SkiFree Remake

1 Introduzione 4

1.1 Informazioni sul progetto 4

1.2 Abstract 4

1.3 Scopo 4

1.3.1 Scopo Principale 4

1.3.2 Scopi didattici 4

2 Analisi 5

2.1 Analisi del dominio 5

2.2 Analisi e specifica dei requisiti 5

2.2.1 Spiegazione elementi tabella dei requisiti: 8

2.3 Use case 9

2.4 Pianificazione 1

2.5 Analisi dei mezzi 1

2.5.1 Software 1

2.5.2 Hardware 1

3 Progettazione 1

3.1 Design dell’architettura del sistema 1

3.2 Design dei dati e database 1

3.3 Design delle interfacce 1

3.4 Design procedurale 2

4 Implementazione 3

4.1 Creazione progetto Unity 2D 3

4.2 Setup ambiente di lavoro 4

4.3 Importazione sprite 4

4.3.1 Separare sprite multiplo in piccoli sprite singoli 5

4.4 Trasformare lo sprite in un GameObject (prefab) 7

4.5 Camera segue il personaggio 8

4.5.1 Creazione personaggio base 8

4.5.2 Implementazione della camera 8

4.6 Parte iniziale con ostacoli predefiniti 9

4.7 Generazione Mappa 11

4.7.1 Script LevelGenerator 11

4.8 Auto gestione degli ostacoli 15

4.8.1 Ostacoli statici 15

4.8.2 Ostacoli dinamici (lupo) 16

4.9 Yeti 16

4.9.1 Script Yeti 16

4.10 Menu principale 18

4.11 Script MenuPrincipale 18

4.12 Grafica del menu principale 19

4.13 Menu di pausa 24

4.13.1 Script MenuPausa 24

4.13.2 Grafica del menu di pausa 26

5 Test 26

5.1 Protocollo di test 26

5.2 Risultati test 30

5.3 Mancanze/limitazioni conosciute 31

6 Consuntivo 31

7 Conclusioni 31

7.1 Sviluppi futuri 31

7.2 Considerazioni personali 31

8 Glossario 31

9 Bibliografia 32

9.1 Bibliografia per articoli di riviste: 32

9.2 Bibliografia per libri 32

9.3 Sitografia 32

10 Allegati 32

# Introduzione

## Informazioni sul progetto

Allievo: Christian Monga

Docente responsabile: Geo Petrini

Scuola: CPT Trevano

Sezione: Informatica

Classe: I3BB

Inizio Progetto: 09.09.2022

Fine Progetto. **23.12.2022**

## Abstract

È una breve e accurata rappresentazione dei contenuti di un documento, senza notazioni critiche o valutazioni. Lo scopo di un abstract efficace dovrebbe essere quello di far conoscere all’utente il contenuto di base di un documento e metterlo nella condizione di decidere se risponde ai suoi interessi e se è opportuno il ricorso al documento originale.

Può contenere alcuni o tutti gli elementi seguenti:

* **Background/Situazione iniziale**
* **Descrizione del problema e motivazione**: Che problema ho cercato di risolvere? Questa sezione dovrebbe includere l'importanza del vostro lavoro, la difficoltà dell'area e l'effetto che potrebbe avere se portato a termine con successo.
* **Approccio/Metodi**: Come ho ottenuto dei progressi? Come ho risolto il problema (tecniche…)? Quale è stata l’entità del mio lavoro? Che fattori importanti controllo, ignoro o misuro?
* **Risultati**: Quale è la risposta? Quali sono i risultati? Quanto è più veloce, più sicuro, più economico o in qualche altro aspetto migliore di altri prodotti/soluzioni?

## Scopo

### Scopo Principale

Riuscire a ricreare il gioco SkiFree usando Unity

### Scopi didattici

Imparare a gestire un progetto completo in modo autonomo, prendendo i requisiti

# Analisi

## Analisi del dominio

Questo capitolo dovrebbe descrivere il contesto in cui il prodotto verrà utilizzato, da questa analisi dovrebbero scaturire le risposte a quesiti quali ad esempio:

* Background/Situazione iniziale
* Quale è e come è organizzato il contesto in cui il prodotto dovrà funzionare?
* Come viene risolto attualmente il problema? Esiste già un prodotto simile?
* Chi sono gli utenti? Che bisogni hanno? Come e dove lavorano?
* Che competenze/conoscenze/cultura posseggono gli utenti in relazione con il problema?
* Esistono convenzioni/standard applicati nel dominio?
* Che conoscenze teoriche bisogna avere/acquisire per poter operare efficacemente nel dominio?
* …

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Visuale del gioco |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il gioco deve essere visualizzato in full screen |
| **Sotto requisiti** | |
| **001** | Deve essere in 2D |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | La mappa è infinita |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sotto requisiti** | |
| **001** | La mappa viene generata automaticamente in base alla posizione del personaggio |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | La mappa contiene degli ostacoli |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sotto requisiti** | |
| **001** | Vengono generati casualmente all’interno della mappa |
| **002** | Gli ostacoli devono essere:   * Alberi (quelli piccoli vengono bruciati quando saltati ad alta velocita) * Sassi * Seggiovie * Altri sciatori o snowboarders * Trampolini * Lupi |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | I movimenti dello sciatore vengono dettati dal mouse |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sotto requisiti** | |
| **001** | La rotazione del personaggio deve essere fluida |
| **002** | Se lo sciatore è messo orizzontalmente si ferma |
| **003** | Con il click del tasto sinistro lo sciatore salta |
| **004** | Se si tiene il tasto destro premuto in volo lo sciatore fa il mortale |
| **005** | Lo sciatore in volo non può cambiare traiettoria |
| **006** | Se atterra male o colpisce gli ostacoli si schianta, si ferma e dopo si rialza |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Jeti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sotto requisiti** | |
| **001** | Compare uno jeti ogni 2500m percorsi |
| **002** | Lo jeti vuole mangiare lo sciatore |
| **003** | Lo jeti è più veloce dello sciatore |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | I dati dell’utente vengono memorizzati e mostrati |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sotto requisiti** | |
| **001** | Deve essere salvato il tempo di gioco della partita |
| **002** | Deve essere salvata la distanza percorsa |
| **003** | Deve essere mostrata la velocità attuale |
| **004** | Devono essere mostrato il punteggio |

|  |  |
| --- | --- |
| **ID: REQ-008** | |
| **Nome** | Il gioco finisce quando lo sciatore viene mangiato |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Lo jeti mangia lo sciatore |

|  |  |
| --- | --- |
| **ID: REQ-009** | |
| **Nome** | Menù di pausa |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Cliccando “esc” si apre il menù di pausa |
| **Sotto requisiti** | |
| **001** | Blocca il gioco |
| **002** | Si può chiudere l’applicazione |
| **003** | Si può tornare all’inizio |

