

Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica



Tesi Magistrale

**Ambient Intelligence per giochi immersivi
di strategia e d'azione**

Relatori:

Prof. Fulvio Corno

Ing. Luigi De Russis

Candidati:

Ten. Alessandro Messina

Ten. Fabio Runci

Anno Accademico 2014-2015

Sommario

Introduzione.....	1
1.1 Contesto Generale.....	1
1.2 Obiettivo della tesi.....	4
1.3 Struttura della tesi.....	6
Game concept.....	8
2.1 Background Storico	8
2.2 Gameplay	10
2.3 Lavoro di tesi	15
2.3.1 Descrizione modalità ON-FIELD.....	15
2.3.2 Schema esplicativo modalità On-Field.....	19
Strumenti e tecnologie.....	21
3.1 Panoramica generale degli strumenti disponibili.....	21
3.1.1 Piattaforme di sviluppo.....	22
3.1.2 Comunicazione	23
3.1.3 Hardware	24
3.1.4 Localizzazione Indoor	26
3.2 Descrizione strumenti e tecnologie utilizzate	27
3.2.1 Piattaforma Arduino	28
3.2.2 Unity	37
3.2.3 Photon Network	46
3.2.4 Estimote Beacons	54
3.2.5 Blender e Stampa3D.....	57
Progettazione e architettura.....	61
4.1 Panoramica generale.....	61
4.2 Implementazione.....	67

4.2.1 Applicazione Mobile	68
4.2.1.1 Connessione al cloud e sincronizzazione	69
4.2.1.2 Connessione Bluetooth e dati ambientali	74
4.2.1.3 Connessione Arduino e Sockets	76
4.2.1.4 Interfaccia utente e feedback di movimento	78
4.2.2 Applicazione Team Leader	84
4.2.2.1 Pannello iniziale	86
4.2.2.2 Selezione squadre	87
4.2.2.3 Lobby	88
4.2.2.4 Pannello di gioco	89
4.2.3 Arduino e sketch	97
4.2.3.1 Pistola Infrarossi	100
4.2.3.2 Pettorina Infrarossi	104
4.2.4 Stampa 3D	108
Valutazioni	111
5.1 Costo	112
5.2 Software	114
5.3 Hardware	116
5.4 Beacons e Ambient Intelligence	118
5.5 Sviluppi futuri	119
Conclusioni	123
Bibliografia	125

Elenco delle figure

Figura 1 - Schema dei ruoli di gioco.....	19
Figura 2 - Prodotti Arduino	30
Figura 3 - Arduino Yun	31
Figura 4 - Schematico Arduino Yun.....	32
Figura 5 – Led Arduino Yun.....	34
Figura 6 - Bottoni di Reset Arduino Yun	35
Figura 7 - Piani abbonamenti Photon Cloud.....	47
Figura 8 - Funzionamento Photon Cloud.....	49
Figura 9 – Player e ricezione dei pacchetti di aggiornamento	52
Figura 10 - Beacons Estimote	54
Figura 11 - Architettura	63
Figura 12 - Architettura di riferimento	67
Figura 13 - Parametri Photon Server	69
Figura 14 - Componenti utilizzate per un Player	71
Figura 15 - Metodo OnPhotonSerializeView	72
Figura 16 - Funzionamento comunicazione Bluetooth.....	74
Figura 17 - Metodo SearchForEstimote.....	75
Figura 18 – Metodo di invio e ricezione stringhe.....	77
Figura 19 - Interfaccia iniziale applicazione mobile	79
Figura 20 - Schermata inserimento codice dell'equipaggiamento	80
Figura 21 - Simboli connessione con equipaggiamento riuscita/fallita	81
Figura 22 - Interfaccia di gioco smartphone	81
Figura 23 - Pannello che segnala al giocatore di essere stato ucciso.....	82
Figura 24 - Funzione per la gestione dell'accelerometro dello smartphone	83
Figura 25 - Creazione della stanza di gioco.....	86
Figura 26 - Pannello di scelta del team di appartenenza.....	87

Figura 27 - Pannello indicativo dei giocatori connessi divisi per squadre	88
Figura 28 - Pannello dei punteggi.....	90
Figura 29 – Pannello Players dell'applicazione web.....	91
Figura 30 - Dati visualizzati dal Team Leader appartenenti al componente di squadra.	92
Figura 31 - Macchina a stati dell'animazione del player	93
Figura 32 - Mappa dislocazione Beacons Estimote.....	95
Figura 33 - Schermata che segnala la sconfitta.....	96
Figura 34 - Schematico pistola infrarossi	98
Figura 35 - Schematico pettorina infrarossi.....	99
Figura 36 - Codice funzionamento emettitore infrarossi	101
Figura 37 - Libreria bridge yun.....	103
Figura 38 - Codice per temporizzazione.....	103
Figura 39 - Codice funzionamento ricevitore infrarossi.....	105
Figura 40 - Codice God Mode	106
Figura 41 - Codice ritorno in vita di un giocatore	107
Figura 42 - Modello su Blender.....	108
Figura 43 - Modello su MakerBot Desktop	109
Figura 44 - Pistola 3D con circuiteria interna.....	110
Figura 45 - Pettorina 3D con circuiteria interna	110
Figura 46 - Kit infrarosso.....	110

Capitolo 1

Introduzione

1.1 Contesto Generale

Grazie al crescente sviluppo tecnologico e informatico termini come “Ambient Intelligence” e “Internet of Things” sono, ormai, di uso quotidiano. Consciamente o meno, la maggior parte dei dispositivi che utilizziamo giornalmente ricade perfettamente in quella che può essere una definizione di “ambient intelligence”. Definizione che racchiude in sé l’utilizzo di interfacce intelligenti, potenziamenti delle possibilità a disposizione dell’utente, supporto ai servizi e interazione umana. Basti pensare che quasi ogni strumento che utilizziamo presenta facili interfacce grafiche che guidano le nostre azioni quotidiane o migliorano determinati servizi. Infatti, l’essere umano sfrutta al meglio i device a disposizione per velocizzare determinate routine e migliorare il risultato ottenibile. Sembrerebbe proprio che la società stessa e lo sviluppo dell’uomo, mirino allo stesso obiettivo che l’Ambient Intelligence si pone, in altre parole, avere un ambiente sensitivo e reattivo che fa della nostra presenza un dato di input da elaborare al fine di migliorare l’azione e la vita quotidiana. Per ottenere questo risultato è necessario che l’utente sia “immerso” in una rete circondata da device intelligenti, non invasivi, che possano percepire, anticipare e magari adattarsi alle necessità degli utenti. Si può affermare quindi, che l’Ambient Intelligence non è una nuova tecnologia, ma una sorta

di nuovo approccio che utilizza, come informazione centrale, le persone e le loro necessità. Un approccio che include in sé tre principali fattori che cooperano per rendere ambiente e dispositivi intelligenti:

Presenza computazionale: i dispositivi che utilizziamo ogni giorno devono contenere microprocessori o unità computazionali per favorire un funzionamento più ampio.

Comunicazione: tutti i dispositivi, immersi nell'ambiente, devono essere in grado di comunicare tra loro in maniera wireless al fine di avere una maggiore interazione.

Interfacce intelligenti: sarà necessario favorire al massimo la semplicità di utilizzo per l'utente, fornendo interfacce grafiche minimali, belle da vedere e in cui risulta facile capire il funzionamento generale senza averne conosciuto a priori le possibilità.

A questi fattori ne va aggiunto un quarto, che pur se secondario risulta importante nel contesto generale, l'invisibilità. Il concetto di invisibilità, va inteso in senso più ampio. L'architettura generale dell'ambiente intelligente dovrà apparire il più possibile invisibile all'utente, in modo tale da favorire una maggiore naturalità del contesto nel quale è immerso. L'utente, entrando in un ambiente intelligente, non dovrà sentirsi alienato ma parte integrante di quel sistema e capace di utilizzarlo rapidamente.

Di conseguenza, tutto deve essere mirato a favorire l'esperienza dell'utente, facilitando le operazioni principali delle sue giornate, il divertimento, la ricerca di determinati punti di interesse, fornendo una rete di dispositivi capaci di collegare tra di loro, accessibili da ogni interfaccia. Oggi vi sono numerosissimi esempi di creazione e introduzione del concetto dell'Ambient Intelligence in tantissimi settori di interesse e si intravede come, questo concetto, rappresenta la base delle future applicazioni. È possibile trovare gli esempi più lampanti nella domotica in cui l'utilizzo di interfacce intelligenti permette agli utenti di avere uno spettro completo di tutti i parametri e funzionalità delle proprie abitazioni. Si potrà effettuare il monitoraggio della temperatura, check di funzionamento di alcune componenti, generazioni di segnali: tutti *ordini* che forniranno un feedback all'utente, sulla base delle sue richieste, agendo direttamente da una comoda interfaccia residente nel dispositivo cellulare o in un PC.

Si intuisce quindi come la componente sensoriale ricopra una fondamentale importanza al fine di ricreare un ambiente intelligente. Sensori di prossimità, di temperatura, di suono, di luce, di gas o pressione sono solo alcuni esempi delle possibilità infinite che si possono utilizzare per rendere l'ambiente intelligente. Fornendo intelligenza, tramite l'utilizzo di microprocessori, sarà possibile immergere l'utente in un ambiente totalmente interattivo, conscio della sua presenza e dell'influenza di ogni altro sensore.

Quello dell'Ambient Intelligence è, dunque, un obiettivo molto ampio che può applicarsi e trovare fondamento in qualsiasi ambito sociale e persino a quello del gaming. Il gaming è divenuto, da parecchi anni, un momento fondamentale nella vita quotidiana dell'uomo che cerca nei videogiochi occasioni di relax, avventura e azione. Con il passare degli anni si è evoluta la grafica dei videogiochi e, da semplici ed essenziali, si è arrivati oggi a veri capolavori grafici che sfiorano il dettaglio reale. A essere approfondito è stato anche il concetto dell'immersività dell'utente. Basti pensare alle varie tecnologie proposte da parte delle grandi software house per "includere", in qualche modo, l'utente e il suo movimento all'interno del mondo virtuale: periferiche di base, WiiMote, PSEye, Microsoft Kinect. E questi sono solo alcuni degli esempi più calzanti, sviluppati per migliorare la componente immersiva dei videogiochi. Molto importanti sono, anche, gli esempi di giochi che hanno fatto della grande immersività un punto di forza per preparare e allenare professionisti nel mondo reale. L'esempio più calzante è quello dato da Gran Turismo, famoso gioco di gare automobilistiche, molto immersivo, utilizzato dalle varie accademie di corsa per preparare i piloti del domani. L'aver un ambiente intelligente all'interno di un videogioco, potrebbe cambiare in maniera definitiva l'esperienza di gioco. Gli sviluppatori potrebbero utilizzare diverse componenti e sensori al solo scopo di aumentare le possibilità del giocatore, potenziare features esistenti e soprattutto migliorare la stessa immersività.

