POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Elettronica, Informazione e Bioingegneria

Sviluppo di un generatore automatico di endless runner personalizzabili

Relatore: **Prof. Pier Luca Lanzi**

Correlatore: **Ing. Luca Bianchi, PhD**

Tesi di Laurea Magistrale:

Christian Miranti

matricola 787042

Anno Accademico 2014/2015

"Che tu creda di farcela o di non farcela, avrai comunque ragione!"
Henry Ford

Sommario

Negli ultimi anni la *brand communication* ha avuto un ruolo preponderante nel mondo del marketing, e consiste nell'insieme di attività che influenzano le opinioni dei clienti nei confronti del brand e dei suoi prodotti. Una delle tecniche più recenti di *brand communication* consiste nello sviluppo di videogiochi con tematiche inerenti al brand o ai suoi prodotti. Gli *endless runner* sono tra i generi di videogiochi maggiormente utilizzati nel campo della *brand communication*. Gli *endless runner* sono un genere di videogiochi che ha riscosso molto successo negli ultimi anni nel mercato delle applicazioni per dispositivi mobili in cui il giocatore comanda la corsa senza sosta di un personaggio su un percorso generato proceduralmente fino all'inevitabile collisione con un ostacolo. Canabalt, Robot Unicorn Attack, Temple Run, Jetpack Joyride, Subway Surfers e Flappy Bird sono gli *endless runner* di maggior successo in termini di numero di utenti e guadagni e sono il punto di riferimento per coloro che sviluppano videogiochi di questo genere. Analizzando le caratteristiche di questi titoli di successo, in questo lavoro di tesi é stato sviluppato uno strumento per generare giochi *endless runner* in maniera automatica e personalizzabile a partire da elementi multimediali, come effetti audio o modelli 3D, di un brand. In particolare, presento le analisi effettuate sugli *endless runner* di maggior successo presenti sul mercato, seguite dalla progettazione e dallo sviluppo del generatore automatico oggetto di questa tesi. Infine, presento un caso pratico in cui il generatore è stato utilizzato per creare un *endless runner* personalizzato senza aver scritto alcuna riga di codice.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il mondo delle app per dispositivi mobili è suddiviso in categorie. Di queste, la categoria col maggior numero di app attive sui dispositivi Apple è quella dei videogiochi [1]. Uno dei generi più popolari di questi videogiochi è quello denominato *endless runner*.

Gli *endless runner* (o *endless platformer*) sono un genere di videogiochi in cui il giocatore controlla un personaggio impegnato in una corsa senza sosta, tentando di percorrere la maggior distanza possibile prima dell'inevitabile collisione con un ostacolo. Le azioni possibili per il personaggio, generalmente, sono limitate a spostamenti sull'asse verticale e, in alcuni casi, sull'asse orizzontale.

Il numero limitato di azioni a disposizione del personaggio corrisponde ad un limitato numero di comandi che il giocatore può utilizzare per comandarlo. Questa caratteristica degli *endless runner* li rende versatili per i *touchscreen* di smartphone e tablet.

Canabalt è il primo titolo di questa categoria di videogiochi ad essere stato pubblicato su uno store per dispositivi mobili [2]. Essendo grafica e meccaniche molto semplici, sono state necessarie poche settimane di lavoro di un unico programmatore per realizzarlo. La popolarità di questo titolo e la sua semplicità nello svilupparlo, ha fatto si che in molti ne emulassero ed estendessero le meccaniche di gioco pubblicando i propri titoli.

L'interesse crescente per gli *endless runner* ne ha inoltre mostrato le potenzialità dal punto di vista della *brand communication* [3]. Con

brand communication si intende l'insieme di attività che influenzano le opinioni dei clienti nei confronti del *brand* e dei suoi prodotti. Sempre più spesso, ad esempio, l'uscita nelle sale cinematografiche di un nuovo film di avventura coincide con la pubblicazione di un *endless runner* basato sul film stesso [4]. Anche altri settori, come televisione, sport e bevande [5], hanno utilizzato questa combinazione tra *endless runner* e *brand communication*.

L'esigenza di offrire a *brand* che operano in settori diversi la possibilità di creare il proprio *endless runner*, ha spinto Neosperience, azienda italina che sviluppa software per la *digital costumer experience*, a proporre lo sviluppo di un generatore automatico di *endless runner* personalizzabili, oggetto di questa tesi.

1.1 Contesto del lavoro

Lo sviluppo di un videogioco o di un generatore di videogiochi si basa su un processo di progettazione dei suoi elementi formali. Gli elementi formali formano la struttura di un videogioco e le relazioni che intercorrono tra loro costituiscono il videogioco stesso [6].

Il giocatore è l'elemento formale che identifica coloro che volontariamente accettano regole e restrizioni del videogioco per giocarci. La combinazione tra numero e ruolo dei giocatori definisce le seguenti tipologie:

- *Single Player vs Game*: un unico giocatore che compete contro il gioco stesso.
- *Multiple Individual Players vs Game*: più giocatori competono contro il gioco.
- *Player vs Player*: due giocatori si affrontano l'uno contro l'altro.
- *Unilateral Competition*: due o più giocatori competono contro un singolo giocatore.
- *Multilateral Competition*: tre o più giocatori competono tra di loro.
- *Cooperative Competition*: due o più giocatori cooperano contro il sistema.

- *Team competition*: due o più gruppi di giocatori si affrontano tra di loro.

L'obiettivo è l'elemento formale che definisce ciò che i giocatori cercano di realizzare all'interno del videogioco. In un gioco possono essere presenti obiettivi, sotto-obiettivi e obiettivi secondari e tutti devono avere la caratteristicha di essere impegnativi ma al contempo raggiungibili. Le tipologie di obiettivi che i giocatori possono affrontare in un videogioco sono i seguenti:

- *Capture*: evitare di essere catturati o uccisi mentre si distruggono alcune proprietà dell'avversario.

- *Chase*: eludere o catturare un avversario.

- *Race*: raggiungere un obiettivo prima degli avversari.

- *Alignment*: allineare degli elementi all'interno di uno spazio o configurazione concettuale.

- *Rescue*: recuperare delle unità o oggetti e portarli al sicuro.

- *Forbidden Act*: spingere gli avversari ad abbandonare una strategia o rompere le regole.

- *Construction*: costruire, mantenere o gestire oggetti di gioco.

- *Exploration*: esplorare aree sconosciute del gioco.

- *Solution*: risolvere un problema o puzzle.

- *Outwit*: utilizzare nuove conoscenze per superare in astuzia i propri avversari.

L'insieme delle procedure di gioco è quell'elemento formale che definisce le azioni che il giocatore può compiere nel videogioco. Le procedure si riferiscono anche all'insieme di controller del gioco, utilizzati per processare l'input del giocatore.

L'insieme delle regole è quell'elemento formale che identifica tutto ciò che il giocatore può o non può fare nel videogioco. Lo scopo delle

regole è quello di definere oggetti e condizioni, limitare le azioni dei giocatori e determinare gli effetti di tali azioni.

Le risorse sono quell'elemento formale che identifica gli oggetti di gioco che hanno valore per i giocatori per raggiungere i propri obiettivi. Il valore di questi oggetti può essere determinato dalla loro scarsità e dalla loro utilità. Alcuni esempi di risorse sono:

- Vite.
- Unità.
- Salute.
- Valuta.
- Inventario.
- Tempo

Il conflitto è un elemento formale che tenta di impedire ai giocatori di raggiungere i propri obiettivi. Un conflitto emerge quando gli obiettivi guidano i giocatori verso regole e procedure che lavorano contro l'obiettivo del giocatore. Nei videogiochi i tre conflitti più comuni sono:

- Ostacoli: possono essere di natura fisica oppure mentale.
- Avversari: altri giocatori o nemici controllati dal sistema.
- Dilemmi: decisioni strategiche le cui possibile conseguenze devono essere valutate prima di procedere.

I confini sono quell'elemento formale che definisce la seprazione tra il videogioco e la realtà. I confini marcano la differenza tra azioni intraprese nel videogioco e le stesse azioni nel mondo reale, le cui conseguenze sarebbero ben diverse.

L'ultimo elemento formale è l'esito del gioco. L'esito deve essere incerto, misurabile e non equo. Le condizioni di vittoria sono diverse rispetto agli obiettivi del giocatore.

La fase di progettazione e sviluppo del generatore di *endless runner*, oggetto di questa tesi, viene preceduta dall'analisi dei principali elementi formali degli *endless runner* di maggior successo.

1.2 Breve descrizione del lavoro

Neosperience è un'azienda italiana che sviluppa piattaforme software innovative nel campo della *costumer experience*. I prodotti sviluppati da Neosperience hanno lo scopo di incrementare la soddisfazione, acquisizione e mantenimento dei clienti e delle community dei *brand*. Con il generatore di *endless runner* sviluppato, Neosperience intende fornire un servizio di *brand communication* basato sullo sviluppo di *endless runner* personalizzati secondo esigenze e caratteristiche dei *brand*.

Neosperience ha già sviluppato in passato il prototipo di un *endless runner*, chiamato Ecnalubma. L'esperienza di Neosperience nella creazione di Ecnalubma ha ispirato l'idea di sviluppare un generatore automatico di *endless runner*, in modo tale da poter fornire videogiochi personalizzati di questo genere in tempi contenuti.

Il progetto nasce da un'iniziale analisi del codice e delle caratteristiche di Ecnalubma. Tale analisi è stata estesa anche a quegli *endless runner* notoriamente utilizzati come base per lo sviluppo di molti titoli di questo genere di videogiochi.

A partire dai risultati di questa analisi è stato possibile progettare il generatore di *endless runner*. In questa fase sono state definite le caratteristiche di ogni elemento della struttura finale di un *endless runner* e la tipologia di interazione. La progettazione è stata estesa anche all'organizzazione dei file che compongono il generatore. L'organizzazione dei file si è resa necessaria per rendere semplice la fase di personalizzazione dell'*endless runner* da parte del *videogame designer* del videogioco.

La fase di sviluppo del progetto è stata realizzata con l'ausilio dell'engine Unity 3D, uno dei software di sviluppo di videogiochi più popolari. Per semplificare la funzionalità di personalizzazione fornita dal generatore di *endless runner*, si è scelto di creare file di configurazione in formato JSON.

Infine, è stato realizzato un prototipo di *endless runner* utilizzando il generatore sviluppato per mostrarne il workflow e le potenzialità.

1.3 Struttura della tesi

La struttura della tesi segue l'ordine con cui sono state svolte le attività necessarie per la creazione del generatore di *endless runner*.

Nel Capitolo 2 vengono citati e analizzati gli *endless runner* di maggior successo. In questo capitolo vengono discussi inoltre i generatori automatici di videogiochi. In particolare, vengono considerati i generatori di Flappy Bird, uno degli *endless runner* di maggior successo.

Nel Capitolo 3 vengono mostrati i risultati ottenuti dalle analisi degli *endless runner* mostrando quali sono le caratteristiche dei videogiochi che ricadono in questo genere.

La fase di progettazione e sviluppo di tale generatore di *endless runner*, viene descritta nel Capitolo 4.

Nel Capitolo 5 viene mostrato un caso pratico di creazione di un *endless runner* utilizzando il generatore realizzato.

Infine, il Capitolo 6 è dedicato alle conclusioni riguardanti il generatore sviluppato e ai possibili sviluppi che questo può avere in futuro.

Capitolo 2

Endless Runner

Un *platform game* (o *platformer*) è un genere di videogioco in cui il giocatore guida un avatar facendogli saltare piattaforme o ostacoli per poter avanzare nel gioco. I *platformer* sono stati il genere più popolare di videogiochi tra gli anni '80 e '90. La loro popolarità è, però, drasticamente calata negli anni a seguire [7].

I *platformer* hanno riscosso nuovamente successo negli ultimi anni grazie all'introduzione di un loro sottogenere chiamato *endless runner*. Con *endless runner* (o *endless platformer*) si intende un genere di videogiochi in cui il giocatore controlla un personaggio impegnato in una corsa senza sosta, tentando di fargli percorrere la maggior distanza possibile prima dell'inevitabile collisione con un ostacolo. Il numero limitato di azioni consentite negli *endless runner* li rende versatili per essere giocati su smartphone e tablet, che sono sprovvisti di tastiere o controller esterni, dove infatti hanno riscosso molto successo.

Il genere degli *endless runner* nasce con grafica 2D a scorrimento orizzontale, evolutasi negli anni anche in 3D. La grafica 3D ha permesso l'introduzione di nuove meccaniche di gioco e dello scorrimento verticale. Negli *endless runner* a scorrimento verticale la visuale è in terza persona, tranne in particolari titoli come Fotonica in cui la visuale è in prima persona [8].

Gli *endless runner* sono di fatto infiniti e utilizzano la *procedural content generation* per generare ambienti ed elementi del percorso fino al termine della partita. La progressione del giocatore viene misurata

in termini di distanza percorsa prima della collisione del personaggio con un ostacolo. Il percorso è composto da ostacoli, valute di gioco e altri elementi che si presentano in pattern.

2.1 Storia degli endless runner

Il primo titolo classificato come *endless runner* risale agli anni '80 ed è B.C.'s Quest for Tires. Il termine *endless runner* è stato però coniato 30 anni dopo, grazie al successo riscosso dai primi due titoli per dispositivi mobili di questo genere: Canabalt e Robot Unicorn Attack [2].

Le loro meccaniche e grafiche sono state riprese ed estese in molti titoli a seguire, incrementando il successo del genere. Tra questi, i titoli più significativi sono Temple Run, Jetpack Joyride, Subway Surfers e Flappy Bird.

2.1.1 B.C.'s Quest for Tires

B.C.'s Quest for Tires è un videogioco per Commodore 64 pubblicato nel 1983. In questo *endless runner*, il giocatore guida la corsa di un uomo delle caverne a bordo di un monociclo di pietra nel tentativo di salvare la propria fidanza rapita.

B.C.'s Quest for Tires è un *endless runner* a scorrimento orizzontale con grafica 2D (Figura 2.1). I comandi a disposizione del giocatore permettono all'uomo delle caverne soltanto di abbassarsi e saltare. A tentare di fermare la corsa del personaggio vi sono ostacoli, la cui collisione con l'uomo delle caverne provoca la perdita di una delle vite a disposizione.