### Spiegazione elementi tabella dei requisiti:

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

**Sotto requisiti**: elementi che compongono il requisito.

## Use case

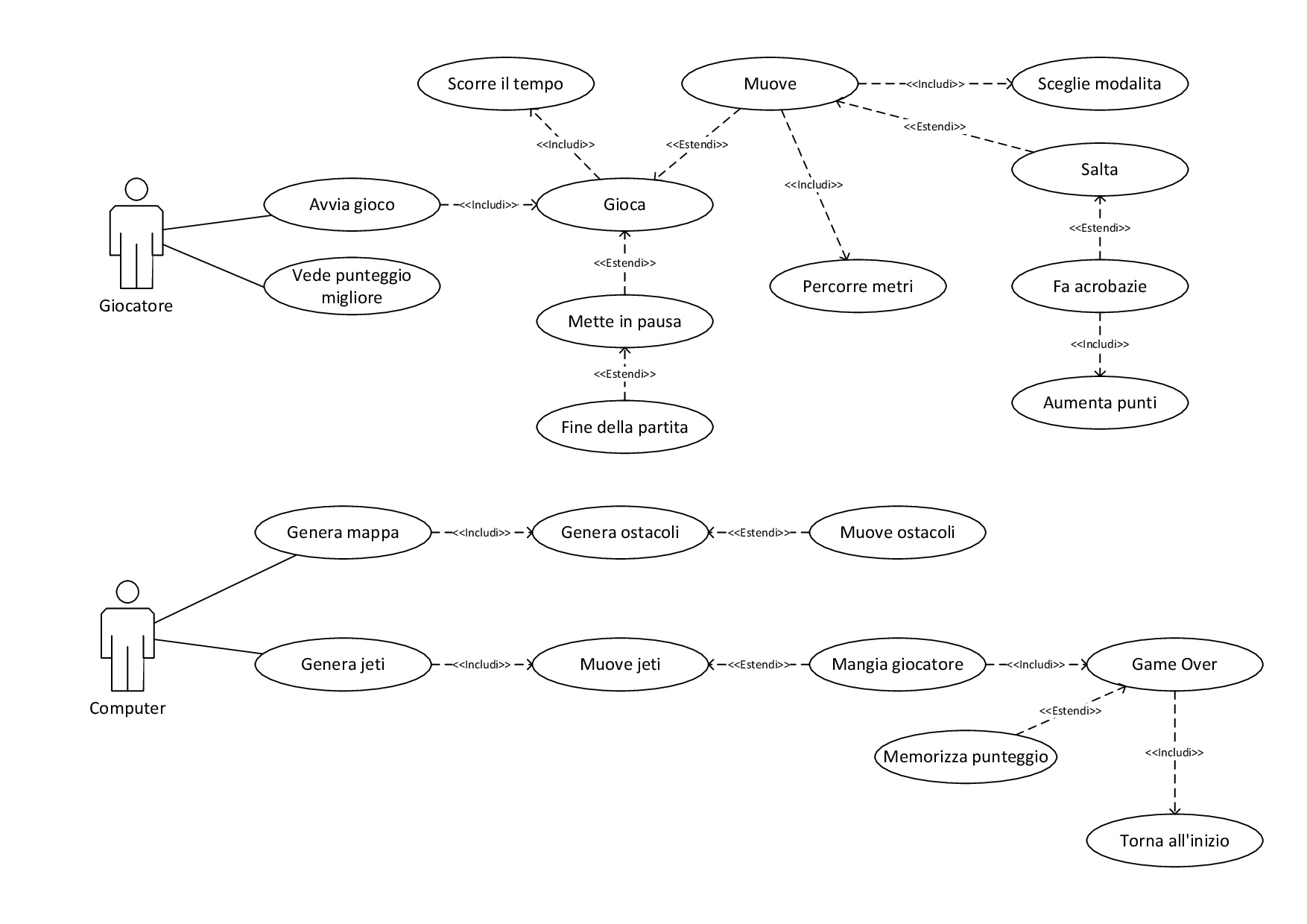
I casi d’uso rappresentano l’interazione tra i vari attori e le funzionalità del prodotto. 

Figura 1 UML Use Case del progetto

## Pianificazione

La pianificazione può essere rappresentata mediante un diagramma di Gantt:

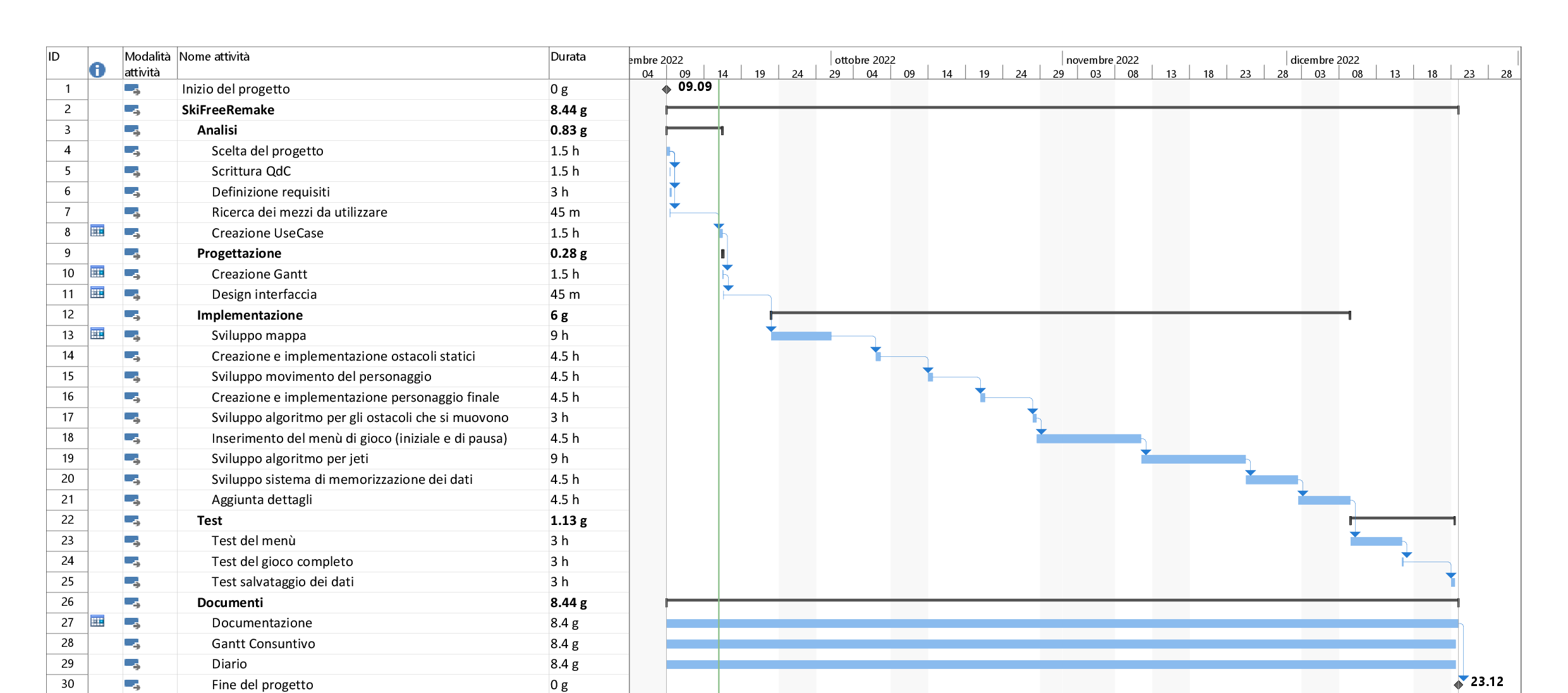


Figura 2 Diagramma di Gantt Iniziale

Per questo progetto utilizzo la pianificazione Iterativa evolutiva.

## Analisi dei mezzi

* Unity 2022.1.1f1
* Microsoft Visio
* Microsoft Project
* Microsoft Word
* Visual Studio Code con estensione C# v1.25.2
* Github con Git

### Software

SDK, librerie, tools utilizzati per la realizzazione del progetto e eventuali dipendenze.

### Hardware

Su quale piattaforma dovrà essere eseguito il prodotto? Che hardware particolare è coinvolto nel progetto? Che particolarità e limitazioni presenta? Che HW sarà disponibile durante lo sviluppo?

Il prodotto potrà essere eseguito su Windows

# Progettazione

Questo capitolo descrive esaustivamente come deve essere realizzato il prodotto fin nei suoi dettagli. Una buona progettazione permette all’esecutore di evitare fraintendimenti e imprecisioni nell’implementazione del prodotto.

## Design dell’architettura del sistema

Descrive:

* La struttura del programma/sistema lo schema di rete...
* Gli oggetti/moduli/componenti che lo compongono.
* I flussi di informazione in ingresso ed in uscita e le relative elaborazioni. Può utilizzare *diagrammi di flusso dei dati* (DFD).
* Eventuale sitemap

## Design dei dati e database

Descrizione delle strutture di dati utilizzate dal programma in base agli attributi e le relazioni degli oggetti in uso.

Schema E-R, schema logico e descrizione.

Se il diagramma E-R viene modificato, sulla doc dovrà apparire l’ultima versione, mentre le vecchie saranno sui diari.

## Design delle interfacce

Descrizione delle interfacce interne ed esterne del sistema e dell’interfaccia utente. La progettazione delle interfacce è basata sulle informazioni ricavate durante la fase di analisi e realizzata tramite mockups.

## Design procedurale

Descrive i concetti dettagliati dell’architettura/sviluppo utilizzando ad esempio:

* Diagrammi di flusso e Nassi.
* Tabelle.
* Classi e metodi.
* Tabelle di routing
* Diritti di accesso a condivisioni …

Questi documenti permetteranno di rappresentare i dettagli procedurali per la realizzazione del prodotto.

# Implementazione

## Creazione progetto Unity 2D

Per creare il progetto Unity in 2D bisogna inizialmente aprire Unity Hub, poi bisogna premere sul tasto “New Project.