1.2 Obiettivo della tesi

Partendo dal concetto di immersività ci si è chiesti quanto, effettivamente, l' Ambient Intelligence può influire nel creare nuove tipologie di gioco in cui l'utente riesce a sentirsi parte del background ricreato e interagire con quest'ultimo in maniera attiva, oppure, modificare i giochi esistenti tramite l'inserimento di nuovi device che rendano l'interazione dell'utente una parte centrale e fondamentale. Localizzazione, interazione con l'ambiente e comunicazione sono alcuni dei concetti da sfruttare se si vuole immergere l'utente in un vero ambiente interattivo nel quale le sue azioni hanno delle ripercussioni nel mondo di gioco. A tale scopo, è stato ideato e creato un nuovo game concept che ha ricoperto il ruolo di un vero esperimento, grazie al quale ci è stato possibile capire quanto l' Ambient Intelligence può essere sfruttato in nuove tipologie di gioco o modificare quelle già esistenti. L'idea del concept è nata partendo dal Laser Game, gioco che, ai giorni nostri, è molto facile trovare in ogni grande città italiana e che, di per sé, rappresenta una tipologia di gioco in cui vengono sfruttate molte componenti sensoriali. Il Laser game prevede due squadre formate da diversi giocatori armati di fucile a infrarossi e pettorina sensibile IR, che combattono tra di loro all'interno di un edificio. L'obiettivo del gioco è essenzialmente accumulare il punteggio più alto colpendo i giocatori della squadra avversaria. Ultimamente, numerose compagnie, specializzate nel creare e fornire equipaggiamento per Laser Game, stanno cercando di potenziare le possibilità di gioco andando a utilizzare nuove componenti come mine di prossimità (sempre infrarossi) o nuove modalità di gioco. Si è voluto potenziare questo modo di giocare tra individui, andando a inserire possibilità di gioco e caratteristiche che sfruttavano l' Ambient Intelligence. Per questo, non avendo a disposizione l'equipaggiamento già funzionante, si è ricreato un laser game utilizzando hardware e tecniche accessibili, costruendo i dispositivi di gioco, aggiungendo componenti di localizzazione e la presenza fisica dell'utente sia nel mondo reale che in quello fittizio. È stato creato un modo di giocare nel quale sono disponibili differenti ruoli all'interno della

stessa partita, in più è contemplata la presenza in rete di ogni utente e l'interazione tra di loro, non solo tramite infrarossi ma anche tramite attiva comunicazione sotto forma di messaggistica e segnalazioni di vario genere. Si è cercato quindi di immergere l'utente all'interno di un'arena intelligente, fornendo metodi di interazione tramite interfacce grafiche, in cui ogni soggetto può comunicare tra loro e in cui sono presenti sensori di prossimità al fine di far diventare l'utente parte integrante dell'azione di gioco.

Il processo di creazione si è sviluppato in varie fasi comuni a quelle della produzione di un normale gioco, però, si è avuta una maggiore attenzione alla fase di progettazione che si è rivelata la parte principale del nostro caso di studio. Questo è stato dovuto principalmente al fatto che, data l'ambizione del progetto, si voleva cercare di ricreare un'architettura il più possibile funzionante e solida in modo tale da permettere un'adeguata valutazione. Il gioco, inoltre, ruota attorno ad un ambiente fittizio che richiama fortemente componenti futuristiche nelle quali ogni scelta dei giocatori può influenzare il corso di ogni partita. Questo background virtuale ha permesso di definire l'idea totale del gioco nell'ottica di una realizzazione più completa del progetto. Questo ha permesso di fissare le basi del vero oggetto della tesi. Una volta stabilito il background, si è passati alla definizione delle regole. Questo processo è avvenuto in maniera libera, evitando di soffermarsi su quelle che potevano essere le limitazioni progettuali e dedicandosi, in maniera importante, a tutte quelle features che permettevano di raggiungere la massima immersività ed interazione con l'ambiente. Una volta realizzato il prototipo del gioco, si è effettuata una valutazione oggettiva su quelle che sono stati i processi di creazione, il funzionamento e le tecnologie utilizzate.

1.3 Struttura della tesi

Nei prossimi capitoli verranno approfondite le fasi della tesi, per fornire una descrizione approfondita di tutte le fasi che hanno ricoperto i mesi di sviluppo. Per fornire uno spettro molto ampio, ogni fase di sviluppo non verrà tralasciata e ci si soffermerà soprattutto sulle scelte progettuali e implementative che hanno permesso di raggiungere il prototipo finale e soddisfare l'obiettivo della tesi.

- Nel capitolo 2 “*Game Concept*”, si è voluto ricreare un documento, tipico e importante negli step di creazione videoludica, in cui viene descritto il background e l'ambientazione attorno a cui ruota il gioco creato, le regole di base e i funzionamenti.
- Nel capitolo 3 “*Strumenti e Tecnologie*”, si è voluta effettuare una panoramica di tutti gli strumenti e tecnologie, disponibili attualmente in commercio, che sono stati valutati in fase di progettazione come possibili scelte definitive. Si effettuerà poi una descrizione dettagliata, con pregi e difetti, di ogni strumento o tecnologia scelta e utilizzata nel corso della tesi.
- Nel capitolo 4 “*Architettura e Implementazione*”, si effettuerà una descrizione generale di quella che è la fase progettuale dell'architettura e messa impiego, di tutti i soggetti che ne fanno parte, specificandone le motivazioni. Infine verrà mostrato come l'architettura è stata implementata utilizzando gli strumenti descritti nel Capitolo 3.
- Nel capitolo 5 “*Valutazione e Futuri Sviluppi*”, verrà effettuata una valutazione oggettiva del prototipo creato, andando ad analizzare il vero funzionamento in maniera critica e verificando le aspettative iniziali di ogni strumento e tecnologia utilizzata. Verranno inoltre descritti i possibili e futuri

sviluppi dell'applicazione, che potranno essere facilmente implementati in caso di una vera realizzazione.

Capitolo 2

Game concept

2.1 Background Storico

Il gioco è ambientato nel Pianeta Terra, in un futuro prossimo, dove l'uomo ha compiuto passi da gigante in termini di sviluppo e tecnologia. Grazie a prosperità e benessere, l'umanità gode di un lungo periodo di pace. Tuttavia un giorno, che oggi è chiamato **Day 0**, nonostante fosse in atto una costante osservazione dello spazio visibile, una grande tempesta di meteore ha colpito la superficie terrestre. I più anziani la ricordano come una pioggia di pietre verdi e fosforescenti, causa di morte e distruzione. La pioggia di meteoriti ha distrutto gran parte di quello che l'uomo aveva creato fino a quel momento, producendo dei grandi buchi in quelle che erano le vie di comunicazione più battute. Come sempre, però, l'uomo ha trovato il modo di risollevarsi ricostruendo città, vie di comunicazione, economia e stabilità.

Dopo qualche settimana un gruppo di ricercatori ha deciso di avventurarsi, nei territori maggiormente colpiti dai meteoriti, al fine di scoprire l'entità e la natura di queste pietre verdi. Scoprirono, dunque, che il materiale in questione era dotato di un grandissimo potenziale e che, se usato correttamente, poteva permettere di sprigionare una grande energia.

Spinte dalla ricerca di potere, le varie comunità hanno iniziato a combattere per contendersi il nuovo materiale scoperto. Inizialmente queste contese furono verbali e sempre più frequenti tra i politici, ma col diffondersi della notizia anche le persone comuni iniziarono a litigare per piccoli pezzi di pietre verdi.

I conflitti con il passare del tempo si sono inaspriti sempre di più raggiungendo il culmine con l'omicidio di un intero team appartenente a una delle comunità più influenti del pianeta. Questo evento, come un effetto farfalla, ne causò degli altri e il mondo conobbe nuovamente le parole odio e morte. I vari conflitti generati, hanno plasmato quelle che sarebbero state le comunità principali del domani che si sarebbero contese il dominio del materiale maledetto. Con il passare del tempo le comunità acquisirono un'identità più vera e distinta che si immedesimava in una diversa vision:

- **Bellicatio:** è una fazione creata da un vecchio colonnello americano che aveva riunito i soldati rimasti sotto un unico vessillo. Non ha un ideale preciso e la sua formazione è principalmente militare. Gli uomini e le donne appartenenti a questa fazione sono spesso utilizzati come mercenari da parte delle altre fazioni.
- **Nuova repubblica:** è una fazione democratica che mira al ritorno alla pace sotto il comando di una repubblica di vecchio stampo. Coloro che ne fanno parte credono che il materiale verde debba essere usato solo per lo sviluppo della società e delle tecnologie e che tutta l'umanità dovrebbe usufruire dei vantaggi creati. Vestono uniformi Blu.
- **Nuovo orizzonte:** è la fazione più piccola tra le tre ma non la meno potente. Rappresenta la fazione più tecnologica e gode di un esercito di soli droidi. È stata creata da un gruppo di persone che credono di poter comunicare con entità aliene, a causa di questo, si chiamano "I Prescelti" e pensano di essere gli unici a meritare l'utilizzo del materiale verde. Vestono uniformi Verdi, che richiamano il colore del materiale.

In questo mondo devastato, il giocatore sceglierà la sua fazione di appartenenza e prenderà le vesti di un eroe di guerra al quale è stata affidata la gestione di un avamposto. La sua missione è quella di potenziare l'avamposto, collezionare il materiale verde, distruggere i nemici e garantire l'egemonia assoluta alla fazione appartenente.

2.2 Gameplay

Una volta selezionata la fazione di appartenenza il giocatore dovrà gestire il suo avamposto selezionando, in primis, gli edifici che decide costruire o sviluppare. Ogni edificio disponibile è specializzato nell'ottenimento di diversi bonus: unità militari, estrazione di risorse, laboratori di ricerca etc. Grazie a questa specializzazione, il giocatore avrà la possibilità di avere una maggiore personalizzazione del suo stile di gioco, andando direttamente a influire su quelle che sono le decisioni intraprese in ambito di sviluppo e tecnologia. Essendo un gioco Online, l'avamposto del giocatore rappresenterà un vero e proprio mini-cosmo all'interno della sua fazione di appartenenza, in cui le scelte intraprese saranno di fondamentale importanza. Inoltre, l'utente avrà la possibilità di comunicare con gli altri giocatori sia in maniera amichevole (chat, messaggistica e commercio) che in maniera offensiva, dichiarando l'attacco o difendendosi da uno di essi. La mappa di gioco sarà infatti caratterizzata da una rappresentazione geopolitica di quelli che sono i territori appartenenti alle fazioni concorrenti; l'obiettivo del giocatore sarà quello di ottenere, assieme all'aiuto degli altri compagni di fazione, più territori possibili al fine di garantire la vittoria alla fazione di appartenenza.

I territori conquistabili saranno inizialmente suddivisi in:

- **Territori di fazione:** 2-3 territori “madre” della fazione di appartenenza non conquistabili dalla fazione antagonista
- **Territori Neutrali:** territori inizialmente “liberi” e conquistabili da parte dei giocatori.

L’operazione di conquista di un territorio libero avviene direttamente in-game. Si ipotizza che all’interno del territorio vi siano forze ostili pilotate da IA da dover affrontare, prima di poter ottenere il territorio scelto. Per evitare problematiche legate alla concorrenza di attribuzione dei territori e conquista, verrà ideato un sistema di uniformazione della difficoltà a seconda del livello del giocatore. In sostanza, la conquista dei territori neutrali richiederà dei **tier** di unità che verranno adattati a seconda del livello del giocatore, per evitare che i giocatori più deboli siano svantaggiati. In ogni caso, data l’alta necessità di collaborazione tra giocatori e anche competitività, il gioco dovrà disporre di vari server o **rooms** in cui ospitare i giocatori dello stesso paese di appartenenza. Ogni server rappresenterà, dunque, un’istanza a se stante di quella che è l’ambientazione di gioco e, all’interno di quest’ultima, le scelte intraprese saranno totalmente indipendenti e limitate al server/room. I server serviranno a evitare sovraffollamento ma, anche, a bilanciare il carico di giocatori in modo tale da fornire un’esperienza gradevole sia a giocatori veterani che neofiti. La progressione del gioco, in caso di esaurimento del contenuto in-game da parte dei giocatori, verrà garantita grazie al concetto di **Classifica**. In altre parole, la conquista di un territorio da parte di un giocatore gli permetterà di acquisire punti per poter scalare la classifica globale del server di appartenenza. La classifica, oltre a rilevarsi come un metodo di fama e grandezza per i giocatori, sarà molto utile per gli stessi sviluppatori che avranno uno strumento rapido per capire quando e come, rilasciare i nuovi contenuti per il progresso del gioco.

