Corso sulla realizzazione di un VIDEOGIOCO

Organizzato da: Tommaso Ripanti-alunno del 5°A
Marco Calvani-Professore di Matematica e Fisica
Anna Ceravolo-Professoressa di Matematica e Fisica

Presentazione progetto

Il progetto che vi proponiamo ha come obiettivo la divulgazione di un sapere e di un'arte che nei licei di Terni è ancora alle prime armi: lo sviluppo di un videogioco e le tante professioni affiliate a quest'ambito.

Cosa ci si deve aspettare? Un primo **corso** da **4 lezioni**, da **2 ore ciascuna**, che vi insegnerà in modo nettamente pratico le skills da apprendere per la creazione di un videogioco. A questo verrà correlata una **game jam a squadre,** dove i vari gruppi si sfideranno nella realizzazione di un gioco digitale, valutato in seguito da una giuria assieme ad un sondaggio generale della scuola.

Introduzione all'IDE (Ambiente di sviluppo)

In queste lezioni utilizzeremo, per la realizzazione della nostra opera, 2 software principali: **Unity**(game engine) e **Visual Studio Community**(editor di codice). Tralasciando il nostro corso, c'è da sapere che esistono svariati altri IDE di sviluppo. Ne cito quindi alcuni: Unreal Engine, Godot, Game Maker, Construct che sono **game engine**(software che facilita lo sviluppo di un videogioco o qualsiasi altra applicazione con grafica a tempo reale e permette che giri su varie piattaforme) e poi Visual Studio Code, Notepad++, Atom, Sublime Text che sono **editor di codice**(sono editor di testo che aiutano nella sintassi e nel debug).

Unity(interfaccia)

Partendo dall'alto abbiamo :

- **Primo menù a barra** con alcune voci: in **File** troveremo per salvare il progetto(**Save Project**) e per compilarlo e trasformarlo in un eseguibile(.exe su Windows o .apk su Android) con **Build and Run.**
- **Nella seconda barra** troviamo a sinistra vari modi per selezionare un oggetto all'interno della scena e centralmente il tasto **play** che permette di avviare il gioco nell'engine.
- **Nella finestra Hierarchy** di sinistra trovo l'elenco di tutti gli oggetti presenti nella scena di gioco attuale.
- **La scena di gioco** centrale mi mostra il gioco stesso al suo inizio. Se spingo il tasto play il gioco in tempo reale.

Unity(interfaccia)

Continuando abbiamo :

- Sulla destra il nostro **Ispector,** in cui troviamo vari parametri e impostazioni dell'oggetto da noi selezionato.
- In basso c'è **la cartella** che contiene i files del progetto: la nostra cartella **Assets** dove posizioneremo le scene, gli scripts, gli sprites e altri elementi che ci serviranno per il gioco. Poi **la Console** che ci mostrerà eventuali problemi durante l'esecuzione della scena di gioco o nostri messaggi di debug. Infine, sulla stessa barra, anche la sezione **Animation** che ci sarà utile quando animeremo alcuni oggetti della scena.

Visual Studio Community(Interfaccia)

Poche le cose da sapere per l'editor di testo:

- i simboli di **salvataggio** degli scripts.
- la **barra degli scripts** aperti
- il vero e proprio **editor di testo**

Un’infarinatura di scripting

Nel nostro script si presentano già alcune espressioni, andiamo a indagare:

- In alto troviamo i **namespaces,** che si possono immaginare come contenitori al cui interno ci sono già linee di codice pronte, scritte da terzi, che noi implementiamo nel nostro script perché utili ai nostri fini.
- con **class** si intende un prototipo con certe caratteristiche con cui si identificano le varie istanze(oggetti) che detengono la stessa classe. La classe può essere **public** se vogliamo usare i suoi elementi in altre classi, altrimenti **private** e cioè inaccessibile da parti esterne(lo stesso vale per un metodo o una variabile). **MonoBehaviour** è la tipologia di classe base di unity che rende possibile l’implementazione dello script ad un oggetto della scena e la disponibilità di alcuni metodi indispensabili per l’engine.

Una infarinatura di scripting

Abbiamo poi 2 **metodi**(o funzioni, che si presentano come uno spazio di codice che elabora un dato preso in ingresso per restituirne uno in uscita. Nonostante quanto affermato, alcune funzioni non ritornano valori: sono di tipo **void**) importantissimi:

- La funzione **Start** esegue il suo codice una volta sola prima del frame iniziale di gioco.
- La funzione **Update** esegue il suo codice ogni frame del gioco.