Lo sviluppo di B.C.'s Quest for Tires ha richiesto appena 8 settimane di lavoro. Dato il periodo della sua pubblicazione, le 50 mila copie vendute di questo *endless runner* sono un notevole successo. Nonostante le cifre raggiunte, non vi sono stati titoli che ne emulassero le meccaniche nei 25 anni successivi.

Figura 2.1: Schermata di gioco di B.C.'s Quest for Tires.

2.1.2 Canabalt

Canabalt è un *endless runner* pubblicato inizialmente per browser e successivamente per dispositivi mobili nel 2009. In questo *endless runner*, il giocatore comanda un avatar attraverso i tetti di una città attaccata da un nemico sconosciuto. Questo *endless runner* viene considerato il primo videogioco ad aver dato origine alla popolarità di questo genere [2][9].

Come B.C.'s Quest for Tires, Canabalt è un *endless runner* a scorrimento orizzontale con grafica 2D (Figura 2.2) che rientra anche nella categoria *one-button game* in quanto l'unica azione a disposizione del giocatore è il salto, attuabile appunto con un unico pulsante. La collisione con ostacoli di piccole dimensioni, come sedie e casse, rallentano temporaneamente la corsa del personaggio, mentre colpire una bomba o non saltare in tempo da un palazzo a all'altro, provoca la fine della partita.

Canabalt è stato sviluppato in appena 7 giorni da un unico programmatore in occasione di una *game jam* chiamata Experimental Game Project [10]. Questo *endless runner* è stato quotato come uno dei migliori giochi pubblicati nel 2009 da diversi siti web del settore, tra cui Rock, Paper, Shotgun [11] ed Eurogamer [12].

Figura 2.2: Schermata di gioco di Canabalt.

2.1.3 Robot Unicorn Attack

Robot Unicorn Attack è stato il primo *endless runner* ad aver emulato le meccaniche di Canabalt [2]. Anche questo titolo è stato pubblicato inizialmente per browser e successivamente per dispositivi mobili. Il giocatore controlla la corsa di un unicorno robotico tra piattaforme sospese, fate e cristalli di stelle. Questi elementi e la grafica cartoonesca hanno avuto come scopo quello di attirare un maggior numero di utenti femminili.

Come Canabalt, questo *endless runner* è a scorrimento orizzontale con grafica a due dimensioni. I comandi a disposizione sono due: il salto e il *dash*, ovvero una spinta che accellera temporaneamente la corsa dell'unicorno e gli permette di distruggere particoli ostacoli. In Robot Unicorn Attack il punteggio finale di un giocatore è dato dalla somma dei punteggi di tre partite consecutive. Il punteggio di ogni partita è dato dalla somma tra distanza percorsa e punti extra ottenuti durante la corsa. Un modo per ottenuere punti extra è quello di raccogliere il maggior numero possibile di elementi chiamati fate. Ogni fata raccolta aggiunge 10 unità al punteggio finale e tale valore aumenta con l'aumentare del numero di fate raccolte consecutivamente. L'altro modo per ottenere punteggi extra è quello di distruggere i cristalli di stelle. Un cristallo di stelle è un ostacolo che è possibile distruggere solo quando il personaggio è in modalità *dash*. Quando un cristallo viene

distrutto, 300 punti vengono aggiunti al punteggio totale della partita.

La popolarità di Robot Unicorn Attack ha portato i suoi sviluppatori a creare diverse versioni dello stesso *endless runner* mantenendone le meccaniche di base e variandone gli asset. Robot Unicorn Attack: Heavy Metal varia dalla versione originale dal punto di vista grafico e di musiche di sottofondo, ispirate appunto al genere musicale heavy metal. Questa versione ha avuto un successo maggiore rispetto a quello riscosso dall'originale [13]. Robot Unicorn Attack: Christmas Edition è invece una versione di Robot Unicorn Attack ambientata in un mondo natalizio. Robot Unicorn Attack: Evolution riprende le meccaniche del gioco originale con l'aggiunta di un evoluzione del protagonista e dell'ambiente dopo aver colpito tre cristalli di stelle consecutivi (Figura 2.3). Retro Unicorn Attack è una versione di Robot Unicorn Attack ispirata ai giochi 8-bit.

Oltre a questi titoli è stato creato un vero e proprio sequel del gioco, chiamato Robot Unicorn Attack 2. Le meccaniche di base rimangono le stesse della prima versione ma con l'aggiunta di quattro caratteristiche: la personalizzazione, le valute virtuali, il *revive* e le sfide. La personalizzazione permette al giocatore di personalizzare il proprio personaggio con accessori oppure di utilizzare un personaggio diverso. In questa versione sono state aggiunte due valute virtuali accumulabili e spendibili all'interno del gioco. La prima valuta, chiamata *teardrop*, è possibile trovarla ad intervalli regolari in sequenze predefinite durante le partite. Questa valuta è possible utilizzarla per acquistare le personalizzazioni del proprio personaggio. La seconda valuta virtuale, chiamata gemma, è possibile trovarla durante una partita ed appare solo di rado. Questa valuta può essere spesa dopo aver colpito un ostacolo per poter riprendere la partita appena interrotta come se l'ostacolo non fosse mai stato colpito. Anche questa è una nuova caratteristica e viene chiamata *revive*, ovvero far tornare in vita il proprio personaggio. L'ultima caratteristica introdotta viene chiamata sfida. Durante ogni partita, con determinate azioni, il giocatore può superare due o più sfide che gli vengono proposte. Una sfida può riguardare il raggiungimento di una distanza specifica durante una partita o il raccogliere un numero minimo di *teardrop*. Il superamento delle sfide viene seguito da un premio

in *teardrop* e dalla loro sostituzione con nuove sfide da superare.

Figura 2.3: Confronto delle schermate di Robot Unicorn Evolution.

2.1.4 Temple Run

Temple Run è stato il primo *endless runner* a scorrimento verticale con grafica 3D e ad essere stato realizzato appositamente per dispostivi mobili. In questo videogioco, il giocatore controlla la corsa di un esploratore intento a scappare da un gruppo di scimmie demoniache custodi della reliquia che ha rubato. Superando il miliardo di download, Temple Run, pubblicato nel 2011, è tutt'ora l'*endless runner* di maggior successo.

I comandi a disposizione in Temple Run permettono di eseguire un numero maggiore di azioni al personaggio rispetto ai titoli citati precedentemente. Oltre a saltare ed abbassarsi, il protagonista può spostarsi lateralmente e cambiare direzione della corsa, girando a destra o a sinistra. Per poter compiere queste azioni, i comandi a disposizione si basano sulle *gesture swipe* e *tilt*. La direzione dello *swipe* definisce l'azione da compiere tra saltare, abbassarsi e cambiare direzione, mentre l'inclinazione del dispositivo quando si effettua il *tilt* definisce il tipo di spostamento. Come in B.C.'s Quest for Tires, vi è una varietà di ostacoli diversi, di cui alcuni superabili solo saltando e altri solo abbassandosi. Grazie alla grafica 3D e allo scorrimento verticale, in Temple Run viene aggiunto anche come ostacolo l'incrocio, ovvero la fine del percorso rettilineo con obbligo di girare a destra o a sinistra per proseguire la fuga. Come in Robot Unicorn Attack, in Temple Run il punteggio finale è dato da una combinazione tra percorso effettuato e punteggi extra,

chiamati moltiplicatori. Inoltre, in questo titolo vi è una valuta virtuale, ottenibile durante le partite, spendibile all'interno del videogioco. Come in Robot Unicorn Attack 2, vi è la possibilità di personalizzare il proprio personaggio. In aggiunta, sono presenti degli elementi di aiuto per il giocatore chiamati powerup. I powerup facilitano il giocatore a raccogliere valute virtuali, a percorrere distanze maggiori e simili. Ad ogni powerup è associato un livello che ne determina la sua durata durante la partita dopo che il personaggio ne è entrato in contatto. Per poter alzare il livello del powerup, ovvero potenziarlo, il giocatore deve pagare una certa quantità di valuta virtuale. All'aumentare del livello aumenta la quantità di valuta richiesta per passare al livello successivo. Vi sono anche degli oggetti chiamati *utility* ad aiutare il giocatore. Questi, a differenza dei powerup, possono essere attivati in qualsiasi momento dopo essere stati acquistati con la valuta virtuale. Infine, Temple Run introduce il concetto di obiettivo. Questo consiste in una sfida che può essere superata in qualsiasi momento e un'unica volta. Una volta superata, l'obiettivo viene evidenziato come tale nella lista degli obiettivi. Superare un obiettivo non conferisce alcun premio in termini di valuta virtuale.

Il successo di Temple Run ha spinto Dinsey a rivolgersi ai suoi creatori per creare *endless runner* basati su due suoi film, Oz il grande e magnifico e Brave - Ribelle. Alla loro uscita nelle sale sono quindi stati rilasciati due spinoff di Temple Run ambientati nei mondi di questi film: Temple Run: Oz e Temple Run: Brave (Figura 2.4).

Dopo il successo di questi titoli, i creatori di Temple Run hanno prodotto un seguito, chimato Temple Run 2. Le meccaniche di base sono rimaste le stesse ma le modifiche apportate sono simili a quelle apportate a Robot Unicorn Attack per la creazione del suo sequel. Gli *utility* sono stati eliminati. Oltre alla valuta virtuale principale, utilizzata per tutti i tipi di acquisti, ne è stata aggiunta una seconda. Questa valuta virtuale, chiamata gemma, è rara da trovare e viene utilizzata per effettuare il *revive* a fine partita. Sono state, inoltre, aggiunte le sfide da superare durante le partite. Come in Robot Unicorn Attack 2, il superamento delle sfide corrisponde ad un innalzamento di livello del personaggio, che viene premiato con valuta virtuale principale. Quando

una sfida viene superata, questa viene sostituita da una nuova sfida.

Come Temple Run, questo sequel ha avuto un grande successo, raggiungendo già nei primi quattro giorni dalla sua pubblicazione i 20 milioni di download e superando in pochi mesi il miliardo [14].

Figura 2.4: Confronto tra Temple Run, Temple Run: Brave e Temple Run: Oz.

2.1.5 Jetpack Joyride

Jetpack Joyride è un *endless runner* a scorrimento orizzontale con grafica 2D pubblicato per dispositivi mobili nel 2011 in cui il giocatore controlla la corsa di un venditore di grammofoni in fuga dopo aver rubato un jetpack da un laboratorio. Jetpack Joyride ha ricevuto prevalentemente recensioni positive e già nel 2013 aveva raggiunto il milione di download. Gli sono stati riconosciuti inoltre molteplici premi.

Jetpack Joyride è un *one-button game* basato sul volo. Premendo sul *touchscreen*, il giocatore attiva il jetpack indossato dal protagonista, che gli permetti di volare. Rilasciando la pressione, il jetpack si disattiva, iniziando la fase di atterraggio. Come in Temple Run, vi è un'unica valuta virtuale, utilizzabile per personalizzare il proprio personaggio, acquistare *utility* e powerup e personalizzare quelli che vengono chiamati veicoli. I veicoli sono una categoria particolare di powerup che, quando vengono attivati in gioco, questi si sostiuiscono al protagonista

e rimangono attivi fino alla collisione con un ostacolo. Tale collisione non provoca la fine della partita ma solo la distruzione del veicolo e il ritorno del protagonista. Nel momento in cui un veicolo viene attivato, l'effetto del *tap* sullo schermo varia rispetto a quello originale, potendo diventare un salto al posto del volo, un atterraggio, un cambio di gravità e altri ancora. Gli altri powerup, invece, vengono decisi prima dell'inizio di una partita e ve ne possono essere attivi solo due contemporaneamente. Come in Temple Run 2 e Robot Unicorn Attack 2, sono presenti le sfide da superare ad ogni partita, il cui superamento fa progredire di livello il giocatore, premiandolo con valuta virtuale. Il concetto, invece, di obiettivi viene esteso in quello di badge. Un badge è una rappresentazione visiva e simbolica del raggiungimento di un particolare obiettivo.

2.1.6 Subway Surfers

Subway Surfers è un *endless runner* a scorrimento verticale con grafica 3D, pubblicato per dispositivi mobili nel 2013, in cui il giocatore veste i panni di un hooligan in fuga da un sorvegliante dopo aver riempito di graffiti un vagone di un treno. Subway Surfers è uno degli *endless runner* di maggior successo, grazie alla sua media giornaliera di 26.5 milioni di utenti attivi [15].

In Subway Surfers i comandi a disposizione sono gli stessi offerti da Temple Run tranne che per gli spostamenti laterali. Infatti, in Subway Surfers non sono presenti incroci a cui dover girare. Il personaggio può occupare tre specifiche posizioni sull'asse orizzontale dell'infinita strada rettilinea che percorre, chiamate *lane*. Per passare da una *lane* all'altra, il giocatore deve utilizzare le *gesture swipe* a destra e *swipe* a sinistra. Come in Temple Run 2 e Robot Unicorn Attack 2, anche in questo *endless runner* è presente il concetto delle due valute virtuali. In questo caso, anche la valuta virtuale più rara può essere utilizzata per effettuare acquisti, ma solo all'interno della sezione personalizzazioni. Anche in questo *endless runner* è presente la possibilità di utilizzare powerup potenziabili grazie ad avanzamenti di livello. Come in Jetpack Joyride, sono disponibili contemporaneamente tre possibili sfide da superare durante le partite e particolari obiettivi vengono premiati con

dei badge. In particolare, le sfide di Subway Surfers vengono sostituite solo quando tutte e tre sono state superate con successo.

La caratteristica di Subway Surfers di limitare il movimento orizzontale del personaggio a tre posizioni specifiche è tra le più emulate nell'ambito degli *endless runner* a scorrimento verticale con grafica 3D.

2.1.7 Flappy Bird

Flappy Bird è un *endless runner* a scorrimento orizzontale con grafica 2D, pubblicato nel 2013 per dispositivi mobili. In Flappy Bird, il giocatore controlla il battito d'ali di un uccellino intento a volare tra un ostacolo a forma di tubo e l'altro (Figura 2.5).