L'obiettivo comune della fazione è, quindi, dominare sulle altre tramite la conquista dei territori; nel caso in cui il territorio scelto sia neutrale l'azione viene svolta in maniera meccanica e senza ripercussioni ma il discorso cambia radicalmente nel caso in cui un giocatore decida di attaccare il territorio appartenente alla fazione avversaria. In questo caso l'azione di attacco può essere dichiarata da tutti quei giocatori che avranno raggiunto un adeguato livello di **Influenza**. L'influenza è una sorta di moneta "spendibile", ottenibile tramite lo sviluppo del proprio avamposto, numero di territori conquistati o tramite il completamento di obiettivi in-game. Lo scopo, dell'attaccare interi territori di fazioni avversarie, è quello di aggiudicarsi il dominio del territorio e di conseguenza i vantaggi derivanti da quest'ultimo. Ogni territorio, infatti, sarà dotato di caratteristiche peculiari che, se in possesso, permetteranno di ottenere vantaggi per l'intera fazione (es: nuove risorse o nuove tecnologie o nuove unità). In questo modo si facilita la coesione tra le fazioni e la collaborazione tra i giocatori.

Se l'azione di attacco viene dichiarata, si andranno a creare 16 slot occupabili dai giocatori, suddivisi in metà per la parte offensiva e metà per la parte difensiva. Il primo slot della controparte offensiva sarà occupato dal dichiarante dell'attacco, mentre quello della controparte difensiva sarà occupato dal giocatore con maggiore punti influenza (o da un suo sostituto). Gli slot rimanenti saranno occupabili in maniera libera da tutti i giocatori appartenenti alla fazione corrispondente e saranno gestiti dal leader dell'azione. Una volta occupati tutti gli slot, la battaglia per la contesa del territorio potrà giocarsi in due modalità:

- **Modalità On-Line:** tutti i giocatori, che occupano slots, inviano le loro truppe e il vincitore verrà stabilito su una differenza di potenza tra le due fazioni, in termini di numero e qualità delle unità in gioco. Questa modalità è già stata vista e viene ripresa da tantissimi giochi online e consiste di un semplice calcolo matematico effettuato sulla base delle regole di gioco.
- **Modalità On-Field:** la battaglia avrà luogo nel mondo reale utilizzando il layer superiore che il gioco mette a disposizione.

Qualsiasi sia la modalità scelta, il territorio verrà comunque assegnato al vincitore della contesa, assieme ad alcuni premi accessori. Considerando che la modalità On-Field richiede una maggiore preparazione e impegno, rispetto alla modalità online, verrà incentivata con maggiori premi inviati ai vincitori della contesa.

Anche nella scelta dei premi da assegnare, si è deciso di influenzare in maniera pesante tutti i giocatori appartenenti alla fazione. Tra i Bonus/Malus disponibili avremo:

- **CRISI:** status assegnato al territorio attaccante nel caso in cui l'operazione di attacco sia fallita. I membri della fazione che hanno partecipato all'attacco avranno un malus del 20% sulla produzione delle risorse per una settimana. Inoltre ogni giocatore perderà il 30% delle risorse contenute all'interno dei loro depositi.
- **CRISI(difensiva):** nel caso in cui i difensori perdano il conflitto, oltre a perdere il territorio conteso, riceveranno una diminuzione del 10% sulla produzione di risorse per una settimana.
- **EGEMONIA:** status assegnato al territorio difensore nel caso in cui l'operazione di difesa abbia avuto successo. In questo caso i membri della fazione otterranno un bonus del 20% sulla produzione delle risorse e verrà assegnata, una tantum, una porzione di risorse bonus.
- **CALO DI MORALE:** status assegnato sia agli attaccanti che ai difensori in caso di perdita del conflitto. Tutte le unità militari subiranno un malus del 20% all'efficacia di attacco/difesa.
- **LEADER INDISCUSSO:** il vincitore del conflitto otterrà il 30% di influenza del leader nemico.

Tra i vari bonus va aggiunta l'attribuzione del territorio, alla fazione attaccante, in caso di attacco riuscito. Tutti questi bonus avranno percentuali maggiorate se la contesa del territorio viene effettuata tramite la modalità on-field.

La modalità ON-FIELD viene “nutrita” da giocatori grazie all’identità data alla fazione Bellicatio. Questa, si differenzia dalle altre perché non si dovrà più gestire un avamposto e conquistare territori, bensì si gestirà un soldato-virtuale che rappresenta il giocatore. Lo scopo di questo sistema è quello di permettere a qualsiasi giocatore, sia facente parte del mondo strategico che non, di poter giocare in qualsiasi momento agli eventi organizzati. Gli eventi corrispondono a conflitti gestiti con la modalità On-Field e saranno visualizzabili in un apposito browser che permette l’utilizzo di filtri per data, ora e giorni preferiti dall’utente. Quando l’utente selezionerà un evento disponibile, che si adatta alle sue esigenze, potrà inviarne l’adesione. Le conferme delle adesioni verranno gestite direttamente dai Team-Leader di ogni squadra. In questo modo, ogni Team Leader potrà scegliere quale giocatore utilizzare all’interno della sua squadra, per cercare di garantirsi la vittoria nel campo reale di gioco. Il sistema, inoltre, prevede un meccanismo di **feedback** per evitare mancate adesioni o partecipazioni all’evento reale di gioco. Questi feedback verranno rilasciati da tutti i giocatori che hanno partecipato ad un determinato evento e serviranno a fornire un servizio di assicurazione della qualità di un giocatore e anche di garanzia. Questo sistema rasenta molto quello già visto per i vari siti web di sharing di esperienze e servizi quali TripAdvisor o BlaBlaCar ma, in questo caso, si parlerà di serietà del giocatore nel partecipare all’evento e di esperienza nel gioco.

Se il giocatore avrà terminato correttamente una partita, al termine di questa gli sarà assegnata dell’esperienza e del denaro utilizzabile per potenziare il suo soldato virtuale. I potenziamenti sono di diverso tipo e gli permetteranno di modificare la sua esperienza di gioco influenzando direttamente sul danno inflitto, sulla vita, su potenziamenti attivabili e sulle munizioni trasportabili.

Nulla vieta che, qualsiasi utente, è libero di poter scegliere di giocare entrambe le modalità. Infatti, la norma Bellicatio è stata pensata per trovare sempre giocatori disponibili a disputare match del Laser-Game e anche permettere a questi ultimi di poter giocare senza il bisogno di un’attiva compagnia, data la natura del gioco.

2.3 Lavoro di tesi

In questa parte viene descritto tutto ciò che risulta essere oggetto della tesi, quindi, ciò che è stato realizzato. L'analisi verrà fatta senza scendere nei dettagli implementativi, di cui ci occuperemo in seguito, ma fornendo un quadro generale di quello che è stato portato a termine.

2.3.1 Descrizione modalità ON-FIELD

La modalità ON-FIELD rappresenta la vera anima del gioco, nonché lo sforzo principale di questa tesi. Infatti, sia la parte strategica che il background ideato servono ad accompagnare il giocatore verso questa modalità innovativa di interazione tra gli utenti. Ogni scelta fatta dall'utente, dalla fazione scelta, permetterà di avere accesso e sbloccare, vari potenziamenti e upgrade utilizzabili direttamente all'interno di questa modalità. Essendo lo sforzo principale di questa tesi, si è cercato di renderla il più possibile indipendente dal resto del gioco, in modo tale da poter essere utilizzata anche in maniera "amichevole" e senza ripercussioni in-game.

Come è già stato affermato, questa modalità fonda le sue basi sulle meccaniche del Laser-Game. Come quest'ultimo, infatti, vi saranno due squadre composte da vari giocatori equipaggiati da pistole laser e pettorine sensibili a infrarossi. L'obiettivo di ogni squadra è quello di sconfiggere i nemici tramite l'utilizzo delle pistole laser. Ogni partita sarà scandita da un timer di gioco di 15 minuti che determinerà la fine dei conflitti tra i

giocatori. La squadra, che avrà eseguito il maggior numero di vittime, sarà dichiarata vincitrice. Oltre a questa meccanica base, già presente nelle nostre città, si è voluto migliorare la giocabilità influenzando su comunicazione, presenza nell'ambiente e immersività. Infatti, ogni utente, oltre ad essere dotato della pistola e pettorina a infrarossi, porterà con sé, fissato al braccio, il suo smartphone, al quale è stata precedentemente installata l'applicazione del gioco. Questa applicazione, dotata di una semplice interfaccia, effettuerà numerosi compiti, tra cui:

- **Fornire una semplice interfaccia rappresentante lo status del giocatore:** saranno visualizzati vita e munizionamento in quell'istante, che verranno continuamente aggiornate sulla base degli eventi di gioco
- **Collegamento con l'equipaggiamento Pistola e Pettorina:** necessario per fornire il feedback diretto in caso in cui si spara o si è colpiti.
- **Raccogliere informazioni sull'ambiente, grazie al Bluetooth:** nella stanza di gioco, verranno dislocati dei Beacon che verranno continuamente interrogati dal device per ottenere informazioni di potenza di segnale, successivamente elaborata in distanza in metri.
- **Comunicare tramite bottoni di emergenza e messaggi con il Leader della squadra**
- **Utilizzare i potenziamenti acquisiti dal suo soldato-virtuale**

L'ultima feature immessa, introduce la nuova figura presente all'interno di questa modalità, il Team Leader. All'interno del mondo fittizio creato, il Team Leader rappresenterebbe il giocatore con maggiore influenza sia della controparte attaccante che di quella difensiva. Questa figura è stata introdotta con lo scopo di creare un nuovo ruolo all'interno delle partite che abbia la responsabilità di gestire la squadra, fornendo ordini, consigli e potenziamenti.

Infatti, a partita avviata, il Team Leader, direttamente dal browser del suo computer, potrà:

- **Osservare lo status dei giocatori appartenenti alla sua squadra:** un indice indicherà il numero di giocatori presenti, la vita e le munizioni di ognuno, il nome del giocatore e tramite un Animazione 3D se il giocatore sta correndo o no.
- **Monitorare lo stato attuale della partita:** vi sarà un menu dedicato con punteggi ottenuti e tempo rimanente alla fine della partita
- **Inviare messaggi di Testo ai giocatori:** grazie a questa possibilità, il team Leader potrà inviare ordini o consigli ai suoi giocatori direttamente dalla sua postazione di gioco
- **Visualizzare la mappa di gioco:** nonostante il Team Leader sarà fisicamente lontano dai giocatori del Laser Game, riuscirà comunque a capire la situazione di gioco grazie all'utilizzo dei Beacons.
- **Inviare potenziamenti e munizioni:** il team Leader potrà decidere di utilizzare i potenziamenti presenti nelle sue scorte per aiutare i membri della sua squadra

Si intravede subito come questa modalità spicchi per un alto livello di comunicazione tra i giocatori con ruoli totalmente diversi. Nonostante si tratti principalmente di un gioco di azione, grazie a questo layer, il laser game acquisisce una grande componente strategica effettuata da parte del Team Leader di ogni squadra. Ogni Team Leader avrà infatti a disposizione un numero limitato di Kit Vita e Kit Munizioni che potranno essere attribuiti “al volo” ad ogni utente selezionato e, di conseguenza, bisognerà farne un uso parsimonioso. Lo stesso vale per i potenziamenti che, se utilizzati, avranno un countdown

da rispettare prima di poter essere nuovamente utilizzabili. Anche la comunicazione diventa importante grazie alla messaggistica o ai pulsanti di aiuto e pericolo, in questo modo ogni giocatore potrà comunicare una situazione pericolosa che potrebbe portare alla morte e di conseguenza richiedere assistenza al Team Leader. Inoltre, sfruttando le potenzialità dei device è stata migliorata l'immersività, garantendo ai Team Leader una visione completa di quello che sta accadendo durante la partita. Ogni giocatore sarà rappresentato da un modello 3D che sarà animato sulla base dei suoi movimenti. Anche l'ambiente presenta una grande importanza. I beacon, infatti, sono stati sfruttati al meglio, nonostante siano stati rilevati come una tecnologia tutt'ora scarna. I beacon serviranno a fornire un'individuazione approssimata della posizione dei giocatori, fornendo ulteriore immersività e strategia ma avranno un ruolo fondamentale anche per il giocatore stesso che prenderà parte fisicamente alla partita. Infatti, ogni squadra avrà a disposizione un Beacon madre che rappresenterà la "Base" della squadra. Questo Beacon Madre, che sancirà il luogo di inizio, sarà il punto di riferimento per i giocatori uccisi. Non appena i punti vita di un giocatore saranno terminati, dovrà precipitarsi il più possibile vicino al Beacon Madre della sua squadra di appartenenza per poter resuscitare e ritornare al combattimento. Tutta la partita, come già accennato, sarà scondita da un Timer di gioco (approssimativamente di 15 minuti) che determinerà lo slot temporale della partita. Una volta terminata, saranno sanciti vincitori e vinti tramite la visualizzazione di un messaggio direttamente nei dispositivi dei giocatori.