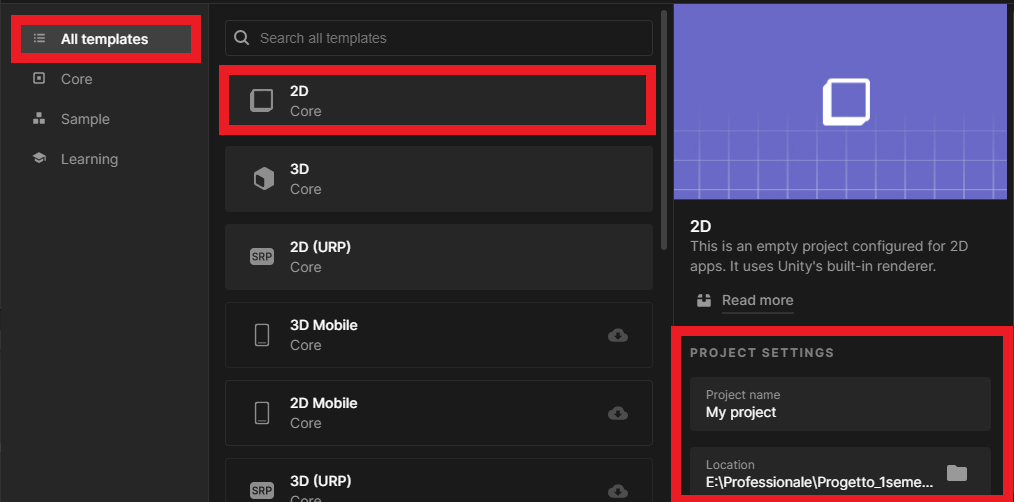


Figura 3 Creazione progetto Unity 2D

Dopodiché bisogna premere su All templates, selezionare 2D e in basso a destra definire il nome del progetto e il percorso di salvataggio.

## Setup ambiente di lavoro

Prima di incominciare è molto comodo ordinare le cartelle e le scene di Unity.

Per prima cosa si deve andare nel menu Project e premere sulla cartella Assets.  
Qui si devono creare le varie cartelle che conterranno i vari sprite, prefabbricati, script e animazioni.

Quindi si deve semplicemente fare tasto destro nello spazio apposito e creare queste cartelle:

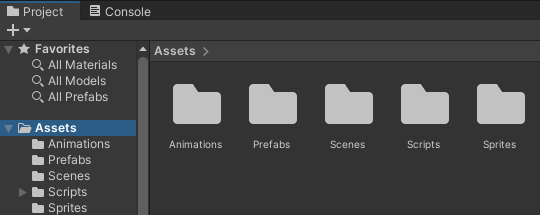


Figura 4 Cartelle di progetto

Dopo aver fatto questo si deve rinominare la scena principale e per farlo si deve entrare nella cartella Scenes, fare tasto destro sull’unica scena presente e rinominarla con un nome a proprio piacimento.

## Importazione sprite

Per poter aggiungere degli oggetti all’interno del gioco come per esempio alberi, animali, case, eccetera, bisogna importarli sotto forma di immagine .png. Questi sprite si possono scaricare da internet, più ufficialmente dal sito di Unitiy[[1]](#footnote-1).

Una volta scaricati basterà andare nella cartella Sprites creata in precedenza e trascinarci all’interno l’immagine scaricata.

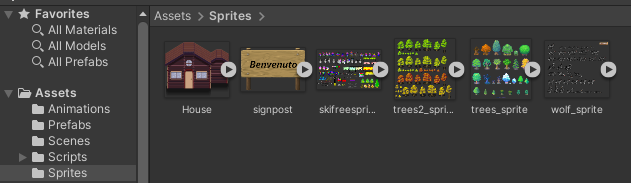


Figura 5 Sprites

Come si può vedere gli sprite non devono essere per forza un’immagine contenente un solo oggetto, ma possono contenere più immagini separate.

### Separare sprite multiplo in piccoli sprite singoli

Per separare questi sprite multipli in molti sprite singoli da poter usare all’interno del workspace bisogna selezionare lo sprite e andare nella sezione Inspector:

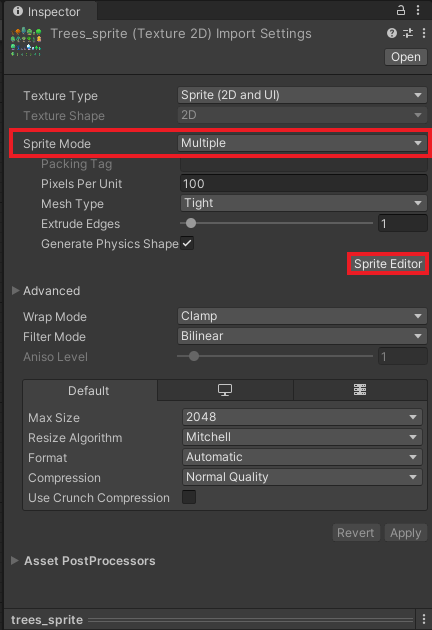


Figura 6 Sprite inspector

Per poter separare gli sprite bisogna impostare la Sprite Mode come Multiple e poi si deve premere sul tasto Sprite Editor.

Una volta premuto Sprite Editor si aprirà una piccola finestra con il proprio sprite aperto:

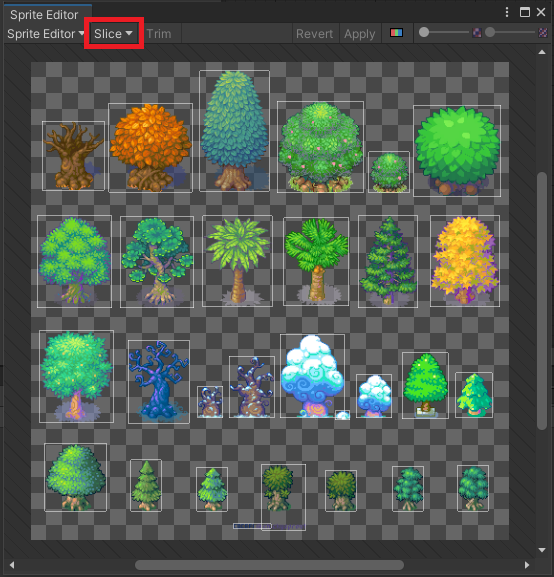


Figura 7 Slice sprite

Quando si apre la finestra bisogna andare sotto Slice e premere il pulsante Slice, dopo aver premuto tutti i vari componenti dello sprite verranno selezionati, come si può vedere nella foto. Questa selezione indica il nuovo sprite singolo di ogni componente. Infine bisogna premere su Apply e nellla cartella Sprites se si premerà la piccola freccia sull’immagine si aprirà una lista con tutti gli sprite singoli di ogni albero da poter utilizzare.

## Trasformare lo sprite in un GameObject (prefab)

Dopo aver separato gli sprite per poterli utilizzare come game object manca ancora un passo, bisogna trasformarli in prefabs (prefabbricati).

Per trasformarli bisogna prendere ogni singolo sprite che si vuole utilizzare, trascinarlo nella sezione Hierarchy e trascinarlo infine nella carrtella Prefabs:

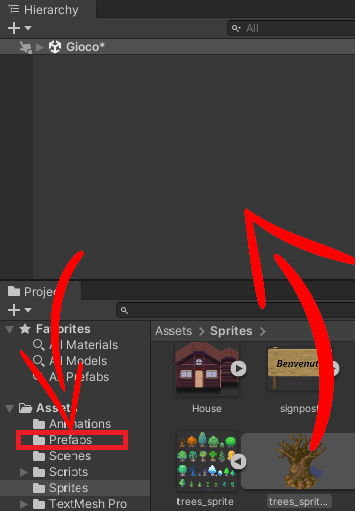


Figura 8 Fare dei prefabbricati

Dopodiché si avranno tutti i vari prefabbricati sotto la cartella Prefabs e per poterli utilizzare basterà ritrascinarli all’interno della scena.

Per cambiare le loro proprietà invece bisognerà selezionarli dalla cartella Prefabs e modificare le varie impostazioni nella sezione Inspector, così facendo le modifiche si applicheranno a tutti i prefabbricati copiati nel workspace di Unity.

## Menu principale

Il menu principale è la prima schermata che si presenta quando si avvia il gioco e permette di iniziare una partita o di uscire dall’applicazione.

### Script MenuPrincipale

Per creare un menu principale bisogna prima di tuto creare una nuova scena, per farlo bisogna andare sotto la cartella Assets/Scenes/ su unity:

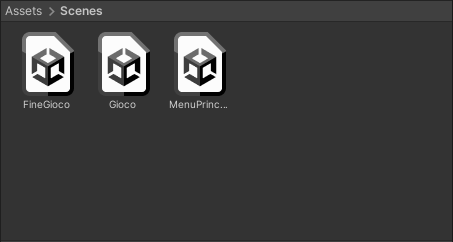


Figura 9 Scenes

Qui bisogna fare tasto destro, poi Create e selezionare Scene. Così si sarà creata una nuova scena che si chiamerà MenuPrincipale. Per aprirla bisogna fare doppio click su di essa.  
Adesso si dovranno aggiungere i pulsanti per poter giocare e uscire dal gioco, quindi si deve creare un nuovo script per la gestione di questo menu.