Flappy Bird, a differenza di quanto visto nei titoli precedenti, al posto di estendere le meccaniche degli altri videogiochi dello stesso genere, le semplifica. Questo *endless runner* è un *one-button game* in quanto l'unico comando a disposizione del giocatore è quello che permette all'uccellino di sbattere le ali. La prograssione del protagonista viene calcolata in numero di coppie di tubi superate con successo. Ciò che il giocatore può fare si limita soltanto all'iniziare una partita, comandare il protagonista fino al contatto con un ostacolo e ricominciarne una nuova.

Flappy Bird è stato sviluppato da un unico programmatore in appena tre giorni, il quale ha affermato di aver guadagnato mediamente 50.000$ al giorno nel corso del primo mese dalla sua pubblicazione.

2.2 Generatori automatici di Flappy Bird

Gli *endless runner* citati nel capito precedente, sono stati di ispirazione per la creazione di videogiochi dello stesso genere. Tra questi, Flappy Bird è stato quello più emulato. Grazie al ridotto numero di meccaniche e caratteristiche, questo *endless runner* può essere sviluppato in breve tempo. Dopo il ritiro di Flappy Bird dagli store per dispositivi mobili, il numero di videogiochi che lo hanno emulato è cresciuto. Per alcune settimane, si è registrata una media di 60 emulatori di Flappy Bird pubblicati ogni giorno sugli store per dispositivi mobili [16].

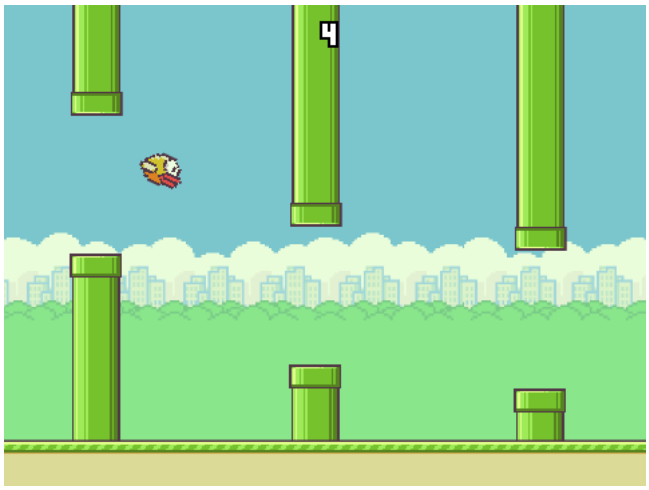

Figura 2.5: Schermata di gioco di Flappy Bird.

Il numero crescente di emulatori di Flappy Bird è stato possibile grazie alla creazione di generatori automatici di questo gioco. Acquistando uno di questi generatori, chiunque è in grado creare una propria versione personalizzata di questo *endless runner* in poche ore.

Un altro generatore automatico di Flappy Bird è stato sviluppato dai creatori di Code.org, sito web dedicato all'insegnamento dell'informatica. Questo generatore viene utilizzato in una serie di lezioni di Code.org, chiamato Code your own Flappy Game, per insegnare a creare emulatori di questo *endless runner* utilizzando sequenze di eventi.

2.2.1 Flappy Crocodile

Flappy Crocodile è un generatore automatico di Flappy Bird. Acquistando la licenza su Chupamobile per 99$, l'acquirente ha a disposizione il codice sorgente e un tutorial su come utilizzare il generatore.

Per poter creare una propria versione di Flappy Bird, utilizzando Flappy Crocodile non è richiesta alcuna conoscenza di programmazione. Seguendo il tutorial acquistato assieme al codice, in 3 ore di lavoro è possibile inserire la propria grafica, animazioni ed effetti audio.

Dopo pochi giorni dalla sua pubblicazione, Flappy Crocodile era già stato acquistato ed utilizzato da 100 diversi acquirenti [17]. Uno degli emulatori di Flappy Bird creato con questo generatore, chiamato Tiny

Flying Drizzy, è stato per alcuni giorni il gioco numero 1 nella classfica delle app gratuite dell'App Store.

2.2.2 Code your own Flappy Game

Code.org è un'organizzazione no profit che mira ad incoraggiare le persone, in particolare studenti, ad imparare l'informatica. Il sito web include lezioni gratuite di programmazione e le iniziative che organizzano hanno come obiettivo principale quello di promuovere l'introduzione di un maggior numero di lezioni di informatica nei programmi scolastici.

Code your own Flappy game è un generatore automatico di Flappy Bird sviluppato da Code.org. Il generatore e le 10 lezioni a cui è associato per il suo utilizzo, ha come scopo quello di insegnare agli utenti a strutturare il codice di un videogioco secondo una sequenza di eventi [19]. Il generatore è composto da due finestre. Nella prima è presente un insieme di blocchi, in cui ogni blocco rappresenta un evento di gioco. La seconda finestra viene dedicata alla composizione in sequenza di questi blocchi. Le composizioni create vengono interpretate dal generatore e utilizzate per creare il videogioco finale. Ogni blocco può essere personalizzato in maniera molto semplice utilizzando un menù a tendina in cui vengono presentate tutte le possibili scelte che possono essere fatte (Figura 2.6).

La creazione di una versione personalizzata di Flappy Bird utilizzando questo generatore e le lezioni ad esso associate, richiede non più di 20 minuti di lavoro.

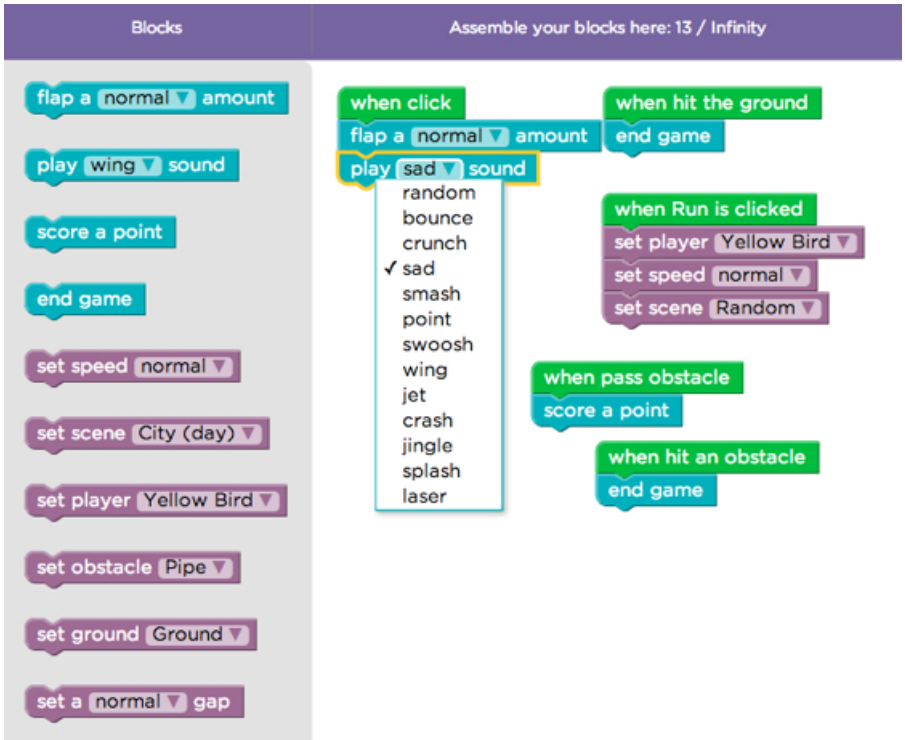

Figura 2.6: Dettaglio struttura a blocchi di Code your own Flappy Game.

Capitolo 3

Elementi degli endless runner

Nel capitolo precedente sono stati descritti alcuni degli *endless runner* di maggior successo. Questi titoli sono tra i più emulati nel mondo degli *endless runner*. Per questo motivo il generatore di *endless runner* sviluppato è stato progettato in modo tale da poter includere nei titoli generati il maggior numero di elementi presenti in questi *endless runner*.

3.1 Elementi di gioco

L'obiettivo principale in un *endless runner* è quello di far percorrere la maggior distanza possibile al proprio personaggio.

Il personaggio è l'avatar che il giocatore comanda per vivere l'esperienza di gioco. Nella maggior parte degli *endless runner*, il giocatore può effettuare quella che viene chiamata personalizzazione, ovvero la possibilità di poter scegliere tra diversi personaggi da comandare.

La corsa del personaggio termina nel momento in cui entra in collisione con un ostacolo. Per poter evitare gli ostacoli che si presentano sul percorso, il personaggio deve compiere delle determinate azioni. Le azioni a disposizione di un personaggio possono variare da un *endless runner* all'altro. Tranne in titoli basati sul volo del protagonista, come Jetpack Joyride e Flappy Bird, l'azione più comune è il salto. In Canabalt, titolo che ha dato origine al successo degli *endless runner*, il salto è l'unica azione a disposizione del giocatore. Negli *endless runner*

basati su due azioni, l'azione affiancata a quella del salto può variare da un titolo all'altro. In Robot Unicorn Attack questa azione è il *dash*, ovvero un'accelerata che permette al personaggio di frantumare alcuni ostacoli. In altri titoli, come Lastronaut, questa azione è lo sparo, che permette di eliminare alcuni ostacoli e nemici, oppure l'abbassarsi del personaggio per passare sotto gli ostacoli, detto scivolamento o *roll*, di B.C.'s Quest for Tires. Per il generatore di *endless runner* si è scelto di implementare lo scivolamento in quanto è l'azione principalmente utilizzata nei titoli a scorrimento verticale. Negli *endless runner* a scorrimento verticale con grafica 3D sono state introdotte le azioni di spostamento laterale a sinistra e a destra. In titoli come Temple Run, quest'azione viene eseguita dal giocatore inclinando il proprio dispositivo nella direazione dello spostamento desiderato, mentre in altri, come Subway Surfers, viene eseguita con la *gesture swipe*. Delle due tipologie di comandi, quella scelta per il generatore di *endless runner* sviluppato è quella basata sulla *gesture swipe*. Negli *endless runner* con spostamento laterle comandato con la *gesture swipe*, il personaggio può occupare tre posizioni prestabilite, chiamate *lane*. L'utilizzo delle *lane* permette di definire con precisione la posizione in cui il personaggio deve trovarsi per evitare il contatto con un ostacolo o per raccogliere una risorsa di gioco.

Le risorse a disposizione del giocatore in un *endless runner* sono tre: powerup, valuta virtuale primaria e valuta virtuale secondaria. Un powerup è un elemento di gioco il cui contatto col giocatore ne provoca l'attivazione delle sue proprietà per un certo numero di secondi. La durata di attivazione di un powerup dipende dal livello di potenziamento raggiunto. Per aumentare il livello di potenziamento di un powerup è necessario spendere una quantità di valuta primaria prestabilita. Ad ogni avanzamento di livello di potenziamento, il prezzo per sbloccare il livello successivo aumenta. Queste caratteristiche sono presenti in molti titoli, come Subway Surfers e Temple Run, che presentano powerup che permettono di saltare più in alto o essere invulnerabile. Le valute virtuali possono essere collezionate ed utilizzate per acquistare personalizzazioni del personaggio. Mentre la valuta virtuale primaria viene utilizzata per effettuare tutti i tipi di acquisti nel gioco, la valuta

virtuale secondaria viene utilizzata in molti titoli, come Temple Run 2 e Subway Surfers, per effettuare il *revive*, ovvero riprendere una partita persa. La valuta virtuale primaria può essere collezionata con facilità, in quanto si presenta spesso e in gruppi, mentre la valuta virtuale secondaria è rara da trovare e quindi di maggiore valore rispetto a quella primaria.

Come visto in titoli come Temple Run 2, Robot Unicorn Attack 2, Jetpack Jiyride e Subway Surfers, negli *endless runner* più recenti sono state aggiunte due tipologie di sotto-obiettivi: le sfide, proposte in maniera casuale e il cui superamento viene premiato con valuta virtuale; gli obiettivi, che sono delle particolari sfide il cui superamento viene premiato un badge. Ad esempio, in Jetpack Joyride sono presenti sfide come:

- Schiva 4 missili in una partita.

- Percorri 700m senza raccogliere monente.

- Raccogli 7 gruppi di monete in una partita.

Alcuni abbinamenti badge - obiettivi presenti sempre in questo titolo sono:

- "Not so green" - Completa 10 missioni.

- "Alpha charlie echo" - Vola per oltre 2Km.

- "Big spender" - Spendi oltre 50000 monente.

Della grande varietà di sfide e obiettivi presenti negli *endless runner*, il generatore sviluppato permette di impostare le due tipologie più comuni: raccogliere un dato numero di valute virtuali primarie in una partita e percorrere almeno una certa distanza in una partita.

3.2 Direzione di scorrimento

Negli *endless runner* a scorrimento orizzontale, il personaggio viene posizionato nella parte sinistra dello schermo, ovvero dalla parte opposta rispetto a dove vengono generati gli altri elementi di gioco. Il

personaggio può muoversi solamente sull'asse verticale, mentre sull'asse orizzontale è vincolato a rimanere nella posizione assegnatagli (Figura 3.1). Per questo motivo i titoli con questa tipologia di scorrimento vengono giocati con lo smartphone in posizione *landscape*, che prevede l'utilizzo di entrambe le mani. I comandi tipicamente utilizzati negli *endless runner* a scorrimento orizzontale sono il *tap* nella metà sinistra dello schermo per effettuare salti e il *tap* sulla metà di destra per gli scivolamenti.