2.3.2 Schema esplicativo modalità On-Field

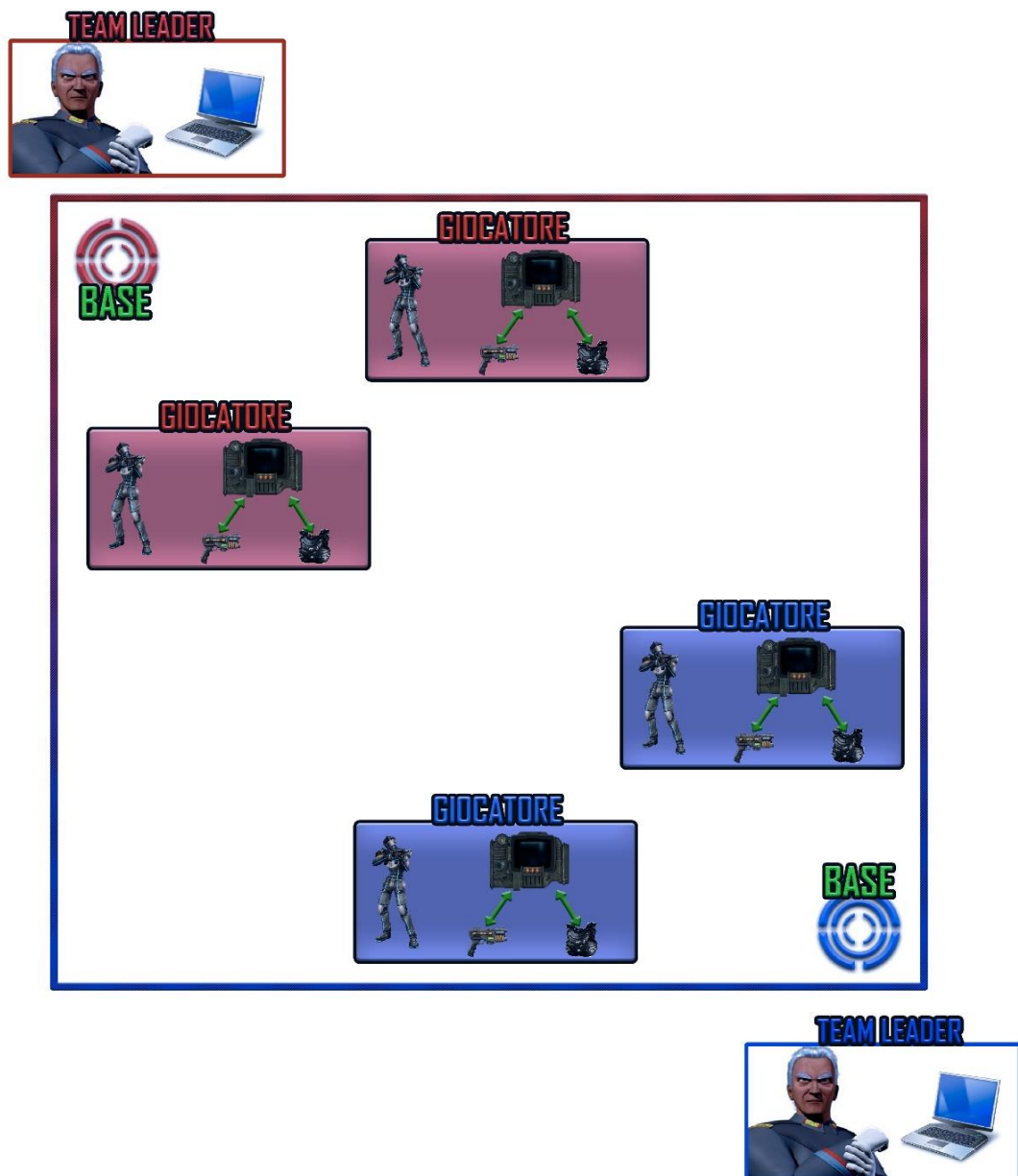


Figura 1 - Schema dei ruoli di gioco

La figura 1 mostra il funzionamento logico del gioco, definendo i soggetti e i device che partecipano ad una partita della modalità On-Field. Vi sono dunque due squadre, indicate dai colori rossi e blu, capitanate da un Team Leader che ha la possibilità di monitorare la situazione nell'arena di gioco tramite il suo computer.

Il suo compito è quello di essere sempre in contatto con i giocatori della sua squadra, inviare rifornimenti e potenziamenti, visualizzare la posizione approssimata sulla mappa grazie ai punti di interesse e inviare ordini a ogni soldato.

Il giocatore che si trova sul campo da gioco, invece, è equipaggiato con un dispositivo portatile (tipicamente uno smartphone) sul quale può visualizzare i suoi parametri (vita e munizionamento) e gli ordini inviati dal comandante. Inoltre, tramite la pressione di un tasto dedicato, può inviare una segnalazione al proprio Team Leader situazioni di pericolo.

Nella mappa del gioco, vengono dislocati vari punti di interesse che servono a fornire al Team Leader un feedback sulla posizione di ogni soldato della propria squadra. Inoltre, è presente un'area dedicata ad ogni squadra (nella figura denominata BASE), in cui i giocatori devono recarsi per ritornare in vita una volta morti. Il termine della partita è scandito da un timer, tipicamente di 15 minuti, all'esaurimento del quale viene stabilita la squadra vincitrice sulla base del numero di uccisioni totali.

Capitolo 3

Strumenti e tecnologie

3.1 Panoramica generale degli strumenti disponibili

La fase di progettazione è la base del lavoro di tesi sul quale si poggiano tutte le scelte fatte per arrivare ai risultati ottenuti. Questa parte è stata fondamentale, in quanto, oggi giorno numerose sono le tecnologie a disposizione degli sviluppatori.

Le scelte sono state fatte definendo, in primis, gli obiettivi finali. Fatto ciò, sono state studiate le diverse opportunità a disposizione che hanno permesso il raggiungimento di quest'ultimi. Il tutto mediante materiali che era possibile reperire, in quanto sprovvisti di risorse finanziarie. Lo sviluppo del progetto di tesi prevede, infatti, la necessità di integrare risorse hardware e software.

Una volta definiti i ruoli dei giocatori, in base alle specifiche del progetto, si è passati alla ricerca e allo studio degli strumenti che ne permettono la realizzazione. Analizzando il game concept, ci si è subito resi conto che vi sono tre principali macro problemi da affrontare. Queste tre macro aree è possibile riassumerle in:

- Comunicazione
- Hardware
- Localizzazione indoor

Questi punti critici devono essere risolti all'interno di un'applicazione che differisce a seconda del ruolo di ogni giocatore.

3.1.1 Piattaforme di sviluppo

Oggi, numerose sono le piattaforme a disposizione degli sviluppatori per la realizzazione di applicativi siano essi web, mobile o per pc. Partendo dal presupposto che era necessario sviluppare software per la piattaforma web e per dispositivi mobile, si è partiti nell'analizzare le risorse a disposizione.

Per la realizzazione di applicazioni web è possibile utilizzare piattaforme come Unreal Engine o Unity, motori grafici che permettono di realizzare applicazioni grafiche in grado di soddisfare le esigenze di progetto. Unreal Engine è un motore grafico che fino a poco tempo fa era totalmente a pagamento e, permette di effettuare il deploy multiplatforma ma in maniera molto più macchinosa rispetto Unity e date le sue qualità grafiche risultava anche più pesante per i dispositivi mobile. Di contro, permette la gestione del multithreading, cosa che Unity non permette di fare.

Ogni giocatore è dotato di un dispositivo Android, nel quale è installato un'applicazione che comunica con l'equipaggiamento in dotazione. La peculiarità di questa scelta sta nel fatto che non è importante che i proprietari del Laser Game siano dotati di questo terzo strumento in quanto ogni giocatore può usare il proprio smartphone durante la partita.

Per la realizzazione dell'applicazione mobile, visto che la scelta è ricaduta su dispositivi Android (in quanto in nostro possesso) è possibile utilizzare l'IDE ufficiale Android Studio. Questo è un software che, in fase di programmazione aiuta i programmatori nello sviluppo del codice sorgente, segnalando errori di sintassi direttamente in fase di scrittura,

oltre a tutta una serie di strumenti e funzionalità di supporto alla fase di sviluppo e debugging.

Vista l'esigenza di dover lavorare contemporaneamente per due differenti piattaforme la scelta è ricaduta su Unity. Grazie a questo motore grafico gratuito è possibile sviluppare del codice facilmente esportabile in qualunque piattaforma oggi a disposizione, quindi, con un'unica stesura di codice è possibile sviluppare software sia per la piattaforma web sia per quella Android.

3.1.2 Comunicazione

La comunicazione risulta il fulcro su cui gira l'intero funzionamento del gioco. Essa è molto importante in quanto il Team Leader deve poter comunicare con i propri componenti di squadra in qualsiasi momento. Inoltre, proprio per come è pensato il game concept, il team leader, avendo il ruolo di stratega nella partita, non deve essere fisicamente presente nell'area dove si disputa il match. Di conseguenza, internet diventa fondamentale per poter effettuare una comunicazione a distanza.

Tra tutti i possibili servizi internet vi era la possibilità di usare un server creato ad hoc o basarsi su un framework presente su Unity. L'implementazione di un server creato ad hoc presenta sicuramente numerosi vantaggi in quanto si ha una migliore gestione del traffico da e verso quest'ultimo e un livello di protezione maggiore. Inoltre, si ha un livello di personalizzazione più elevato sotto il diretto controllo del programmatore.

Perseguire questa strada rischia di distogliere l'attenzione sul vero oggetto della tesi, in quanto risulta un lavoro non necessario per raggiungere l'obiettivo prefissato, togliendo tempo al perseguimento di quest'ultimo. Di conseguenza, si è scelto di utilizzare un

framework di rete già disponibile per la piattaforma Unity. Tra tutti quelli disponibili Photon Network si è rivelato adatto alle esigenze in quanto, grazie a Photon Cloud, non vi è la necessità di gestire un server poiché viene fornito già pronto per l'utilizzo da questa framework.

Photon Cloud è un Software as a Service (SaaS), un modello di distribuzione del software applicativo dove un produttore di software sviluppa, opera (direttamente o tramite terze parti) e gestisce un'applicazione web che mette a disposizione dei propri clienti via internet. Tale servizio viene gestito completamente dai realizzatori permettendo di concentrarsi completamente sulla parte client dell'applicazione, mentre l'hosting, le operazioni del server e il ridimensionamento è tutto curato da Exit Games. Inoltre, la presenza di un cloud garantisce una bassa latenza e tempi più brevi di andata e ritorno per i giocatori in tutto il mondo.