Lo script è molto semplice:

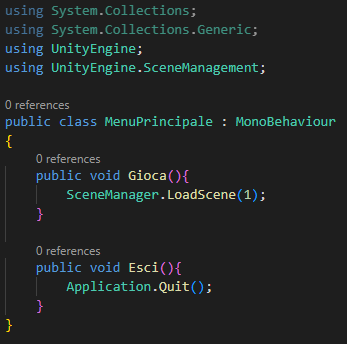


Figura 10 Script MenuPrincipale

La prima riga da aggiungere all’interno dello script è “using UnityEngine.SceneManagement” per permettere a Unity di gestire le scene.   
In seguito basta implementare i seguenti metodi:

* Gioca() 🡪 serve per caricare la scena di gioco
* Esci() 🡪 sere per uscire e chiudere l’applicazione

Come si vede dallo screenshot precedente il metodo esci contiene semplicemente un comando di sistema che permette di chiudere l’applicazione, mentre nel metodo gioca bisogna caricare una scena.

Per sapere l’ordine in cui le proprie scene sono impostate bisogna andare sotto il menu File e poi aprire Build Settings. Qui bisogna prendere le proprie scene e trascinarle sotto Scenes In Build:

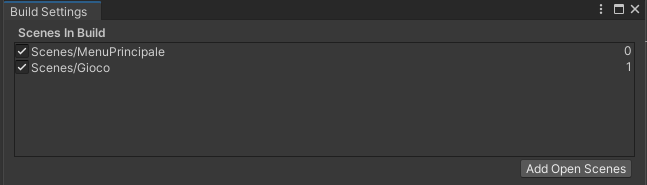


Figura 11 Ordine delle scene

A sinistra viene mostrato il nome della scena mentre a destra viene mostrato l’indice corrispondente alla scena.

Tornando quindi al metodo Gioca() si deve inserire il numero 1 negli argomenti del metodo .LoadScene() così prenderà la scena corrispondente al valore 1, quindi caricherà la partita.

### Grafica del menu principale

A questo punto lo script è pronto e si può passare alla parte un po’ più grafica del menu.

Per iniziare si deve creare un Canvas che conterrà i pulsanti del menu, quindi si deve andare sotto la sezione Hierarchy, fare tasto destro e sotto UI selezionare canvas.

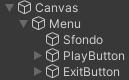


Figura 12 Canvas menu principale

All’interno del canvas si deve poi creare un Empty GameObject, chiamato Menu, che farà da contenitore per i pulsanti e per lo sfondo del menu.  
Per prima cosa bisogna aggiungere lo script MenuPrincipale a questo GameObject, quindi AddComponent, si cerca lo script e lo si aggiunge.

Questo GameObject dovrà adattarsi automaticamente al cambiamento di risoluzione quindi bisogna andare nel suo Inspector e nella proprietà Rect Transform bisogna impostare Stretch da entrambi i lati:

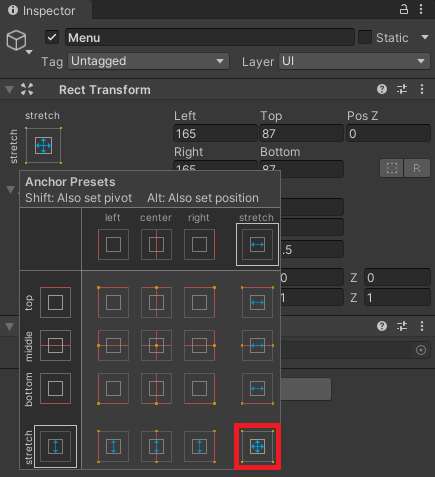


Figura 13 Rect Transform del menu

Dentro il GameObject Menu bisogna creare due Button: uno per iniziare la partita e l’altro per chiudere l’applicazione. Per creare un pulsante bisogna fare tasto su Menu, andare sotto UI e premere su   
Button – TextMeshPro, questo creerà un pulsante con all’interno un oggetto Text che contiene il testo che verrà visualizzato sul pulsante.

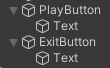


Figura 14 Pulsanti del menu principale

Nell’inspector dell’oggetto Text si può modificare a proprio piacimento lo stile della scritta del pulsante:

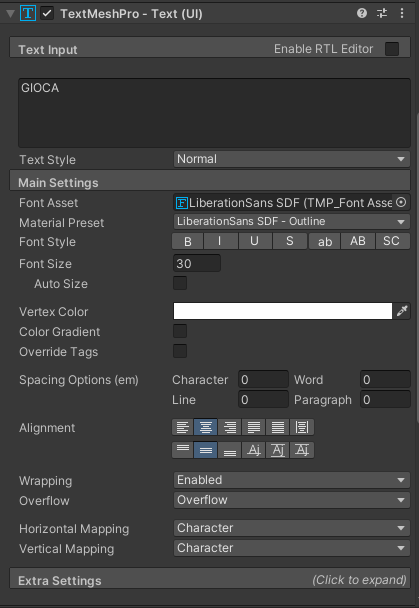


Figura 15 Inspector di Text

Nel riquadro iniziale si deve scrivere il testo che si vuole veder visualizzato sul pulsante, come per esempio GIOCA o ESCI. Sotto ci sono tutte le varie impostazioni per lo stile della scritta e, in particolare, Vertex Color rappresenta il colore di essa.

Una volta modificata la scritta dei pulsanti bisogna andare nell’Inspector di quest’ultimi si può modificare lo stile e le funzioni del pulsante:

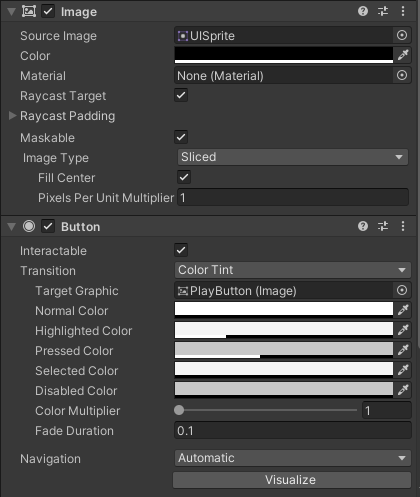


Figura 16 Inspector del pulsante

Per prima cosa bisogna impostare il colore della proprietà Image come nero, poi bisogna cambiare i colori nella proprietà color in questo modo:

* Normal Color 🡪 bisogna impostare la trasparenza a 0



Figura 17 Trasparenza colore pulsanti

* Highlighted Color 🡪 bisogna impostare la trasparenza a 60
* Pressed Color 🡪 bisogna impostare la trasparenza a 100 con un colore grigio chiaro
* Selected Color e Disabled Color 🡪 non verranno usati quindi si mette la trasparenza a 0

In questo modo i pulsanti non avranno un colore di sfondo ma quando si andrà sopra di essi con il mouse ci comparirà uno sfondo poco acceso e quando si premeranno lo sfondo diventerà più visibile, dando così un effetto di pressione del pulsante.

In seguito si deve impostare l’azione che il pulsante deve svolgere al momento della pressione. Quindi bisogna andare sotto il componente Button del pulsante e bisogna premere il piccolo + nella proprietà  
On Click().

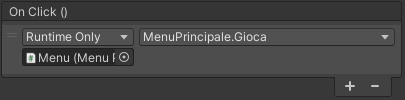


Figura 18 On Click() dei pulsanti

Qui bisogna trascinare il GameObject Menu, che contiene lo script MenuPrincipale, e bisogna selezionare il metodo corrispettivo al pulsante.

A questo punto si deve creare un GameObject Image che conterrà lo sfondo del menu, esso va creato sempre sotto il GameObject Menu.  
Per prima cosa nella proprietà Image bisogna inserire l’immagine che si vuole avere come sfondo, per farlo bisogna semplicemente trascinare l’imagine che si vuole impostare nel Source Image:

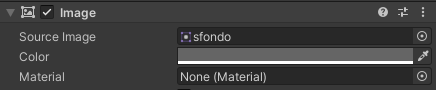


Figura 19 Immagine di sfondo

Poi per rendere lo sfondo un po’ più scuro bisogna impostare come colori RGB tutti a 100 e la trasparenza a 255, così da creare un grigio che rende più scuro lo sfondo, in questo modo:



Figura 20 Sfondo menu principale

Per inserire altre immagini a proprio piacimento basta creare un GameObject Image sotto il Menu, dare le dimensioni all’immagine e trascinare l’immagine sotto Source Image.