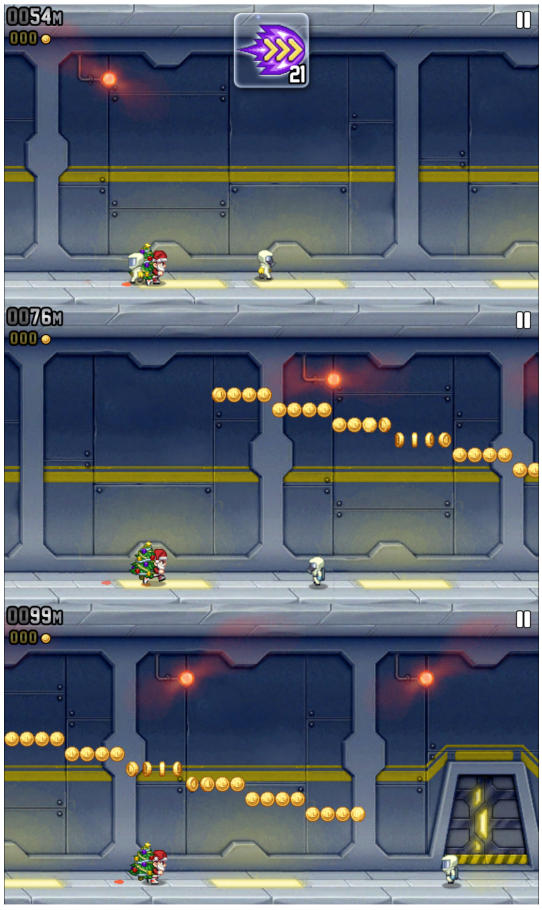

Figura 3.1: Esempio di progressione di Jetpack Joyride, titolo a scorrimento orizzontale.

Negli *endless runner* a scorrimento verticale, il personaggio può muoversi sia sull'asse verticale sia su quella orizzontale (Figure 3.2) . I movimenti sull'asse orizzontale sono tipicamente vincolati alla sola occupazione di tre posizioni specifiche chiamate *lane*, che definiscono

anche le posizioni che gli altri elementi di gioco, come valute virtuali e ostacoli, possono occupare. Per queste caratteristiche, tipicamente, i titoli a scorrimento verticale vengono giocati con lo smartphone in posizione *portrait*, che permette l'utilizzo di una sola mano. Questa caratteristica permette ai titoli a scorrimento verticale di poter essere giocati anche in situazioni in cui è necessario utilizzare una mano, ad esempio, per reggersi in metropolitana o sul tram. Le azioni comunemente utilizzate in questi titoli prevedono salti, scivolamenti e spostamenti sulle *lane*. Queste azioni possono essere eseguite dal giocatore utilizzando la *gesture swipe*.

Figura 3.2: Esempio di progressione di Subway Surfers, titolo a scorrimento verticale.

3.3 Animazioni e audio

Gli *endless runner* sono dei videogiochi frenetici e ripetivi in cui ogni partita può durare anche soli pochi secondi. La frenesia di un *endless runner* è data dal ritmo sostenuto di ogni partita, dalla dinamicità degli elementi di gioco e dall'ambientazione. La dinamicità degli elementi di gioco è data dall'insieme delle loro animazioni. Ogni personaggio e powerup deve essere dotato di almeno un'animazione per ogni azione che può compiere o subire. Questi devono quindi essere in pos-

Tabella 3.1: Animazioni ed effetti audio per elementi di gioco

	Idle	Salto	Atterraggio	Scivolamento	Spostamenti	Collisione
Personaggio	Si	Si	Si	Si	Si	Si
Ostacolo	Si	No	No	No	No	Si
Powerup	Si	Si	Si	Si	Si	Si
Valute	Si	No	No	No	No	Si
Ambiente	Si	No	No	No	No	No

sesso di animazioni per ogni tipo di movimento che possono effettuare e per quando entrano in contatto con un ostacolo. Le due valute virtuali e gli ostacoli sono dotati di un'animazione standard per rendere visibile la loro presenza in scena e un'animazione per quando entrano in contatto col personaggio. Ognuna di queste animazioni viene accompagnata da un effetto audio che viene riprodotto in concomitanza della riproduzione dell'animazione.

Anche l'ambiente contribuisce a rendere le partite frenetiche e può essere dotata di animazioni. Nella maggior parte degli *endless runner*, l'ambiente non può entrare in contatto col personaggio, tranne in casi particolari come Jetpack Joyride, e ha come scopo quello di mostrare il mondo da cui il personaggio è in fuga. L'utilizzo di audio e animazioni per i rispettivi elementi di gioco é sintetizzato nella Tabella 3.1.

Il senso di frenesia è dato anche dalle musiche utilizzate negli *endless runner* come sottofondo. Queste, tipicamente, hanno un ritmo sostenuto e ripetitivo.

Capitolo 4

Sviluppo e progettazione

4.1 Strumenti di sviluppo

L'architettura del sistema è stata creata con l'ausilio dell'engine Unity 3D. L'ambiente di sviluppo e le librerie che mette a disposizione ben si prestano per la realizzazione di videogiochi e generatori di videogiochi.

Dei linguaggi di programmazione messi a disposizione da Unity 3D, il codice per lo sviluppo del generatore di *endless runner* è stato realizzato in C#. Il codice del progetto è stato scritto utilizzando l'editor MonoDevelop, che viene fornito di default con l'installazione di Unity 3D.

I salvataggi dei progressi di gioco e le impostazioni degli elementi che lo compongono sono stati gestiti col formato JSON. Una delle caratteristiche principali del formato JSON è quella di fornire una struttura delle informazioni di facile lettura e modifica.

4.1.1 Unity 3D

Unity 3D è una flessibile e potente piattaforma di sviluppo di videogiochi ed esperienze interattive 2D e 3D multipiattaforma. L'ambiente di sviluppo Unity 3D è compatibile sia con Microsoft Windows sia con Mac OS X. I videogiochi sviluppati in Unity 3D possono essere eseguiti su Windows, Mac, Linux, Xbox 360, Xbox One, PlayStation 3,

Playstation 4, PlayStation Vita, Wii, Wii U, iPad, iPhone e Android. Unity 3D è in grado anche di produrre videogiochi per browser web che utilizzano il plugin Unity Web Player, supportato sia su Mac che su Windows.

MonoDevelop è l'ambiente di sviluppo integrato (IDE) in dotazione con Unity 3D. Un IDE combina le operazioni tipiche degli editor di testo con funzionalità di debugging e di gestione di progetti. MonoDevelop viene installato automaticamente insieme a Unity 3D e viene preimpostato come editor di default per la stesura del codice.

Grazie a questa piattaforma, nel 2009 Unity Technologies è stata nominata da Gamasutra tra le "Game Companies of 2009" e l'anno successivo, Unity 3D è stato premiato con il "Technology Innovation Award" del Wall Street Journal nella categoria software.

Unity 3D permette di gestire i videogiochi prodotti in scene. Ogni scena è composta da un'insieme di oggetti, chiamati Game Object, organizzati in gerarchie. I Game Object hanno diverse proprietà e componenti che ne definisco l'aspetto e il comportamento, che possono essere modificati utilizzando una particolare sezione dell'interfaccia grafica di Unity 3D chiamata Inspector. Una volta ultimata la scena, è possibile testare e visualizzare il prodotto finito grazie ad un'anteprima del videogioco. Unity 3D permette l'importazione automatica di molte tipologie di formati di file. Nel momento in cui un software esterno modifica un file precedentemente caricato nel proprio progetto, Unity 3D effettua nuovamente e in maniera automatica l'importazione del file aggiornato. Unity 3D supporta l'integrazione con i seguenti software: 3D Studio Max, Maya, Softimage, Blender, Modo, ZBrush, C4D, Cheetah3D, Photoshop, Adobe Fireworks e Allegorithmic Substance.

Unity 3D è stato utilizzato come motore per la creazione di molti titoli di successo come i già citati Temple Run e Temple Run 2 di Imagine, Bad Piggies e Angry Birds Epic di Rovio, Hearthstone: Heroes of Warcraft di Blizzard Enterteinment e Monument Valley di Ustwo.

4.1.2 JSON

JSON, o JavaScript Object Notation, è un formato standard che utilizza una struttura *human-readable* per la conservazione e organiz-

zazione di informazioni.

Le informazioni riportate nei file JSON possono essere composte da una singola proprietà ad un oggetto complesso, che è un insieme di proprietà diverse. Una proprietà è costituita da una coppia attributo - valore, come illustrato di seguito:

Algorithm 1 Proprietà di un oggetto in JSON

```
"nome" : "valore"
```

Per definire, invece, un oggetto è necessario elencare le proprietà che lo caratterizzano, separate da virgole e racchiuse tra parentesi graffe, come mostrato nell'esempio seguente:

Algorithm 2 Definizione di un oggetto in JSON

```
{
"nome" : "Christian",
"cognome" : "Miranti",
"età" : "25"
}
```

Un oggetto può essere anche il valore di un attributo di un altro oggetto. Questo può essere definito in formato JSON nel modo seguente:

Algorithm 3 Oggetti come propietà di oggetti in JSON

```
{
"nome" : "Christian",
"cognome" : "Miranti",
"età" : "25",
"università":
{
"nome" : "Politecnico",
"sede" : "Milano Leonardo"
}
}
```

Infine, è anche possibile definire liste di diversi oggetti o proprietà che hanno gli stessi attributi utilizzando sempre la separazione con le virgole ma racchiudendoli all'interno di parentesi quadre:

Algorithm 4 Liste di oggetti in JSON

```
[
{
"nome" : "Christian",
"cognome" : "Miranti",
"età" : "25"
},
{
"nome" : "Davide",
"cognome" : "Falco",
"età" : "25"
}
]
```

4.2 Organizzazione degli Assets

4.2.1 Resources

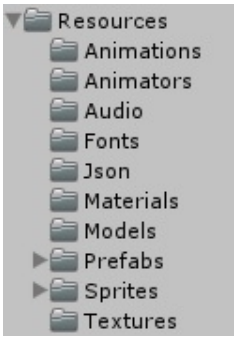

Figura 4.1: Dettaglio sul contenuto della cartella Resources

La libreria UnityEngine.Resources di Unity 3D permette di accedere via codice agli asset caricati all'interno del proprio progetto. Per potervi accedere è necessario raccoglierli all'interno di una cartella chiamata, appunto, Resources. Questa cartella è stata organizzata in sottocartelle in modo tale da gestire separatamente tipologie diverse di file (Figura 4.1).

La sottocartella Animations è quella preposta a contenere le animazioni dei modelli 3D che si desidera importare. Anche se inizialmente i modelli 3D possono essere importati con tutte le proprie animazioni in un unico file, è importante suddividerle in file separati e nominarli ade-

guadatamente per poterli rendere facilmente utilizzabili e rintracciabili in seguito.

La sottocartella Animators contine quelli che Unity 3D definisce Animator Controller, il cui scopo è quello di organizzare e mantenere un insieme di animazioni per personaggi ed altri elementi di gioco animati. Il controller contiene riferimenti alle animazioni che gestisce e riproduce sfruttando una struttura a macchina a stati. Ad ogni stato corrisponde una particolare animazione e le transizioni da uno stato all'altro avvengono al verificarsi di condizioni programmabili.

Nella sottocartella Audio vengono inseriti gli effetti audio e le musiche che vengono riprodotte all'interno del videogioco generato. In Unity 3D i file audio vengono importati come Clip Audio e gestiti attraverso dei componenti chiamati Audio Source.

Nella sottocartella Fonts vengono inseriti i font che possono essere utilizzati per gli elementi testuali dell'interfaccia grafica.

La sottocartella JSON contiene i file di salvataggio e di configurazione del videogioco generato. Ciascuno di questi file si occupa della configurazione e salvataggio di una singola tipologia di elemento di gioco. I dettagli di questi file vengono trattati nel Paragrafo 4.3 e nel Pragrafo 4.4.

Le sottocartelle Materials, Models e Texture sono quelle preposte all'importazione dei modelli 3D con i propri materiali e textures. La suddivisione in cartelle di questi file permette di tenere separati i diversi aspetti del modello finale e poterlo personalizzare con più facilità.

I modelli finali, ricostruiti in Scene, vengono importati nella sottocartella Prefabs, che è la sottocartella a cui il codice del progetto fa riferimento per il caricamento degli elementi 3D che utilizzati nel videogioco finale.

Infine, nella sottocartella Sprites sono presenti i singoli elementi grafici, come bottoni e finestre, riguardanti l'interfaccia di gioco.

4.2.2 Scenes

All'interno della cartella Scenes sono presenti le scene, o livelli, che compongono il videogioco finale. In ognuno di essi sono presenti i Game

Object essenziali per il corretto caricamento e relativo funzionamento del livello stesso. Le scene previste sono le seguenti:

- Loading: si occupa della lettura e interpretazione dei file JSON e del relativo caricamento degli elementi di gioco.
- MainMenu: presenta il menù principale del gioco, dove è possibile raggiungere tutte le altre sezioni.
- Personalization: dove è possibile acquistare e selezionare nuovi personaggi e aumentare il livello di potenziamento dei powerup.
- Run: è la partita vera e propria.
- Showcase: dove visualizzare i badge conquistati grazie alle proprie performance e quelli ancora da conquistare.
- Temp: scena utilizzata in fase di creazione dei Prefab degli ostacoli, è composta da diversi Game Object che facilitano la creazione dei Collider.

4.2.3 Scripts

Nella cartella Script sono presenti tutti gli script utilizzati nel sistema. Gli script sono stati suddivisi in sottocartelle in base agli elementi di gioco a cui vengono assegnati. Le sottocartelle degli script sono quindi:

- Badge
- Challenge
- Character
- Composition
- Environment
- FirstVirtualValue
- Obstacle
- Powerup

- SecondVirtualValue
- Match

In ognuna di queste cartelle associate ad elementi di gioco è presente in particolare uno script chiamato Manager che si occupa del caricamento dei salvataggi e delle impostazioni presenti nei file JSON relativi agli elementi di gioco stessi. Per esempio, nella sottocartella Character di Scripts è presente uno script denominato CharacterManager che si occupa del caricamento di tutte le informazioni riguardanti i personaggi presenti nel file Characters.json. Gli altri script si occupano della gestion del comportamento di ogni elementi di gioco e della loro eventuale visualizzazione sull'interfaccia grafica.

4.2.4 Templates

Nella cartella Templates sono presenti file di diverse tipologie che vengono utilizzati come punto di partenza per creare elementi grafici e di gioco. Ad esempio, sono presenti i template per ognuno dei file JSON della sottocartella JSON, oppure il template per l'Animator Controller dei personaggi, che presenta tutti gli stati che un personaggio può possedere e le condizioni di transizione da uno stato all'altro.