3.1.3 Hardware

Nessuna invenzione può realmente considerarsi tale se non è supportata da hardware in grado di supportare le tesi progettuali. Proprio per questo, nella creazione di un gioco con caratteristiche di immersività, strategia e Ambient intelligence l'hardware ricopre un ruolo fondamentale.

Durante la fase di analisi della tesi si è partiti con l'idea di poter utilizzare dispositivi già in commercio per tutto ciò che concerne il mondo del Laser Game. Quindi, si è cercato di prendere contatti diretti con i titolari di Laser Game di zona, ai quali risultava molto difficile poter fornire i loro dispositivi di puntamento e ricezione. Visto l'esito negativo si è cercato di trovare dispositivi in commercio ma questo secondo tentativo non è andato a buon fine dato il budget (pari zero) a disposizione. Preso coscienza della difficoltà nel

reperire dispositivi già funzionanti, si è partiti da zero studiando il funzionamento di questi ultimi e perseguendo la strada che portava all'intera realizzazione dei device.

Grazie allo studio dei device di puntamento e ricezione, abbiamo compreso che la tecnologia utilizzata per simulare l'azione di fuoco da un'arma e la segnalazione di centramento di un avversario, avviene tramite un emettitore ed un ricevitore a infrarossi.

Sia l'arma che la pettorina, sensibile al laser infrarosso, devono avere un minimo di intelligenza in modo da poter mantenere il conteggio delle munizioni a disposizione e dei punti vita dei giocatori. Tenendo conto degli obiettivi prefissati, tutto questo non basta in quanto è fondamentale che l'equipaggiamento a disposizione del giocatore deve essere in grado di comunicare con un dispositivo mobile in modo da poter aggiornare in tempo reale i parametri vitali e di munizionamento del giocatore. Inoltre, deve essere in grado di recepire, da parte del comandante di squadra, tutti i potenziamenti ad esso indirizzati. In base a queste esigenze, si è scelto di dotare sia il dispositivo di puntamento che quello di ricezione di un Arduino Yun.

Questo particolare tipo di Arduino risulta consono alle nostre esigenze, in quanto oltre alle normali caratteristiche è dotato di un secondo processore Linux che gestisce un modulo Wi-Fi. In aggiunta, questa piattaforma risulta molto utilizzata, infatti, è molto semplice reperire informazioni sui possibili usi fatti da utenti amatoriali e non.

3.1.4 Localizzazione Indoor

Per raggiungere l'obiettivo, ovvero la realizzazione di un gioco in grado di sfruttare le caratteristiche di Ambient Intelligence, si è partiti dal voler riproporre, su una mappa virtuale all'interno dell'applicazione a disposizione del Team leader, la posizione dei singoli componenti di squadra.

Inizialmente ci si è posti l'obiettivo di effettuare la localizzazione indoor attraverso gli strumenti a disposizione. I Beacon Estimote sono dei dispositivi hardware LE Bluetooth con l'aspirazione, attraverso la potenza del segnale, di poter fornire la distanza di un dispositivo dotato anch'esso di Bluetooth e connesso a quest'ultimo. Tuttavia, nonostante le ottime premesse, la tecnologia risulta ancora scarna per una localizzazione approssimata e di conseguenza si è andati alla ricerca di altre tecnologie disponibili.

Una possibile alternativa sembrava essere IndoorAtlas, un sistema per il tracciamento della posizione e la creazione di percorsi utili a raggiungere una determinata destinazione mediante smartphone, il tutto basato sul rilevamento di informazioni relative al campo magnetico terrestre.

Il principio di funzionamento di tale tecnologia è piuttosto immediato: ogni particolare configurazione della struttura di una determinata stanza crea un'anomalia del campo magnetico ben definita, la quale può essere utilizzata per identificare specifiche aree all'interno di un edificio. Ogni zona, insomma, è come se avesse una sorta di impronta digitale magnetica, rilevabile mediante smartphone ed utilizzabile per indicare all'utente la strada migliore per raggiungerla.

L'iniziativa ha, dunque, sulla carta importanti potenzialità, potendo idealmente rivoluzionare il mondo della navigazione assistita e della geolocalizzazione, soprattutto in ambienti chiusi, tipicamente ostici per il rilevamento di simili informazioni. Il progetto, tuttavia, è ancora in fase di maturazione e poco sviluppato. Inoltre, ai fini della tesi non permette di ottenere risultati accettabili in quanto ci si scontra con una realtà altamente

dinamica e teoricamente sempre differente in quanto le arene di gioco possono essere i più disparate possibili.

Essendo l' Ambient Intelligence uno dei punti fondamentali del progetto di tesi, non si poteva prescindere dall' avere un' interazione con l' ambiente esterno, quindi i continui studi hanno portato a sfruttare i Beacon Estimote (che avevamo a disposizione come dispositivi di prossimità), in grado di giocare un ruolo peculiare nello svolgimento della partita. Infatti, posizionando questi dispositivi in punti strategici della mappa è possibile rilevare quando un giocatore si trova nelle estreme vicinanze di un particolare punto di interesse.

3.2 Descrizione strumenti e tecnologie utilizzate

Per realizzare tutto quello che è necessario per raggiungere gli obiettivi della tesi, sono stati utilizzati diversi strumenti di sviluppo gratuiti nonché applicativi di programmazione, dispositivi hardware e strumenti per la modellazione 3D.

In questa parte di studio, ci occuperemo di fare una panoramica generale su ogni singolo strumento, con lo scopo di fornire le basi per comprendere il reale funzionamento.

3.2.1 Piattaforma Arduino

Arduino è una piattaforma di prototipazione open-source basata su hardware e software facili da usare. Le schede Arduino sono in grado di leggere i dati di input di trasformarli in un output. È possibile dire alla board cosa fare con l'invio di una serie di istruzioni al microcontrollore della scheda. Per fare questo si utilizza il linguaggio di programmazione Arduino (basato su Wiring) e il software di Arduino (IDE).

Nel corso degli anni Arduino è stato il cervello di migliaia di progetti, da oggetti di uso quotidiano a strumenti scientifici complessi. Una comunità mondiale di produttori come studenti, hobbisti, artisti, programmatori e professionisti, si è raccolta intorno a questa piattaforma open-source. I loro contributi danno un valore aggiunto grazie alla quantità incredibile di informazioni accessibili che risultano di grande aiuto per principianti ed esperti.

Arduino è nato all'Ivrea Interaction Design Institute come uno strumento semplice per la prototipazione rapida, rivolto a studenti senza un background in elettronica e programmazione. Non appena è stata raggiunta una comunità più ampia, la scheda Arduino ha incominciato a cambiare per adattarsi alle nuove esigenze e sfide, differenziando la propria offerta a partire da boards semplici ad 8 bit, fino ai prodotti per le applicazioni dell'IoT, dispositivi indossabili, stampa 3D e ambienti embedded. Tutte le schede Arduino sono completamente open-source, abilitano gli utenti a sviluppare i propri progetti in modo indipendente ed al fine li adattarsi alle loro particolari esigenze. Il software, inoltre, è open-source, ed è in crescita grazie ai contributi degli utenti di tutto il mondo.

Grazie alla sua esperienza utente semplice e accessibile, Arduino è stato utilizzato in migliaia di diversi progetti e applicazioni. Il software Arduino è facile da usare per i principianti ma abbastanza flessibile per gli utenti avanzati. Funziona su Mac, Windows e Linux. Insegnanti e studenti lo usano per costruire strumenti scientifici a basso costo, per dimostrare i principi chimica e fisica, o per iniziare con la programmazione e la robotica.

Ci sono molti altri microcontrollori e piattaforme di microcontrollori disponibili per il physical computing. Parallax Basic Stamp, BX-24 di Netmedia, Phidgets, Handyboard del MIT, e molti altri offrono funzionalità simili. Tutti questi strumenti prendono gli intricati dettagli di programmazione dei microcontrollori e lo avvolgono in un pacchetto facile da usare. Arduino oltre a semplificare il processo di lavoro con i microcontrollori, offre molti vantaggi interessanti rispetto ad altri sistemi:

- **A buon mercato:** le schede Arduino sono relativamente poco costose rispetto ad altre piattaforme di microcontrollori. La versione meno costosa del modulo Arduino può essere assemblata a mano, e anche i moduli pre-assemblati Arduino costano meno di 50 \$
- **Cross-platform:** Il Software Arduino (IDE) funziona sui sistemi operativi Windows, Macintosh OSX e Linux. La maggior parte dei sistemi di microcontrollori sono limitati a Windows.
- **Semplice ed ambiente di programmazione chiaro:** Arduino software (IDE) è facile da usare per i principianti, ma abbastanza flessibile da sfruttare pure per gli utenti più esperti.
- **Open source e software estensibile:** Il software di Arduino è pubblicato come open source, disponibile per estensione, da programmatori esperti. Il linguaggio di programmazione può essere espanso tramite librerie C++ e le persone che vogliono capire i dettagli tecnici possono fare il salto da Arduino al linguaggio di programmazione AVR C su cui è basato. Allo stesso modo, se si vuole, è possibile aggiungere codice AVR C direttamente nel programma Arduino.
- **Open source e hardware estensibile:** I piani delle schede Arduino sono pubblicati sotto licenza Creative Commons, così i progettisti di circuiti esperti possono fare la propria versione del modulo, estendendola e migliorandola. Anche gli utenti relativamente inesperti possono costruire le proprie

applicazioni su breadboard al fine di capire come funziona e risparmiare denaro.

L'intera gamma di prodotti ufficiali Arduino è variegata e comprende numerosi articoli tra cui schede, moduli (un form-factor più piccolo di pannelli classici), Shields (gli elementi che possono essere inseriti su una board per dare funzionalità aggiuntive), e kit.

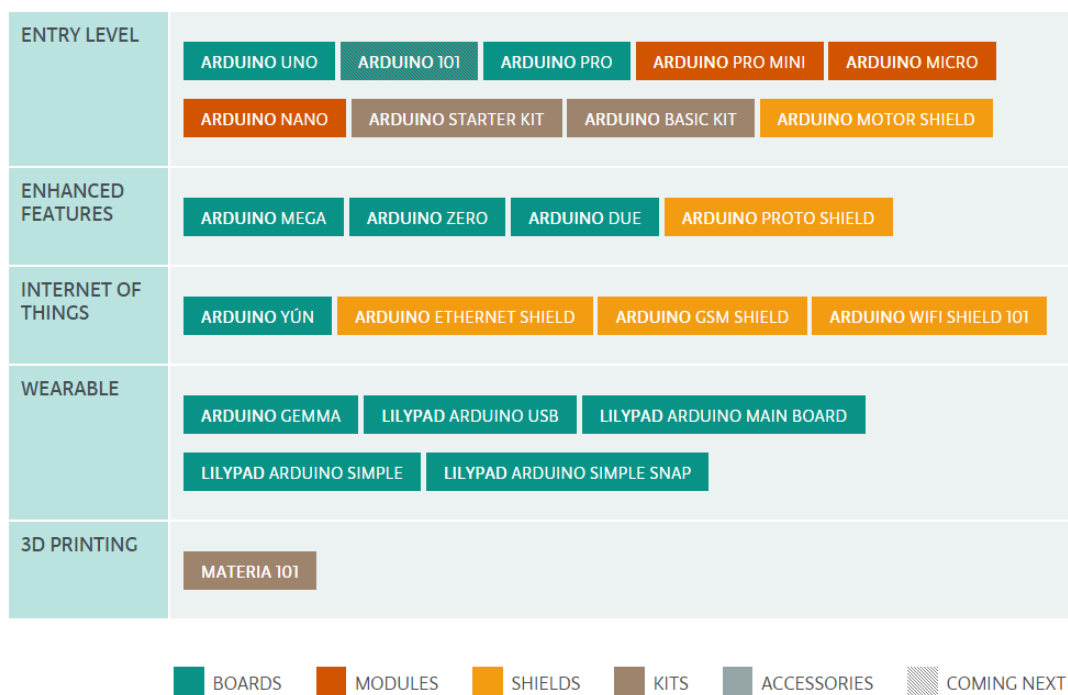


Figura 2 - Prodotti Arduino

Tra la vasta gamma dei prodotti ci siamo orientati su quella relativa all'Internet of Things in quanto fornisce la possibilità di creare facilmente dispositivi connessi sfruttando le opportunità messe a disposizione del world wide web. Quindi la scelta è ricaduta su Arduino Yun.

