Alma Mater Studiorum Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

MecWilly3D 2.0: riscrittura ed ottimizzazione di un serious game in Unity3D per dispositivi mobili

Relatore: Chiar.mo.
Prof. Stefano Ferretti

Correlatore: Chiar.ma
Prof. Silvia Mirri

Presentata da:
Matteo Muratore

Sessione III
Anno Accademico 2016/2017

A Luca, Mamma, Papà e ai miei nonni che non

hanno mai smesso di credere in me.

Indice

Introduzione

Le applicazioni per smartphone e tablet e la robotica sono ormai parte integrante della società e della cultura degli ultimi anni e, applicati nello stesso environment, possono essere utili per svariati compiti tra cui lo sviluppo e il potenziamento delle abilità socio-cognitive nei bambini.

A partire dagli ultimi anni della scuola dell'infanzia il bambino è stimolato dalla scoperta e dalla collocazione topografica degli oggetti, con un occhio di riguardo alla visione degli stessi rispetto ad un determinato punto focale (destra, sinistra).

La difficoltà che riscontra il bambino è proprio quella del riuscire a distinguere la posizione di un oggetto osservato da una prospettiva diversa dalla propria. Inoltre, a quell'età è difficile riuscire a mettersi nei panni di un'altra persona o di un altro oggetto, riuscendo quindi a modificare la propria prospettiva in favore di una diversa dalla propria.

Per questo motivo l'utilizzo delle ICT e della robotica consentono di offrire molteplici soluzioni per il supporto e la gestione di queste abilità di orientamento topografico a partire già dai 5 anni per indurre il bambino ad un conflitto socio-cognitivo (tramite il rapporto con un altro soggetto).

MecWilly, un robot umanoide alto 1 metro e 20 cm, realizzato con materiali di recupero, grazie alla collaborazione con il dipartimento di Psicologia dell'Università di Bologna e il Comune di Rimini, viene utilizzato proprio per studiare il conflitto socio-cognitivo nei bambini in età prescolare e scolare. Riesce a riconoscere, tramite apposite fotocamere situate in corrispondenza degli occhi, le

persone dai tratti del viso, per poi seguirle con lo sguardo e focalizzarsi su specifici punti dell'ambiente limitrofo.
Lo scopo del bambino è quello di impartire degli ordini prestabiliti a MecWilly in modo da raggiungere un obiettivo prefissato. Viene utilizzato in particolare nei progetti di supporto per i bambini e nell'educazione alla raccolta differenziata e al riciclo dei rifiuti.
In questo progetto MecWilly ha in mano un oggetto e si muove in una griglia di gioco (in cui sono situati dei bidoni per la raccolta differenziata) fino ad arrivare al bidone corretto, in modo da gettare l'oggetto al suo interno. L'esperimento consiste nel valutare il numero totale di mosse per raggiungere l'obiettivo e il tempo impiegato.
In passato sono stati sviluppati due videogiochi con lo scopo di verificare l'esistenza di una differenza tra l'ambiente reale e un ambiente virtuale in termini di risultati.
La prima versione sviluppata è stata un videogioco bidimensionale per tablet: questa versione però non riproduceva fedelmente l'esperimento originale, svolto in un ambiente reale (una classe della scuola primaria), poiché la telecamera di gioco era situata sopra il labirinto e non di fronte al robot, impedendo al bambino di immedesimarsi pienamente in un contesto più affine a quello reale (impedendo quindi il realizzarsi del conflitto socio-cognitivo).
La seconda versione sviluppata è stata un videogioco in 3D che risolveva il problema principale della prima versione, ponendo la prospettiva del giocatore di fronte al robot e avvicinandosi alla realizzazione di un ambiente virtuale molto simile a quello reale.
L'obiettivo di questa tesi è aggiornare la seconda versione del gioco riscrivendo il nucleo del codice sorgente, migliorando l'ottimizzazione di esso per smartphone e tablet, aggiungendo nuove feature quali l'audio (dando ascolto ai feedback della scorsa

versione e avvicinandoci ancor di più all'ambiente reale), un level editor (in modo da rendere possibile all'insegnante la creazione di livelli direttamente dall'applicazione senza dover toccare il codice sorgente) e una classifica per livello (in modo da invogliare il giocatore a mettersi in competizione con gli altri bambini); inoltre si tenterà di perfezionare la grafica generale del gioco ridisegnando i modelli utilizzati. Infine, verrà effettuato un paragone tra la versione precedente e quella appena sviluppata, seguendo il percorso tracciato dal lavoro svolto dal developer della scorsa versione [1].

CAPITOLO 1

Videogiochi ed educazione

La “net generation”, grazie ad un più accessibile benessere economico e alla crescita sia finanziaria che di notorietà delle aziende videoludiche, ha potuto usufruire sin da dalla gioventù dei prodotti sviluppati da esse. Questo ha dato adito alla possibilità di godere di articoli quali gli Educational Games, specialmente nel mercato dei dispositivi mobili, ormai a disposizione della maggior parte dei giovani. Al giorno d’oggi le applicazioni per l’apprendimento create considerando un target di bambini della scuola primaria contengono elementi di gaming o di gamification [2].

1.1 Edutainment

Educazione e al contempo divertimento sono le basi del cosiddetto edutainment (parola composta da “education” ed “entertainment”), termine coniato da Bob Heyman durante il suo lavoro presso National Geographic; questo concetto, inizialmente rivolto ad un contesto ludico ma didattico, si è poi esteso a tutto ciò che può essere considerato produttivo e costruttivo attraverso il gioco (ad

esempio indottrinando su tematiche delicate quali l'etica, il sesso, le sostanze stupefacenti).

1.2 Serious game

Con serious game ci si riferisce ai cosiddetti "giochi seri", in cui il contenuto di intrattenimento viene meno rispetto a quello educativo.

Similmente agli edutainment l'obiettivo della sensibilizzazione su tematiche importanti è prioritario, affiancato anche da attività promozionali e campagne sociali, nonostante l'origine di questa forma ludica sia da ricondursi a simulazioni di guerra o giochi da tavolo dei secoli scorsi (quali "Kriegsspiel" o "Monopoly").
L'utilizzo di questi giochi varia in base ai contesti: da un punto di vista commerciale vengono utilizzati per simulare vendite sia telefoniche che porta a porta, colloqui o situazioni di rapporti diretti; in campo militare invece possono essere usati per l'addestramento del personale o per simulazioni belliche (come i war game). A parte gli utilizzi citati sopra, si è pensato di sfruttare questo tipo di giochi per contesti più quotidiani come la formazione generale dell'individuo.

1.3 GWAP

Un GWAP (Game With A Purpose) consiste in un gioco multiplayer online progettato per divertire ma al contempo compiere operazioni che sono semplici da svolgere per l'essere umano, ma non altrettanto

per i computer odierni. Ideati dal professor Luis Von Ahn, i GWAP "[…] aiutano a migliorare le ricerche online di immagini e audio, intensificando l'intelligenza artificiale e insegnando ai computer ad osservare, ma tutto ciò non interessa al giocatore poiché egli scopre che questi giochi sono molto divertenti." [3]
Tra le attività naturali per l'essere umano ma complesse per un computer, possono rientrare categorie come risoluzione di rompicapi, riconoscimento di immagini e così via; queste vengono inserite in un contesto di gamification in modo da non risultare noiose.

I GWAP mostrano generalmente queste caratteristiche:

- Timed response: l'obiettivo del gioco è stimolare l'utente a superare le prove entro un determinato limite di tempo per migliorare la competizione personale e con altri giocatori.
- Score Keeping: il giocatore è invogliato a proseguire nel livello grazie al conseguimento di obiettivi e all'assegnazione di punti.
- Player skill levels. Completando obiettivi e guadagnano punti, il giocatore può salire di livello in modo da essere invogliato a continuare il proprio percorso verso la fine del gioco.
- High-score lists. Classifiche di diverso tipo che attestano i risultati ottenuti dal giocatore durante le prove lo mettono in confronto altri utenti da tutto il mondo.
- Randomness. Il fine della casualità è quello di rendere unica una partita (come la scelta dello sfidante o le prove da sostenere) e evitare comportamenti scorretti (come il cheating); inoltre l'incertezza circa il proseguimento della

partita invoglia il giocatore a continuare a giocare.

Alcuni esempi di GWAP sono ESP game (che consiste nel riconoscimento delle etichette assegnate ad un'immagine da uno dei due giocatori scelti randomicamente) e Peekaboom (che sfrutta i dati ottenuti da ESP game per determinare la precisa posizione degli oggetti nell'immagine in un confronto tra due giocatori).

1.4 Gamification

Lo scopo della gamification è prendere in prestito elementi di giochi per coinvolgere maggiormente l'utente in contesti che nella vita quotidiana risulterebbero noiosi e poco divertenti.

1.4.1 Elementi base della gamification

Nonostante la continua evoluzione di meccaniche di gioco è possibile riscontrare comunque elementi caratteristici della gamification:

- Punti: guadagnare punti nel corso del gioco aumenta il coinvolgimento dell'utente, che può utilizzarli per ricompense, oggetti di gioco o premi. In questo modo il giocatore si convince di spendere in maniera fruttuosa il proprio tempo per guadagnare più punti.

- Livelli: col procedere del gioco e l’accumulo di punti il giocatore può avanzare di grado in una scala gerarchica ed essere stimolato a impegnarsi sempre di più.
- Beni virtuali: L’acquisto di beni virtuali, con valuta reale o fittizia, consente al giocatore di ottenere oggetti in-game elitari per distinguersi dagli utenti.
- Classifiche: la competizione è un elemento della gamification che invoglia il giocatore a raggiungere i primi posti nelle classifiche generali.

CAPITOLO 2

Specifiche di progetto

2.1 MecWilly

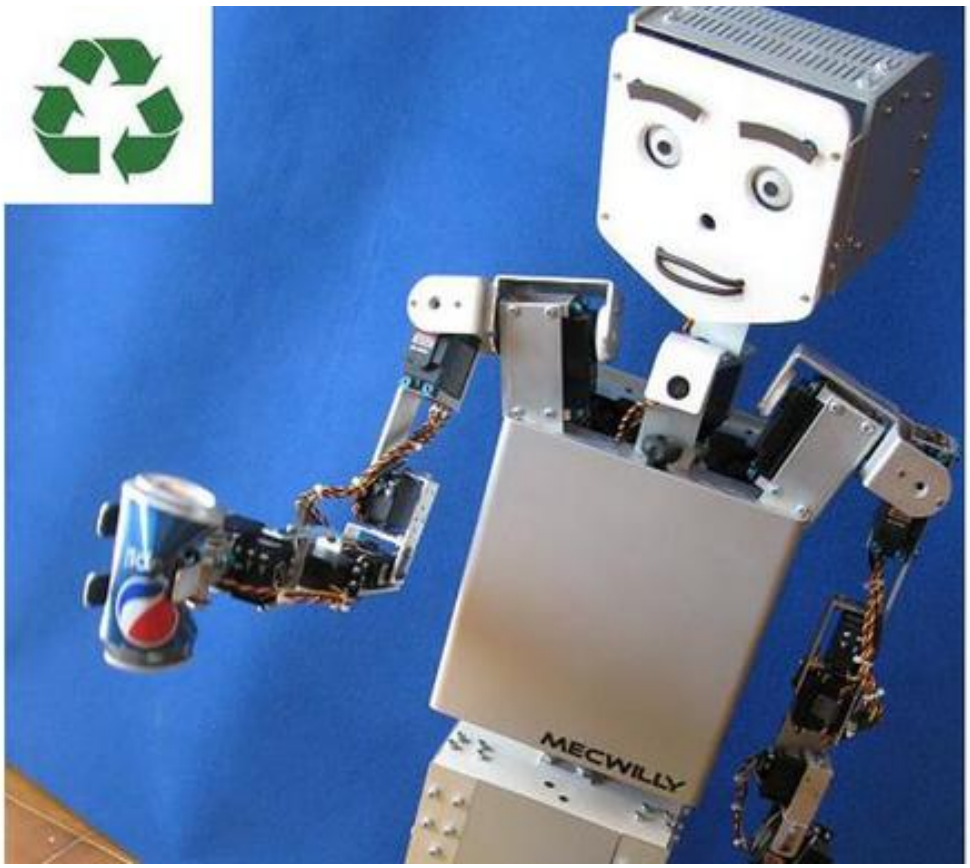

Figura 2.1: MecWilly nella realtà

2.1.1 Chi è MecWilly?

MecWilly è un robot umanoide alto 1 metro e 20 cm, realizzato da Roberto Masini con materiali economici e di recupero, grazie alla collaborazione con il Dipartimento di Psicologia dell'Università di Bologna e il Comune di Rimini. Dotato di apposite fotocamere posizionate in corrispondenza degli occhi, il robot è in grado di riconoscere i tratti del viso delle persone riuscendo a seguirle con lo sguardo, di capire se gli si sta sorridendo e di concentrare l'attenzione su particolari zone dell'ambiente circostante. Grazie ad una sintesi vocale è in grado di dialogare autonomamente con le persone su argomenti generici e può essere addestrato su specifici argomenti conservando comunque la sua capacità di rispondere a qualunque richiesta generica. MecWilly è wireless e connesso ad Internet ed è possibile chiedergli di prelevare in tempo reale informazioni quali previsioni del tempo, ultime notizie o qualunque altra informazione utile. Funziona a batteria con un'autonomia di circa 4 ore ed è possibile ricaricarla tramite rete elettrica (senza interrompere la propria esecuzione). Nella nuova versione è provvisto di un touch screen sul retro che permette di tenere sotto controllo il sistema e di gestire le impostazioni [4].