4.2.5 UI

All'interno della cartella UI sono presenti i Prefab creati dei singoli elementi grafici inseriti all'interno delle scene del videogioco. Ogni pulsante, barra, o area con particolari informazioni, è presente all'interno di questa cartella e modificabile direttamente dall'Inspector. Le modifiche apportate ad un Prefab vengono propagate automaticamente a tutti i Game Object creati a partire da quel particolare Prefab.

4.3 Impostazioni iniziali

Per poter gestire i movimenti del personaggio, la sua posizione e quella degli altri elementi di gioco, sono state utilizzate delle coordinate specifiche come punti di riferimento. Sono state definite 3 altezze di riferimento:

- Low: corrisponde all'altezza 0, ovvero il livello del piano di gioco.

- Medium: corrisponde all'altezza +1, ovvero l'altezza del personaggio quando è in fase di scivolamento.

- High: corrisponde all'altezza +2, ovvero l'altezza del personaggio quando è in fase di corsa.

Sull'asse orizzontale, invece, le 3 posizioni di riferimento sono:

- Left: corrisponde alla posizione -1.

- Middle: corrisponde alla posizione 0.

- Right: corrisponde alla posizione +1.

Le 3 posizioni sull'asse orizzontale vengono chiamate *lane* e vengono utilizzate per posizionare elementi di gioco e definire dove può trovarsi il personaggio effettuando spostamenti laterali quando la tipologia di scorrimento scelta per il gioco è quella verticale (Figura 4.2). Nel caso in cui, invece, la tipologia di scorrimento scelta è quella orizzontale, il generatore allinea tutti gli elementi di gioco alla posizione orizzontale chiamata Middle e gli spostamenti laterali da parte del personaggio vengono disabilitati (Figura 4.3).

Il file JSON preposto alle impostazioni iniziali è chiamato Game.json, ed è il file in cui vengono definite le caratteristiche generali del gioco. Il primo attributo riguarda il titolo che si vuole dare al videogioco. Il nome scritto in questo campo viene visualizzato nel menù principale. Il campo successivo riguarda la tipologia di scorrimento del videogioco. Come detto in precedenza, un *endless runner* può essere a scorrimento orizzontale oppure a scorrimento verticale. Per questo motivo gli unici valori ammessi in questo campo sono "horizontal" e "vertical".

Gli attributi successivi riguardano informazioni generali sulla situazione attuale del giocatore. Vengono quindi indicate le quantità di valute primaria e secondaria che sono state accumulate e quante di queste non sono ancora state spese e quindi utilizzabili per fare acquisti nel gioco. Successivamente vengono salvati i dati riguardanti i metri che sono stati percorsi in totale dal giocatore a partire dalla sua prima

partita e quale è stata la sua prestazione migliore, che ne definisce il record personale.

Viene inoltre indicato l'indice corrispondente al personaggio che si sta utilizzando attualmente e gli indici di tre diverse sfide da superare nelle partite. Tali indici fanno riferimento alla posizione degli elementi indicati (in questo caso il personaggio e le sfide) negli appositi file JSON (Characters.json e Challenges.json, rispettivamente), ricordando che il sistema di numerazione degli indici in un file JSON inizia da 0.

Algorithm 5 Game.json

```
{
"title" : "titolo del gioco",
"scrolling" : "horizontal/vertical",
"total_first_virtual_value" : "0",
"actual_first_virtual_value" : "0",
"total_second_virtual_value" : "0",
"actual_second_virtual_value" : "0",
"total_meters" : "0",
"record" : "0",
"running_character" : "0",
"challenges" : [
"0",
"0",
"0"
]
}
```

4.4 Elementi di gioco

4.4.1 Personaggi

I personaggi sono gli avatar che il giocatore utilizza per vivere l'esperienza di gioco (Figura 4.4). Tipicamente negli *endless runner*, viene data la possibilità di cambiare il personaggio di default o, almeno, di poterlo personalizzare. Inoltre non tutti i personaggi e le personalizzazioni sono disponibili immediatamente al primo accesso al gioco, ma sono usufruibili solo dopo averli acquistati con la valuta virtuale richiesta. I personaggi vengono rappresentati con modelli 3D che possiedono

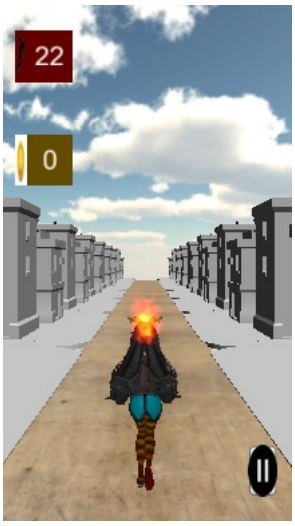

Figura 4.2: Risultato impostazione dello scorrimento ad ”vertical”

Figura 4.3: Risultato impostazione dello scorrimento ad ”horizontal”

le stesse caratteristiche, come le tipologie di animazioni e di effetti audio, ma personalizzate da personaggio a personaggio.

In particolare, ogni personaggio è contraddistinto da un nome e da un modello 3D che lo rappresenta. Il nome del modello 3D riportato all'interno del file JSON come campo dell'attributo ”model”, deve corrispondere al nome del Prefab desiderato all'interno della cartella Resources/Prefabs.

E' possibile selezionare o acquistare un personaggio all'interno della sezione Personalization. Per far ciò, deve essere quindi indicata anche un'immagine di presentazione di ogni personaggio, importata nella car-

tella Sprites. Deve essere inoltre indicato se il giocatore è già in possesso del personaggio oppure se ancora non è stato acquistato. L'acquisto di un personaggio può avvenire utilizzando una delle due valute virtuali a disposizione, ognuna indicata con una quantità diversa a causa della diversa facilità con cui queste due valute possono essere raccolte.

Le azioni che un personaggio può compiere, come già accennato nei capitoli precedenti, sono: correre, saltare, atterrare, girare a destra, girare a sinistra, scivolare e morire. Per ognuna di queste sarà quindi necessario indicarne animazioni e relativi effetti audio. Gli Audio Clip corrispondenti, vengono definiti direttamente all'interno del file JSON, che li cercherà nella cartella Resources/Audio. Le animazioni, invece, devono essere impostate utilizzando il template dell'Animator Controller dei personaggi già predisposto. Questo deve quindi essere copiato nella cartella Resources/Animators, rinominato a piacere e impostato con un semplice *drag and drop* delle animazioni su ogni stato della macchina a stati. Nel file JSON è quindi sufficiente indicare il nome dato all'Animator Controller per poterlo abbinare al modello 3D. Nel caso in cui si dovesse scegliere come tipologia di scorrimento quella orizzontale, allora animazioni e audio impostati per gli spostamenti a destra e a sinistra verrebbero ignorati in quanto queste azioni non sono possibili in questa tipologia di scorrimento.

Figura 4.4: Dettaglio di un personaggio utilizzato nel generatore

Algorithm 6 Character.json

```
[
{
"name" : "nome del personaggio",
"model" : "nome del modello presente nalla cartella Prefabs",
"presentation_image" : "nome dell'immagine presenti nella cartella
Sprites",
"is_owned" : "true/false",
"coins_value" : "0",
"gemes_value" : "0",
"audio" : {
"run":"nome dell'Audio Clip presente nella cartella Audio",
"jump":"nome dell'Audio Clip presente nella cartella Audio",
"land":"nome dell'Audio Clip presente nella cartella Audio",
"turn_right":"nome dell'Audio Clip presente nella cartella Audio",
"turn_left":"nome dell'Audio Clip presente nella cartella Audio",
"roll":"nome dell'Audio Clip presente nella cartella Audio",
"die":"nome dell'Audio Clip presente nella cartella Audio"
},
"animator_controller" : "nome dell'Animato Controller persente nella
cartella Animator"
}
]
```

4.4.2 Ostacoli

Gli ostacoli sono quegli elementi di disturbo che tentano di interrompere la corsa del personaggio. Questi vengono caratterizzati in particolare da un nome e dal modello 3D che li rappresenta in scena. Le altre caratteristiche sono l'Audio Clip utilizzato quando è in gioco e l'Audio Clip utilizzato quando entra in collisione col personaggio. Per entrambe queste situazioni, viene prevista un'animazione, che viene impostata e controllata dall'Animator Controller specificato all'interno del file JSON. La modalità di impostazione dell'Animator Controller è la stessa descritta nel personaggio ma utilizzando un altro template, che prevede appunto il passaggio tra le animazioni di idle e di collisione.

Algorithm 7 Obstacle.json

```
[
{
"name":"nome dell'ostacolo",
"model":"nome Prefab in Prefabs",
"presentation_image":"nome immagine in Sprites",
"audio":
{
"idle":"nome AudioClip in Audio",
"hit":"nome AudioClip in Audio"
},
"animator_controller":"nome AnimatorController in Animators"
}
]
```

La creazione degli ostacoli necessita di un passo aggiuntivo, che è la creazione dei suoi Collider. Per far ciò è sufficiente impostare il proprio modello all'interno della scena appositamente creata chiamata Temp. Importato il modello nella scena e posizionato in modo tale che la sua base corrisponda alla posizione (0, 0, 0), è sufficiente creare dei Game Object senza mesh con i soli collider, posizionarli nelle posizioni in cui si prevede di avere una collisione, impostarli come figli del modello principale e importare il Game Object creato all'interno della cartella Prefabs. Ad ogni oggetto che rappresenta un Collider, deve essere aggiunto lo script denominato ObstacleHitCollider, presente all'interno della cartella Scripts/Obstacle.

4.4.3 Valuta virtuale primaria

La valuta primaria è la valuta virtuale che viene utilizzata per effettuare tutte le tipologie di acquisti all'interno di un videogioco. Questa valuta può essere accumulata entrandovi in contatto durante le partite oppure superando le sfide proposte.

La valuta primaria è rappresentata da un nome e da un modello 3D. Per poterla facilmente individuare anche all'interno della UI, deve essere speciaficata anche un'immagine di presentazione. Come per gli ostacoli, possono essere specificati gli Audio Clip della modalità idle e collisione e l'Animator Controller contente le animazioni corrispondenti.

Una particolarità delle valute primarie è che durante una partita si presentano a gruppi ordinati. Per questo motivo, all'interno del file JSON, è possibile definire tutte le tipologie di raggruppamenti, o pattern, che possono presentarsi. Per far ciò è sufficiente definire un nome nella sezione "pattern" e indicare la posizione sull'asse delle z e sull'asse delle y di ciascuna valuta primaria che lo compone.

4.4.4 Valuta virtuale secondaria

La valuta virtuale secondaria è quella valuta virtuale che viene utilizzata per effettuare il *revive* al termine di una partita e per acquistare personalizzazioni.

Come per la valuta primaria, questa è rappresentata da un nome, da un modello 3D, un'immagine di presentazione che la rappresenta nell'interfaccia grafica, dagli Audio Clip e dall'Animator Controller da impostare con le animazioni delle modalità idle e collisione. A differenza degli altri elementi di gioco citati fino ad ora, la valuta virtuale secondaria ha un attributo in più in cui definire la probabilità che venga istanziata in partita quando è prevista la sua presenza. Infatti in ogni composizione viene sempre definita la posizione della valuta secondaria, ma con questo campo aggiuntivo viene garantita la sua peculiarità di poter essere trovata raramente.

Algorithm 8 Coin.json

```
{
"name":"nome valuta primaria",
"model":"nome Prefab in Prefabs",
"presentation_image":"nome immagine in Sprites",
"audio":
{
"idle":"nome AudioClip in Audio",
"hit":"nome AudioClip in Audio"
},
"patterns":
[
{
"name":"nome pattern",
"coins":
[
{
"Deep":"0.0",
"FloorLevel":"Low/Medium/High"
}
]
}
]
}
```

Algorithm 9 Gem.json

```
{
"name":"nome seconda valuta",
"model":"nome Prefab in Prefabs",
"presentation_image":"nome immagine in Sprites",
"probability":"0.0",
"audio":
{
"stay":"nome AudioClip in Audio",
"hit":"nome AudioClip in Audio"
},
"animator_controller":"nome AnimatorController in Animators"
}
```

4.4.5 Powerup

Il powerup è quell'elemento di gioco che, durante una partita, aiuta il personaggio, donandogli proprietà che gli facilitano la corsa o altri compiti secondari, come la raccolta di valute virtuali primarie.

Come gli altri elementi di gioco citati fino ad ora, questo è caratterizzato da un nome e un modello 3D, che rappresenta ciò che prende temporaneamente il posto del personaggio principale quando un powerup viene attivato. L'attivazione di un powerup avviene quando si entra in contatto di un particolare Game Object a forma di cubo che può presentarsi sul percorso durante una partita, raffigurante l'immagine di presentazione del powerup stesso. Tale immagine, viene anche utilizzata all'interno della sezione Personalization per rappresentare nella UI il powerup.

Come per la valuta virtuale secondaria, anche il powerup è un oggetto che non si trova frequentemente durante una partita. Per questo motivo, anche all'interno del file JSON è necessario necessario impostare la probabilità con cui questo può apparire.

Le impostazioni dell'audio e delle animazioni è invece molto simile a quello visto per il personaggio principale, in quanto è un oggetto controllabile e che quindi può effettuare una varietà di azioni. Di base, tali azioni corrispondono a quelle che il personaggio può effettuare. L'impostazione dell'Animator Controller viene effettuata a partire dallo stesso template messo a disposizione per i personaggi.

Infine viene considerato il fatto che ogni powerup ha una durata di utilizzo limitata e che tale durata è direttamente proporzionale al suo livello di potenziamento. Per impostare questa caratteristica, all'interno del file JSON è previsto un campo riguardante il livello che ogni powerup può avere (non superiore a 4) e il suo valore iniziale per poter effettuare il primo potenziamento. Il valore del passaggio di livello e il tempo di utilizzo del powerup, impostato a 10 secondi, eṕroporzionale al livello di potenziamento del powerup stesso.

Algorithm 10 Powerup.json

```
[
{
"name" : "nome powerup",
"category" : "Vehicle/Magnet",
"presentation_image" : "nome immagine in Sprites",
"probability" : "0.0",
"level" : "0",
"initial_value" : "0",
"model" : "nome Prefab in Prefabs",
"audio_substitute" :
"run" : "nome AudioClip in Audio",
"turn_right" : "nome AudioClip in Audio",
"turn_left" : "nome AudioClip in Audio",
"die" : "nome AudioClip in Audio"
}
}
]
}
```

4.4.6 Composizioni

Una composizione è un insieme di ostacoli, pattern di valuta primaria, valuta secondaria e powerup che si presenta contemporaneamente al giocatore ad intervalli regolari e a distanze l'uno dall'altro prestabilite (Figura 4.5).