Arduino Yun è una scheda basata sul microcontrollore ATMEGA32U4 e Atheros AR9331. Il processore Atheros supporta una distribuzione Linux basata su OpenWrt chiamato OpenWrt-Yun. La scheda è dotata di Ethernet e il supporto Wi-Fi, una porta USB-A, slot per schede micro-SD, 20 ingressi digitali / pin di uscita (di cui 7 possono essere utilizzate come uscite PWM e 12 come ingressi analogici), un oscillatore di 16 MHz, una connessione micro USB, un header ICSP e 3 pulsanti di reset.

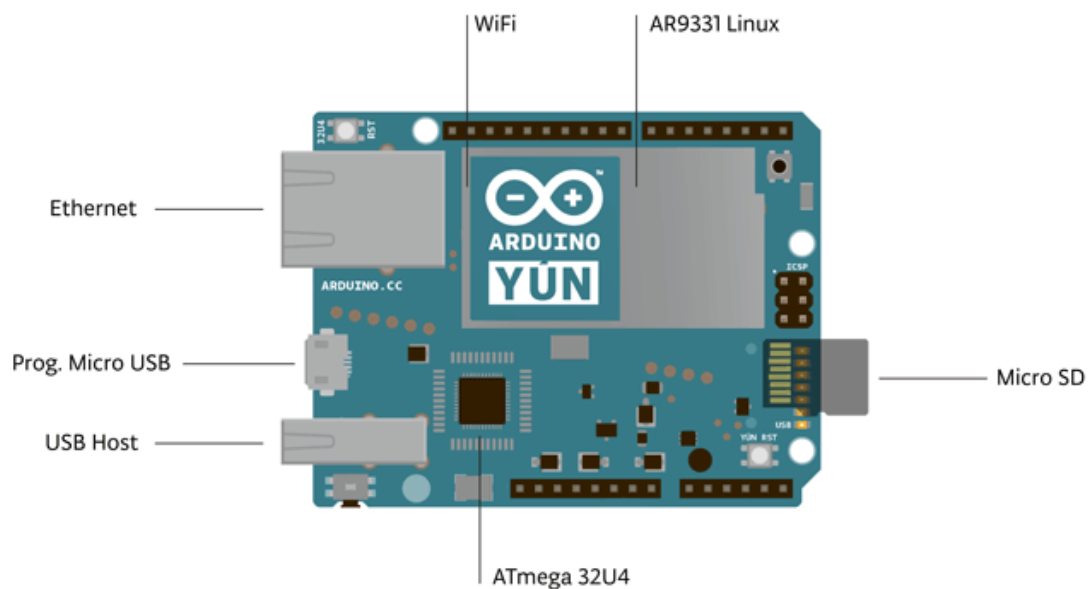


Figura 3 - Arduino Yun

Yun si distingue dalle altre schede Arduino, nel senso che può comunicare con la distribuzione Linux a bordo, offrendo un computer di rete potente. Oltre ai comandi Linux come cURL, è possibile scrivere i propri shell e python script per interazioni robuste.

La Yun è simile al Leonardo, un'altra scheda della famiglia Arduino, in quanto il ATMEGA32U4 ha incorporato la comunicazione USB, eliminando la necessità di un processore secondario. Questo permette alla Yun di apparire come un computer collegato come un mouse e una tastiera, oltre a una porta seriale virtuale (CDC) / porta COM.

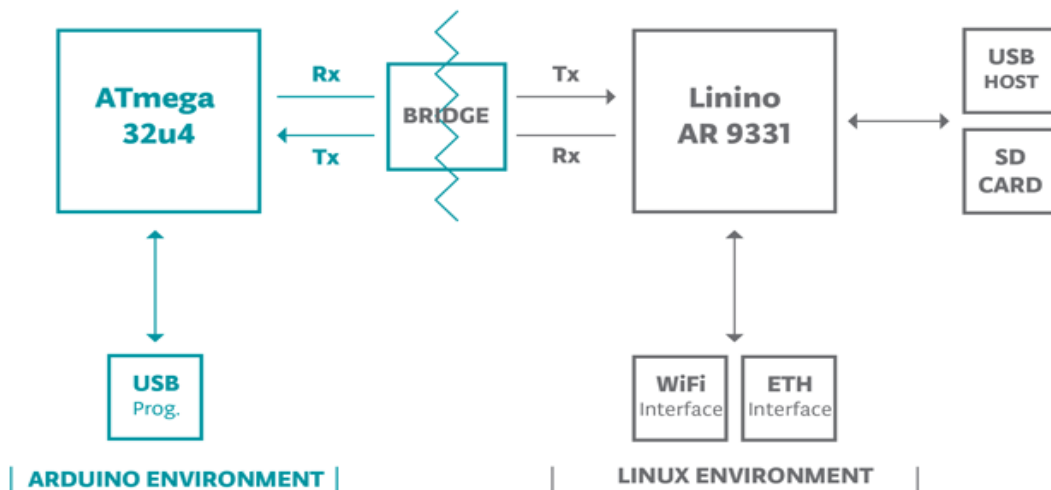


Figura 4 - Schematico Arduino Yun

Le librerie del Bridge facilitano la comunicazione tra i due processori, dando agli sketches Arduino la possibilità di eseguire gli script di shell, comunicare con le interfacce di rete e ricevere informazioni dal processore AR9331. L'host USB, l'interfaccia di rete e la scheda SD non sono collegati al 32U4 ma all'AR9331 e le librerie del Bridge consentono all'Arduino di interfacciarsi con le periferiche.

Lo Yun presenta due processori e di seguito vengono mostrate le tabelle riepilogative delle loro caratteristiche:

AVR microcontroller Arduino	
Microcontroller	ATMEGA32U4
Tensione di funzionamento	5V
Tensione di ingresso	5
Pins Digital I / O	20
Canali PWM	7
Pins di ingresso analogico	12
Corrente di CC per Pin O / I	40 mA
Corrente di CC per Pin 3.3V	50 mA
Memoria flash	32 KB (di cui 4 KB utilizzata dal bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Frequenza di clock	16 MHz

Microprocessore Linux	
Processore	Atheros AR9331
Architettura	MIPS @ 400MHz
Tensione di funzionamento	3.3V
Ethernet	IEEE 802.3 10 / 100Mbit / s
Wifi	IEEE 802.11b / g / n
USB tipo A	2.0 Host
Lettore di schede	Solo Micro-SD
RAM	64 MB DDR2
Memoria flash	16 MB
SRAM	2.5 KB
EEPROM	1 KB
Frequenza di clock	16 MHz
Supporto della scheda 802.3af PoE compatibile	Vedere <i>Potenza</i>

Il ATMEGA32U4 dispone di 32 KB (con 4 KB utilizzati per il bootloader). Ha anche 2.5 KB di SRAM e 1 KB di EEPROM (che può essere letta e scritta con la libreria di EEPROM).

Lo Yun ha 64 MB di RAM DDR2 e 16 MB di memoria flash. La memoria flash è precaricata con una distribuzione Linux, basata su OpenWrt, chiamata OpenWrt-Yun.

L'installazione OpenWrt-Yun occupa circa 9 MB di 16 MB disponibili della memoria flash interna. È possibile utilizzare una scheda micro SD, se si ha bisogno di più spazio su disco per l'installazione delle applicazioni.

Non è possibile accedere ai pin I / O del Atheros AR9331. Tutte le linee di I / O sono legati al 32U4.

Ciascuno dei 20 piedini digitali può essere utilizzato come ingresso o uscita.

- LED: 13. Vi è un built-in LED guidato da pin digitale 13. Quando il perno è alto, il LED è acceso, quando il perno è basso, è spento.
- Ci sono diversi altri LED di stato sul Yun che indicano l'alimentazione, la connessione WLAN, la connessione WAN e USB.

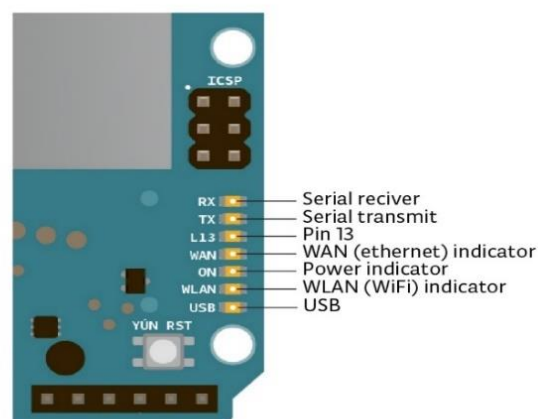


Figura 5 – Led Arduino Yun

Ci sono 3 pulsanti di reset con differenti funzioni sulla scheda:

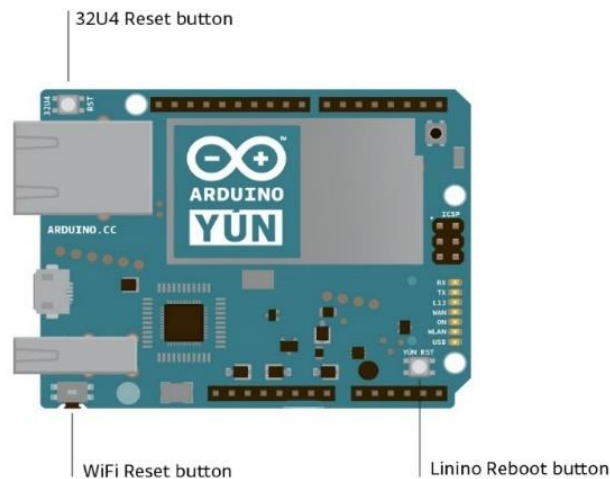


Figura 6 - Bottoni di Reset Arduino Yun

- Yun RST: permette di resettare il microprocessore AR9331. Il ripristino del AR9331 provoca il riavvio del sistema Linux. Tutti i dati memorizzati nella RAM andranno persi e tutti i programmi in esecuzione saranno terminati.
- 32U4 RST: permette di resettare il microcontrollore ATMEGA32U4. Tipicamente utilizzato per aggiungere un pulsante di reset per schemi che bloccano quello sul bordo.
- WLAN RST: questo tasto ha una doppia funzione. Quella primaria serve a ripristinare il Wi-Fi alla configurazione di fabbrica. La configurazione di fabbrica consiste nell'impostare il Wi-Fi del Yun in modalità Access Point (AP) e assegnare l'indirizzo IP di default che è 192.168.240.1, in questa condizione è possibile connettersi con il proprio computer alla rete della connessione Wi-Fi che appare con il nome SSID "Arduino Yun-XXXXXXXXXXXX", dove gli X rappresentano dodici cifre dell'indirizzo MAC dello Yun. Una volta connesso è possibile raggiungere il pannello web

dello Yun, con un browser, all'indirizzo 192.168.240.1 o "http://arduino.local". Si noti che il ripristino della configurazione Wi-Fi provocherà il riavvio dell'ambiente Linux. Per ripristinare la configurazione Wi-Fi è necessario premere, e tenere premuto, il pulsante RST WLAN per 5 secondi. Quando si preme il pulsante, il LED blu WLAN inizia a lampeggiare e continuerà ancora a lampeggiare fin tanto che non termina la procedura di reset. La seconda funzione del tasto WLAN RST è quello di ripristinare l'ambiente Linux alle impostazioni di fabbrica. Per ripristinare l'ambiente Linux è necessario premere il pulsante per 30 secondi. Da notare che, effettuando questo ripristino, si perdono tutti i file o software installati sulla memoria flash collegata al AR9331.