Nel nostro viaggio incontreremo altri metodi, come ne creeremo altri noi stessi e spesso usufruiremo di quelli void anche solo per organizzare meglio e rendere più comprensibile il nostro script, al posto di ammucchiare il tutto all'interno dell' Update.

Una infarinatura di scripting

Altro elemento di grande rilevanza sono le **variabili**(pensiamole come delle scatole che contengono degli oggetti di vario genere) che possono essere:

- **int:** numeri interi.
- **float:** numeri a virgola mobile(decimali).
- **char:** caratteri(lettere e simboli).
- **string:** parole o frasi(concatenazioni di caratteri).
- **bool:** valore booleano (true o false, 1 o 0)

Una infarinatura di scripting

- Fondamentale è anche il costrutto **if-else**, funziona così:

```
if(condizione 1){
        //fa qualcosa
}else if(condizione 2){
        //fa qualcos'altro
}else{
        //fa qualcos'altro
}
```

Spesso nella condizione sono utili questi **operatori**:

- if(**a == b**): “se a è uguale a b”, sono i **2 uguali di confronto**(Per assegnare ad una variabile un valore usiamo solo 1 uguale!)
- **if(a!=b)**: “se a è diverso da b”
- **if(a>b)**: “se a è maggiore di b”
- **if((a == b && c==d) || b==c)**: “se (a è uguale a b e c è uguale a d) o b é uguale a c

Una infarinatura di scripting

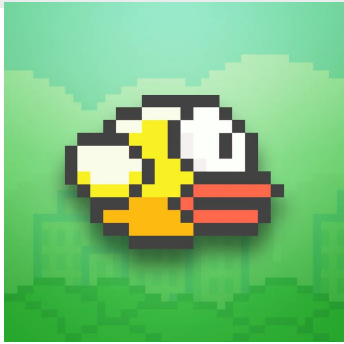
- Importanti sono anche i **cicli** :
  - ciclo **for**: si ripete finché la variabile inizializzata tra parentesi tonde, che aumenta di un tot ad ogni iterazione, non soddisfa più la condizione di accesso al ciclo.
  - ciclo **while:** si ripete finché la condizione tra parentesi tonde viene soddisfatta.
  - ciclo **foreach:** si ripete per ogni elemento della tipologia indicata presente nell'array della stessa tipologia.

- https://youtu.be/Pp4B7GAj78k
- https://youtu.be/9tMvzrqBUP8
- https://unity3d.com/learning-c-sharp-in-unity-for-beginners

```
for ( int i = 0; i < 10; i ++ ) {
        //fa qualcosa
}
```

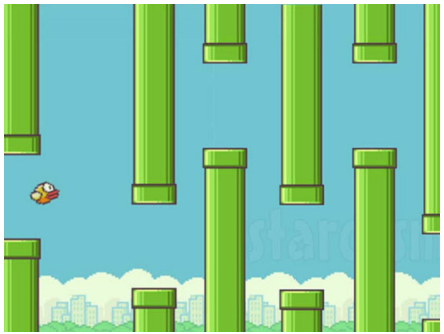
```
while ( gameover ) {
        //fa qualcosa
}
```

```
foreach ( int enemy in enemies){
        //fa qualcosa
}
```



Creiamo il nostro clone!

Siamo quindi pronti per realizzare il nostro gioco. Avete presente il mitico **Flappy bird**? Considerate che il suo creatore Dong Nguyen guadagnava giornalmente 50.000 dollari nonostante la semplicità del videogioco per smartphone. Divertente ma dannatamente difficile: a lavoro!



Prepariamo gli sprites e la scena di gioco

- Scaricare gli assets da unity asset store: **Landscape Tiles & Birds (Free)**. In una nuova cartella Sprites inseriamo i nostri assets insieme ad altre due icone per i controlli audio.
- Inseriamo lo sprite **“bird301”** nella scena dopo aver cambiato i **pixel per Unit** da 100 a **20**(20 pixel ogni unità di Unity-in questo modo lo sprite occuperà più unità e aumenterà le sue dimensioni rispetto alla camera). Rinominiamo l’oggetto in scena come **“Player”** e poniamolo a -**5 sull’asse x e sulla z** dal centro e a 0 sull’asse y.
- Inseriamo il **“med block 02”** anche lui con 20 pixel per Unit. Duplichiamo lo sprite più volte formando **2 colonne** con una distanza che permetta l’ingresso del player tra le due. Creiamo un **empty object** e chiamiamolo **“column”**. Selezioniamo i blocchi e trasciniamoli nella colonna(ora la colonna è l’oggetto **parent** e i blocchi sono i suoi **children**). Se il suo **pivot**(centro di rotazione) non è centrato, clicchiamo su **“Pivot”**. Posizioniamolo a **-5 sull’asse z** e dopo aver creato una cartella **“Prefabs”**, trasciniamo l’oggetto al suo interno e eliminiamolo dalla scena.