2.1.2 Utilizzo di MecWilly

Background teorico e metodologico

Le applicazioni per smartphone e tablet e la robotica sono ormai parte integrante della società e della cultura degli ultimi anni e,

applicati nello stesso environment, possono essere utili per svariati compiti tra cui lo sviluppo e il potenziamento delle abilità socio-cognitive nei bambini [5] [6].
A partire dagli ultimi anni della scuola dell'infanzia una delle abilità più importanti ad essere scoperte ed immagazzinate dal bambino è l'orientamento topografico, con un occhio di riguardo alla visione degli oggetti rispetto ad un determinato punto facendo attenzione alle posizioni (destra, sinistra) relative all'oggetto.
La difficoltà che riscontra il bambino è proprio quella del riuscire a distinguere la posizione di un oggetto osservato da una prospettiva diversa dalla propria. Tali posizioni infatti non sono assolute, bensì sono relative rispetto ad una determinata posizione. Inoltre, a quell'età è difficile riuscire a mettersi nei panni di un'altra persona o di un altro oggetto, riuscendo quindi a modificare la propria prospettiva in favore di una diversa dalla propria.
A questo punto entrano in gioco l'utilizzo delle ICT e della robotica che riescono ad offrire molteplici soluzioni per supportare al meglio l'acquisizione e la gestione di queste abilità di orientamento topografico a partire già dai 5 anni.

Conflitto socio-cognitivo

Molto si è detto e scritto negli ultimi anni sull’importanza del computer nello sviluppo cognitivo del bambino e molto si è detto circa i vantaggi e gli svantaggi, i pericoli dell’utilizzo del computer in ambito educativo.
Il computer oltre ad essere un “magazzino di informazioni” è anche capace di presentare situazioni di “problem-solving” strumento efficace di apprendimento. Cioè il bambino scompone un problema,

sperimenta una soluzione e, raggiuntala, riceve un feedback immediato, così che la soluzione del problema funge da rinforzo positivo e stimola verso ulteriori apprendimenti. [7]

Le nuove teorie cognitive dimostrano come l'apprendimento non avviene attraverso semplici conoscenze mnemoniche bensì attraverso l'acquisizione di procedure che favoriscono lo sviluppo intellettivo, si può pensare ad una sorta di "intelligenza in azione". Quindi un giusto approccio didattico è "imparare facendo" e, soprattutto, farlo insieme agli altri, cioè "imparare cooperando".

Il rapporto tra pari rappresenta un'opportunità e uno strumento per sollecitare un maggiore sviluppo intellettivo attraverso meccanismi di conflitti socio-cognitivi, innescati dalla scoperta-scambio di prospettive diverse dalla propria che producono "squilibri". Confronti e conflitti interindividuali tra diverse soluzioni conducono i bambini a coordinare i loro punti di vista in un nuovo sistema su cui giungono ad accordarsi attraverso il superamento delle singole posizioni, con una negoziazione che porta ad una decisione condivisa. [8]

L'interazione sociale qui è vista principalmente come uno strumento per accelerare lo sviluppo delle strutture dell'intelligenza, ma anche, sottolinea la psicologia sociale, come il superamento del lavoro individuale con una forte valenza comunicativa e sociale. Essa porta ad aiutarsi reciprocamente, all'autonomia, alla corresponsabilità del risultato-apprendimento, migliora le relazioni sociali e il livello di autostima individuale. [9]

Scopo di questa trattazione è mettere in evidenza l'importanza che possono assumere i videogiochi nell'ambito di una didattica attiva ove il bambino, con i suoi tempi e i suoi modi, sperimenta il valore dell'esperienza concreta e aiuta il docente a svolgere un curricolo fondato sulle procedure e non sulle semplici conoscenze.

In questo contesto è stato sviluppato MecWilly.

2.1.3 Esperimenti software precedenti

Così come un compagno o tutore può supportare e aiutare il bambino a raggiungere un obiettivo o un risultato che non riuscirebbe a raggiungere individualmente, la tecnologia può sostituire l'uomo per questo compito, riuscendo a portare un bambino ad agire all'interno della propria zona di sviluppo prossimale [10] per arrivare a risultati cui non arriverebbe da solo.
Sono state realizzate due versioni in precedenza per riuscire a rappresentare in un ambiente virtuale ciò che accade nell'ambiente reale [11].
La prima versione sviluppata è stata un videogioco bidimensionale per tablet in cui era possibile muovere MecWilly all'interno di un labirinto per raggiungere il bidone corretto dove poter buttare il rifiuto tenuto in mano dal robot.
Il problema di questa versione è la prospettiva del giocatore rispetto a MecWilly dato che la telecamera nel gioco era situata sopra il labirinto e non di fronte al robot, non riuscendo a riprodurre fedelmente l'esperimento originale svolto in una classe della scuola primaria (dunque in un ambiente reale). In questo modo venivano meno alcuni aspetti fondamentali del conflitto socio-cognitivo (creato dal posizionamento frontale del bambino rispetto al robot).
La seconda versione sviluppata è stata un videogioco tridimensionale per tablet. Il fine di questa versione era ovviare al problema del videogioco 2D ponendo la prospettiva del giocatore di fronte al robot, posizionando la telecamera del gioco di fronte ad esso. In questo modo l'esperimento è stato riprodotto in maniera più

fedele riuscendo a generare il conflitto socio-cognitivo generato dall'esperimento effettuato in ambiente reale.
Una feature importante non replicata nella precedente versione è stata la mancanza dei suoni riprodotti da MecWilly nella realtà, aspetto pesantemente criticato dai bambini durante i test effettuati in una scuola elementare di Cesena [1].
Un'altra lacuna tra le feature del gioco è l'impossibilità da parte dell'insegnante o del bambino di poter creare nuovi livelli di gioco (feature invece esistente nell'ambiente reale).
Per quanto riguarda il codice scritto per questa versione tridimensionale esso presenta dei gravi problemi a livello di ottimizzazione (non sono state utilizzate alcune tecniche fondamentali per l'ottimizzazione su tablet e smartphone).
Ponendo la telecamera esattamente di fronte al robot viene risolto il problema principale del conflitto socio-cognitivo, ma va a penalizzare il comparto grafico (riuscendo a vedere bene solo il robot e pochi bidoni vicino ad esso, andando a tagliare alcune parti della griglia di gioco).
La seconda versione di questa applicazione 3D vuole risolvere tutti questi aspetti riscrivendo il codice sorgente in modo da avere un occhio di riguardo all'ottimizzazione e in modo da aggiungere tutte le feature mancanti. Inoltre, la telecamera avrà proiezione ortografica, cosicché la dimensione degli oggetti non venga influenzata dalla distanza dalla telecamera, in modo da non perdere d'occhio l'obiettivo originale (generare il conflitto) e così da rendere visibile tutta la griglia di gioco come nell'ambiente reale. In questo modo si riesce a riprodurre ancor più fedelmente l'esperimento originale.

2.2 MecWilly3D 2.0

L'applicazione MecWilly3D 2.0 ha la finalità di aggiornare e ottimizzare per dispositivi mobili il videogioco MecWilly3D, creato per sviluppare e migliorare le abilità nell'orientamento topografico e spaziale dei bambini della scuola primaria, oltre che per voler educare il bambino alla raccolta differenziata.
Il progetto è la riscrittura del videogioco 3D originale, in modo da risolvere tutti i problemi della prima versione di MecWilly3D e aggiungere alcune feature mancanti nella versione precedente.

2.3 Modellazione 3D

La modellazione tridimensionale è il processo di sviluppo di una rappresentazione matematica di qualunque superfice di un oggetto (sia animato che inanimato) in tre dimensioni attraverso software specializzati. Il risultato di questo processo è chiamato "modello 3D".
Il modello 3D è interpretato dall'hardware tramite la sua topologia, cioè l'insieme di vertici, lati e facce atti a determinarne la forma.
La modellazione 3D può essere usata per la creazione di oggetti 3D utilizzabili per diversi fini. Il fine determina l'approccio da utilizzare nella modellazione. In caso di modelli 3D destinati alla stampa è richiesta una particolare attenzione allo sviluppo della topologia del modello che deve essere "reale", cioè non presentare caratteristiche non replicabili nella realtà (quali ad esempio intersezioni di lati o sovrapposizioni di vertici).

In caso di modelli destinati ad un utilizzo rappresentativo del prodotto (Concept Art) può venir meno la cura della topologia a favore di un approccio più semplice per lo sviluppo dei dettagli.
Nel caso dello sviluppo di un videogioco è necessario tenere in considerazione entrambi gli aspetti.

2.3.1 Scelta del software di modellazione

Per modellare gli oggetti da utilizzare nel gioco sono stati considerati due software di modellazione, il primo totalmente gratuito e open source, l'altro a pagamento:

- Blender: disponibile per vari sistemi operativi come Microsoft Windows, macOS, GNU/Linux e FreeBSD, si propone come alternativa open source e gratuita per la modellazione, il rigging, l'animazione, il compositing e il rendering di immagini tridimensionali.

- Autodesk Maya: disponibile tramite un modello di sottoscrizione a pagamento per sistemi operativi quali Microsoft Windows, macOS e GNU/Linux, è una delle applicazioni di modellazione grafica tridimensionale più utilizzate dall'industria videoludica e cinematografica.

Data la natura e la complessità del progetto si è optato per l'utilizzo di Blender, nonostante la controparte a pagamento abbia a disposizione una UI migliore e delle funzionalità avanzate semplificate rispetto a Blender.

2.4 Requisiti di un modello per un videogioco

Un modello sviluppato per un videogioco non deve avere una topologia necessariamente reale tuttavia deve presentare una topologia "pulita", cioè composta prevalentemente di quadrati o triangoli e un corretto utilizzo degli Edge-Loop. Il rispetto di queste regole permette di avere un modello facile da animare o deformare e facile da sviluppare in 2D per l'applicazione delle texture. È al contempo importante che il modello presenti il minor numero di poligoni possibile per raggiungere l'equilibrio tra una piacevole resa grafica e un'ottimizzazione adeguata. Al fine di ottenere ciò il workflow si divide in "concept modeling" e "production modeling".

2.5 Concept Modeling VS Production Modeling

Come esposto sopra, un modello da utilizzare in un videogioco deve rispondere prima di tutto a requisiti di ottimizzazione, ciò tuttavia non deve andare a discapito della resa estetica del modello.
Il workflow attualmente adottato nell'industria prevede lo sviluppo di un modello High-Poly (Concept Modeling), con un elevato numero di dettagli (ma non utilizzabile in un engine) e di un modello Low-Poly (Production Modeling) utilizzabile in un engine.
Il modello High-Poly presenta dettagli quali ad esempio, viti, crepe, pieghe, che vengono resi tramite lo sviluppo di geometrie, dunque

accrescono il conto totale dei poligoni. Questi dettagli, per quanto appaganti per l'occhio, rischiano di influire sull'ottimizzazione complessiva del prodotto finale. A questo riguardo si è sviluppato l'utilizzo di tecniche per salvare questi dettagli in texture speciali (ad esempio le normal map) che, interpretate dall'engine, riescono a riprodurli anche sul modello Low-Poly, dunque in assenza di vera geometria ma semplicemente cambiando il modo in cui la luce interagisce col modello (figura 2.2).