## Menu di pausa

Il menu di pausa permette di riprendere la partita o di tornare al menu principale bloccando temporaneamente la partita

### Script MenuPausa

Per fare il menu di pausa bisogna creare un nuovo script chiamato MenuPausa e anche qui, come nello script MenuPrincipale, bisogna implementare lo “using.UnityEngine.SceneManagement”.  
In seguito bisogna creare due variabili pubbliche, una di tipo GameObject che conterrà il menu da mostrare o da nascondere mentre l’altra è una variabile booleana per controllare se il gioco è in pausa oppure no.

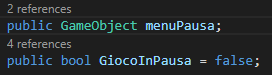


Figura 21 Variabili iniziali dello script MenuPausa

In seguito bisogna creare tre metodi:

* void Riprendi() 🡪 disattiva il menu di pausa e riprende la partita
* void Pausa() 🡪 attiva il menu di pausa e blocca la partita
* void TornaAlMenu() 🡪 torna al menu principale

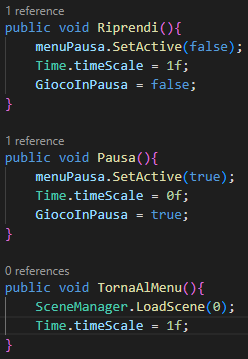


Figura 22 Metodi per il menu di pausa

All’interno del metoto Riprendi() bisonga disattivare la schermata del menu principale e grazie al .SetActive(false) essa viene nascosta e disattivata, poi viene reimpostato il Time.timeScale al valore di default, cioè 1, e infine viene impostata la variabile GiocoInPausa a false (questa variabile serve anche

per il movimento del personaggio).

Nel metodo Pausa(), invece, viene svolto il contrario del metodo Riprendi(): viene attivato il menu, viene bloccato il tempo e viene impostata a true la variabile GiocoInPausa.

Infine nel metodo TornaAlMenu() viene semplicemente caricata la scena del menu principale reimpostando il tempo a 1.

A questo punto bisogna lavorare nel metodo Update():

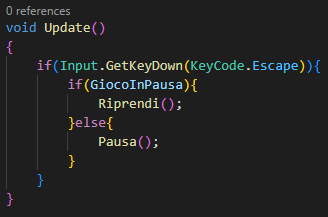


Figura 23 Metodo Update() dello script MenuPausa

Nel metodo Update() bisogna controllare periodicamente se l’utente preme ESC e poi deve essere svolto uno dei due metodi creati in precedenza.

Se il menu è aperto e viene premuto ESC la partita riprende, mentre se la partita è in corso e viene premuto ESC si apre il menu di pausa e la partita si ferma.

### Grafica del menu di pausa

Il menu di pausa deve trovarsi nella scena della partita, quindi nella scena Gioco. Come il menu principale bisogna creare un canvas contenente un Empty GameObject che conterrà i due pulsanti.

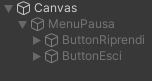


Figura 24 GameObject contenente il menu di pausa

Quando inizialmente si creano i due pulsanti se si prova ad avviare il gioco essi saranno sempre visibili.

Per questo bisogna andare sotto l’Inspector del GameObject MenuPausa e disattivarlo, per farlo bisogna togliere il visto nel primo quadratino visualizzato in alto:

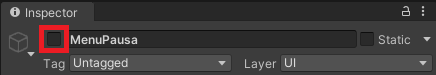


Figura 25 Disattiva il menu di pausa di default

In questo modo, di default, sarà nascosto.

Per collegare lo script alla grafica bisogna andare nel GameObject Canvas creato in precedenza, bisogna premere Add Component e bisogna semplicemente aggiungere lo script MenuPausa.

Infine bisogna trascinare il GameObject MenuPausa nella variabile apposita chiamata Menu Pausa:

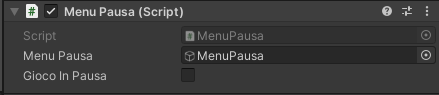


Figura 26 MenuPausa assegnato alla sua variabile

Infine bisogna configurare i pulsanti nello stesso modo del menu principale usando i metodi dello script MenuPausa.

## Camera segue il personaggio

### Creazione personaggio

Prima di tutto bisogna creare un Empty GameObject chiamato Player che conterrà il Player\_Body, così da rendere più semplice la gestione del personaggio.

In Player\_Body nel componente Sprite Renderer si deve inserire lo sprite dello sciatore e nel componente Transform si deve impostare la rotazione X a 180°. Questo perché dopo si andrà ad utilizzare un metodo dove la parte più in alto dello sprite seguirà il mouse e così facendo lo sprite seguirà il mouse con gli sci e non con la testa.

Nel GameObject Player invece bisogna inserire sia il RigidBody2D sia il PolygonCollider, il RigidBody2D deve essere di tipo Dynamic e con gravità a 0 altrimenti il player cadrebbe all’infinito.

### Implementazione della camera

Adesso bisogna andare sotto l’opzione Window 🡪 Package Manager e impostare come “Packages:” Unity Reistry. Una volta fatto questo sotto la sezione dei Packages bisognerà installare “Cinemachine”.

Finita l’installazione si deve aggiungere un nuovo oggetto alla scena, bisogna aggiungere una 2D Camera, per aggiungerla bisogna sempre fare tasto destro sulla scena poi Cinemachine e infine 2D Camera.

Dopo averla aggiunta per fissarla sul personaggio bisogna trascinare il game object Player all’interno del Follow nell’Inspector della camera 2D creata in precedenza.

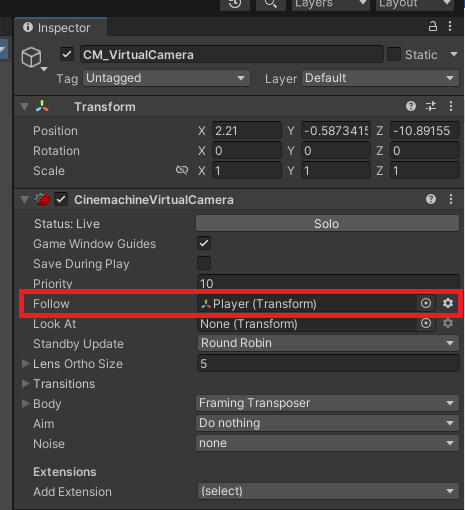


Figura 27 Collegare il player alla camera 2D

### Script PlayerMovement

Questo script serve per gestire il movimento del personaggio e per funzionare ha bisogno della posizione del mouse, della posizione della camera e del menu di pausa.

Inizialmente deve solamente memorizzare la camera:

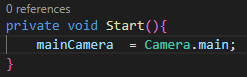


Figura 28 Memorizzazione della camera

Per far si che il player segua i movimenti del mouse servono 3 metodi:

* PrendiPosizioneMouse() 🡪 ritorna la posizione del mouse
* SeguiMouseDelay(float speed) 🡪 prende la velocità e applica il movimento al player
* PuntaMouse() 🡪 punta il player verso il mouse

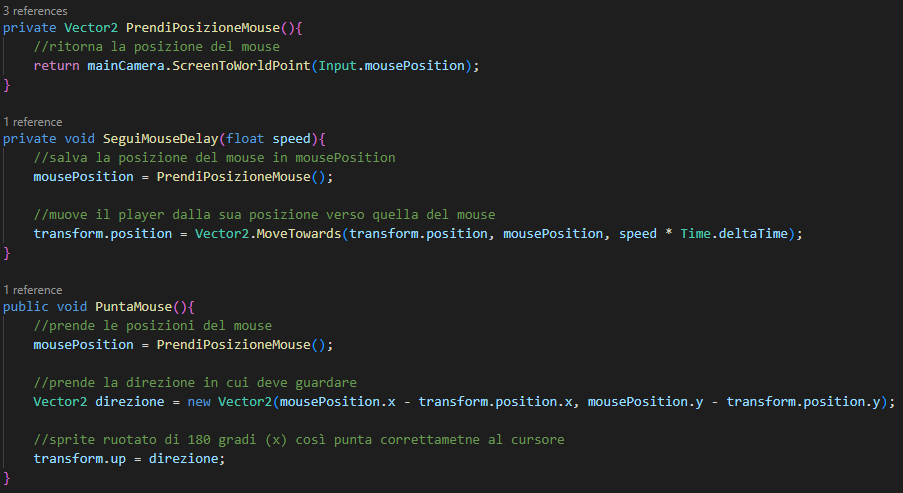


Figura 29 Metodi per il movimento del personaggio

Questi tre metodi permettono di muovere e ruotare il player in funzione del mouse, in particolare “transform.up = direzione” fa si che il punto più in alto dello sprite punta al cursore, per questo prima è stato necessario ruotare lo sprite del player di 180°.