Prepariamo gli sprites e la scena di gioco

- Inseriamo un **GameObject “Game_manager”**(gestirà alcuni parametri della scena)
- Aggiungiamo alla scena un **Canvas** con **Render Mode** impostata su **Screen Space-Camera**(scegliere quindi la camera, in modo da avere una **UI responsive**)con all’interno:
  - 1 **text “Score”**
  - 1 **text “High_score”**
  - 1 **panel “Touch_panel”**
  - 1 **button “Audio_control”** con lo sprite del volume attivo incorporato
  - 2 **empty object “Tap_to_play”** e **“Tap_again”** con un Rect Transform nell’Inspector.
- Ricordiamo di settare Il display del gioco su una risoluzione fissa (**risoluzione di riferimento** - la stessa da settare nel canvas - , se vogliamo che il gioco si orienti in orizzontale facciamo si che la larghezza sia più ampia dell'altezza !)

Il player

- Nell’Inspector del player aggiungiamo questi componenti:
  - **Rigidbody 2D**(rende l’oggetto a cui è legato vulnerabile alla fisica dell’engine, impatti da collisione e gravità per esempio). Settiamo in **Costraints** la **Freeze position** sull’asse x.
  - **Capsule collider 2D** che rileva le collisioni con altri elementi dotati di un collider(per rilevare una collisione è anche necessario che uno dei due corpi abbia un rigidbody!). Editiamo il collider secondo lo sprite.

Il player

- Animiamo il player andando sulla tab **Animation** e cliccando su **Create**(assicuriamoci che sia selezionato il player nella hierarchy!)**.** Diamo un nome all'animazione e salviamola in una cartella che chiameremo **"Animations".** Nella **timeline** inseriamo a distanza debita gli altri due sprites correlati al primo(**"bird302" e "bird303"**). Assicuriamoci che la nostra **Animation Clip** sia in **Loop Time** e che al nostro player sia attaccato un componente **Animator**(vedremo poi come utilizzarlo nelle animazioni dei testi).

Il player

- Creiamo in un'apposita cartella **"Scripts"** il nostro **"Player"** script e aggiungiamolo al player. Apriamo il file con **Visual studio**.
- Nello script utilizzeremo varie classi come:
  - **Input** per il rilevamento e la gestione degli input esterni
  - **transform** per la gestione di posizione, rotazione, dimensioni dell'oggetto e delle relazioni parent-children
- Vi troveremo un nuovo metodo: **FixedUpdate()** da utilizzare per codice che agisce sulla fisica dell'engine e perciò su operazioni sui rigidbody.
- Utilizziamo **Time.deltaTime** per rendere il movimento dipendente dal tempo e non dai frame.

A differenza dell'**Update** che si ripete ad intervalli irregolari in quanto il delta di tempo tra ogni frame puó variare, il **FixedUpdate** si basa su Physical frame che si presentano a intervalli costanti

Le colonne, il loro spawner e l'object pooling

- Aggiungiamo **2 box collider 2D** e editiamoli in base alla forma delle colonne
- Aggiungiamo **1 box collider 2D** nel foro in modalità **trigger**(la collisione viene rilevata ma non produce effetti fisici)
- Creiamo il nostro **"Column"** script e aggiungiamolo al prefab colonna. Apriamo il file con **Visual studio**.

- Aggiungiamo nella scena un empty object e lo chiamiamo **"Columns- spawner".**
- Creiamo il nostro **"Spawner"** script e aggiungiamolo al nostro GameObject Spawner. Apriamo il file in **Visual Studio**.

Le colonne, il loro spawner e l'object pooling

- Utilizzeremo un nuovo elemento: la classe **Queue**(): un array in cui gli oggetti che vengono inseriti per ultimi sono anche gli ultimi ad uscirne.