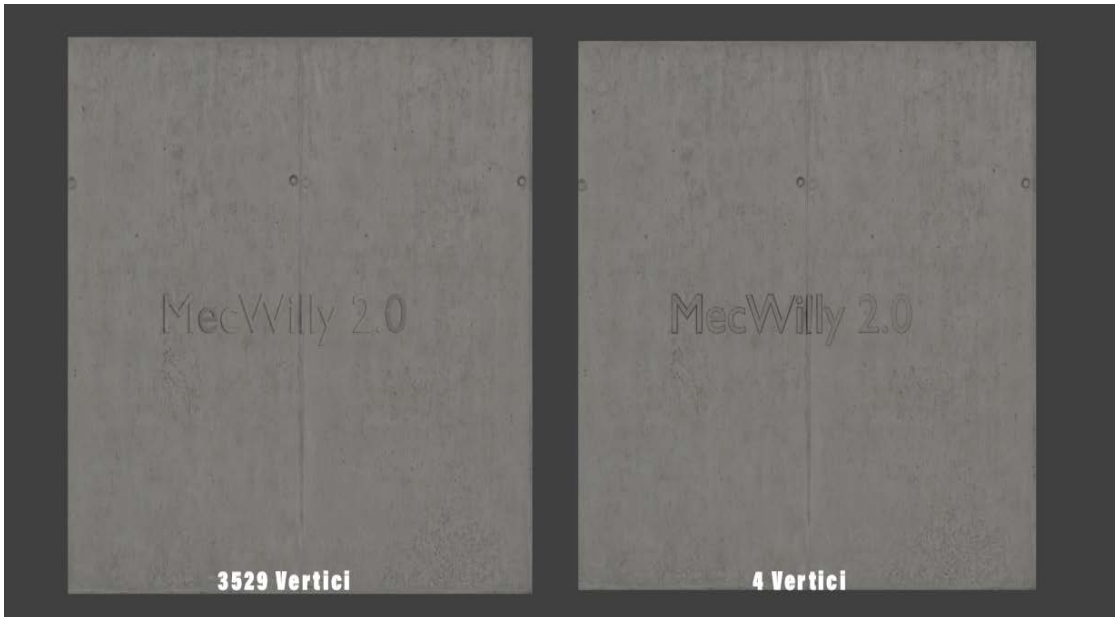

Figura 2.2: Modelli High-Poly e Low-Poly a confronto

2.6 Definizione di un motore grafico

Il motore grafico è il nucleo software di un videogioco o di qualsiasi altra applicazione con grafica in tempo reale. Esso fornisce le tecnologie di base, semplifica lo sviluppo, e spesso permette al gioco di funzionare su piattaforme differenti come le console o sistemi operativi per personal computer. La funzionalità di base fornita tipicamente da un motore grafico include un motore di

rendering ("renderer") per grafica 2D e 3D, un motore fisico o rilevatore di collisioni, suono, scripting, animazioni, intelligenza artificiale, networking, e scene-graph.
Già dagli anni ottanta le software house necessitavano di un software che semplificasse la realizzazione di videogiochi più o meno simili (come ad esempio lo SCUMM di LucasArts che semplificava la realizzazione delle sue avventure grafiche, a discapito di una sensazione di déjà-vu nei comandi e nella UI dei propri giochi), ma è dagli anni novanta, specialmente con la nascita e la popolarità acquisita dagli sparatutto in 3D quali Doom e Quake, che le aziende preferivano acquistare motori grafici di terze parti per lo sviluppo di un proprio videogioco piuttosto che creare tutto da zero.
Negli anni questa crescente popolarità ha fatto sì che più aziende sviluppassero motori grafici pensati per un uso personale piuttosto che commerciale, distribuiti quindi in maniera gratuita. Alcuni esempi sono Unity3D, Unreal Engine 4, Amazon Lumberyard e Godot.

2.7 Scelta dell’engine

2.7.1 Unity

Unity è uno strumento di authoring integrato multipiattaforma sviluppato da Unity Technologies e scritto in C++ per la creazione di videogiochi 3D o altri contenuti interattivi, quali visualizzazioni architettoniche o animazioni 3D in tempo reale. La licenza del

software è EULA ed esso gira sia su Microsoft Windows sia su macOS, con la possibilità di produrre giochi multipiattaforma quali Microsoft Windows, Linux, Mac, Xbox 360, PlayStation 3, PlayStation Vita, Wii, iPad, iPhone, Android, Windows Mobile, Playstation 4 Xbox one e Wii U.
Pensato quasi esclusivamente per gli sviluppatori indipendenti, da quando è nato (la versione 1.0 è stata distribuita nel 2005) ad oggi si è riservato una grossa fetta di mercato grazie alle ottime funzioni di export multipiattaforma, alla possibilità di programmare con 3 linguaggi diversi ad alto livello (C#, JavaScript, BOO) e alla sua interfaccia intuitiva, una commUnity sempre più attiva e un costo per la versione “PRO” minore rispetto alla concorrenza.

2.7.2 Unreal Engine 4

Unreal Engine è un motore grafico multipiattaforma (disponibile per Microsoft Windows, Linux e MacOS) sviluppato da Epic Games e scritto in C++, utilizzato per la prima volta nel 1998 per sviluppare il videogioco 3D sparatutto in prima persona Unreal. Nonostante fosse principalmente utilizzato per gli FPS, con il passare del tempo è riuscito a conquistare anche altri generi, in particolar modo i giochi di ruolo. Con la versione 4 rilasciata nel 2014, la Epic Games ha rilasciato un nuovo modello di sottoscrizione (Gratuito con royalty del 5% sopra i 3000$ di guadagno) per l'utilizzo del proprio engine, in modo da concorrere anche nel mercato degli sviluppatori indipendenti. Così come in Unity, anche in UE4 c'è completo supporto per la realizzazione di videogiochi multipiattaforma. L'unico linguaggio utilizzato per la realizzazione dei giochi con

questo engine è il C++. Inoltre, il codice sorgente con la versione 4 è stato reso disponibile a tutti.

2.7.3 Amazon Lumberyard

Amazon Lumberyard è un motore grafico gratuito (disponibile solo per Microsoft Windows) e sviluppato da Amazon, scritto in C++ e LUA, basato sull'architettura del CryEngine (motore grafico sviluppato da Crytek) e pubblicato nel 2016. La particolarità di questo engine, oltre alla possibilità
di sviluppare videogiochi multipiattaforma come Unity e UE, è l'integrazione degli Amazon Web Services in modo da permettere agli sviluppatori di compilare o hostare i propri giochi sui server di Amazon, oltre che ad avere pieno supporto di Twitch per i livestreaming e per il Twitch ChatPlay (dove le persone possono interagire con il videogioco in streaming attraverso dei comandi predefiniti scritti in chat). Il codice sorgente è disponibile gratuitamente ma con alcune limitazioni come l'impossibilità di utilizzarlo per creare un altro game engine.

2.7.4 Godot

Godot è un motore grafico open source e multipiattaforma con licenza MIT (disponibile per Microsoft Windows, macOS, Linux, FreeBSD, OpenBSD) e sviluppato dalla sua stessa commUnity, scritto in C++ e rilasciato nel 2015.
I giochi in godot possono essere scritti in C#, C++ o GDScript (simile al Python).

L'obiettivo principale di Godot è la semplificazione del processo di progettazione di qualsiasi videogioco. Mettendo da parte i classici modelli di progettazione, lo strumento crea un ambiente di sviluppo che consente di realizzare progetti nella maniera più naturale possibile partendo dagli elementi che saranno visibili all'utente finale. Questo modo di progettare videogiochi è stato pensato per favorire ed incoraggiare sia i singoli sviluppatori che i piccoli team di sviluppo ed è il fattore che rende Godot fondamentalmente diverso dagli altri motori grafici. Grazie al sistema di scene e nodi organizzati gerarchicamente in alberi è possibile progettare videogiochi partendo dalla schematizzazione del gioco su di un semplice foglio di carta.

2.7.5 Scelta

Tutti gli engine menzionati, nella loro versione gratuita, permettono di usare liberamente tutti gli strumenti utili per questo progetto.
In questo caso la scelta del motore è ricaduta su Unity per diversi motivi:

- La documentazione di Unity e la commUnity, oltre che l'ampia disponibilità di tutorial di terze parti, sono nettamente migliori rispetto agli altri engine dato che è stato il primo tra quelli menzionati ad essere stato rilasciato gratuitamente.

- Dato che i target del videogioco sono i tablet, rispetto agli altri engine, Unity è molto più leggero ed è compatibile anche con versioni di Android e iOS più datate.

- L'AssetStore di Unity è lo store più ricco e completo rispetto alla sua concorrenza, in cui è possibile trovare molti assets e scripts sviluppati da terze parti sia gratuitamente che a pagamento.

- Molto semplice da imparare ad utilizzare grazie alla possibilità di inserire i propri scripts all'interno degli oggetti in gioco (rendendo il Workflow decisamente più semplice da gestire) e grazie all'uso di un linguaggio ad alto livello come il C#.

- Essendo un engine molto leggero, è ben ottimizzato anche per essere utilizzato su elaboratori di vecchia data.

- Permette il deploy come web app.

2.8 Unity

2.8.1 Interfaccia

L'interfaccia principale di Unity è costituita da 6 elementi principali che possono essere spostati, riorganizzati e raggruppati a piacere (ad eccezione della toolbar) (figura 2.3):

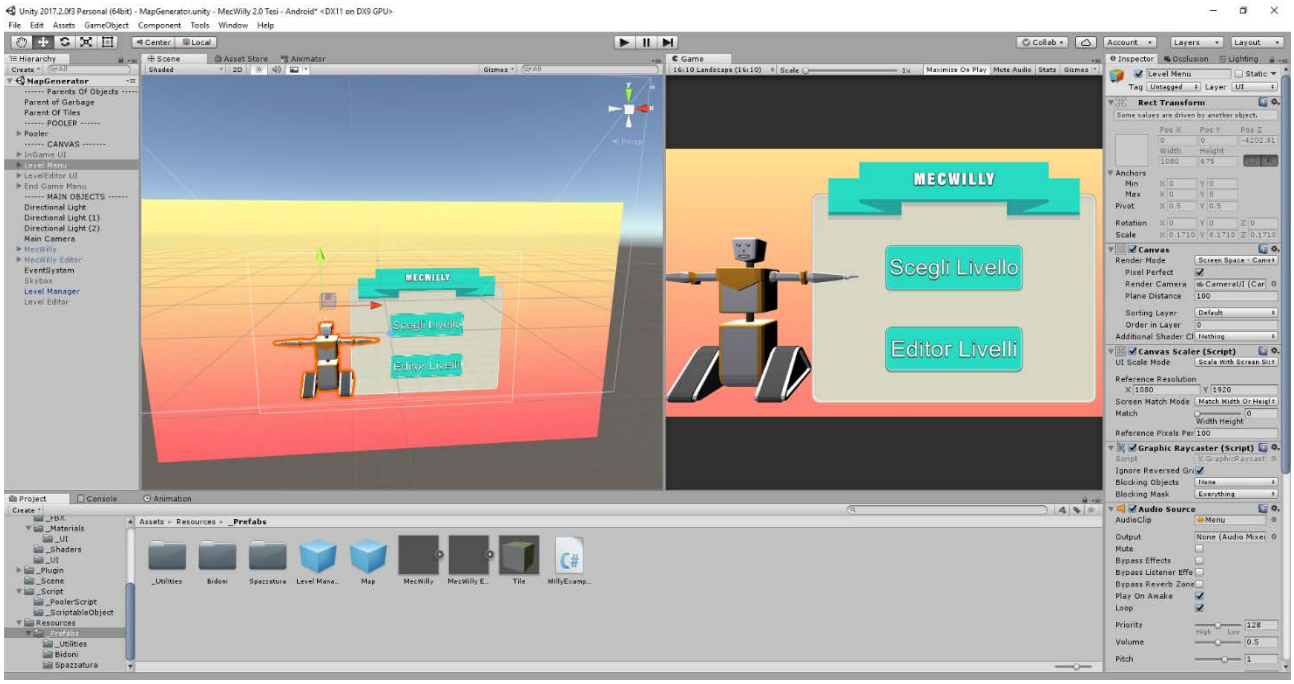

Figura 2.3: Interfaccia di Unity

- Project Window.