Poi per gestire le collisioni si deve inserire un metodo OnCollisionEnter2D che controlli le collisioni del player. Se esso si scontra con lo yeti si distruggono entrambi i GameObject e viene ricaricato il menu principale.  
Inoltre al momento dello scontro lo sprite del player deve cambiare e viene quindi attivata l’animazione che lo fa passare dalla posizione di movimento a quella di scontro, infine inizia una nuova Coroutine che blocca il personaggio per 2 secondi e che lo resetta prima di ripartire.

Nel metodo Update() bisogna controllare che, dopo un salto, sia atterrato correttamente e, nel caso in cui fosse così, deve aumentare il punteggio altrimenti deve iniziare la Coroutine dello scontro e bloccare il personaggio per 2 secondi.

Poi nel caso in cui ci fosse il menu di pausa aperto il player non deve seguire e puntare il cursore ma deve bloccarsi completamente.

## Parte iniziale con ostacoli predefiniti

Per generare la parte iniziale del gioco in modo statico bisogna innanzitutto creare un nuovo GameObject Empty all’interno della scena principale.

Per fare ciò bisogna andare nella Hierarchy, premere tasto destro e cliccare su Create Empty:

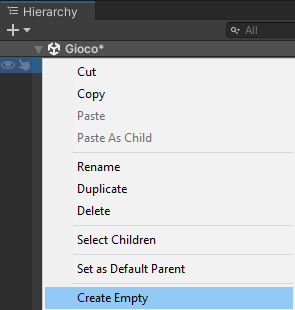


Figura 30 Create Empty GameObgject

Il GameObject appena creato si può chiamare come InizioGioco o InizioLivello.  
In seguito per aggiungerci dei prefabbricati in modo statico basterà trascinarli sopra al proprio game object. Per controllare se si sono aggiunti all’interno basta vedere se è stato rappresentato come un dropdown:



Figura 31 GameObject InizioGioco

In questo caso all’interno del game object sono presenti due alberi, una casa e un cartello.  
Nel workspace di Unity si mostra così:



Figura 32 InizioGioco visivo

Se si seleziona il game object padre, quindi InizioGioco, si dovrebbe vedere come nella foto: tutti gli oggetti contenuti all’interno di InizioGioco sono contornati di blu, sono tutti selezionati.

## Generazione Mappa

### Script LevelGenerator

Per generare la mappa si inizia creando un nuovo GameObject che si chiamerà “LevelGenerator”:

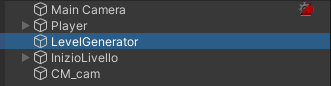


Figura 33 GameObject LevelGenerator

Questo GameObject permetterà la creazione della mappa generata in modo casuale. Dopo averlo creato, nella cartella Scripts, si deve creare un nuovo C# Script: LevelGenerator.cs.

All’interno di questo script si devono creare delle variabili iniziali che permetteranno di:

* Memorizzare gli ostacoli statici da generare
* Memorizzare gli ostacoli dinamici da generare
* Memorizzare lo yeti
* Memorizzare le posizioni della camera
* Definire delle costanti che aiuteranno nella generazione
* Memorizzare le posizioni, verticali, degli ostacoli già generati

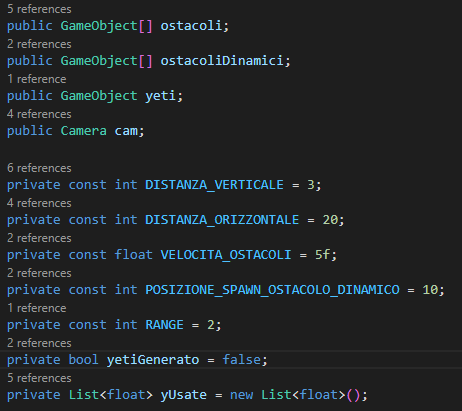


Figura 34 Variabili Iniziali per LevelGenerator

In questo script viene utilizzato solamente il metodo “Update()” quindi il metodo “Start()” può essere eliminato.

#### Generazione ostacoli statici

All’interno del metodo “Update()” per prima cosa bisogna memorizzare le posizioni della camera, x e y, e poi bisogna creare un numero random che sia compreso nella grandezza della lista per poter generare ogni volta un ostacolo differente:

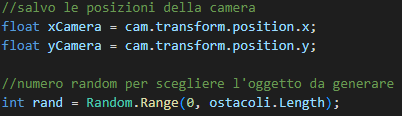


Figura 35 Posizioni della camera e numero random per gli ostacoli

A questo punto si può iniziare a generare la mappa.

Per generare la prima riga di ostacoli bisogna innanzitutto aggiungere alla lista delle yUsate la y di dove bisogna generare gli ostacoli:

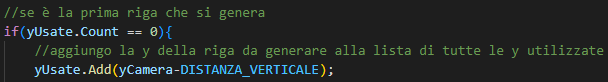


Figura 36 Inserire y nella lista

Come si vede dall’immagine la y da usare si calcola utilizzando la yCamera e la costante che contiene il valore di quanto distante si devono generare gli ostacoli. Facendo questo calcolo si trova una y minore (che si trova più in basso) di quella della camera e gli oggetti verranno generati fuori dal campo visivo di essa.

Quindi dopo aver salvato la posizione y per generare la riga di ostacoli si possono effettivamente generare gli alberi.

Per farlo bisogna fare un loop che generi al massimo 40 alberi (costante della distanza orizzontale \* 2):

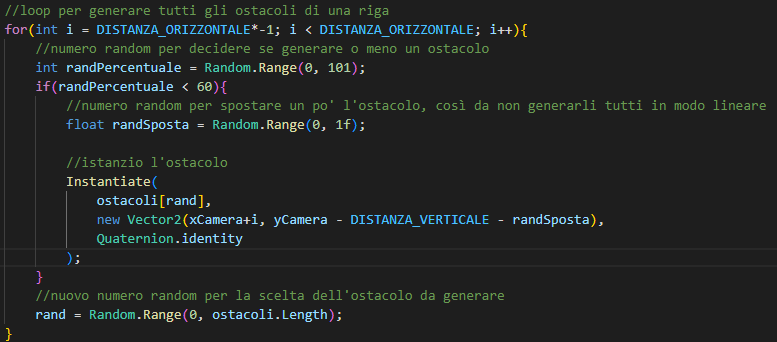


Figura 37 Loop per la generazione degli ostacoli

All’interno del loop, grazie alla variabile “randPercentuale”, si può decidere la percentuale di generazione degli ostacoli sulla stessa riga. In questo caso gli ostacoli verranno generati con una percentuale del 60%.

La variabile “randSposta” permette di spostare l’ostacolo più in alto o più in basso così da non avere una linea retta di ostacoli.

Questo funziona correttamente nel caso in cui si dovesse generare la prima riga di ostacoli.

Nel caso in cui avesse già creato i primi ostacoli bisogna controllare che la nuova y sia ad una distanza minima da quella precedente:

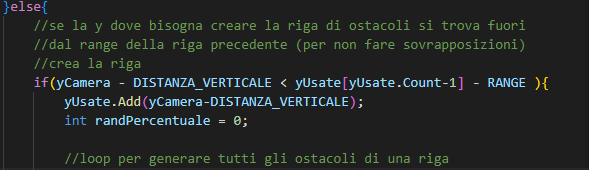


Figura 38 Controllo per la generazione degli ostacoli

Per fare questo controllo bisogna controllare che la nuova y che si vuole utilizzare, cioè il calcolo   
“yCamera-DISTANZA\_VERTICALE”, si trovi fuori dal range di spawn degli ostacoli precedenti. Quindi la nuova y deve trovarsi più in basso del range dell’ultima utilizzata.

Dopodiché basterà inserire lo stesso loop precedente per generare gli ostacoli e la generazione sarà completa.

#### Generazione Oggetti Dinamici

Dopo aver generato gli oggetti statici si può iniziare a generare anche quelli dinamici che si muovono da un lato della mappa all’altro.

Prima di tutto si deve creare un metodo che contenga il codice per creare e far muovere questi oggetti.