Ogni composizione è caratterizzata da un nome e, per ognuno degli elementi che lo compongono, viene definito l'indice dell'elemento all'interno del rispettivo file JSON. Ad ogni elemento di gioco viene assegnata una *lane* (se si sta parlando di scorrimento verticale, altrimenti la *lane* viene impostata di default su quella centrale), l'altezza dal terreno e la sua profondità rispetto al centro della composizione.

4.4.7 Sfide

Le sfide sono il primo elemento di gioco descritto a non avere un modello 3D che le rappresenta. Ogni sfida è un obiettivo da raggiungere, il cui superamento viene premiato con delle valute primarie oppure con un badge.

Algorithm 11 Composition.json

```
[
{
"name":"nome della composizione",
"obstacle":
{
"number":"indice in Obstacle.json dell'ostacolo scelto",
"position":
{
"Lane":"Left/Middle/Right",
"Floor_Level":"Low/Medium/High",
"Deep":"un numero qualsiasi di tipo float"
}
},
"powerup":
{
"number":"indice in Powerup.json del powerup scelto",
"position":
{
"Lane":"Left/Middle/Right",
"Floor_Level":"Low/Medium/High",
"Deep":"un numero qualsiasi di tipo float"
}
},
"coins":
{
"number":"indice in Coin.json del pattern di valuta primaria scelto",
"position":
{
"Lane":"Left/Middle/Right",
"Floor_Level":"Low/Medium/High",
"Deep":"un numero qualsiasi di tipo float"
}
},
"geme":
{
"position":
{
"Lane":"Left/Middle/Right",
"Floor_Level":"Low/Medium/High",
"Deep":"un numero qualsiasi di tipo float"
}
}
}
]
```

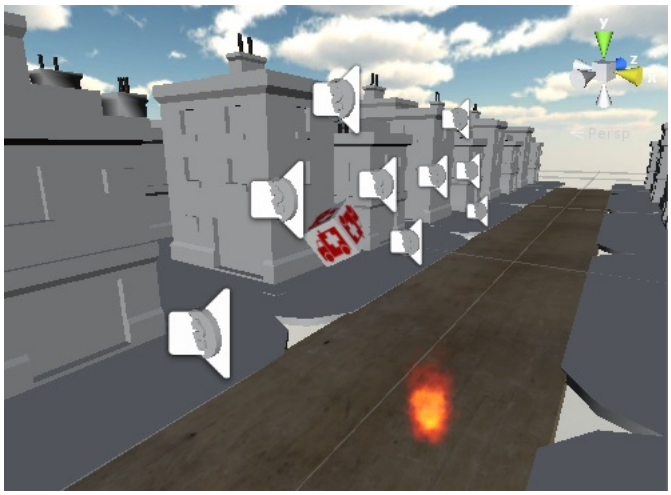

Figura 4.5: Dettaglio di una composizione ricreata durante una partita creata dal generatore

Una sfida viene rappresentata da un nome, da una tipologia e da un numero. La tipologia deve essere una di quelle preimpostate e corrisponde ad un concetto quale "percorri almeno *tot* metri in un match" oppure "arriva a possedere fino a *tot* valute primarie". Il numero da definire identifica il valore che quel *tot* deve assumere per far sì che la sfida sia considerata superata.

Infine viene indicato se la sfida è utilizzabile per i badge o per le partite. Se si sceglie di utilizzare una sfida per un badge, vuol dire che è possibile definire un badge che viene vinto se tale sfida viene superata. In caso contrario, tale sfida può essere utilizzata per le sfide proposte durante le partite. Quando una di queste viene superata, allora il giocatore viene premiato con della valuta primaria e tale sfida viene sostituita da una nuova, diversa da quella appena superata e dalle altre ancora da superare.

4.4.8 Badge

I badge sono delle rappresentazioni di premi per aver superato particolari sfide. Ogni badge è quindi rappresentato da un nome e da una sfida. La sfida è indicata dall'indice che questa occupa all'interno della lista di sfide del rispettivo file JSON.

Algorithm 12 Challenge.json

```
[
{
"name":"nome sfida",
"type":"Match_Meters/Match_Coins/Total_Meters/Total_Coins",
"number":0,
"for_bedge":"true/false"
}
]
```

A queste caratteristiche viene aggiunta un'immagine del badge, che viene visualizzata all'interno della sezione Showcase e, per pochi istanti, durante la partita in cui la sfida è stata superata, e un valore booleano che indica se tale sfida è stata già superata, e quindi se il giocatore è in possesso del badge, o meno.

Algorithm 13 Badge.json

```
[
{
"name" : "nome badge",
"presentation_image" : "nome immagine in Sprites",
"challenge_number" : "0",
"unlocked" : "true/false"
}
]
```

4.4.9 Ambiente

L'ambiente identifica tutti quegli elementi con cui il personaggio e agli elementi delle composizioni non possono interagire e il cui unico scopo è quello di rappresentare il mondo in cui è ambientato il videogioco (Figura 4.6).

In particolare viene definito il Material utilizzato nella Skybox e la Texture utilizzata per il terreno su cui il personaggio si trova a correre. Inoltre è presente una lista di modelli 3D che vengono presentati a lato della zona di gioco, identificati da un nome, il nome del modello 3D, la sua lunghezza, la distanza rispetto al centro della scena, un eventuale Audio Clip e un eventuale Animator Controller.

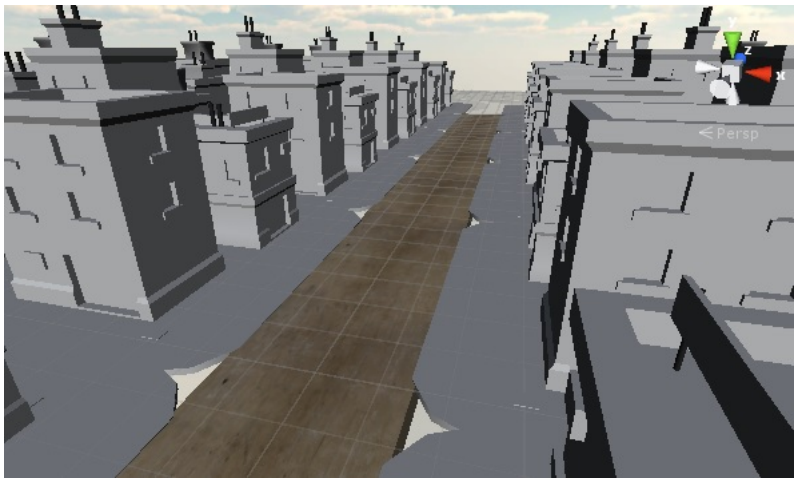

Figura 4.6: Dettaglio di una serie di ambienti ricreati dal generatore per un endless runner a scorrimento verticale

4.5 Sezioni di gioco

4.5.1 Menù principale

Il menù principale è la prima sezione ad essere visualizzata subito dopo aver effettuato il caricamento di tutti gli elementi di gioco e dei salvataggi. Questo è il punto centrale del sistema in quanto vengono visualizzate le informazioni principali dello stato attuale del giocatore ed è l'unica sezione dalla quale è possibile accedere a tutte le altre sezioni.

La prima informazione che viene fornita è il titolo del gioco, visualizzato nella parte alta dello schermo, affianco del quale è possibile trovare le quantità di valuta primaria e di valuta secondaria attualmente in possesso del giocatore. Viene anche mostrato il miglior punteggio ottenuto dal giocatore e le tre sfide attualmente attive che si possono superare durante le partite. A fare da sfondo al pulsante che permette di iniziare una nuova partita, vi è l'immagine di presentazione del personaggio attualmente selezionato per giocare. Nella parte inferiore dello schermo, sono presenti i pulsanti dedicati al passaggio alla sezione riguardante la personalizzazione e la sezione riguardante la visualizzazione dei badge.

L'interfaccia grafica del menù principale e delle altre sezioni sono state realizzate in due versioni, una per ogni possibile tipologia di

scorrimento (Figura 4.7 e Figura 4.8).

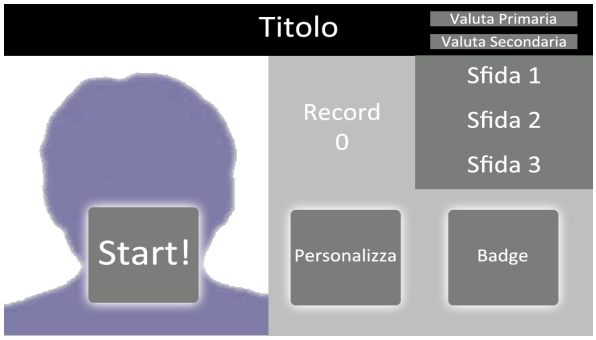

Figura 4.7: Composizione elementi grafici del menù principale in orizzontale.

4.5.2 Partita

La partita corrisponde al gioco vero e proprio, ovvero il momento in cui il giocatore può utilizzare il personaggio selezionato per tentare di battere il proprio record di distanza percorsa. Il giocatore può effettuare svariate azioni oltre al comandare il personaggio, come mettere in pausa o ricominciare la partita. Dovendo gestire i salvataggi dei progressi fatti e la creazione procedurale di ambiente ed elementi di gioco, la partita è stata sviluppata e gestita come una macchina a stati finiti (Figura 4.9).

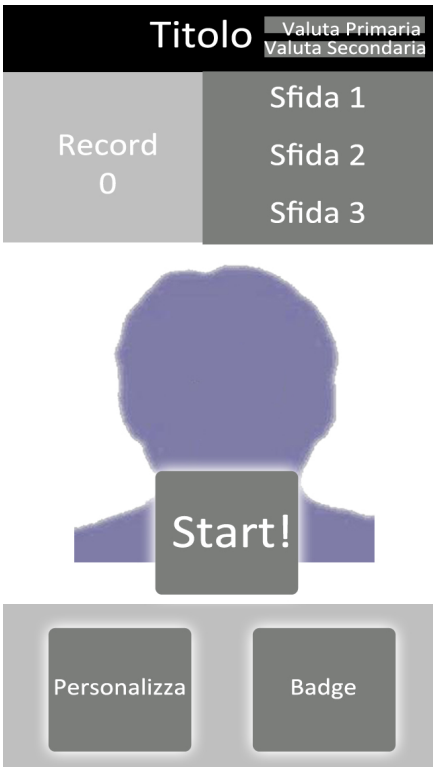

Figura 4.8: Composizione elementi grafici del menù principale in verticale.

4.5.2.1 Start

Lo stato iniziale della macchina a stati della partita è stato chiamato Start. Questo è lo stato dedicato al caricamento in scena degli elementi che compongono ciò che il giocatore vede durante la corsa a partire dal primo frame. Viene caricato il Material della Skybox e la Texture del terreno di gioco. Se la tipologia di scorrimento scelta è quella verticale, la Camera, ovvero ciò che definisce cosa vede il giocatore, si trova già nella posizione corretta, il personaggio viene posizionato e impostato con tutti gli script creati per comandarlo sia sull'asse verticale che su quello orizzontale e l'ambiente di gioco viene posizionato su ambo i lati del terreno. Nel caso dello scorrimento orizzontale, invece, la Camera viene posizionata in modo tale da vedere lateralmente a distanza il personaggio, il quale viene dotato dei soli script creati per i movimenti

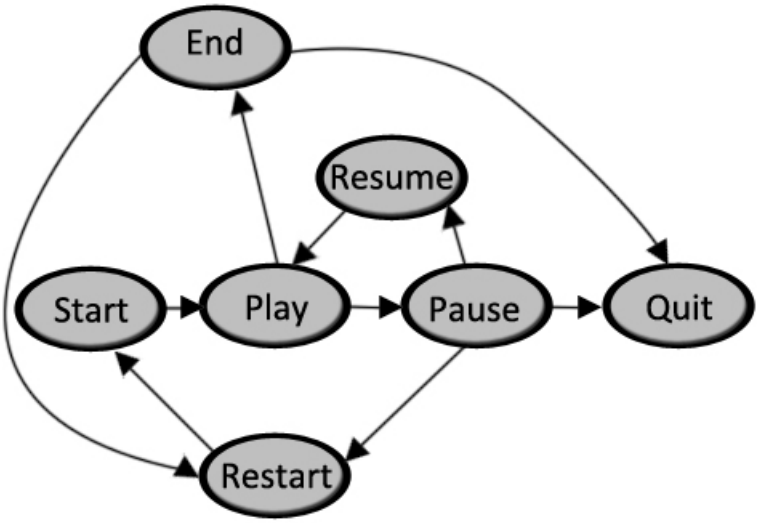

Figura 4.9: Macchina a stati della partita.

sullasse verticale. L'ambiente viene ricreato lungo il terreno di gioco sul lato opposto rispetto alla posizione della Camera.

4.5.2.2 Play

Quando le operazioni svolte dallo stato Start sono vengono ultimate, lo stato successivo è quello denominato Play. Questo è lo stato in cui il giocatore prende il comando del proprio personaggio per tentare di battere il proprio record. In questo stato, gli script di controllo del personaggio sono attivi, potendo quindi muoversi secondo i limiti previsti per la tipologia di scorrimento del gioco. Ad intervalli regolari, il sistema seleziona una composizione casuale e ne carica gli elementi che la compongono nella parte più lontana del terreno di gioco. All'ostacolo, al gruppo di valute primarie e agli eventuali powerup e valute secondarie di ogni composizione viene assegnato in aggiunta uno script che ne controlla il movimento sull'asse z. Nella scena, infatti, non è il personaggio che avanza, ma sono gli altri elementi che si dirigono verso di esso. Questo script viene assegnato anche all'ambiente, che però viene caricato in modo tale che tra un elemento dell'ambiente e il successivo non vi si siano sovrapposizioni o spaziature non desiderate.