  In questa finestra vengono visualizzati tutti gli assets all'interno del progetto. Così come in un file manager, si possono creare o eliminare cartelle e assets, inserirli nei preferiti per velocizzare la ricerca degli assets più utilizzati, ricercare gli assets tramite la funzione di ricerca e, cliccando con il tasto destro su un asset, sono disponibili diverse opzioni quali:

  - creare un nuovo asset;
  - mostrarlo in Explorer;
  - aprirlo e cancellarlo;
  - cercare le referenze di quell'asset all'interno della Scene View;

- reimportare l'asset.

Inoltre, è possibile creare una cartella speciale chiamata "Resources" che si interfaccia con le API di Unity e via codice è possibile istanziare gli assets all'interno di questa cartella in maniera dinamica.

- Scene View.

  Questa finestra permette di muoversi all'interno di una scena e di modificarla rendendo possibile la visualizzazione di ogni asset presente nella scena. C'è anche la possibilità di modificare la visuale della scena da 3D a 2D e viceversa oltre che cambiare la visualizzazione della griglia e delle gizmos.

- Hierarchy Window.

  Questa finestra contiene una lista di tutti gli oggetti presenti nella scena. Alcuni di essi sono istanze di Assets (come i modelli 3D) e altri possono essere dei Prefabs (oggetti come gli altri, ma ogni modifica effettuata su uno di questi, viene applicata anche a tutti gli altri oggetti dello stesso tipo). Come suggerito dal nome, gli oggetti sono raggruppati in maniera gerarchica.

- Inspector Window.

  Questa finestra mostra tutte le informazioni di un determinato oggetto selezionato nella Hierarchy Window, ovvero vengono mostrati il Transform dell'oggetto e tutti gli script e i

component attaccati ad esso. Tramite questa interfaccia possiamo aggiungere un component o uno script o modificare i valori di ogni component esistente.

- Toolbar.

  La toolbar è diviso in sette controlli basilari e si interfaccia con diverse parti dell'editor:

  - Transform Tools che si interfaccia alla Scene View per il controllo degli oggetti all'interno della scena;

  - Transform Gizmos Toggles che modifica la visuale nella Scene View;

  - I bottoni Play/Pause/Step che si interfacciano con la Game View;
  - Cloud button che apre la finestra di Unity Services;

  - Account Drop-Down per accedere all'account Unity;

  - Layers Drop-Down per controllare quali oggetti devono essere visualizzati nella Scene View;

  - Layout Drop-Down per controllare l'arrangiamento di tutte le Views.

- Game View.

  La Game View si "attiva" nel momento in cui premiamo il pulsante Play della Toolbar. Essa è renderizzata dalle telecamere presenti nel gioco e rappresenta il prodotto finale. Unity fa in modo di rendere impossibile l'avvio del gioco nel caso in cui ci siano errori all'interno del codice e quindi nel caso in cui la compilazione non va a buon fine. Da qui è possibile selezionare quante e quali telecamere renderizzare, cambiare la risoluzione della View, scalare la risoluzione e leggere alcune statistiche importanti quali il framerate attuale.

2.8.2 Scene

Le scene contengo gli ambienti e i menù del gioco. Possono essere create più scene ad esempio per suddividere i vari livelli all'interno del gioco o i vari menù. In ogni scena è possibile inserire in forma gerarchica gli asset presenti nel Project Window.

2.8.3 Components

I components sono ciò che vanno a caratterizzare e che rendono funzionale un determinato GameObject (che funge da contenitore per essi). Il Transform Component è inserito di default all'interno di ogni singolo GameObject ed è l'unico component che non è possibile rimuovere dall'oggetto. Se ad esempio viene creato all'interno della Scene un GameObject vuoto e un altro con un solo

component aggiuntivo, il Light Component, si può notare che, nonostante un GameObject contiene solo un component in più dell'altro, i due oggetti sono totalmente diversi fra loro (infatti il GameObject con il Light Component definisce l'illuminazione della scena).

2.8.4 GameObjects

I GameObjects, ovvero gli oggetti presenti nel gioco, sono l'elemento più importante in Unity. Presi da soli non sono molto utili, ma sono dei potenti "contenitori" per i Components, i quali rendono questi GameObjects realmente utilizzabili. Esso ha sempre un Transform attaccato per rappresentare (e modificare) la rotazione, la grandezza e l'orientamento dell'oggetto. In seguito, è possibile aggiungere nuovi component o script sia a runtime che tramite l'Inspector Window dopo aver selezionato l'oggetto da voler modificare.

2.8.5 Prefabs

All'interno di un videogioco possiamo avere diversi elementi identici che si ripetono (ad esempio dei proiettili in uno sparatutto, nemici dello stesso tipo). Per poter modificare tutti gli oggetti identici fra loro senza dover applicare le modifiche singolarmente e manualmente, Unity mette a disposizione i prefabs. Essi infatti sono un tipo di risorsa che permette di memorizzare un GameObject insieme a tutti i suoi components, scripts e proprietà.

2.8.6 Cameras

Le telecamere nei videogiochi sono gestite in maniera molto simile a quelle dei film, infatti esse servono per mostrare il mondo di gioco al giocatore. In una scena ci deve essere minimo una telecamera attiva e se ne possono inserire più di una (da poter attivare contemporaneamente o a runtime disattivandone un'altra). Esse sono dei GameObject contenenti Camera Component, Flare Layer, Audio Listener e un Physics Raycaster.
All'interno del Camera Component si possono modificare l'ampiezza di visione della telecamera, il tipo di proiezione (Ortographic o Perspective) e lo Skybox (il cielo) oltre che altre funzioni quali MSAA, il culling e l'HDR.

2.8.7 Lights

Mentre le mesh e le texture definiscono la forma e il look degli oggetti in scena, le luci definiscono il colore e l'umore di un ambiente tridimensionale. Sono dei component che vengono attaccati ad un GameObjects.

2.8.8 ScriptableObjects

Uno ScriptableObject è una classe che permette di immagazzinare una grande quantità di dati indipendenti dalle istanze dello script. Serve per serializzare qualsiasi tipo di dato facente parte del mondo di Unity. Ad esempio, oggetti come i GameObjects, Vector3 e classi

non predefinite di Unity possono essere salvati in questa classe per essere richiamati ed utilizzati in seguito, a differenza di JSON o dei semplici file di testo (che possono salvare solo dati generici come stringe e interi, ma non oggetti che derivano da MonoBehaviour). Una volta creata la classe è possibile inizializzare i dati di questa classe creando una nuova istanza di essa via codice oppure tramite il ProjectWindow, facendo click con il tasto destro del mouse su una cartella, si ha la possibilità di creare un nuovo file di quel tipo e tramite l'inspector è possibile editare tutti i valori di quella classe.

ScriptableObjects dei livelli

```
[CreateAssetMenu]
public class LevelScriptableObject : ScriptableObject {
    public GameObject rifiuto;
    public Map map;
    public Position startPlayerPos;
    public Position[] garbagePos;
}

[System.Serializable]
public class Map
{
    public float x;
    public float z;
}

[System.Serializable]
public class Position
{
    public GameObject target;
    public bool garbage;
    [HideInInspector]
    public bool isMovedInEditor;
    [HideInInspector]
    public bool insideMap;
    public float x;
```

```
    public float z;
    public float rotationY;
}
```

CAPITOLO 3

Implementazione

In questo capitolo verranno illustrate le scelte implementative effettuate descrivendo tutti gli elementi che compongono il videogioco, facendo attenzione in particolar modo a confrontare questa nuova versione con quella precedente riprendendone ogni punto e ampliando descrivendo le nuove feature.

3.1 Modelli 3D

A differenza della vecchia versione del gioco, è stato utilizzato un workflow più diretto e semplice vista la natura dei modelli da creare. Ad esempio la bottiglia di vetro che nella versione originale è stata modellata partendo dal disegno di una linea del contorno facendola ruotare tramite l'utilizzo di modificatori per determinarne la geometria, nella versione attuale è stata modellata partendo da una geometria primitiva (un cilindro, già disponibile all'interno del software) e applicando due estrusioni alla faccia superiore del cilindro e inserendo tre edge-loop all'interno della prima, in seguito scalati proporzionalmente per creare il collo della bottiglia.
Il workflow partendo da geometrie primitive è stato quello preferito nello sviluppo dei modelli.

Per quel che concerne il texturing, tutti i modelli (tranne banana e MecWilly) condividono un'unica texture atlas, cioè una texture che contiene le diverse texture degli oggetti. Questo permette di caricare nella RAM una sola texture condivisa piuttosto che diverse texture per ogni modello (figura 3.1).

Figura 3.1: Texture Atlas

3.2 Unity

3.2.1 Prefabs

I modelli 3D sono stati importati in Unity e sono stati realizzati i prefabs per ogni oggetto creato in Blender, in modo da poterli sia utilizzare da editor che a runtime.

3.2.2 Logica Di Gioco

L'obiettivo del gioco è dare dei comandi a MecWilly in modo da fargli raggiungere il bidone corrispondente al rifiuto situato nella propria mano.
L'area di gioco di MecWilly è una griglia a scacchiera. In questa griglia sono situati vari bidoni (ma solo uno è quello corrispondente al rifiuto da gettare) e MecWilly è libero di muoversi all'interno di quest'area di gioco. Non esistono limiti di tempo o di mosse per terminare il livello, ma una volta terminato comparirà il punteggio ottenuto in base al tempo e alle mosse effettuate, verrà richiesto di inserire il proprio nome e verrà visualizzata una classifica per quello specifico livello.
Una volta lanciato il gioco si è messi di fronte a una scelta: è possibile cominciare a giocare scegliendo dei livelli creati precedentemente oppure è possibile avviare l'editor dei livelli, in modo da poter creare un livello personalizzato. Di seguito saranno analizzate entrambe le feature.

3.2.2.1 Livelli

Una volta selezionata l'opzione "Scegli livello" si passa alla schermata di selezione dei livelli. Di base ne sono presenti cinque, partendo dal primo livello più facile al quinto livello più difficile, e in ogni livello il robot deve gettare un rifiuto diverso. In questo menù è presente anche il livello "LE" (sigla che sta per "Level Editor") che corrisponde al livello creato grazie all'editor dei livelli. Sia questo menù sia il livello vero e proprio sono gestiti da un oggetto di classe LevelManager che è un Singleton. Questo oggetto si occuperà di istanziare il livello giusto e inserire all'interno dei livelli tutto ciò che è scritto nello Scriptable Object corrispondente al livello selezionato e di gestire il menù di pausa.
Appena inizia il livello viene attivato il timer dalla classe Timer, è possibile passare al menù di pausa premendo il bottone corrispondente e la classe PlayerController gestisce il movimento del robot rilevando la pressione dei bottoni presenti a schermo effettuando dei controlli nel caso in cui il giocatore stia uscendo fuori dalla mappa o vada a sbattere contro i bidoni, o ancora nel caso in cui il giocatore si trova di fronte al bidone corrispondente al rifiuto da gettare (in questo caso si avvia un animazione in cui il robot getta via il rifiuto).
Nel momento in cui il rifiuto è stato gettato entra in funzione la classe RifiutoManager che gestisce i menù di fine livello: viene istanziato un InputText dove poter inserire il proprio nome e in seguito viene visualizzata la classifica generale aggiornata nel caso in cui il punteggio appena ottenuto rientri nella Top 5.
L'audio del gioco è gestito in due modi diversi:

- la musica in background è inserita come clip in un component AudioSource che viene settato a Play On Awake e Loop, in modo che la musica inizi appena l'oggetto su cui è attaccato il component viene attivato e in modo che una volta terminata la clip ricominci da capo;

- gli effetti sonori sono gestiti a runtime utilizzando la funzione PlayOneShot presente nella classe AudioSource di MonoBehaviour come si può notare nel frammento riportato di seguito della classe RifiutoManager.

Sia le classifiche che i livelli creati con l'editor all'interno del gioco vengono salvati in locale e caricati al momento opportuno.