All’interno di questo metodo bisogna creare una nuova variabile che contenga la percentuale per decidere se generare o meno un ostacolo dinamico, dopo si può fare il solito controllo per decidere se generare l’ostacolo oppure no:

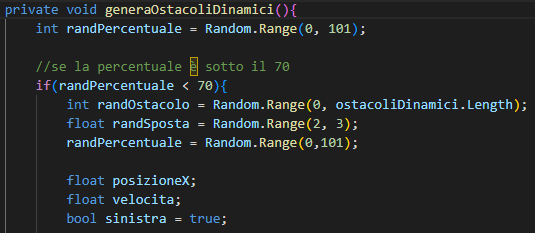


Figura 39 Inizio metodo generaOstacoliDinamici

Se la pecentuale è sotto al 70% verranno create nuove variabili:

* randOstacolo 🡪 valore casuale che rappresenta un indice dell’array degli ostacoli dinamici
* randSposta 🡪 valore float casuale tra 2 e 3 per aumentare la distanza di generazione dal player
* randPercentuale 🡪 viene generato un nuovo valore all’interno della variabile per poterla utilizzare in seguito per decidere se generare l’ostacolo a sinistra dello schermo oppure a destra
* posizioneX 🡪 variabile che conterrà la posizione di generazione dell’ostacolo
* velocita 🡪 conterrà la velocità dell’ostacolo
* sinistra 🡪 contiene un valore booleano per sapere se dover ruotare lo sprite dell’ostacolo oppure no, questo controllo è necessario visto che l’ostacolo potrebbe essere generato sia a sinistra sia a destra

Dopo aver creato queste variabili si deve controllare se impostare la posizione di generazione a sinistra o a destra del player, per fare questo basta controllare il valore di randPercentuale:

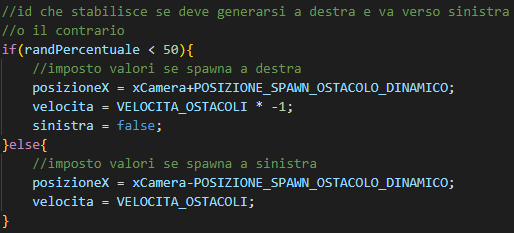


Figura 40 Controllo generazione a destra o sinistra

Se il valore è minore di 50 viene generata una posizione alla destra schermo altrimenti viene generata una posizione a sinistra.

Per calcolare il punto di generazione bisogna sommare o sottrarre la posizione xCamera e la costante per la distanza laterale dello spawn dell’ostacolo dinamico (questo valore può essere scelto in modo indifferente).

In seguito viene impostata la velocità con un valore scelto in base a questi criteri:

* in caso di generazione a destra la velocità deve essere negativa
* in caso di generazione a sinistra la velocità deve essere positiva

Infine viene impostata a false la variabile sinistra che permetterà di ruotare l’ostacolo nella direzione corretta.

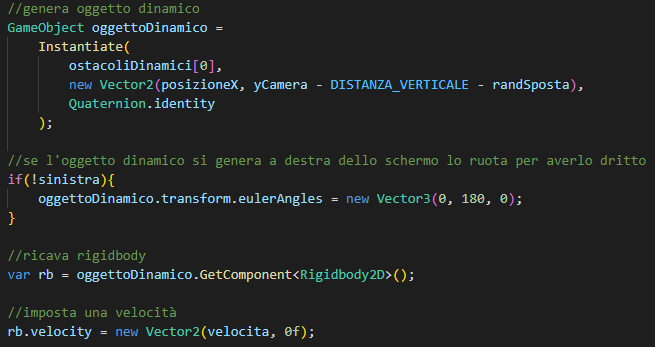


Figura 41 Creazione oggetto dinamico e asegnazione proprietà

A questo punto si può generare effettivamente l’oggetto.

All’interno della variabile oggettoDinamico vengono salvate tutte le proprietà e informazioni del nuovo oggetto istanziato.   
Viene istanziato un ostacolo casuale tra quelli presenti, alla posizione calcolata sull’asse delle x, e ad una posizione y più in basso rispetto al player.

Se lo sprite dell’ostacolo deve essere ruotato viene ruotato solamente sull’asse delle y di 180°.

Infine viene impostata una velocità al rigidbody, solamente sull’asse delle x, per permettere il movimento dell’ostacolo.

Come ultima cosa si deve aggiungere lo script al GameObject creato in precedenza, per farlo bisogna andare nel suo Inspector, bisogna premere su “Add Component”, bisogna cercare lo script LevelGenerator e bisogna aggiungerlo facendo semplicemente doppio click su di esso.

## Auto gestione degli ostacoli

### Ostacoli statici

Bisogna inizialmente creare un nuovo script chiamato GestoreOstacoli che dovrà essere applicato a tutti i prefabbricati degli ostacoli presenti.

Questo script contiene il GameObject LevelGenerator e il collider dell’ostacolo:

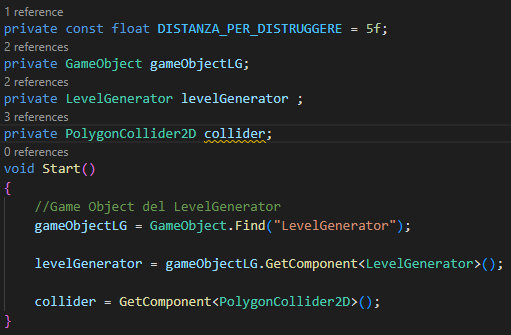


Figura 42 Variabili dello script GestoreOstacoli

In seguito, all’interno del metodo update, bisogna prima controllare se il personaggio ha superato l’ostacolo attuale e se è stato premuto il tasto sinistro del mouse. Questi controlli si possono fare con due semplici if:

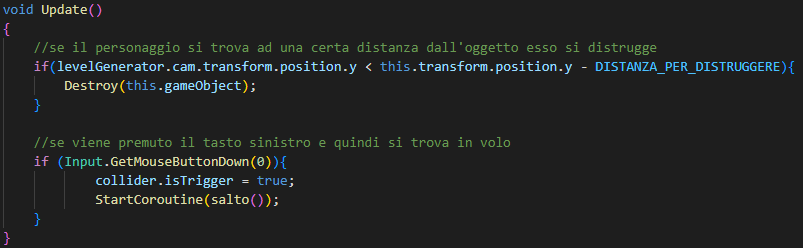


Figura 43 Controlli dello script GestoreOstacoli

Se il personaggio ha superato l’ostacolo di una certa distanza esso si distrugge e se è stato premuto il tasto sinistro del mouse il collider dell’oggetto non farà più collisione con il player per 2 secondi.

Questo tempo è dato grazie alla Coroutine salto():

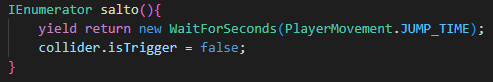


Figura 44 Coroutine salto() in GestoreOstacoli

### Ostacoli dinamici (lupo)

Per i lupi che corrono si deve creare un piccolo script apposito per loro che dovranno implementare inseme a GestoreOstacoli.

Questo script permette di far partire l’animazione di morte del lupo e di renderlo un ostacolo statico al momento di una collisione con qualsiasi cosa.

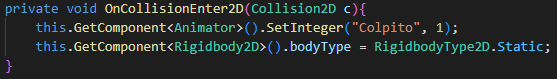


Figura 45 OnCollisionEnter2D() per i lupi

## Yeti

Per lo yeti si deve scaricare uno sprite animato dal sito di Unity[[2]](#footnote-2) e bisogna importarlo dentro al progetto.

Prima di tutto bisogna prendere lo sprite chiamato Cow2 (si trova sotto la cartella scaricata/Prefabs/) e bisogna rinominarlo in Yeti (per esempio) e poi bisogna mettergli un RigidBody2D Dynamic senza gravità:

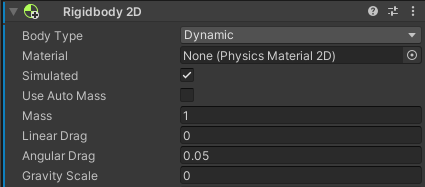


Figura 46 RigidBody2D dello Yeti

Un PolygonCollider2D che prenda bene tutto il modello e con isTrigger a false, un Animator e uno script Yeti.

### Script Yeti

All’interno dello script bisogna inizialmente trovare e salvare il GameObject del player all’interno di una variabile e bisogna stabilire la velocità di movimento:

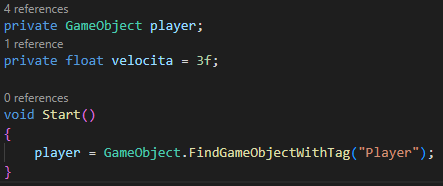


Figura 47 Variabili script Yeti

A questo punto bisogna fare in modo che lo yeti segua e punti sempre il player.

