisionSnap(Collider other) : controlla se c’è collisione tra il collider e un secondo collider indicato, restituendone la direzione e bloccando il collider al collider confrontato, se per esempio sta arrivando una palla dal cielo e arriva ad una piattaforma allora la palla si deve fermare sopra alla piattaforma, lo stesso vale per le altre dimensioni, i calcoli devono essere diversi in base al tipo di collider rilevato

## CircleCollider extends Collider

Indica un collider circolare(NON ELLISSE, X E Y SONO UGUALI)

### Metodi

* CollisionDirection CheckCollision(Collider other) : controlla se c’è collisione tra il collider e un secondo collider indicato e ne restituisce la direzione, i calcoli devono essere diversi in base al tipo di collider rilevato
* CollisionDirection CheckCollisionSnap(Collider other) : controlla se c’è collisione tra il collider e un secondo collider indicato, restituendone la direzione e bloccando il collider al collider confrontato, se per esempio sta arrivando una palla dal cielo e arriva ad una piattaforma allora la palla si deve fermare sopra alla piattaforma, lo stesso vale per le altre dimensioni, i calcoli devono essere diversi in base al tipo di collider rilevato

## CollisionDirection

Il lato del collider su cui è avvenuta la collisione

* NONE
* RIGHT
* LEFT
* UP
* DOWN

# 

# Gruppo UI

Tutti il sistema dell'interfaccia grafica verrà gestito da questo gruppo, a partire da un elemento generico non definito fino ad arrivare a elementi più specifici come dei bottoni

## UIElement

LA POSIZIONE È IL CENTRO DELL’OGGETTO

Un elemento generico dell’interfaccia grafica, ha il riferimento al suo oggetto e al suo transform(per comodità anche se si potrebbe ottenere dal gameObject), deve avere riferimenti a tutto ciò che lo definisce graficamente.

### Attributi

* GameObject gameObject : l’oggetto a cui è attaccato l’elemento
* Transform transform : posizione e dimensione dell’oggetto a cui è attaccato l’elemento
* String text : il testo scritto
* int color : il colore dell’elemento UI
* PImage texture : la texture dell’elemento UI

### Metodi

* abstract void Display() : mostra l’elemento dell’UI

## Button extends UIElement

Elemento UI semplice che sovrascrive il display e indica un modo per capire se il mouse si trova sopra al bottone.

### Attributi

* int id : l’id del bottone

### Metodi

* void Display() : mostra il bottone
* boolean MouseOver() : indica se il mouse è sopra il bottone

## 

## Panel extends UIElement

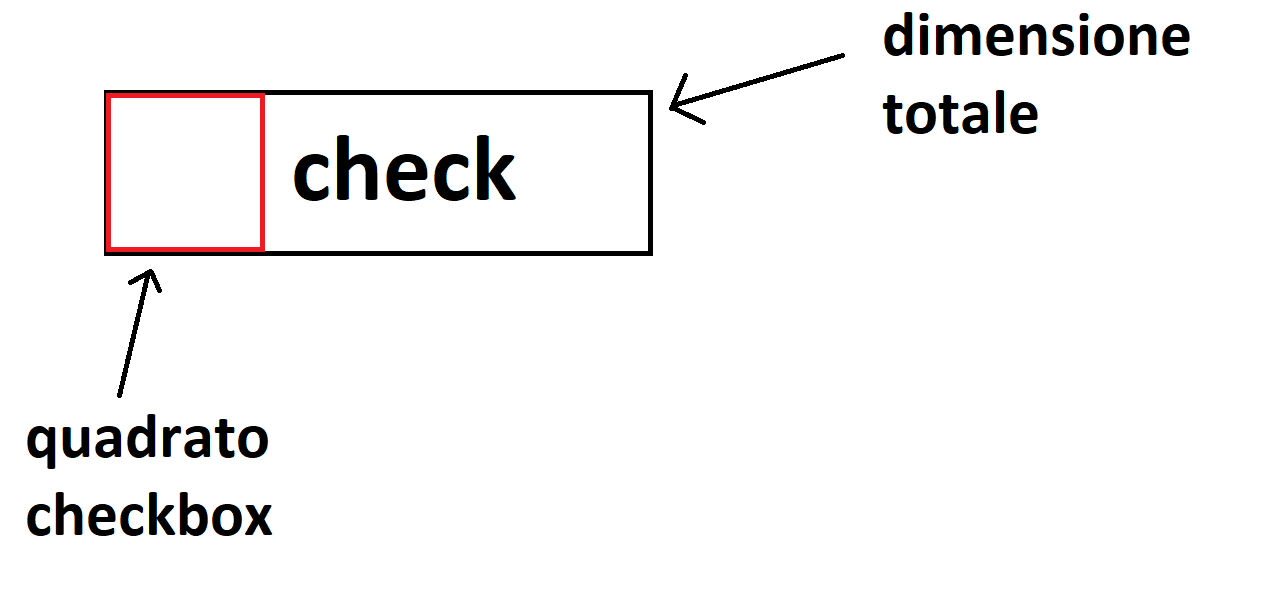
Elemento UI semplice che sovrascrive il metodo display per mostrare un pannello (con o senza testo), è l'equivalente di una TextBox