Frammento del LevelManager – Selezione del livello

```
public void OnClickLevel(string levelName)
{

    if (levelName == "LevelSelect")
    {
        mainMenuUI.SetActive(false);
        levelSelectMenu.SetActive(true);
    }

    else
    {
        LevelEditor.sceneLoadedManuallyLevelEditor =
           false;
        sceneLoadedManuallyInsideLevel = false;

        if (levelName == "UserLevel")
        {
            LevelEditor.disableTouch = true;
            foreach (LevelScriptableObject level in
```

```
                levelList)
            {

                if (level.name == "UserLevel")
                {
                    LevelEditor.LoadAssets(level);
                }

            }
        }

        levelSelected = AssignLevel(levelName);
        playerHand = levelSelected.rifiuto;
        tilesPos = new
           Vector3[(int)levelSelected.map.x *
           (int)levelSelected.map.z];
        GenerateMap();
        levelMenu.SetActive(false);

        if (levelName != "LevelEditor")
        {
            LevelEditor.disableTouch = true;
            inGameUI.SetActive(true);
        }

        else
        {
            LevelEditor.disableTouch = false;
            levelEditorUI.SetActive(true);
            levelEditorObject.SetActive(true);
            gameObject.SetActive(false);
        }
    }
}

private LevelScriptableObject AssignLevel(string
      levelName)
{

    foreach(LevelScriptableObject level in levelList)
    {

        if (level.name == levelName)
        {
```

```
                return level;
            }

        }

        return null;
    }
```

Frammento del LevelManager – Generazione del livello

```
private void GenerateMap()
    {

        for (int x = 0; x < levelSelected.map.x; x++)
        {
            for (int z = 0; z < levelSelected.map.z; z++)
            {

                Vector3 tilePosition = new Vector3((int)-
                    levelSelected.map.x / 2 + x, -10f,
                    (int) - levelSelected.map.z / 2 +
                         z);
                GameObject newTile =
                    TileObjectPooling.current.
                    GetPooledObject(tilePrefab);
                tileQueue.Enqueue(newTile);
                newTile.transform.position = tilePosition;
                newTile.transform.rotation =
                    Quaternion.Euler(90f, 0f, 0f);
                newTile.transform.localScale = new
                    Vector3(0.95f, 0.95f, 20f);
                tilesPos[tileArrayCounter] =
                    newTile.transform.position;
                tileArrayCounter++;
                newTile.SetActive(true);
            }
        }

        //spawn player in livello
        if (levelSelected.name != "LevelEditor")
        {
            playerObject.SetActive(true);
```

```
            playerObject.transform.position = new
               Vector3(levelSelected.startPlayerPos.x,
               0.21f,
                    levelSelected.startPlayerPos.z);
            playerObject.transform.rotation =
               Quaternion.Euler(0f,
                    levelSelected.startPlayerPos.
                    rotationY,0f);
        }

        //spawn playerEditor in editor
        else
        {
            playerEditorObject.SetActive(true);
            playerEditorObject.transform.position = new
                    Vector3(levelSelected.startPlayerPos.x
                    , 0.21f, levelSelected.
                         startPlayerPos.z);
            playerEditorObject.transform.rotation =
               Quaternion.Euler(0f,
                    levelSelected.startPlayerPos.
                    rotationY, 0f);
        }

        //Spawn bidoni
        for(int i = 0; i <
          levelSelected.garbagePos.Length; i++)
        {
            garbageSpawning = true;
            SpawnGarbage(i);
        }
    }

    private void SpawnGarbage(int i)
    {

        if (levelSelected.garbagePos[i].garbage) {
            GameObject newGarbage =
                    GarbageObjectPooling.current.
                         GetPooledObject(levelSelected.
                         garbagePos[i].target);
            garbageQueue.Enqueue(newGarbage);
            newGarbage.transform.position = new
                    Vector3(levelSelected.garbagePos[i].x,
```

```
                    0.2f,
                levelSelected.garbagePos[i].z);
            newGarbage.transform.rotation =
                    Quaternion.Euler(0f,
                        levelSelected.
                         garbagePos[i].rotationY,
                    0f);
            newGarbage.SetActive(true);
        }

        else
        {
            GameObject newRifiuto =
               Instantiate(levelSelected.garbagePos[i]
                    .target);
            newRifiuto.transform.position = new
                    Vector3(levelSelected.garbagePos[i].x,
                        newRifiuto.transform.position.y,
                            levelSelected.garbagePos[i].
                             z);
            newRifiuto.transform.rotation =
                    Quaternion.Euler(0f,
                        levelSelected.
                         garbagePos[i].rotationY
                              , 0f);
            newRifiuto.transform.localScale = new
                    Vector3(0.6f, 0.6f, 0.6f);
                         newRifiuto.SetActive(true);
        }
    }
```

Frammento del PlayerController – Movimento e rotazione

```
   private void CheckCollisionAndMove(Vector3
     posNextTile, float step)
   {

        foreach(Transform tile in
          LevelManager.current.parentOfTiles.transform)
        {

            If (LevelManager.current.tilesPos.
```

```
        Contains(posNextTile))
{

    if (tile.position.x == posNextTile.x &&
        tile.position.z == posNextTile.z &&
             !tile.gameObject.
             GetComponent<GarbageObjectByTile>
             ().cannotMove)
    {
        if (moveForward)
        {
            StartCoroutine(MovePlayerNextTile
             (posNextTile.x,
             posNextTile.z, step));
        }
        else if (moveBackward)
        {
            StartCoroutine(MovePlayerNextTile
             (posNextTile.x,
             posNextTile.z, step));
        }
        break;
    }

    else if (tile.position.x == posNextTile.x
             && tile.position.z ==
                  posNextTile.z &&
                  tile.gameObject.GetComponent
                  <GarbageObjectByTile>().
                  cannotMove)
    {
        moveForward = false;
        moveBackward = false;
        break;
    }
}


else
{
    moveForward = false;
    moveBackward = false;
    break;
}
```

```
    }
}

public IEnumerator RotatePlayer(float rotationY)
{
    GetComponent<Rigidbody>().DORotate(new
           Vector3(transform.rotation.x, rotationY,
            transform.rotation.z), 0.3f);
    DeactivateButtons();
    isRotating = true;

    yield return new WaitForSeconds(0.4f);

    transform.eulerAngles =
      TakeRotationInspector(gameObject.transform);
    isRotating = false;
    ActivateButtons();
}

private IEnumerator MovePlayerNextTile(float
 playerPosX, float playerPosZ, float step)
{

    Vector3 targetPos = new Vector3(playerPosX,
      playerPosY, playerPosZ);
    transform.position =
       Vector3.MoveTowards(transform.position,
      targetPos, step);
    if (moveForward)
    {
        isMovingForward = true;
    }
    else if (moveBackward)
    {
        isMovingBackward = true;
    }

    DeactivateButtons();

    yield return new WaitUntil(() =>
      Vector3.Distance(transform.position, targetPos)
           < 0.05f);
```

```
        transform.position = targetPos;
        ActivateButtons();
    }
```

Frammento del PlayerController – Gettare il rifiuto

```
  private void OnTriggerStay(Collider other)
  {
       if (other.CompareTag("OpenGarbage") &&
         other.gameObject.
              GetComponent<GarbageObjectByTile>()
              .garbageToAnimate.CompareTag(RifiutoManager
                   .rifiutoObject.tag) && !isRotating)

       {
           garbageObject =
              other.gameObject.
                   GetComponent<GarbageObjectByTile>().
                        garbageToAnimate;

           if ((Mathf.Abs(transform.eulerAngles.y) –
              Mathf.Abs(garbageObject.transform.
                   eulerAngles.y ) > 170f &&
                   Mathf.Abs(transform.eulerAngles.y) –
                   Mathf.Abs(garbageObject.transform.
                   eulerAngles.y) < 190f) ||
                   (Mathf.Abs(transform.eulerAngles.y) -
                   Mathf.Abs(garbageObject.transform.eule
                   rAngles.y) < -170f &&
                   Mathf.Abs(transform.eulerAngles.y) -
                   Mathf.Abs(garbageObject.transform.eule
                   rAngles.y) > -190f))
           {

               gameObject.GetComponent<AudioSource>().
                    PlayOneShot(animationThrowClip);
               Animator garbageAnim =
                   other.gameObject.
                        GetComponent
                             <GarbageObjectByTile>().
                             garbageToAnimate.
                             GetComponent<Animator>();
```

```
            DeactivateButtons();
            garbageAnim.Play("B_OpenClose");
            Animator playerAnim =
                GetComponent<Animator>();
            playerAnim.CrossFade("C_Throw", 0.2f);

            other.enabled = false;
        }

        else
        {
            if (!isHittingTrash)
            {
                gameObject.
                     GetComponent<AudioSource>().
                          PlayOneShot(hitTrashClip);
                isHittingTrash = true;
            }
        }

    }

    else if(other.CompareTag("OpenGarbage") &&
      !other.gameObject.
           GetComponent<GarbageObjectByTile>().
                garbageToAnimate
                     .CompareTag(RifiutoManager.
                          rifiutoObject.tag)
                               && !isRotating)

    {

        if (!isHittingTrash)
        {

            gameObject.
                GetComponent<AudioSource>().
                     PlayOneShot(hitTrashClip);

            isHittingTrash = true;
        }
    }
}
```

```
    private void OnTriggerExit(Collider other)
    {
        isHittingTrash = false;
    }
```

Frammento del RifiutoManager - Classifica

```
   public void ShowLeaderBoard(InputField input)
   {
        if (input.text.Length > 0)
        {

          LevelManager.current.playerNameMenu.
               SetActive(false);
          LevelManager.current.highScoreMenu.
               SetActive(true);

            string nameLBJson = "LeaderBoard" +
               LevelManager.levelSelected + ".json";
            string filePath =
               Path.Combine(Application.persistentDataPath
                    , nameLBJson);

            LeaderboardSO thisLB =
               ScriptableObject.
                    CreateInstance<LeaderboardSO>();
            thisLB.name = "LeaderBoard" +
               LevelManager.levelSelected;
            InitializeLBSO(thisLB);

            if (!File.Exists(filePath))
            {
                thisLB.PositionInLB[0].namePlayer =
                    input.text;
                thisLB.PositionInLB[0].score =
                    PlayerController.score;
                SaveLeaderboard(thisLB);
                InsertLBOnPanel(thisLB);
            }
            else
            {
                LoadLeaderBoard(thisLB);
```

```
    LBloaded = true;
}


if (LBloaded)
{

    if (PlayerController.score <=
        thisLB.PositionInLB[4].score)
    {
        LBloaded = false;
    }

    else
    {

        PlayerScoreData[] temp = new
             PlayerScoreData[5];
        for (int n = 0; n < 5; n++)
        {
            temp[n] = new PlayerScoreData();
        }

        int actualPos = 0;
        for (int i = 0; i<5; i++)
        {
            if(thisLB.
               PositionInLB[actualPos].score
                  > PlayerController.score
                     || scoreInserted)
            {
                temp[i] =
                  thisLB.
                  PositionInLB[actualPos];
                actualPos++;
            }
            else if (!scoreInserted &&
                  thisLB.
                  PositionInLB[actualPos].
                  score <=
                  PlayerController.score)
            {
                temp[i].namePlayer =
                   input.text;
```

```
                        temp[i].score =
                          PlayerController.score;
                        scoreInserted = true;
                    }
                }

                thisLB.PositionInLB = temp;
            }
            SaveLeaderboard(thisLB);
            InsertLBOnPanel(thisLB);
            LBloaded = false;
        }
    }

}
```

Frammento del RifiutoManager – Load/Save classifica

```
public void CopyLeaderBoardSO(LeaderboardSO
 toCopy, LeaderboardSO toPaste)
{
    toPaste.PositionInLB = toCopy.PositionInLB;
}

public void SaveLeaderboard(LeaderboardSO toSave)
{
    string filePath =
      Path.Combine(Application.persistentDataPath,
           toSave.name + ".json");
    if (File.Exists(filePath))
    {
        string dataAsJson =
           File.ReadAllText(filePath);
        LeaderboardSO saveData =
           ScriptableObject.
                CreateInstance("LeaderboardSO") as
                     LeaderboardSO;

        JsonUtility.
           FromJsonOverwrite(dataAsJson, saveData);
        CopyLeaderBoardSO(toSave, saveData);
        string saveStatePath =
```

```
            Path.
                Combine(Application.persistentDataPath
                    , toSave.name + ".json");

        File.WriteAllText(saveStatePath,
            JsonUtility.ToJson(saveData, true));
    }
    else
    {
        LeaderboardSO saveData =
            ScriptableObject.
                CreateInstance("LeaderboardSO") as
                    LeaderboardSO;
        CopyLeaderBoardSO(toSave, saveData);
        string saveStatePath =
            Path.
                Combine(Application.persistentDataPath
                    , toSave.name + ".json");
        File.WriteAllText(saveStatePath,
            JsonUtility.ToJson(saveData, true));
    }
}

public void LoadLeaderBoard(LeaderboardSO toLoad)
{
    string filePath =
      Path.Combine(Application.persistentDataPath,
            toLoad.name + ".json");
    string dataAsJson = File.ReadAllText(filePath);
    LeaderboardSO loadData =
      ScriptableObject.CreateInstance("LeaderboardSO")
       as LeaderboardSO;
    JsonUtility.
      FromJsonOverwrite(dataAsJson, loadData);
    CopyLeaderBoardSO(loadData, toLoad);
}
```

3.2.2.2 Editor dei livelli

Una delle nuove feature create per la seconda versione del gioco è l'editor dei livelli, gestita dalla classe LevelEditor. La UI dell'editor è divisa in quattro sezioni sui quattro angoli dello schermo:

- In basso a sinistra è possibile spostare i bidoni all'interno dell'area di gioco nella posizione desiderata. Nel caso in cui i bidoni vengono spostati ma non vengono inseriti all'interno della griglia, al salvataggio del livello vengono automaticamente eliminati.