La velocità con cui questi elementi si muovono sull'asse z è la stessa e viene controllata dal Match Manager, un Game Object dotato di uno script che ha il compito di aumentare gradualmente la velocità fino ad arrivare ad un limite massimo prestabilito. Tutti i Game Object che escono dal campo visivo della Camera vengono eliminati. Ad ogni frame viene inoltre fatto un controllo sulle sfide attive e sui badge ancora non in possesso del giocatore. Nel momento in cui risulta che una sfida di gioco o sfida associata ad un badge è stata superata, il giocatore viene avvisato con un messaggio su schermo che rimane visibile per pochi secondi.

4.5.2.3 Pause

Nello stato Play è presente un pulsante per passare allo stato denominato Pause. In questo stato, tutti gli elementi di gioco vengono interrotti e un'interfaccia grafica riassuntiva dei metri percorsi, valute primarie raccolte, sfide superate e sfide ancora da superare, si presenta al giocatore.

4.5.2.4 End

Nel momento in cui il personaggio entra in contatto con un ostacolo, dallo stato di Play si passa allo stato di End. Come nello stato Pause, tutti gli elementi di gioco vengono interrotti e la situazione generale della partita viene mostrata al giocatore. A differenza dello stato di Pause, però, qui avvengono i salvataggi dei progressi del giocatore, salvando quindi i risultati raggiunti durante la partita in fatto di sfide superate, metri percorsi e valute raccolte. Viene data inoltre la possibilità di riprendere la partita da dove è stata interrotta spendendo, inizialmente, un'unità di valute secondarie. Ad ogni ripresa durante lo stesso match, la quantità di valute secondarie richieste viene raddoppiata.

4.5.2.5 Restart

Sia dallo stato End sia da quello Pause è possibile raggiungere lo stato Restart. Questo è lo stato preposto all'abbandono della partita attuale e all'inizio di una nuova. Le operazioni che vengono effettuate

sono quindi quelle di salvataggio dei dati, distruzione degli elementi dinamici presenti ancora sul terreno di gioco e reset dei dati della partita che si sta abbandonando. Concluse queste operazioni, dallo stato di Restart si passa allo stato di Start.

4.5.2.6 Quit

Lo stato Quit, come Restart, è raggiungibile sia dallo stato End e sia dallo stato Pause e si occupa del reset dei dati della partita e salvataggio dei dati. La sezione Partita viene abbandonata per tornare al menù principale.

4.5.2.7 Resume

Dallo stato Pause è possibile raggiungere un ultimo stato che è quello denominato Resume. In questo stato, il giocatore attende 3 secondi prima di poter riprendere la partita che precedentemente interrotta prima di passare allo stato Pause. Allo scadere dei 3 secondi, la velocità del Match Manager, che era stata impostata a 0 in Pause, torna ad essere quella precedente all'interruzione della partita e i comandi del personaggio vengono riabilitati.

4.5.3 Personalizzazioni

Nella sezione Personalizzazioni è possibile utilizzare le valute virtuali raccolte per poter effettuare acquisti. Questa sezione è stata divisa in due parti distinte: una dedicata ai personaggi e una dedicata ai powerup.

La sezione dei personaggi carica al proprio avvio la sequenza di immagini di presentazione dei personaggi disponibili nel gioco che si possono scorrere. Ad ogni immagine è associato anche un insieme di pulsanti, la cui visualizzazione e relativo utilizzo è vincolato allo stato del personaggio. Se, infatti, il personaggio non è ancora stato acquistato dal giocatore o non è tra quelli disponibili di default, vengono visualizzati due pulsanti che permettono il suo acquisto. Il primo pulsante riguarda il suo acquisto utilizzando la valuta primaria, mentre

l'altro riguarda l'acquisto con la valuta secondaria. Le quantità richieste di valute per questi acquisti corrispondono a quelle impostate nel file JSON del personaggio. Se, invece, il personaggio dovesse essere disponibile ma non è quello impostato come quello attualmente utilizzato per giocare, viene visualizzato il pulsante di selezione. Se il personaggio è già stato acquistato ed è quello selezionato attualmente per correre, viene visualizzata un'immagine che indica la sua selezione.

La sezione delle personalizzazioni dedicata ai powerup presenta la lista di tutti i powerup disponibili nel gioco, rappresentati dalla loro immagine di presentazione e dal nome. Sotto al nome è sono presenti 4 indicatori che rappresentano fino a quale livello il powerup è stato potenziato. Per poter effettuare l'acquisto dei potenziamenti, è presente un pulsante accompagnato dalla quantità di valute primarie necessarie per effettuare l'avanzamento di livello. La quantità di valute primarie necessarie per avanzare di livello viene incrementato all'aumentare del livello di potenziamento raggiunto dal powerup.

4.5.4 Bacheca

La bacheca è la sezione preposta alla visualizzazione dei badge che si possono guadagnare o che sono stati guadagnati superando determinate sfide. Ogni badge è rappresentato dalla sua immagine di presentazione e dal suo nome. A questi viene accompagnata anche la descrizione della sfida che deve essere superata per ottenere tale badge. Se un badge è già stato vinto, viene visualizzata al suo fianco un immagine che ne indicata tale stato.

Ogni badge può essere già stato conquistato oppure la sua sfida non è ancora stata vinta. Per rendere evidente questa differenza, è stata aggiunta una spunta che si attiva solo nel momento in cui la sfida del relativo badge è stata portata a termine con successo.

Capitolo 5

Realizzazione di un caso pratico

Il generatore automatico sviluppato è stato utilizzato per la realizzazione di un prototipo di *endless runner*. Gli asset utilizzati sono stati in parte reperiti dall'Asset Store di Unity 3D e in parte messi a disposizione da Nesoperience.

Le caratteristiche di base scelte per l'*endless runner* da generare sono le seguenti:

- Scorrimento verticale.

- 2 personaggi, di cui uno utilizzabile di default e uno da acquistare.

- 2 ostacoli.

- 1 powerup.

- 5 sfide per le partite e 2 sfide per i badge.

- 2 badge associati a 2 sfide differenti.

5.1 Impostazioni di base

Le impostazioni di base riguardano l'assegnamento di un titolo al prototipo da generare e una tipologia di scorrimento. Questi assegnamenti possono essere fatti impostando il file Game.json, presente nella sottocartella JSON di Resources. Avendo deciso di creare un *endless*

runner a scorrimento verticale intitolato Test Runner, il campo "title" è stato impostato con la stringa "Test Runner", mentre il campo "scrolling" con la stringa "vertical".

I campi del file Game.json riguardanti il personaggio attualmente attivo e le sfide da superare nelle partite, invece, sono stati modificati dopo aver impostato i file Characters.json e Challenges.json.

5.2 Importazione degli asset

Le immagini di presentazione di personaggi, della valuta primaria, della valuta secondaria e del powerup sono stati importati nella sottocartella Sprites di Resources. Per ognuno di essi è stato impostato il metodo di importazione Sprite al posto di Texture, che è l'importazione di default per le immagini. L'immagine utilizzata per il terreno di gioco è stata, invece, importata nella sottocartella Textures.

Il Material della Skybox è stato importato all'interno della sottocartella Materials.

Gli effetti audio, tutti in formato mp3, sono stati importati all'interno della sottocartella Audio.

I modelli 3D di personaggi, ambienti, powerup, valuta primaria, valuta secondaria e ostacoli sono stati importati nella sottocartella Models. Dei primi modelli 3D sono stati creati i Prefab da utilizzare nel videogioco e importati nella sottocartella Prefabs. La realizzazione dei Prefab degli ostacoli, invece, ha richiesto un passaggio aggiuntivo descritto nel Paragrafo 5.2.1.

Le animazioni dei modelli 3D sono state importate nella sottocartella Animations di Resources.

5.2.1 Creazione degli ostacoli

Nella scena presente nella cartella Scenes chiamato Temp, sono presenti 3 piani di dimensioni (1, 0.1, 1) posizionati, rispettivamente, alle coordinate (0,0,0), (0,1,0) e (0,2,0). Il piano in (0,0,0) rappresenta il piano di gioco, il piano in (0,1,0) rappresenta l'altezza del personaggio quando è in modalità scivolamento, mentre il piano (0,2,0) rappresenta l'altezza del personaggio durante la corsa. Questi 3 piani fungono da

punto di riferimento per individuare con facilità le altezze in cui posizionare i Collider degli ostacoli da inserire nel gioco per ottenere il comportamento desiderato quando entrano in contatto col personaggio.

Ognuno degli ostacoli è stato aggiunto nella scena Temp e posizionato tra i 3 piani. All'ostacolo presente nella scena sono stati aggiunti come suoi figli a livello gerarchico un Cube per ogni Collider previsto per quell'ostacolo. Dopo aver scalato e posizionato opportunamente questi Cube, il loro MeshRender è stato disattivato, in modo tale da renderli invisibili durante la partita, e il campo isTrigger del loro BoxCollider è stato attivato. Infine è stato aggiunto ad ognuno di questi Cube lo script ObstacleHitCollider, presente nella sottocartella Obstacle di Scripts, che gestisce il comportamento dell'ostacolo nel momento in cui entra in contatto col personaggio.

I modelli ottenuti con questa procedura sono stati importati nella sottocartella Prefabs di Resources in modo tale da poter essere richiamati, impostati ed istanziati durante le partite.

5.2.2 Impostazione delle animazioni

Gli unici modelli 3D ad essere in possesso di animazioni sono quelli dei due personaggi. Per ognuno dei personaggi è stata creata una copia dell'AnimatorController presente nella cartella Templates chiamato CharacterController. Dopo aver rinominato opportunamente queste copie, sono state importate nella cartella Animators. Per ognuno degli AnimatorController sono state impostate le animazioni dei personaggi trascinandole all'interno degli opportuni stati di cui l'AnimatorController è composto.

5.3 Assegnazione dei ruoli

In questa fase, ad ogni effetto audio, Prefab, immagine e AnimatorController è stato assegnato un ruolo all'interno del videogioco. Ad esempio, il ruolo di un'immagine può essere quello di venir utilizzata come immagine di presentazione di un personaggio. Per impostare i ruoli è sufficiente indicare i nomi dei file importati all'interno degli appositi campi dei file JSON. Il generatore automatico cerca all'interno

delle sottocartelle di Resources i file indicati nei file JSON e li utilizza secondo il ruolo assegnatogli.

In questo prototipo è previsto che vi siano 2 personaggi. Nel file Characters.json sono quindi state definite 2 descrizioni di personaggi, separati da una virgola. Per poter utilizzare le informazioni riguardanti il primo personaggio, l'indice di riferimento è lo 0, mentre per accedere al secondo è l'1. Il primo personaggio definito nel file JSON è quello di default, ovvero quello utilizzabile sin dal primo accesso al videogioco. Per questo motivo non è necessario definire alcun valore per poterlo acquistare in gioco e il campo "is_owned", che ne indica il possesso da parte del giocatore, è stato impostato col valore "true". Il secondo personaggio, invece, deve essere acquistato prima di poterlo utilizzare nel gioco e quindi il valore del campo "is_owned" è stato impostato a "false". Il valore di acquisto del secondo personaggio utilizzando la prima valuta è stato impostato a 100, mentre il valore di acquisto utilizzando la seconda valuta è stato impostato a 5. I campi riguardanti le immagini di presentazione, gli effetti audio e l'AnimatorController sono stati impostati riportando i nomi dei file importati nelle rispettive sottocartelle di Resources.

Gli ostacoli presenti in questo endless runner sono 2 e la loro impostazione rispecchia quella utilizzata per i personaggi per quanto riguarda i nomi, gli effetti audio, l'immagine di presentazione e l'AnimatorController.

La procedura di impostazione del powerup segue la stessa utilizzata per gli ostacoli. E' stato inoltre necessario impostare la probabilità con cui il powerup può presentarsi durante il caricamento in partita di una composizione a cui è stato assegnato. Il livello del powerup è stato impostato a 1, mentre il valore di acquisto con valuta virtuale primaria iniziale per poterlo potenziare al livello 2 è stato impostato a 500. Infine è stata scelta la tipologia di powerup impostandola col valore "Vehicle".

Anche la procedura di impostazione della valuta virtuale primaria segue quella utilizzata per l'impostazione degli ostacoli. A differenza di tutti gli altri elementi di gioco, però, la valuta primaria si presenta in pattern, ovvero in gruppi ordinati. Nel file Coins.json sono quindi state impostate 2 tipologie di pattern, entrambe composte da 3 unità di valu-

te primarie, con l'unica differenza che nella prima queste si presentano alla stessa altezza, mentre nella seconda sono ad altezza crescente.

Anche l'impostazione della valuta secondaria segue la procedura utilizzata per l'impostazione degli ostacoli con, in aggiunta, la probabilità con cui questa può presentarsi al caricamento in gioco di una composizione che ne prevede la presenza.

L'impostazione dell'ambiente ha seguito una procedura diversa. Sono stati indicati il nome del Material utilizzato per la Skybox nel campo "skybox" e il nome della Texture da applicare sul terreno nel campo "road_texture". L'impostazione dell'ambiente si è conclusa con la definizione di un unico modello 3D da mostrare ai lati del terreno di gioco durante le partite. Dopo aver indicato il nome, il modello 3D di riferimento e gli eventuali audio e AnimatorController, sono state definite le distanza che questo modello deve avere dal centro della scena e quale è la sua lunghezza. Questi valori sono indispensabili per poter garantire una distanza appropriata dal terreno di gioco e poter controllare la distanza tra un modello 3D dell'ambiente generato e l'altro. Questi valori sono stati ottenuti dopo aver effettuato alcune partite di prova.

Infine sono state definite le composizioni, ovvero l'insieme composto da ostacolo, powerup, pattern di valuta primaria e valuta secondaria che vengono generati contemporaneamente e vengono presentati al personaggio durante la corsa. Per ognuno di questi elementi, sono stati impostati gli indici della posizione che occupano nei rispettivi file JSON e la loro altezza dal terreno, la *lane* su cui devono trovarsi e la distanza rispetto al centro della composizione. Sono state definite due tipologie di composizioni.

5.4 Sfide e badge

Nel file Challenge.json sono state impostate 5 sfide da superare in partita e 2 sfide assegnabili ai badge. Sono state impostate 4 sfide di tipo MatchMeters e 3 di tipo MatchCoins. I campi riguardanti i valori minimi richiesti per considerare superata la sfida sono stati impostati con quantità diverse da sfida a sfida.