Yun ha un certo numero di modalità per la comunicazione con un computer, un altro Arduino o altri microcontrollori. Il ATMEGA32U4 fornisce una UART TTL (5V) seriale dedicata. Il 32U4 consente inoltre di effettuare una comunicazione via USB e appare come una porta COM virtuale sul computer al quale è collegato. Il chip funziona anche come un dispositivo di velocità USB 2.0, utilizzando i driver USB COM standard. Il software Arduino include, inoltre, un monitor di serie che consente l'accesso a dati testuali semplici da inviare da e per la scheda Arduino: i LED RX e TX sulla scheda lampeggiano quando i dati vengono trasmessi attraverso la connessione USB, al computer.

I pin digitali 0 e 1 vengono utilizzati per la comunicazione seriale tra il 32U4 e AR9331. La comunicazione tra i processori è gestita dalle librerie dei Bridge.

Le interfacce Ethernet e Wi-Fi a bordo sono esposte direttamente al processore AR9331. Per inviare e ricevere dati attraverso di loro, vengono utilizzate le librerie del Bridge. Per configurare le interfacce, è possibile accedere al pannello di controllo di rete.

Yun ha anche funzionalità USB. Infatti, è possibile collegare periferiche come dispositivi flash USB per storage aggiuntivo, tastiere o webcam.

Yun può essere programmato con Arduino Software (IDE). Le ATMEGA32U4 sull'Arduino Yun vengono pre-inizializzate con un bootloader che permette di caricare il nuovo codice senza l'uso di hardware esterno.

Arduino Software (IDE) è un software open-source che rende facile la scrittura di codice e il caricamento, di quest'ultimo, sulla board. Funziona su Windows, Mac OS X e Linux e può essere utilizzato con qualsiasi scheda Arduino.

3.2.2 Unity

Unity è un motore grafico all-in-one contenente tutto ciò che è necessario per sviluppare giochi o applicazione grafiche 3D. E' nato nel 2005 e sin da allora è diventato sempre più popolare, tanto che oggi è possibile definirlo come l'engine più utilizzato al mondo, almeno per gli sviluppatori mobile. Il motivo di questa grande diffusione è stata la scelta filosofica intrapresa da Unity e dal suo Team. Il Team di sviluppo voleva creare una piattaforma il più possibile semplice da utilizzare e con un modello di business adeguato anche per piccole società indipendenti. Nonostante, infatti, Unity non si classifichi al primo posto per le performance, grazie a queste scelte progettuali e di business si è rilevato un software parecchio amato dagli sviluppatori. Inoltre, soprattutto tra i primi anni di rilascio (all'incirca 2005-2010), si è dimostrato l'unico engine in grado di poter sviluppare qualsiasi genere di gioco con uno sforzo relativamente piccolo, al contrario, invece, di Unreal Engine e CryEngine che sembravano fossero più "specializzati" negli FPS. Tutte queste premesse gli hanno permesso di emergere tra tutti i concorrenti come engine flessibile, completo e soprattutto gratuito. Infatti, Unity è disponibile in due versioni: **Free** e **Pro**. Fino alla versione 4, le differenze tra free e pro si facevano sentire

in termini di disponibilità di tools e anche di rilascio su alcune piattaforme, era infatti impossibile per uno sviluppatore free di unity poter rilasciare giochi su mobile.

Dalla versione 5 in poi, la versione Free si differisce dalla Pro solo su alcune componenti di gestione del team, legati allo sviluppo e sulla impossibilità di cambiare lo splashscreen iniziale che resta sempre quello con il marchio Unity. Ad ogni modo, che si scelga la versione Free o la versione Pro, è possibile accedere sempre all'Asset Store. Quest'ultimo rappresenta un altro punto di forza della piattaforma e consiste di uno store in cui vengono venduti o resi disponibile gratuitamente, i cosiddetti **assets**. Su Unity per assets si intendono tutti quegli elementi in grado di popolare una scena di gioco, in altre parole, ci riferiamo a modelli 3d, animazioni, effetti particellari, scripts, audio etc. L'Asset Store è accessibile gratuitamente, sia da cliente che da publisher, a tutti gli utenti e in questo modo chiunque può guadagnare offrendo e preparando degli asset da pubblicare direttamente nello store.

Il Team di Unity, assiste con particolari guide la preparazione di asset adeguati che verranno avallati prima di essere resi disponibili al pubblico. Inoltre, indirizza sul corretto prezzo (nel caso di asset messo in vendita) in modo tale da massimizzare le vendite in caso di pubblicazione. Questo servizio è molto usato da tantissime case e sviluppatori indipendenti sia come mezzo di pubblicità che come mezzo di guadagno. Di conseguenza, grazie all'Asset Store, si può affermare che Unity rappresenta un engine potenzialmente infinito, sia per la quantità adeguatamente sufficiente di tool già incorporati ma anche per tutti gli asset che è possibile acquistare e che sono sicuramente di ottima qualità.

Nonostante l'Asset Store e di conseguenza il continuo sviluppo di tools e assets "esterni" da parte della community, il team di sviluppo mantiene sempre il software in continuo aggiornamento rilasciando, a scadenze annuali, numerose roadmaps che riguardano diverse parti di Unity. Sotto questo punto di vista, si può affermare che il team di sviluppo fa di tutto per mantenere un grande contatto con community e utilizzatori tramite forum, piattaforma ask&answers, youtube, video tutorial e roadmaps si è perennemente in discussione diretta con i creatori e sviluppatori. Il supporto è dunque ottimo ed è effettuato

non solo dai diretti interessati ma anche dai membri di questa infinita community che cresce ogni giorno.

Tra tutti gli aggiornamenti, rilasciati fino ad ora, di particolare importanza e rilievo sono il **4.6 UI** e **UNET**. Il primo rappresentò un grande passo in avanti in quella che era l'utilizzabilità e semplicità di creazione di interfacce grafiche all'interno della piattaforma Unity. Infatti, fino a quel momento, tutte le interfacce grafiche venivano create tramite la api **GUI** in cui, benché permettesse una grande personalizzazione, era comunque necessario scrivere del codice alle volte troppo sequenziale e poco chiaro. Grazie all'aggiornamento 4.6 sono stati inseriti una serie di tool grafici che permettono di inserire in maniera rapida la propria grafica e assegnare, tramite l'utilizzo dell'editor, il comportamento corrispondente o le animazioni richieste. UNET invece, è un insieme di features e api che permettono di creare giochi online senza l'utilizzo di framework esterni.

Questo tool, ha rappresentato per tanti anni il sogno di ogni sviluppatore di Unity che voleva avere un framework di rete nativo e sviluppato da un team di sviluppo serio che avesse tutta l'intenzione di proseguirne lo sviluppo e il miglioramento. UNET è stato rilasciato solo quest'anno, ma le premesse sono ottime. Infatti, facilità di utilizzo, protezione e soprattutto concetto di autorità, sono alcune delle caratteristiche che finora sono state mostrate. Addirittura, entro la metà del 2016, UNET dovrebbe essere in grado di garantire la creazione di MMORPG: tipologie di giochi che, per la loro complessità architeturale della rete, non sono quasi mai stati sviluppati su Unity. Si spera che, nel lungo termine, UNET possa sostituire tutti i framework di rete e imporsi come unica soluzione ma purtroppo la roadmap di sviluppo sembra un pò troppo lunga, obbligando gli sviluppatori di oggi a utilizzare framework di terze parti.

Unity è dunque un ottimo software di sviluppo di giochi e, come si è già detto, il più utilizzato. La grande diffusione del software non è una cosa da sottovalutare soprattutto se si è alle prime armi. Sono presenti miriade di tutorials accessibili gratuitamente creati sia dal team di sviluppo ma anche da sviluppatori indipendenti che utilizzano piattaforme come Facebook e Youtube per creare validissimi spunti e guide da seguire.

Parlando di funzionalità, si può affermare che Unity ruota attorno al concetto di **GameObject**. Il **GameObject** rappresenta un oggetto di gioco che può essere popolato, direttamente dall'Editor o in realtime tramite script, da varie componenti che serviranno ad aggiungere funzionalità e caratterizzazione.

Tra le varie componenti le più utilizzate sono sicuramente:

- **Mesh Renderer:** permette la visualizzazione di una determinata mesh o modello 3D, appartenente al **GameObject**
- **Colliders:** consente di scegliere una forma primitiva come Collider. Lo scopo dei Collider è quello di creare una sorta di bounding box (o altro) che includa all'interno la Mesh. È proprio nella gestione delle collisioni che si utilizza questo componente. Degno di nota è anche la possibilità di rendere il collider un **Trigger** che servirà a rilevare la collisione ma non reagire in termini fisici. Ottimo per piattaforme invisibili o per far scatenare determinati eventi.
- **Rigidbody:** questo componente è ciò che permette la simulazione della fisica e delle collisioni. Un **GameObject** dotato di un **Rigidbody** si comporterà come un oggetto dotato di gravità e di conseguenza risponderà a tutte quelle che sono le leggi della fisica, all'interno della scena. Si possono impostare vari parametri come la frizione, la gravità e il peso.
- **Animator Controller:** introdotto con l'aggiornamento **4** di Unity, l'**Animator Controller** è un componente del servizio **Mecanim** per gestire le animazioni. Prima di questo aggiornamento, tutte le animazioni andavano elencate nell'editor e successivamente fatte avviare da script agendo direttamente sul modello. Grazie a Mecanim e all'**Animator Controller** sarà possibile specificare una macchina a stati che specifica il comportamento di un determinato modello andando ad agire su variabili di transizione. Le variabili di transizione possono essere booleani, float e interi e consentono un'ottima personalizzazione e caratterizzazione del

comportamento dei modelli all'interno della scena. A livello di scripting, basterà settare la variabile di transizione desiderata per passare allo stato/animazione ricercato.

- **Audio Listener e Source** il Listener fornisce una componente di “ascolto” di audio ad un determinato gameObject. Il Source, invece, identifica una fonte audio nella quale verrà specificato un determinato suono da parte dello sviluppatore.
- **Scripts:** Ad ogni GameObject è possibile inserire qualsiasi tipo di script creato. La grande flessibilità di Unity consente di utilizzare, anche contemporaneamente, script scritti in Javascript o C#. All'interno degli script si può accedere in maniera nativa a tutte le componenti presenti tramite l'interfaccia **GetComponent**, assicurando una grandissima personalizzazione dei comportamenti.

Ogni GameObject può essere anche categorizzato in **TAG** diverse. Lo scopo delle TAG è appunto quello di fornire un metodo efficace per trovare GameObject all'interno della scena. Assieme ai TAG si menziona l'utilizzo dei **Layer** della fisica.

I layers ricoprono un ruolo fondamentale quando si parla di collisioni. Specificare il layer di appartenenza e cambiare la matrice di collisione, permette di capire, per ogni oggetto, con quali GameObject avverrà la collisione e specificare anche delle eccezioni. È possibile anche modificare il materiale dell'oggetto tramite l'utilizzo di vari **Shader**. Gli shader possono essere programmati direttamente dall'ambiente di sviluppo selezionato oppure importati dall'esterno, utilizzando il linguaggio GLSL.

La maggior parte delle funzionalità è, dunque, accessibile tramite Editor, che di per sé è completo e chiaro.

L'editor è suddiviso in varie finestre accessibili, ognuna con uno scopo e un significato particolare, tutte sono personalizzabili in dimensione e disposizione in diversi layout, preimpostati e non.