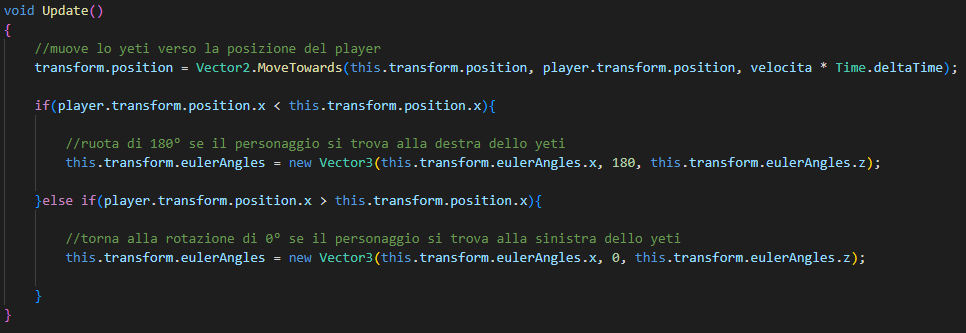


Figura 48 Update() dello script Yeti

Così sia che il player si trovi alla sua sinistra o alla sua destra lo yeti riuscirà sempre a puntarlo e a correre verso il giocatore.

Infine nel momento in cui lo yeti si scontra con un qualsiasi oggetto lo deve distruggere:

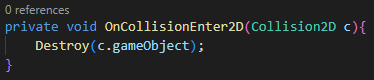


Figura 49 OnCollisionEnter2D() dello script Yeti

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-01 | **Nome:** | Controllare la risoluzione della finestra del gioco |
| **Descrizione:** | Controllare che il gioco si adatti sempre al full screen del monitor | | |
| **Prerequisiti:** |  | | |
| **Procedura:** | 1. Avviare il gioco 2. Cambiare risoluzione e controllare che si adatti sempre | | |
| **Risultati attesi:** | Il gioco si adatta ad ogni risoluzione in tempo reale | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-02 | **Nome:** | Movimenti del personaggio |
| **Descrizione:** | Controllare se i movimenti del personaggio funzionano correttamente | | |
| **Prerequisiti:** | Avvio del gioco | | |
| **Procedura:** | 1. Usare il mouse 2. Posizionare il mouse sotto il personaggio e verificare che vada in avanti 3. Spostare il mouse verso sinistra e verificare che il personaggio giri e che in posizione orizzontale si fermi 4. Spostare il mouse verso destra e verificare che il personaggio giri e che in posizione orizzontale si fermi | | |
| **Risultati attesi:** | Il personaggio si muove verso la direzione del mouse e curva in modo più definito rispetto al gioco originale | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-02 | **Nome:** | Movimenti del personaggio (in aria) |
| **Descrizione:** | Controllare se i movimenti del personaggio quando si trova in aria funzionano correttamente | | |
| **Prerequisiti:** | Aver verificato i movimenti base | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Cliccare una volta il tasto sinistro 3. Quando il personaggio si trova in volo tenere premuto il tasto destro | | |
| **Risultati attesi:** | Il personaggio salta e prova a fare un mortale, se ci riesce continua a sciare mentre se non ci riesce si schianta al suolo, si ferma e poi si rialza | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-03 | **Nome:** | Movimenti del personaggio (in aria) |
| **Descrizione:** | Controllare che il personaggio non possa cambiare traiettoria in volo | | |
| **Prerequisiti:** | Aver verificato i movimenti base | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Cliccare una volta il tasto sinistro 3. Mentre si trova in volo muovere il mouse a destra e a sinistra per far curvare il personaggio | | |
| **Risultati attesi:** | Il personaggio mentre si trova in volo non deve poter cambiare traiettoria | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-03 | **Nome:** | La mappa si genera correttamente |
| **Descrizione:** | Controllare se la generazione della mappa avviene prima del passaggio effettivo del personaggio | | |
| **Prerequisiti:** | Aver verificato i movimenti base | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Andare avanti e verificare che si generi una mappa con tutti gli ostacoli 3. Andare a sinistra e verificare che si generi una mappa con tutti gli ostacoli 4. Andare a destra e verificare che si generi una mappa con tutti gli ostacoli | | |
| **Risultati attesi:** | Man mano che il personaggio si sposta la mappa genera gli ostacoli automaticamente | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-03 | **Nome:** | Collisione con gli ostacoli |
| **Descrizione:** | Controllare se esiste una collisione con i vari ostacoli | | |
| **Prerequisiti:** | Aver verificato i movimenti base  Aver verificato la generazione della mappa | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Andare a sbattere contro ad ogni ostacolo e provare il trampolino | | |
| **Risultati attesi:** | Il personaggio deve fermarsi alla collisione con tutti gli ostacoli e deve saltare più in alto guadagnando velocità se salta su un trampolino. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-04 | **Nome:** | I dati vengono visualizzati a schermo |
| **Descrizione:** | Controllare se durante la partita i dati vengono visualizzati a schermo | | |
| **Prerequisiti:** | Aver verificato i movimenti base e quelli aerei  Aver verificato la generazione della mappa | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Saltare su un trampolino 3. Fare un’acrobazia | | |
| **Risultati attesi:** | Ogni volta che si fa un’acrobazia aumenta il punteggio di 10 punti | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-04 | **Nome:** | Controllare punteggio |
| **Descrizione:** | Controllare se quando si fa un’acrobazia vengono assegnati i punti in modo corretto | | |
| **Prerequisiti:** | Aver verificato i movimenti base e quelli aerei  Aver verificato che i dati vengano visualizzati a schermo | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Saltare su un trampolino 3. Fare un’acrobazia | | |
| **Risultati attesi:** | Ogni volta che viene effettuata un’acrobazia vengono aggiunti 10 punti | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-05 | **Nome:** | Controllare lo Jeti |
| **Descrizione:** | Controllare se dopo 2500m percorsi compare uno jeti | | |
| **Prerequisiti:** | Aver verificato i movimenti base e quelli aerei  Aver verificato la generazione della mappa  Aver verificato la visualizzazione dei dati della partita | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Percorrere 2500m | | |
| **Risultati attesi:** | Dopo i 2500m deve comparire uno jeti | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-009  REQ-05 | **Nome:** | Controllare lo jeti |
| **Descrizione:** | Controllare che lo jeti voglia uccidere lo sciatore | | |
| **Prerequisiti:** | Aver verificato i movimenti base e quelli aerei  Aver verificato la generazione della mappa  Aver verificato la visualizzazione dei dati della partita  Aver verificato che lo jeti compare dopo 2500m | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Percorrere 2500m 3. Muoversi | | |
| **Risultati attesi:** | Lo jeti prende lo sciatore e lo mangia | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-010  REQ-05 | **Nome:** | Fine della partita |
| **Descrizione:** | Controllare che quando lo sciatore viene mangiato la partita finisce e si torna in cima | | |
| **Prerequisiti:** | Aver verificato i movimenti base e quelli aerei  Aver verificato la generazione della mappa  Aver verificato la visualizzazione dei dati della partita  Aver verificato che lo jeti compare dopo 2500m  Aver verificato che lo jeti mangia lo sciatore | | |
| **Procedura:** | 1. Posizionare il mouse sotto il personaggio 2. Percorrere 2500m 3. Farsi mangiare dallo jeti | | |
| **Risultati attesi:** | La partita deve finire e il personaggio torna in cima alla montagna | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 1.7) (ad esempio Gantt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc.

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc.

# Glossario

Inserite una semplice tabella con due colonne che spieghi i termini specifici del progetto (lista dei termini in ordine alfabetico A-Z)

Esempio:

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| AJAX | **Asynchronous JavaScript And XML**: una tecnica che permette di eseguire richieste ed ottenere dati da una pagina web in modo asincrono. |
| CSS | **Cascading Style Sheets**: linguaggio che permette di definire il layout e la grafica di una pagina web. |

# Bibliografia

## Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo dell’articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell’articolo.

## Bibliografia per libri

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell’editore,
5. Anno di pubblicazione,
6. ISBN.

## Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

**Esempio:**

* http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente/documentazione macchine virtuali
* Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
* Documentazione di prodotti di terzi
* Eventuali guide utente / Manuali di utilizzo
* Mandato e/o QdC
* Prodotto
* …

1. <https://assetstore.unity.com/> [↑](#footnote-ref-1)
2. <https://assetstore.unity.com/packages/2d/characters/enemycow-227480> [↑](#footnote-ref-2)