L'object pooling è uno stratagemma per ridurre l'affaticamento della **CPU**. Al posto di spawnare e distruggere oggetti di continuo, quest'ultimi si attivano o disattivano secondo necessità. È chiamato così perché queste istanze vengono create tutte di seguito all'inizio del gioco e inserite in una sorta di "piscina-contenitore". All'uso vengono espulse dal contenitore per poi essere reinserite alla loro disattivazione.

Effetto parallasse - nuvole

- Posizioniamo una riga di nuvole lungo l'asse x.
- Creiamo il nostro **“Parallax”** script e aggiungiamolo ai nostri GameObject parent che contengono una serie di nuvole children. Apriamo il file in **Visual Studio**.

L’**effetto paralasse** non è altro che una finzione dell movimento del player, che in realtà non si muove assolutamente: è di fatti l’ambiente a lui circostante che si modifica.

Score e high score

- È ora di creare lo script del **“GameManager”** dove gestiremo il nostro score e il salvataggio dell’high score. Apriamolo in **Visual studio**.
- Per la gestione della **UI** è necessario inserire un nuovo namespace: **using UnityEngine.UI**.
- Le variabili dello score devono essere considerate **statiche** in quanto in gioco ci servirà uno e un solo score.
- Per salvare una variabile int, float, string, in modo tale da preservare il suo valore ogni volta che si riapre l’applicazione, usiamo la classe **“PlayerPrefs”** con i suoi metodi **SetInt(**“nome variabil”**,** valore**)** e **GetInt(**“nome variabile”, valore di default**)** nel caso di un intero.

Una **variabile statica** è un elemento che viene associato solamente alla classe in sè e non alle sue istanze. Questo significa che utilizzeremo il **modifier static** per individuare variabili uniche in gioco e che quindi non si ripresentano per ogni oggetto che possiede quella classe. Questo tipo di variabili non si resettano al caricamento di una nuova scena.

Touch Panel

In realtà per identificare un tocco sullo schermo o un click del mouse non serve necessariamente un tasto digitale nel Canvas. Si poteva tranquillamente usare la funzione **Input.GetMouseButtonDown(0)**, dove per 0 si intende il bottone di sinistra del mouse(funziona anche per il rilevamento touch nonostante ci siano altri metodi ancora). Ho preferito peró questo pannello touch affinchè si possano cliccare o toccare altri buttons del Canvas senza interagire anche con le dinamiche del gioco: considerato certamento come un bug dal giocatore.

- Nel component **Button,** all'interno dell'oggetto, troviamo la sezione **OnClick().** Clicchiamo due volte sul **+** per aggiungere le funzioni da chiamare al click del tasto. Trasciniamo quindi gli oggetti contenenti la classe in cui è presente il metodo di interesse.

Tap to play e le Courotines

- Creiamo nel Canvas un **empty object** con all'interno un **text**: sarà la nostra lettera modello da animare per comporre poi l'intera frase.
- Animiamo la lettera creando un'animazione come già fatto per il player ma questa volta attivando la modalità di registrazione e mutando la sua posizione verticalmente in frame differenti. Aprendo l'**animator** correlato, pigiamo su **+** per creare un nuovo parametro **trigger**(è come una variabile booleana ma con la differenza che dal valore true torna in automatico false al termine dell'animazione). Aggiungiamo uno stato **"Idle"** e lo impostiamo di default, per poi creare due **transizioni**(una di avvio e una di ritorno) con lo stato animato.
- Inseriamo in **Tap_to_play** e **Tap_again** copie della lettera animata, cambiandone ovviamente il contenuto di testo. Disporre quindi ogni lettera con una giusta distanza dall'altra. **Disattivare** poi **"Tap_again".**

Tap to play e le Courotines

- Creiamo lo scrip **TextEffect** e inseriamolo su ogni lettera. Apriamo **Visual Studio.**
- Abbiamo 2 nuovi elementi da indagare:
  - **List:** un array dinamico(che non necessita una dimensione prestabilita)
  - **Courotine:** una funzione chiamabile per via del metodo StartCourotine(nome IEnumerator) che possiede la capacità di riniziare da dove era rimasta dopo una pausa e non ricominciare da capo. Faremo uso di cicli per realizzare una animazione in loop.