- In basso a destra è possibile selezionare il rifiuto da far tenere al robot. All'apertura dell'editor il robot ha già in mano un rifiuto predefinito in modo da non generare errori.

- In alto a sinistra è possibile cambiare il numero di quadrati presenti nella griglia (da un minimo di 1x1 a un massimo di 10x10).

- In alto a destra sono presenti tre pulsanti: (rispettivamente da sinistra verso destra) il primo indica il salvataggio del livello creato, il secondo indica la possibilità di cancellare tutte le modifiche effettuate e di ricominciare da capo, il terzo indica l'uscita dall'editor senza salvare le modifiche effettuate.

Nel caso in cui il robot dovesse essere spostato in un’area fuori dalla griglia di gioco e il livello dovesse essere salvato, una volta avviato il livello "LE" MecWilly si troverà nella sua posizione standard (ovvero X = 0 e Z = 0).

Frammento del LevelEditor – Pulsanti in alto a destra

```
 IEnumerator Confirm(AudioSource audio)
{
    yield return new
      WaitForSeconds(audio.clip.length);

    disableTouch = false;
    areYouSure.GetComponent<Image>().enabled = false;

    if
      (EventSystem.current.currentSelectedGameObject.
          transform.parent.gameObject ==
               discardChangesUI)
    {

        CopyScriptableObject(LevelManager.current.
           defaultLevelEditorSO,
               LevelManager.current.levelEditor);

        foreach (Position garbage in
           LevelManager.current.
               levelEditor.garbagePos)
        {
            garbage.isMovedInEditor = false;
        }
        EventSystem.current.currentSelectedGameObject.
           transform.parent.gameObject.
               SetActive(false);
        sceneLoadedManuallyLevelEditor = true;
        SceneManager.LoadScene(0);
    }

    else if
      (EventSystem.current.currentSelectedGameObject.
```

```
            transform.parent.gameObject == exitUI)
        {

            foreach (Position garbage in

          LevelManager.current.levelEditor.garbagePos)
            {
                garbage.isMovedInEditor = false;
            }
            disableTouch = false;
            SceneManager.LoadScene(0);
        }


        else
        {

            CopyScriptableObject(LevelManager.
               current.levelEditor, userLevel);
            foreach (Position garbage in
               LevelManager.current.
                    levelEditor.garbagePos)
            {
                garbage.isMovedInEditor = false;
            }


            File.Delete(Path.Combine(Application.
               persistentDataPath, "LeaderBoardUserLevel
                    (LevelScriptableObject).json"));

            SceneManager.LoadScene(0);
        }
    }

    IEnumerator ConfirmMapSize(AudioSource audio)
    {

        yield return new
          WaitForSeconds(audio.clip.length);

        if (!disableTouch)
        {
            int xSize = 0;
```

```
            int zSize = 0;
            string xString = new
               String(xText.GetComponent<Text>().
                    text.Where(Char.IsDigit).ToArray());
            string zString = new
               String(zText.GetComponent<Text>().
                    text.Where(Char.IsDigit).ToArray());
            int.TryParse(xString, out xSize);
            int.TryParse(zString, out zSize);
            LevelManager.current.levelEditor.map.x =
               xSize;
            LevelManager.current.levelEditor.map.z =
               zSize;
            SaveAssets(LevelManager.current.levelEditor);
            sceneLoadedManuallyLevelEditor = true;
            SceneManager.LoadScene(0);
        }
    }

    IEnumerator DontConfirm(AudioSource audio)
    {
        yield return new
          WaitForSeconds(audio.clip.length);
        LevelManager.current.
          levelEditorUI.GetComponent<AudioSource>().
               UnPause();
        disableTouch = false;
        areYouSure.GetComponent<Image>().enabled = false;
        EventSystem.current.
          currentSelectedGameObject.transform.parent
               .gameObject.SetActive(false);
    }
```

3.2.3 Differenze con la versione precedente

La principale differenza rispetto alla versione precedente è l'implementazione dell'object pooling per i bidoni e la griglia, utile ad ottimizzare meglio il gioco per dispositivi mobili.
Le funzioni di base esistenti in Unity per istanziare e distruggere oggetti a runtime sono Instantiate e Destroy e richiedono in genere

poche risorse quando vengono richiamate. Però ci sono casi in cui gli oggetti da creare e da distruggere hanno un ciclo di vita molto ridotto dalla creazione ed è necessario distruggerne una quantità molto vasta nel giro di pochi secondi (si pensi ad esempio a dei proiettili in uno sparatutto). In questo scenario le risorse richieste sono numerose e la CPU fa fatica a gestirle tutte. Inoltre, Unity utilizza la Garbage Collection per deallocare memoria che non viene più utilizzata, per cui un eccessivo numero di chiamate della Destroy rallenta ancora di più la CPU creando fenomeni di stuttering e lag specialmente nei dispositivi mobili.

L'object pooler è la classe dove vengono preistanziati tutti gli oggetti che vengono utilizzati più spesso in momenti specifici prima del gameplay vero e proprio (come ad esempio all'avvio del gioco o durante una schermata di caricamento). Invece di istanziare e distruggere nuovi oggetti, vengono riutilizzati gli oggetti che non fanno più parte della scena (attivandoli e disattivandoli al momento opportuno).

Lo stile grafico è stato modificato in maniera opportuna utilizzando colori più vivaci per i menù e gli oggetti in generale. Inoltre, la posizione della telecamera è stata modificata per rendere ben visibile sia su smartphone che su tablet (anche cambiando risoluzione del dispositivo) tutti gli oggetti presenti nella scena e la sua proiezione è passata da "prospettiva" a "ortografica", in modo che gli oggetti non variassero di dimensione al variare della distanza dalla telecamera.

Sono stati creati cinque livelli al posto di tre in modo da utilizzare tutti e cinque i prefabs dei rifiuti per educare il bambino alla raccolta differenziata e per ognuno dei livelli viene generata una classifica in modo da incoraggiare il bambino a migliorare la propria prestazione nel gioco.

L'editor dei livelli è stato aggiunto in modo tale che l'insegnante possa creare dei livelli ad hoc per il bambino e in modo tale che il bambino possa divertirsi a creare un livello che sia per lui piacevole senza dover mettere mano al codice sorgente.
L'audio è un'altra feature assente nella versione precedente. Sono state selezionate e inserite melodie in background ed effetti sonori consoni alla fascia di età dei bambini su cui sono stati svolti i test di gradimento del videogioco.

TileObjectPooling – classe che gestisce il pooler della griglia

```
[System.Serializable]
public class TileObjectPoolItem
{
    public GameObject pooledObject;
    public int pooledAmount;
    public bool shouldExpand;
    public bool canRotate;
}

public class TileObjectPooling : MonoBehaviour
{

    public List<TileObjectPoolItem> itemsToPool;
    public static TileObjectPooling current;
    List<GameObject> pooledObjects;
    public GameObject parentOfPool;

    void Awake()
    {
        current = this;
    }

    void Start()
    {
        pooledObjects = new List<GameObject>();
        foreach (TileObjectPoolItem item in itemsToPool)
        {
            for (int i = 0; i < item.pooledAmount; i++)
```

```
            {
                GameObject obj =
                    Instantiate(item.pooledObject);
                obj.transform.parent =
                    parentOfPool.transform;
                obj.SetActive(false);
                pooledObjects.Add(obj);
            }
        }
    }

    public GameObject GetPooledObject(GameObject segment)
    {
        for (int i = 0; i < pooledObjects.Count; i++)
        {
            if (!pooledObjects[i].activeInHierarchy &&
               (pooledObjects[i].name == segment.name +
                    "(Clone)" || pooledObjects[i].name ==
                        segment.name))
            {
                return pooledObjects[i];
            }
        }
        foreach (TileObjectPoolItem item in itemsToPool)
        {
            if (item.pooledObject.name == segment.name)
            {

                if (item.shouldExpand)
                {
                    GameObject obj =
                        Instantiate(item.pooledObject);
                    obj.transform.parent =
                        parentOfPool.transform;
                    obj.SetActive(false);
                    pooledObjects.Add(obj);
                    return obj;
                }
            }
        }
        return null;
    }

}
```

3.2.4 Esempio di partita

All'avvio del gioco viene visualizzato un menu con le opzioni di selezione di un livello esistente (o creato con l'editor dei livelli) o di passaggio all'editor dei livelli (figura 3.2).

Figura 3.2: Menù iniziale

Se viene premuto il bottone per passare all'editor, esso viene caricato ed è possibile spostare i bidoni all'interno della griglia, spostare MecWilly e scegliere il rifiuto da fargli tenere in mano (figura 3.3).

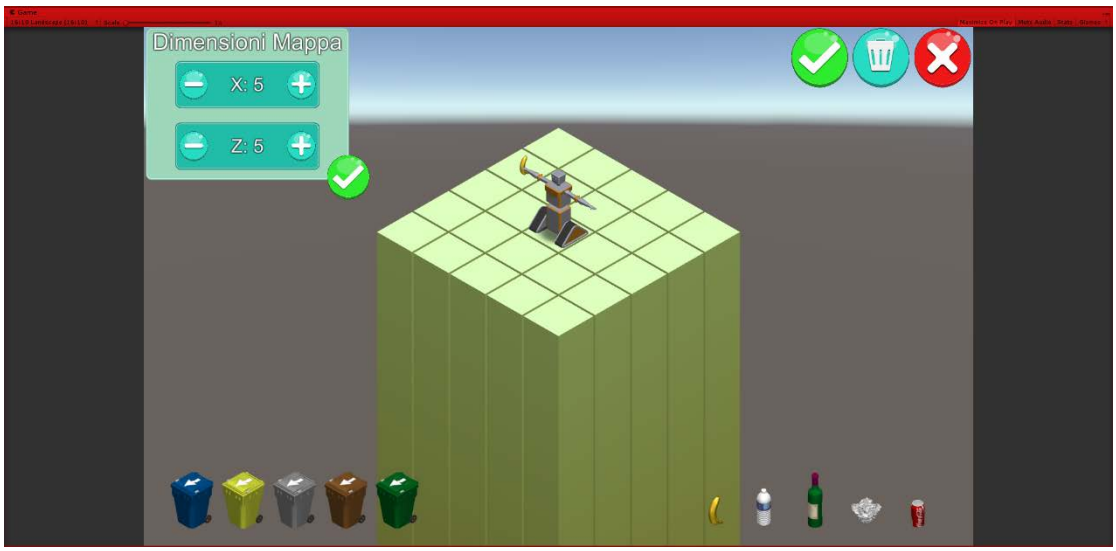

Figura 3.3: Editor dei livelli

Nel caso in cui si clicca il bottone per la selezione di un livello esistente, viene aperto un menù di selezione livello formato da 6 quadrati (ognuno rappresentante un livello diverso). Il livello “LE” si avvia solo se è stato precedentemente creato un livello utilizzando l’editor, altrimenti il menù rimane attivo fino a che non viene scelto un livello esistente (figura 3.4).

Figura 3.4: Menù di selezione livello

Una volta selezionato il livello esso viene caricato e il gioco si avvia, facendo partire il timer e attivando i pulsanti della UI (figura 3.5).

Figura 3.5: Livello 1 avviato

Attraverso i pulsanti “Avanti”, “Indietro”, “Sinistra”, “Destra” è possibile far interagire il giocatore con il robot in modo da raggiungere l'obiettivo. Durante il movimento o la rotazione, i bottoni si disattivano in modo da poter dare tempo a MecWilly di completare la propria azione e vengono riattivati al termine della stessa (figura 3.6).