Le due sfide riguardanti i badge sono state impostate a 2 diversi badge ai quali sono stati anche associati un nome e un'immagine di presentazione importata precedentemente nella sottocartella Sprites di Resources.

Con limpostazione delle sfide e dei badge, si conclude la fase di modifica dei file JSON preposti allassegnazione dei ruoli e descrizione del gioco.

5.5 UI

L'interfaccia grafica è stata modificata solo a livello dei colori. Per far ciò è stato sufficiente agire direttamente sui Prefabs degli elementi grafici creati nella cartella UI, variandone appunto i colori.

Nelle immagini che seguono, vengono mostrate alcune schermate del videogioco generato con la procedura descritta.

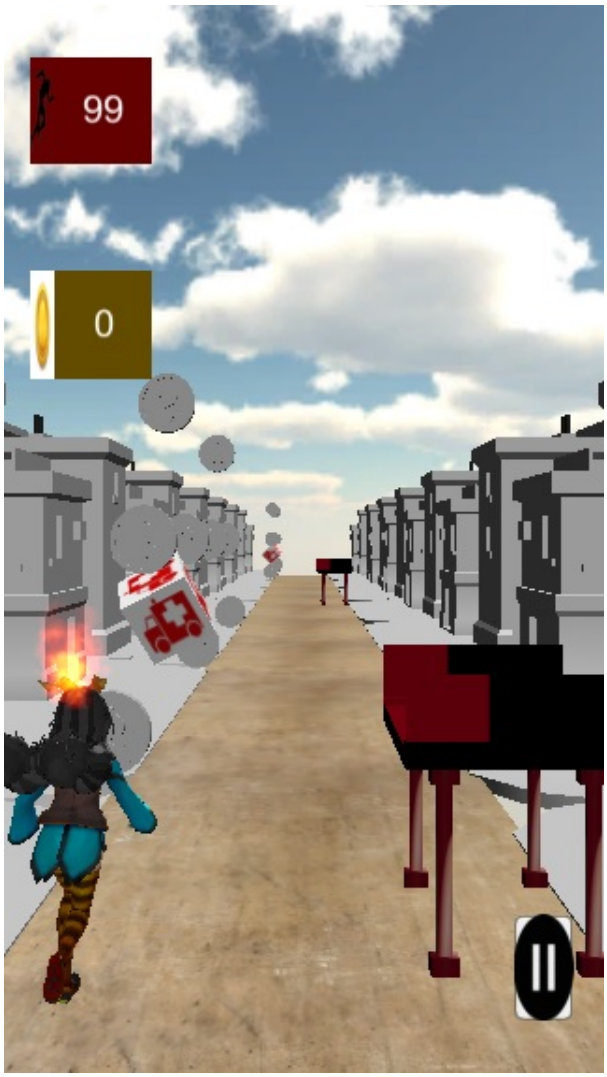

Figura 5.1: Schermata durante una partita con l'enedless runner generato

Figura 5.2: Schermata principale dell'enedless runner generato

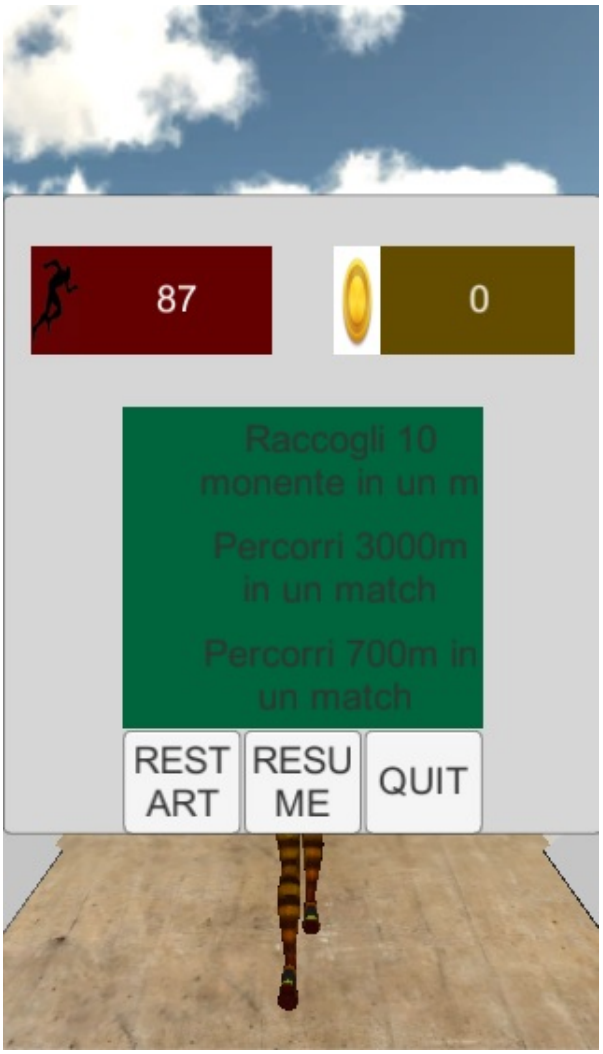

Figura 5.3: Schermata di pausa dell'enedless runner generato

Figura 5.4: Schermata dopo aver collisono con un ostacolo dell'enedless runner generato

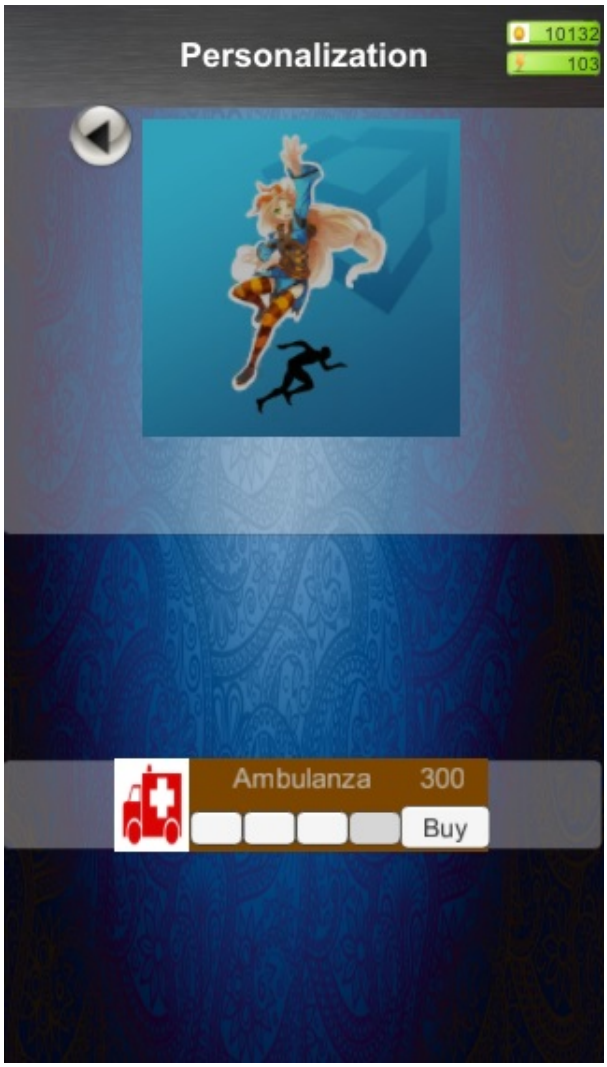

Figura 5.5: Schermata di personalizzazione dell'enedless runner generato

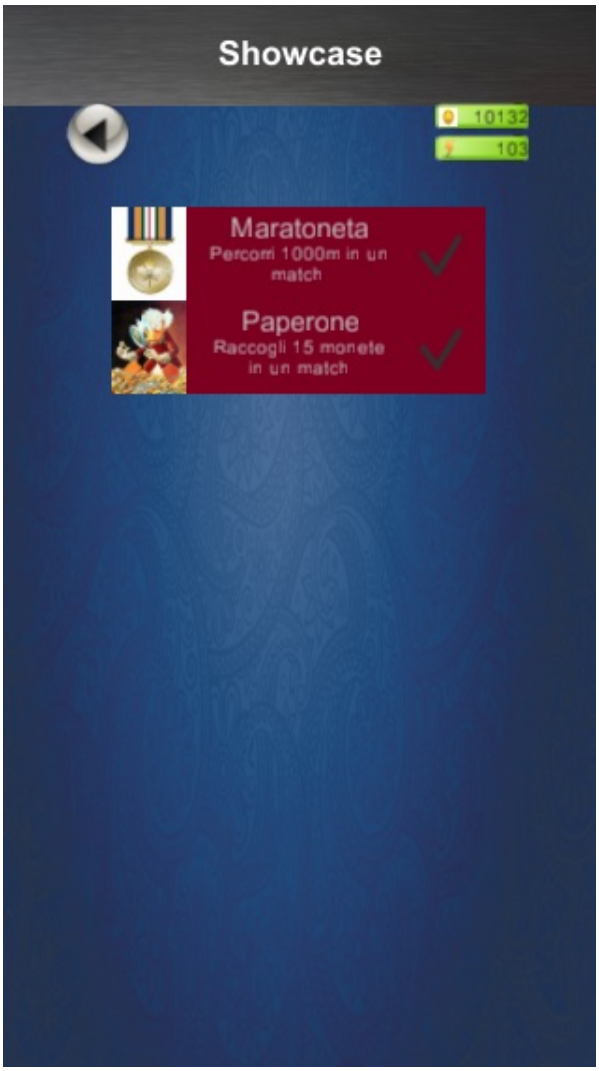

Figura 5.6: Schermata di visualizzazione dei badge dell'enedless runner generato

Capitolo 6

Conclusioni e sviluppi futuri

Il generatore sviluppato permette di creare un *endless runner* senza dover scrivere alcuna riga di codice ma semplicemente importando gli asset desiderati e configurandoli attraverso i file JSON preposti a questo compito.

Il generatore rende quindi superflua tutta la fase di definizione, progettazione e sviluppo di meccaniche degli elementi caratteristici di un *endless runner* in quanto già definite all'interno del generatore. Grazie a queste caratteristiche, il generatore è in grado di ridurre sensibilmente il tempo di sviluppo di un videogioco di questa tipologia ma, al contempo, di renderlo altamente personalizzabile a livello dei contenuti in base alle particolari esigenze dei brand che ne fanno richiesta.

Questo generatore ha come scopo sia quello di generare in maniera rapida *endless runner* sia quello di dare la possibilità di creare nuove meccaniche di gioco che arricchiscano quelle giá fornite dal generatore. Il codice del generatore permette di poter cambiare agevolmente i comandi dei personaggi e powerup agendo direttamente sugli script.

Un possibile sviluppo è quello di sostituire le meccaniche implementate per un tipico personaggio che si muove su un terreno con quelle di un personaggio volante. Per realizzare cambiamenti come questo o simili è sufficiente cambiare il solo script di controllo del personaggio per ottenere il risultato desiderato.

Per rendere ancora più competitivi i titoli creati con questo sistema, è possibile aggiungere servizi social, come il collegamento a Facebook,

per poter rendere partecipi anche i contatti dei giocatori dell'esistenza del gioco e delle sue performance. Questo darebbe la possibilità di rendere i videogiochi prodotti col generatore appetibili anche per altre categorie di giocatori e di creare quindi una community di cui beneficerebbe il gioco stesso e il *brand* per il quale è stato realizzato.

L'introduzione di servizi social permetterebbe anche di introdurre nuovi elementi, come la possibilità di scambiarsi valute tra giocatori con cui si è collegati.

Bibliografia

[1] *Most popular Apple App Store categories in March 2015*, 2015, http://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

[2] *Dont Stop: The Game That Conquered Smartphones*, 2013, http://www.newyorker.com/tech/elements/dont-stop-the-game-that-conquered-smartphones

[3] *7 Innovative Mobile Gaming Opportunities for Brands*, 2011, http://www.forbes.com/sites/michaelmatthews/2011/09/13/7-innovative-mobile-gaming-opportunities-for-brands/

[4] *Endless Runner Apps*, https://www.commonsensemedia.org/lists/endless-runner-apps

[5] *Advergames: The New Mobile Advertising Trend in 2015*, 2015, http://aointeractive.co.za/advergames-new-mobile-advertising-trend-2015/

[6] *The formal systems of games and game design atoms*, 2014, http://www.acagamic.com/courses/infr1330-2014/the-formal-systems-of-games-and-game-design-atoms/

[7] *Platform Game*, http://en.wikipedia.org/wiki/Platform_game

[8] *Fotonoica*, http://www.fotonica-game.com/

[9] *Temple Run 2 Review*, 2013, http://www.eurogamer.net/articles/2013-01-21-temple-run-2-review

[10] *Experimental Gameplay Project*, http://experimentalgameplay.com/blog/about/

[11] *The Games Of Christmas: December 2nd*, 2009, http://www.rockpapershotgun.com/2009/12/02/the-games-of-christmas-december-2nd/

[12] *Games of 2009: Canabalt*, 2009, http://www.eurogamer.net/articles/games-of-2009-canabalt-article

[13] *Robot Unicorn Attack*, http://en.wikipedia.org/wiki/Robot_Unicorn_Attack

[14] *'Temple Run' has been downloaded 1 billion times, and most players are women*, 2014, http://www.theverge.com/2014/6/4/5776232/temple-run-1-billion-downloads

[15] *Kiloo's Subway Surfers Making Mobile Waves*, 2013, http://www.forbes.com/sites/danieltack/2013/03/07/kiloos-subway-surfers-making-mobile-waves/

[16] *Over Sixty 'Flappy Bird' Clones Hit Apple's App Store Every Single Day*, 2014, http://www.forbes.com/sites/insertcoin/2014/03/06/over-sixty-flappy-bird-clones-hit-apples-app-store-every-single-day/

[17] *How to Make a no. 1 App with $99 and Three Hours of Work*, 2014, http://www.wired.com/2014/03/flappy-bird-clones/

[18] *Tiny Flying Drizzy Cheats and Tips*, 2014, http://appcheaters.com/tiny-flying-drizzy-cheats-and-tips/

[19] *Code your own Flappy Game*, 2014, http://studio.code.org/flappy/1