### Metodi

* void Display() : mostra il pannello di testo

## CheckBox extends UIElement

Definizione di una checkbox. La texture di base è la texture in caso la box non sia checked, se è checked usa fillTexture, come icona della checkbox usa checkIcon, se non è utilizzata utilizza quella di default “defaultCheck”. Se non ci sono texture usa color per fare una base e fillColor come colore quando è checked. Il testo va scritto a destra della checkbox. La dimensione indica tutta l’area della checkbox ovvero la checkbox in se più il testo, la dimensione del quadrato di check dev’essere quanto l’altezza. Per capire meglio basarsi sull’immagine di riferimento.



### Attributi

* static PImage defaultCheck : l’icona di default per il check, fornita da PM
* int fillColor : il colore di fill per la checkbox
* PImage fillTexture : la texture di fill per la checkbox
* PImage checkIcon : l’icona del check da visualizzare quando attivo
* boolean checked : indica se checked oppure no

### Metodi

* void Display() : mostra la checkbox
* boolean MouseOver() : indica se il mouse è sopra la checkbox(compreso il testo)
* void setChecked(boolean state) : setter per il checked, indica se lo deve attivare o disattivare
* void toggleChecked() : se è checked allora lo unchecka, se no allora diventa checked
* boolean isChecked() : restituisce se è checked oppure no

# Gruppo GameObject

La gestione del sistema degli oggetti, delle loro coordinate e dello scripting è gestita da questo gruppo. Renderà possibile la creazione di oggetti che popoleranno la scena e degli script per poterne ampliare le funzionalità

## GameObject

LA POSIZIONE È IL CENTRO DELL’OGGETTO

Indica un oggetto qualsiasi della scena, l’oggetto ha una posizione e una dimensione, un nome identificativo ed una lista di componenti che ne ampliano il funzionamento. Un componente può essere un oggetto qualsiasi, come un collider oppure uno script personalizzato

### Attributi

* ArrayList<Object> componentList : i vari componenti attaccati all’oggetto
* Transform transform : la posizione e la dimensione dell’oggetto
* String name : il nome dell’oggetto

### Metodi

* T GetComponent(Class<T> type) : restituisce il componente del tipo richiesto, se non esiste deve dare errore
* void AddComponent(T component) : aggiunge il componente all’oggetto, se esiste già un componente dello stesso tipo allora da errore. Se il componente è uno script, un collider, un renderer o un elemento UI allora inizializza i suoi attributi “transform” e “gameObject”. Importante fare in modo che non si possa aggiungere un Transform come componente. ATTENZIONE NON POSSONO ESISTERE DUE ELEMENTI CHE DERIVANO DA “UIElement” O “Renderable”
* ArrayList<Object> GetComponents() : restituisce i componenti dell’oggetto
* String GetName() : restituisce il nome dell’oggetto
* boolean HasComponent(Class<T> type) : restituisce true se l’oggetto ha un componente del tipo richiesto, altrimenti false

## Script

LA POSIZIONE È IL CENTRO DELL’OGGETTO

Il modello per uno script personalizzato da attaccare ad un oggetto come componente, può essere esteso per creare una funzionalità personalizzata

### Attributi

* GameObject gameObject : l’oggetto a cui è attaccato questo componente
* Transform transform : la posizione e la dimensione dell’oggetto a cui è attaccato questo componente

### Metodi

* void Start() : il metodo che viene eseguito al caricamento della scena, da lasciare vuoto, si può fare l’override alla creazione di uno script, non serve che sia un metodo astratto
* void Update() : il metodo che viene eseguito ogni frame, da lasciare vuoto, si può fare l’override alla creazione di uno script, non serve che sia un metodo astratto
* void OnCollisionEnter(Collider self, Collider other) : eseguito quando l’oggetto entra in collisione con qualcosa

## Transform

LA POSIZIONE È IL CENTRO DELL’OGGETTO

Il modello per la posizione e la dimensione di un oggetto, contiene 2 vettori bidimensionali per poterle definire

### Attributi

* Vector2 position : la posizione dell’oggetto
* Vector2 scale : la dimensione dell’oggetto

### Metodi

* Vector2 GetPosition() : restituisce la posizione
* Vector2 GetScale() : restituisce la dimensione
* void SetPosition(float x, float y) : imposta la posizione dell’oggetto
* void SetPosition(Vector2 pos) : imposta la posizione dell’oggetto
* void SetScale(float x, float y) : imposta la dimensione dell’oggetto
* void SetScale(Vector2 scale) : imposta la dimensione dell’oggetto

## 

## Vector2

Il modello di un vettore bidimensionale, contiene il riferimento ai valori su due assi e tutti i metodi di utility per impostazioni e operazioni