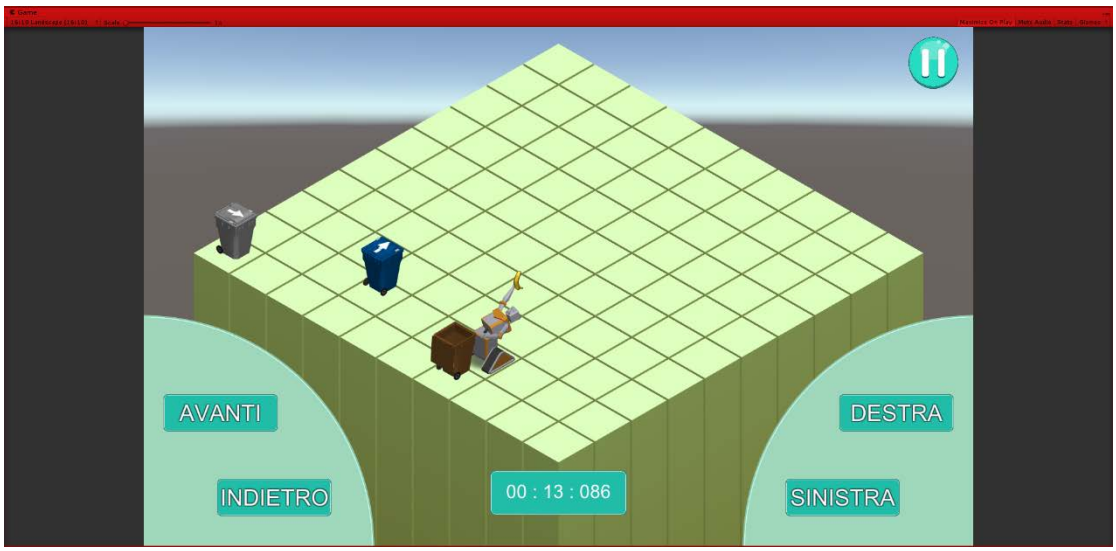

Figura 3.6: MecWilly getta il rifiuto nel bidone giusto

Quando termina l'animazione e termina la clip corrispondente all'effetto sonoro del rifiuto che viene gettato, compare l'input text dove poter inserire il proprio nome per entrare in classifica (figura 3.7).

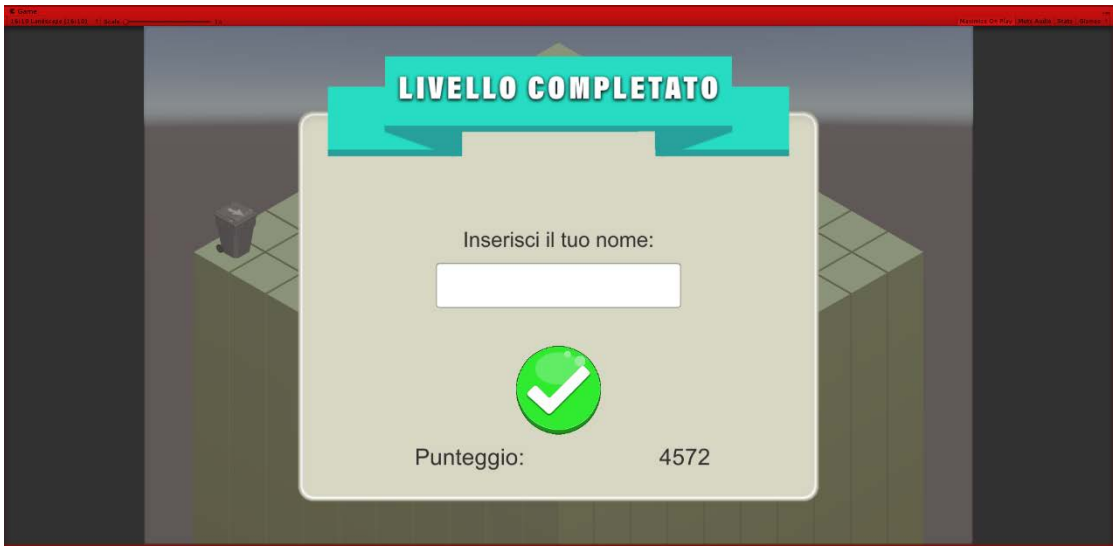

Figura 3.7: Inserire nome per entrare in classifica

Appena viene inserito il nome, compare la classifica (top 5) in cui è possibile vedere se il proprio nome compare in essa. Il punteggio

viene calcolato a runtime, partendo da un limite massimo di 5000 punti e diminuendo ogni secondo e ogni volta che viene premuto un bottone per il movimento (figura 3.7).

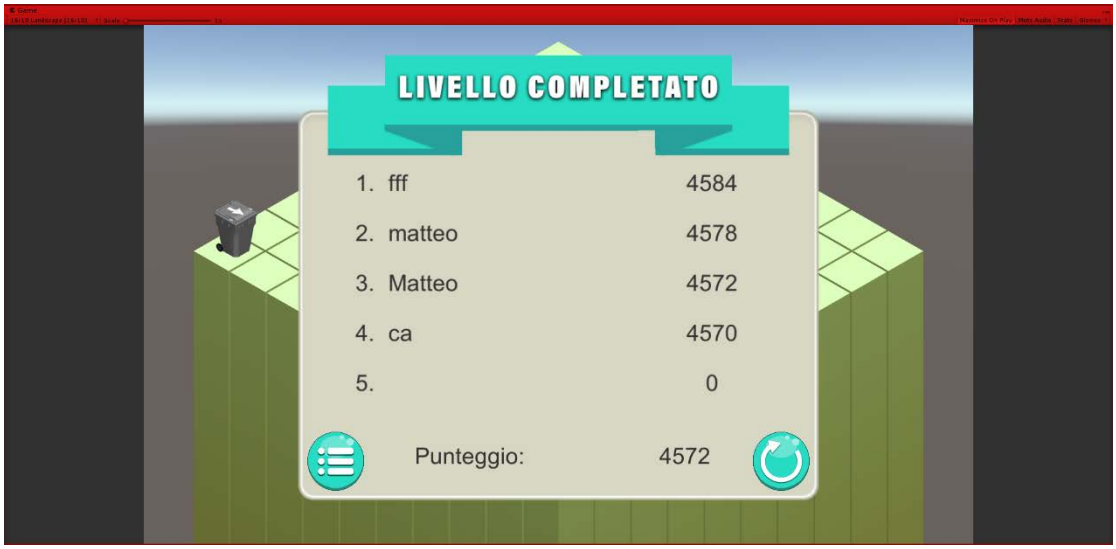

Figura 3.7: Classifica del livello completato

3.3 Livelli di gioco

Nel primo livello MecWilly deve raggiungere il bidone marrone per gettare la banana che ha in mano. La griglia di gioco è 10x10 (figura 3.5).

Nel secondo livello MecWilly deve raggiungere il bidone giallo per gettare la bottiglia di plastica che ha in mano. La griglia di gioco è 5x7 (figura 3.8).

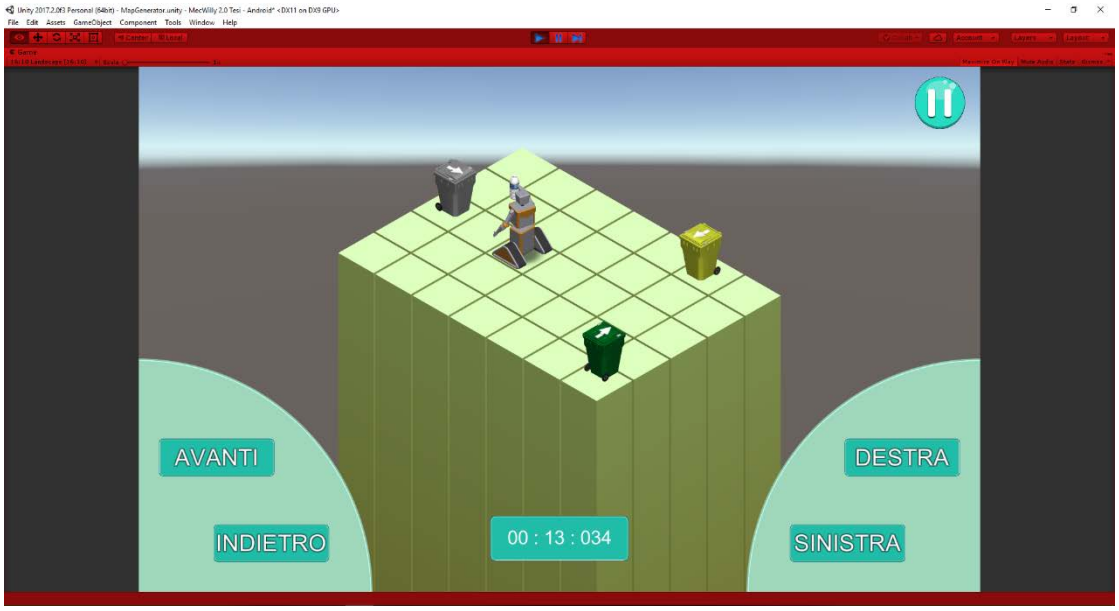

Figura 3.8: Livello 2

Nel terzo livello MecWilly deve raggiungere il bidone verde per gettare la bottiglia di vetro che ha in mano. La griglia di gioco è 8x5 (figura 3.9).

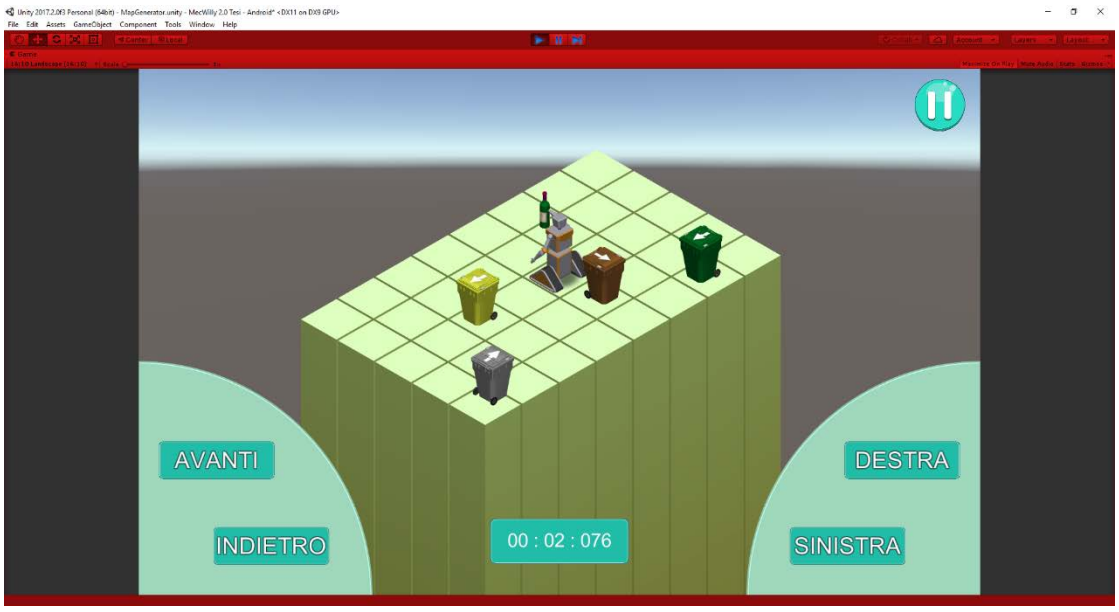

Figura 3.9: Livello 3

Nel quarto livello MecWilly deve raggiungere il bidone blu per gettare la carta che ha in mano. La griglia di gioco è 8x8 (figura 3.10).