Audio Control

- Creiamo nella scena un nuovo **empty object** e lo chiamiamo **“Audio_manager”**. Al suo interno inseriamo tre **Audio Source** e dopo aver aggiunto in una cartella **Audio** tre file audio e un **AudioMixer** trasciniamo all’interno delle sorgenti la musica o gli effetti scelti e in **output** l’apposito **Master** del **Mixer**.
- Apriamo un nuovo script su **Visual studio**: **“AudioManager”**.
- Esaminando il codice, troviamo la funzione **FindGameObjectsWithTag(**“tag”**)** che individua nella scena tutti i GameObject presenti con quel tag e ritorna un array.
- Il metodo **DontDestroyOnLoad(**gameobject**)** fa si che il gameobject tra parentesi non si distrugga al caricamento di una nuova scena di gioco.
- La funzione **GetChild(**index dell’oggetto figlio**)** ottiene l’oggetto figlio specificato dall’indice numerico che rappresenta il posizionamento del gameobject nella coda di figli partendo dall’alto.
- Nello script del GameManager controlliamo il volume(muto o non) della musica.

Pipeline, shaders e effetto bloom

- Cliccare su **Window-->Package Manager** e selezionare poi **Unity Registry** per la voce **Packages.** Quindi cercare il pacchetto **Universal RP** e installarlo insieme allo **Shader Graph.**
- Creare una cartella **Pipeline** in cui inserire un **Pipeline Asset(2D Renderer).**
- Cliccare su **Edit-->Project Settings-->Graphics** dove in **Scriptable Render Pipeline Setting** inseriamo il nostro **Renderer.**
- Andiamo poi sull'ispector della **Main Camera** e attiviamo il **Post Processing.**

Una render pipeline è uno script che svolge una serie di operazioni prendendo gli oggetti della scena, elaborandoli per poi proiettarli sul display

Il post processing è un insieme di elaborazioni tramite effetti per il miglioramento della qualità video

Pipeline, shaders e effetto bloom

- In una nuova cartella **Shaders&Materials** creiamo uno **Shader “Dissolve”.** Con il tasto destro del mouse clicchiamoci sopra e creiamo un **Material** dello stesso nome già legato allo shader.
- Attacchiamo il materiale al player.
- Apriamo lo shader nello **ShaderGraph** e iniziamo a editarlo.
- Controlliamo alcune variabili create via script del player con funzioni del tipo material.SetFloat(“reference parametro”, valore)
- Aggiungiamo alla scena un **Global Volume** e inseriamo l’effetto **Bloom** con i parametri attivi: **Threshold**(1) e **Intensity**

Creare la build del gioco

- Su **Edit-->Project Settings-->Player** fissiamo un orientamento di default e aggiungiamo **l'icona** del gioco e la **splash image**(Immagine di avvio).
- Per ridurre le dimensioni del gioco, eliminare tutti gli **assets superflui** e i **pacchetti** che non utilizziamo sul Package Manager. Altra mossa che possiamo tentare è quella di cambiare lo **Scripting Backend** su **IL2CPP** e mettere una spunta su **Split APKs by target architecture**.(crea più file **apk** con diverse architectures).
- Cliccare su **File-->Build Settings.** Cambiamo piattaforma su **Android** se vogliamo la versione mobile. Clicchiamo su **Build**.

Unity collaborate o GitHub for Unity

Plug-in per Unity che vi permetteranno di lavorare in team rimanendo sincronizzati dopo ogni modifica del vostro progetto.

- Per Collaborate è sufficiente attivare il servizio dall'editor e invitare i membri dalla dashboard. Dopo ció pubblicare i file.
- Per GitHub è necessario scaricare la versione desktop e iscriversi. Dopodiché creare una repository e inserire lí i file.

Altre utilità

Programmi PC

- Piskel
- Krita
- MagicaVoxel
- Blender

Programmi Android

- Huion Sketch

Video

- Unity
- Brackeys
- CodeMonkey
- Dani
- Playlist scelta

Asset Store

- Unity
- Brackeys

Game Jam

- In data ? si terrà la Donatelli Game Jam presso ?
- Ogni squadra partecipante di minimo 2 componenti si dovrà iscrivere mediante modulo Google indicando:
  - Nome squadra
  - Nome e cognome dei partecipanti
  - Nome Videogioco
  - 1 immagine-logo della squadra
  - 1 immagine-vetrina del videogioco
- Il punteggio assegnato ad ogni gioco presentato sarà costituito da:
  - Sondaggio generale della scuola a cui potranno partecipare studenti , insegnanti e personale scolastico.
  - Voto della giuria su tali punti specifici:
    - Originalità
    - Performance
    - Altro