### Attributi

* float x : il valore sull’asse x
* float y : il valore sull’asse y

### Metodi

* void Set(float x, float y) : imposta i valori del vettore
* void Set(Vector2 vec) : imposta i valori del vettore
* void SetX(float x) : imposta il valore della x
* void SetY(float y) : imposta il valore della y
* float GetX() : restituisce il valore della x
* float GetY() : restituisce il valore della y
* boolean Equals(float x, float y) : controlla se i valori del vettore sono uguali a quelli passati
* boolean Equals(Vector2 vec) : controlla se il vettore è uguale al vettore passato
* static Vector2 RIGHT() : restituisce un nuovo vettore che indica la direzione verso destra (probabilmente [1,0])
* static Vector2 LEFT() : restituisce un nuovo vettore che indica la direzione verso sinistra (probabilmente [-1,0])
* static Vector2 DOWN() : restituisce un nuovo vettore che indica la direzione verso il basso (probabilmente [0,1])
* static Vector2 UP() : restituisce un nuovo vettore che indica la direzione verso l’alto (probabilmente [0,-1])
* Vector2 Sum(float value) : somma sia x che y con il valore indicato
* Vector2 Sum(float x, float y) : somma il vettore con i valori specificati
* Vector2 Sum(Vector2 vec) : somma il vettore con il secondo vettore specificato
* Vector2 Multiply(float value) : moltiplica sia x che y per il valore indicato
* Vector2 Multiply(float x, float y) : moltiplica il vettore per i valori specificati
* Vector2 Multiply(Vector2 vec) : moltiplica il vettore per il secondo vettore specificato
* Vector2 Divide(float value) : divide sia x che y per il valore indicato
* Vector2 Divide(float x, float y) : divide il vettore con i valori specificati
* Vector2 Divide(Vector2 vec) : divide il vettore con il secondo vettore specificato

# Gruppo Graphic System

Gestione del sistema della grafica e renderizzazione degli oggetti, fornisce un metodo per il disegno di oggetti inanimati oppure un modo per creare oggetti in movimento

## Renderable

LA POSIZIONE È IL CENTRO DELL’OGGETTO

Modello per un oggetto renderizzabile, generalizza un metodo di rendering e ha un riferimento al suo oggetto e transform

### Attributi

* GameObject gameObject : il riferimento all’oggetto a cui è attaccato
* Transform transform : il riferimento alla posizione e alla dimensione

### Metodi

* abstract void Render() : disegna l’oggetto

## Renderer extends Renderable

Gestione della visualizzazione di un oggetto, può essere un oggetto semplice dotato di forma (rettangolo e cerchio) con un colore, oppure può essere dotato di texture

### Attributi

* int color : il colore dell’oggetto
* PImage texture : la texture dell’oggetto
* DrawType renderType : il tipo di render che deve avere l’oggetto

### Metodi

* void Render() : disegna l’oggetto in base al tipo definito

## Animator extends Renderable

Gestore delle animazioni di un oggetto, contiene tutte le animazioni e i metodi per gestirle

### Attributi

* ArrayList<Animation> animationList : la lista delle animazioni per l’animator
* Animation currentAnimation : l’animazione attualmente in riproduzione

### Metodi

* void Render() : Esegue le animazioni in base agli state handler
* void PlayAnimation(String name) : ferma l’animazione in corso e fa partire l’animazione in base al nome
* void StopAnimation() : ferma l’animazione
* void PauseAnimation() : mette in pausa l’animazione in base al nome
* void SetAnimationList(ArrayList<Animation> list) : imposta la lista delle animazioni
* void AddAnimation(Animation anim) : aggiunge un’animazione alla lista

## 

## Animation

Il modello per un'animazione, contiene i vari frame, può essere loopato. So può far partire, mettere in pausa o fermare a piacimento. La velocità del cambiamento dei frame è personalizzabile per personalizzare il tutto nel modo migliore possibile :)

### Attributi

* ArrayList<PImage> frames : i frame dell’animazione
* boolean loop : se l’animazione deve andare in loop oppure no
* boolean isPlaying : indica se l’animazione è in esecuzione oppure no
* boolean isEnded : indica se l’animazione è finita
* int currentFrame : il frame attuale dell’animazione
* float frameDelay : ogni quanto deve cambiare frame(in secondi)
* float timer : il timer per il cambio del frame(in secondi, gestito con la libreria del DeltaTime)
* String name: nome/id dell’animazione

### Metodi

* void StateHandler() : gestisce l’animazione aggiornando i frame in base allo stato attuale e al timer definito, se deve andare in loop l’animazione ricomincia
* void Play() : fa partire l’animazione
* void Stop() : ferma l’animazione e la riporta al primo frame
* void Pause() : mette in pausa l’animazione
* int GetCurrentFrameIndex() : restituisce il numero del frame attuale dell’animazione
* void SetFrames(ArrayList<PImage> frames) : imposta i frame dell’animazione in base alla lista definita
* void AddFrame(PImage frame) : aggiunge il frame all’animazione

## DrawType

I possibili tipi di disegno(usato nel Renderer)

* TEXTURED
* CIRCLE
* RECT