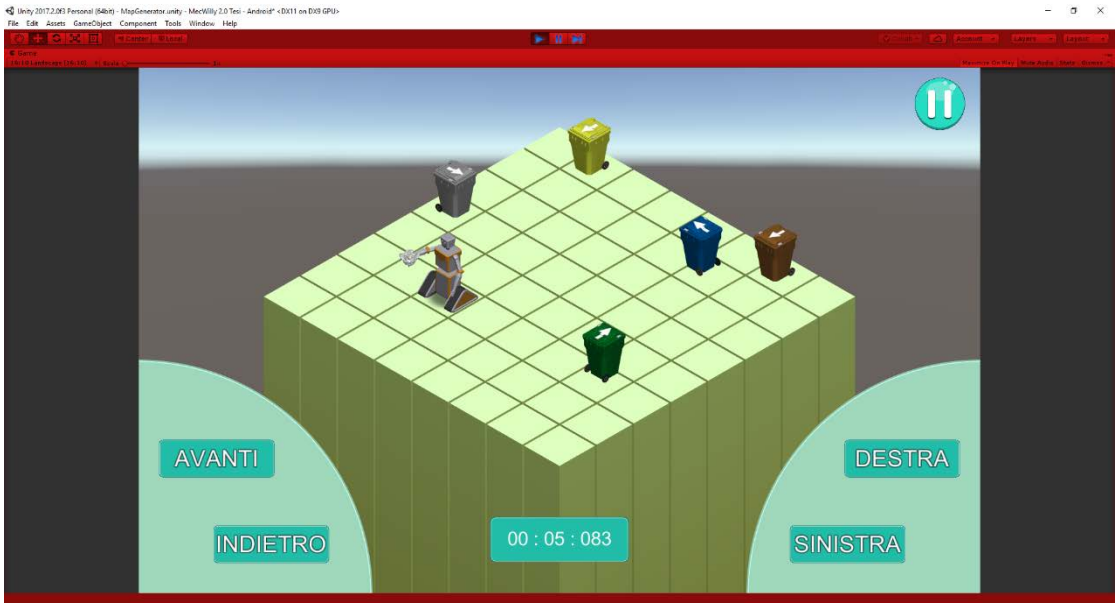

Figura 3.10: Livello 4

Nel quinto livello MecWilly deve raggiungere il bidone grigio per gettare la lattina che hai in mano. La griglia di gioco è 10x10 (figura 3.11). In “LE” è possibile giocare un livello creato nel level editor.

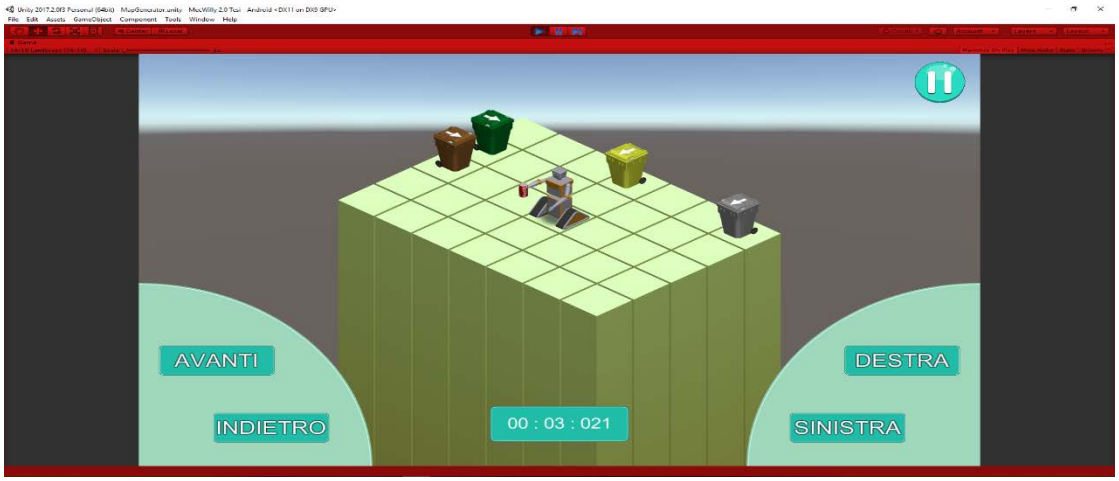

Figura 3.11: Livello 5

Capitolo 4

Conclusioni

Il videogioco presentato in questa tesi ha l'obiettivo di sviluppare e migliorare le abilità nell'orientamento topografico e spaziale dei bambini della scuola primaria. La realizzazione di una seconda versione di un videogioco tridimensionale che simulasse la sperimentazione con il robot fisico è stata fondamentale poiché è andata a risolvere dei problemi con la simulazione stessa, dati dalla mancanza di un comparto audio, di una visione non perfetta del mondo di gioco (a causa della telecamera) e dalla mancanza di dinamicità nella creazione dei livelli, e ha risolto problemi di ottimizzazione del gioco nel campo del mobile gaming.
Sviluppi futuri possono essere:

- Interazione vocale.
- Implementazione di una modalità multiplayer.
- Migliorie generali nel level editor.
- Test ed esperimenti che coinvolgano scuole primarie.

Bibliografia

[1]. Matteo Corradin. MecWilly3D: supporto all'orientamento topografico nei bambini in età scolare attraverso un serious game per la raccolta differenziata, 2016.

[2]. Sam S. Adkins. The 2012-2017 worldwide game-based learning and simulation-based markets. http://www.ambientinsight.com/Resources/Documents /AmbientInsight_SeriousPlay2013_WW_GameBasedLearning _Market.pdf, 2013.

[3]. Carnegie Mellon University. Website Taps into Human Factor. https:// www.cmu.edu/homepage /computing/2008/summer/games-with-a-purpose.shtml, 2008.

[4]. MecWilly Project. Mecwilly project. http://www.mecwilly.it/, 2010-2018

[5]. Benvenuti M. Mazzoni, E. A robot-partner for preschool children learning English using socio-cognitive conflict. educational technology & society, 2015.

[6]. Sapio B. Mazzoni E., Nicolò E. Children, multimedia content and technological artefacts: An exploratory study using text analysis tools. interactive technology and smart education, 2015.

[7]. D’Alessio Maria. Psicologia dell’età scolare, ed. Carocci, 2002.

[8]. Hans-Furth. Piaget per gli insegnanti, ed. Giunti Barbera, 1980.

[9]. David W. Johnson, Roger T. Johnson, Edythe J. Holubec. Apprendimento cooperativo in classe, ed. Erickson, 2015.

[10]. Vygotskij L.S. Mind in society: The development of higher psychological processes, 1978.

[11]. Paola Salomoni Elvis Mazzoni Catia Prandi, Silvia Mirri. Mecwilly in your pocket: on evaluating a mobile serious game for kids. http://ieeexplore.ieee.org/document/7543737/, 2013.

Ringraziamenti

A mio fratello Luca, la miglior persona e la miglior guida che potesse capitarmi durante tutto il mio percorso di vita. Anche se a volte non andiamo d'accordo e la pensiamo in maniera diversa su tutto, è la persona più buona che io conosca. Mi ha sempre aiutato nel momento del bisogno senza mai chiedermi nulla in cambio e mi ha sempre spronato a dare il massimo in tutti i campi, anche se non sempre gli ho dato ascolto…

A mia madre, senza di lei non avrei mai raggiunto questo traguardo così importante. Ha sempre dato tutto e anche di più per far sì che io fossi felice, insegnandomi giorno per giorno i veri valori della vita e guidandomi sempre verso le scelte migliori che potessi fare. Grande amante delle tradizioni, l'unica vera Donna/Befana/Babbo Natale della mia vita, mi ritengo estremamente fortunato ad avere lei come mamma, perché è perfetta con tutti i suoi pregi e i suoi difetti e non la cambierei per nulla al mondo.

A mio padre, la persona più simpatica, sorridente e felice che io conosca. Al mio migliore amico, colui che mi ha reso così come sono e alla quale mi ispiro sempre cercando di essere come lui un giorno, ma forse è un traguardo davvero irraggiungibile. Al miglior musicista che io conosca, facendomi "strimpellare" la chitarra quando le mie dita erano più piccole del manico, anche se i nostri gusti musicali non sono mai andati d'accordo. Al miglior informatico che io conosca, trasmettendomi fin da piccolo la sua passione per l'informatica e i videogiochi senza la quale non avrei mai scelto un percorso universitario e lavorativo come questo.

A nonna Nina, la persona più forte e la migliore cuoca che io abbia mai incontrato, dotata di un cuore d'oro, mi ha insegnato a vedere sempre il lato buono delle persone. Ogni volta che entro nel suo appartamento ha una storia nuova da raccontarmi ed è sempre molto felice quando viene a sapere una notizia positiva mia o di mio fratello.

A Daniela, Alessandra, Cristiana, Lucia, Silvia, Daiana, zia Gianna, zia Lina, zia Lucia e zio Peppino, con cui faccio sempre mille risate durante i pranzi e le cene di famiglia.

Ad Alessandro e Martina, i miei due nipotini, i più energici della famiglia. È sempre una gioia vederli giocare insieme e ogni volta ho sempre tanto da imparare da loro.

Al Presidente e all’ex segretario (ma amico) della Vas Mass Davide Vasile e Alessandro Magnani, che mi hanno dato ottimi consigli riguardo la scrittura della tesi.

A Fina, che grazie ai suoi erasmus mi ha fatto scroccare letti e cene in tutta Europa senza mai chiedermi nulla in cambio e a Silvio, con cui sono libero di parlare dei Pokèmon senza essere giudicato e a cui devo ridare Final Fantasy X da circa 6 anni.

A Mattia e Leonardo, che tra alti e bassi sono sempre stati al mio fianco dal giorno 0 in tutti i momenti tristi e felici delle nostre giornate.

A Angelo e Antonio, che considero i miei “secondi” coinquilini, sono riusciti a sopportarmi insieme a Vas e Magna per periodi molto lunghi nella loro casa e mi hanno fatto sentire sempre come se fosse la mia.

A Marco e Daniele, che sono stati costretti ad ascoltare trap tutto il giorno, tutti i giorni.

A Fabiana, Eleonora, Giulia, Virginia e Ruggio che sono così uniche da essere ciò che considero le mie migliori amiche e le mie “comare” preferite (si Ruggio, anche tu sei la mia “comara” preferita).

A Federica, che a partire dall’ultimo esame prima della laurea mi ha dato la forza per continuare a studiare e mettercela tutta, credendo in me fino all’ultimo.

Ai DogeLoopers, che mi hanno aiutato a studiare e superare più della metà degli esami e con cui ho passato le migliori serate a giocare e divertirci. Ricordatevi di non hodlare mai altrimenti facciamo la fine di Erik.

Al gruppo “Le Amiche Della Gang”, con le quali ormai faccio le più belle partite di poker della storia.

Agli iFools, con i quali sono cresciuto sia a livello artistico sia a livello umano. Con loro e le loro canzoni al mio fianco ricorderò sempre quant’era bella la vita a 16 anni e quanto è bello tutt’ora provare per suonare alle classiche tre serate all’Amenà.

Alla Nuke Crew. Se sul dizionario cercate il significato di fratellanza, band che spaccano, organizzazione e comicità, trovate la loro foto pre-Bolowood a Zona Roveri.

Al BOCA 2.0. Con loro ogni partita è un’emozione unica dato che riescono a vincere tranquillamente con le squadre più forti e a soffrire con quelle più deboli.

Al gruppo “Mura Bufu”, con cui sono certo che, anche se passassero 50 anni e non ci sentissimo da 49, mi basterebbe andare al Bounty per incontrare di nuovo tutti.

Ai Blink-182, che mi hanno insegnato come sembrare fighi suonando gli stessi tre accordi per 20 anni e che mi hanno fatto viaggiare sia in Italia che all’estero per ascoltare per la milionesima volta gli stessi 8 album.

A Vasto, anche se ti ho lasciato fisicamente, nel cuore ci sarai sempre e solo tu, il luogo migliore che questo mondo potesse mai regalarmi.

A Bologna, che mi ha offerto tutte le opportunità che non avrei mai sognato di avere e che continua a stupirmi ogni giorno di più.

All’Inter, che non riesco ad abbandonare anche se mi fa soffrire ed arrabbiare ogni settimana da quando ho 6 anni. Se non è amore questo…

A tutte le persone con cui ho condiviso un sorso di birra, una risata, una parata, una notte o 24 anni della mia vita.

Ma il ringraziamento più sincero lo voglio dedicare ai miei nonni che non ci sono più:

a nonna Anna di Fara, non esistono parole per descriverla, mi ha cresciuto donandomi tutto l’amore possibile e facendomi passare le giornate che io ricordo con più piacere. Mi hai visto partire per Bologna e spero che adesso veda il traguardo che ho raggiunto anche grazie a te, al tuo coraggio, al tuo sorriso contagioso e alla tua voglia di vivere;

a nonno Peppe, sono certo che saresti stato fiero di come è diventato il “birichino di nonno”;

a nonno Antonio, che non ho mai avuto il piacere di incontrare, ma grazie ai racconti della nostra famiglia sono convinto che saresti stato per me una persona eccezionale.

A tutti voi.