Le finestre principali sono:

- **Scene:** rappresenta la finestra sul mondo di gioco in cui è possibile creare e collocare i `GameObject` desiderati. Ogni `GameObject` potrà essere spostato/ruotato/scalato utilizzando degli appositi tool accessibili sia tramite keybind che tramite interfaccia. Unity consente di avere tanti oggetti **Scene** che possono essere popolati in maniera differente a seconda delle necessità. Il dividere in scene il gioco, è un metodo utile per mantenere una gerarchia di oggetti piuttosto piccola e non appesantire il lavoro. Si può passare da una scena all'altra utilizzando la stringa **`Application.LoadLevel(..)`**.
- **Game:** rappresenta ciò che viene “visto” dal giocatore a gioco avviato. Questo viene stabilito grazie all'oggetto **Camera** che può essere personalizzato in termini di frustum, posizione, rotazione praticamente come qualsiasi altro `GameObject`. Nella maggior parte dei giochi, l'oggetto Camera viene spesso associato al personaggio principale, andando a creare le principali disposizioni visuali viste nei giochi quali Prima Persona, Terza Persona o Visuale Isometrica. La Camera può anche essere animata per creare corti o filmati all'interno dell'applicazione 3D.
- **Project:** consiste dell'insieme di cartelle e file presenti nella cartella principale del progetto. È consigliabile mantenere una cartella di progetto ordinata per facilitare l'accesso e la comprensione ai file di progetto.
- **Hierarchy:** in questa finestra vengono indicati tutti i `GameObject` utilizzati all'interno della scena corrente. La finestra è chiamata Gerarchia, perché effettivamente ogni `GameObject` può rappresentare i “padre” di altri oggetti `GameObject` “figli” andando a creare effettive gerarchie. Il creare gerarchie è utile per ordinare gli oggetti nella scena, ma c'è da ricordare che ogni figlio eredita la posizione e la rotazione dell'oggetto padre.

- **Inspector:** questa è la finestra nella quale sono presenti la maggior parte delle personalizzazioni e scelte di sviluppo che riguardano i GameObject. Ogni qual volta selezioniamo un differente GameObject, la finestra Inspector viene popolata dai dettagli, componenti e script che lo riguardano. Qui possiamo inserire nuove componenti o rimuovere quelle presenti.
- **Console:** come qualsiasi ambiente di sviluppo, questa finestra mostra sia il risultato della compilazione che eventuali mancanze progettuali tra le componenti del GameObject
- **Animator e Animation:** Animator è la finestra della macchina che governa gli stati del controller selezionato. Animation rappresenta un metodo per creare rapide animazioni direttamente all'interno di Unity; è composto da una timeline in cui è possibile inserire dei keyframe su qualsiasi proprietà accessibile dall'inspector.

Parlando di ambiente di sviluppo inerente lo scripting, Unity utilizza **MonoDevelop** un IDE open-source che consente lo sviluppo multiplatforma e anche l'utilizzo di diversi linguaggi tra cui C#, Visual Basic o Javascript. Monodevelop è un buon IDE nel caso di progetti piccoli e con script composti da poche righe di codice. Tuttavia si percepisce la mancanza di professionalità, in alcune funzioni, nel caso in cui il progetto diventa corposo.

Una delle falle più grandi è l'impossibilità di ridurre le funzioni e, inoltre, spesso si verificavano vari crash del sistema. Tuttavia, dall'ultimo aggiornamento, Unity ha integrato la versione di **VisualStudio** Community come IDE principale, ovviando a tutti i problemi legati all'utilizzo di MonoDevelop.

All'interno dell'IDE è possibile sfruttare le API di base del linguaggio scelto (JavaScript o C#) assieme all'API di Unity, includendo all'interno dello script il namespace **UnityEngine**. L'API è completa e permette di accedere a qualsiasi componente, settarla, modificarne i parametri ed effettuare operazioni complesse.

In generale è possibile affermare che il funzionamento di uno script di Unity è legato a 2 macro-funzioni:

- **void Start():** funzione eseguita una volta sola all'inizializzazione dello script. Normalmente, si utilizza questa funzione per ottenere l'accesso alle componenti necessarie per lo scripting o fare verifiche di vario genere. Esiste anche nella versione **Awake** che permette di effettuare una inizializzazione, addirittura prima che le altre componenti vengano inizializzate.
- **void Update():** funzione che viene eseguita ad ogni frame. All'interno di questa funzione vanno inseriti tutti quei controlli e comportamenti da eseguire ad ogni ciclo. Esiste anche nella forma **FixedUpdate** che viene effettuata a intervalli specifici di tempo per facilitare le operazioni della fisica e dei Rigidbody.

Tra tutte le funzionalità dell'API offerte da Unity le **Coroutine** sono uno strumento molto interessante. Una Coroutine è come una funzione che dà la possibilità di interrompere l'esecuzione e effettuare il ritorno a Unity, per poi continuare il lavoro da dove viene interrotto. Le Coroutine garantiscono una grande personalizzazione della scena di gioco e permettono di effettuare operazioni in maniera più controllata. Si vuole menzionare anche la possibilità di stabilire **Navigazione** e **Pathfinding** per un determinato oggetto. È possibile, infatti, aumentare l'IA di un determinato oggetto andando a creare determinati **waypoints** sulla scena che possono essere perseguiti e percorsi, il tutto modificando dei parametri direttamente dall'editor.

Concludendo, si può affermare che Unity è un ottimo strumento per realizzare applicazioni, giochi e interfacce grafiche di ogni genere. La possibilità di effettuare il deploy su qualsiasi piattaforma, evitando di scrivere codice diverso, è un ottimo vantaggio quando si vuole cercare di raggiungere il bacino di utenza più grande possibile. Attualmente, inoltre, rappresenta l'unica alternativa completa nel caso si voglia

programmare e creare giochi o applicazioni mobile. Tuttavia vi sono alcune piccole mancanze che rendono a volte lo sviluppo un po' ostico. Parlando di performance, la mancanza della gestione multithread risulta essere una grande pecca, nonostante il team di sviluppo faccia di tutto per dimostrare di poter creare anche giochi pesanti o effettuare calcoli complessi. Durante lo sviluppo della tesi si è proprio appurata questa mancanza, soprattutto quando si è affrontata la gestione del Bluetooth, effettuando lo scan dei dispositivi, assieme al normale ciclo di gioco. Anche utilizzando il pacchetto C# e i thread, non si riesce a bypassare il problema questo perché Unity permette di eseguire le sue funzionalità solo all'interno del thread principale. Inoltre, nonostante l'UI 4.6 si dimostra un tool di facile utilizzo, presenta una modalità di funzionamento un po' scarna che fa uso di troppe variabili pubbliche per dichiarare i comportamenti di ogni pulsante o immagine. La gestione della gerarchia tra i vari oggetti, spesso, diventa un vero nemico. Ricercare un oggetto figlio o risalire all'oggetto padre diventa pressoché impossibile e spesso si è costretti ad usare sotterfugi o exploits.

Infine, sarebbe comodo avere un sistema di reset delle componenti e dei GameObject nel caso in cui una determinata scena permene, per più tempo, all'interno di una partita di gioco. Si è notato, infatti, come fosse difficile rendere una stessa scena ri-giocabile, nel caso in cui si volessero mantenere alcuni dati intatti. Anche la componente della fisica presenta qualche problema. È difficile, infatti, raggiungere ottimi livelli di simulazione utilizzando la fisica nativa di Unity. La cosa che lascia un po' pensare è che vi sono asset esterni, accessibili dall'Asset Store, che garantiscono un risultato migliore con una migliore efficienza. Si intravede, dunque, come Unity è un ottimo engine per il Mobile ma che presenta qualche pecca per giochi più complessi. Nonostante questo, viene utilizzato da tantissime software house su qualsiasi piattaforma e le roadmap di sviluppo sono molto promettenti, rendendo Unity un prodotto più che consigliato.

3.2.3 Photon Network

Photon è un framework di sviluppo di giochi o applicazioni multiplayer online. È stato creato dall'azienda tedesca ExitGames nel 2003 e continua ad evolversi grazie al costante sviluppo supportato anche dalle vendite e dall'acclamazione del prodotto da parte dei clienti. La società fornisce due servizi multiplayer, con caratteristiche e profondità diverse, che si adattano alle diverse esigenze dei clienti:

- **Photon Server:** pacchetto completo che permette lo sviluppo di giochi multiplayer scalabili utilizzando un server creato ad hoc e personalizzabile tramite l'api fornita.
- **Photon Cloud:** servizio basato su Cloud che consente di dimenticarsi della componente server e dedicarsi totalmente ai Client di gioco.

Qualsiasi soluzione si scelga, le librerie restano le stesse visto che si connettono allo stesso servizio di backend che condivide gli stessi comandi e la stessa logica. Grazie al servizio di backend, Photon permette lo sviluppo multipiattaforma e la connessione, ad uno stesso server, da parte di dispositivi diversi nello stesso momento. Tra le due soluzioni offerte la più appetibile per facilità di utilizzo, velocità nell'effettuare test e velocità di rilascio è Photon Cloud. I principali benefici di Photon Cloud sono:

- **Evitare di programmare la parte server:** grazie al Cloud ci si può dedicare esclusivamente al client e alla sua logica di gioco.
- **Latenza bassa:** il Cloud è sviluppato in diversi centri globali, infatti, durante il setup si può scegliere a quale servizio associarsi che sia esso europeo, americano o asiatico.

- **Alta Efficienza:** il servizio è dotato di una grande efficienza e garantisce la consegna di pacchetti tra i destinatari ad una bassissima latenza.
- **Scalabilità automatica:** qualsiasi sia il numero di giocatori connessi, il Cloud farà in modo di garantire sempre un servizio ottimale scalando le risorse hardware quando necessario.

Le caratteristiche esposte hanno permesso al servizio di imporsi come uno dei migliori framework multiplayer disponibili, anche grazie al piano gratuito di utilizzo. Photon Cloud è, infatti, disponibile gratuitamente a tutti i clienti fino ad un massimo di 20 CCU. L'acronimo CCU sta per Concurrent Users e corrisponde al numero di giocatori collegati in un determinato istante, all'interno della porzione di Cloud. Di conseguenza, qualunque sviluppatore può azzerare i costi di server e iniziare a sviluppare il gioco all'interno della componente Cloud. Dopo aver terminato lo sviluppo, in caso di rilascio, si può passare ad un piano a pagamento.

Il più interessante risulta il secondo piano disponibile che permette, con un singolo pagamento, di ottenere per sempre una porzione di Cloud fino ad un massimo di 100 CCU.

20 CCU	100 CCU	500 CCU	Per 1000 CCU
FREE always	\$95 once	\$89 monthly	\$169 monthly per 1,000 CCU .
20 Connections ~4k Monthly Actives 500 Msg/s per Room Overage not included	100 Connections ~20k Monthly Actives 500 Msg/s per Room Overage not included	500 Connections ~100k Monthly Actives 500 Msg/s per Room Overage is included	\$199 monthly per 1,000 CCU for 5,000 CCU and up. Contact us for enterprise plans and private clouds .

Figura 7 - Piani abbonamenti Photon Cloud

L'altro vantaggio sta nel fatto che risulta molto facile effettuare il passaggio da Photon Cloud a Photon Server. Questo avviene grazie alla condivisione della stessa API del Client tra le due tecnologie proposte. Di conseguenza, spesso il Cloud è sfruttato, dalle grandi case, come uno strumento di test di meccaniche, per creare rapidi prototipi da rilasciare nelle piattaforme di crowdfunding o per verificare i vari game concept in maniera rapida ed efficiente.

Nel caso di un progetto abbastanza serio, il passaggio a Photon Server è quasi obbligatorio, poiché il Cloud manca di quella personalizzazione e concetto di autorità necessari a evitare problemi di vario genere, soprattutto legati ai cheats. Nonostante questo, vi sono numerosi esempi di giochi creati utilizzando Photon Cloud, soprattutto nel Mobile.

Da qualche anno ExitGames si è anche alleata con l'azienda PlayFab che fornisce un servizio simile dedicato però ai database e alla persistenza di dati. Grazie a questa alleanza, gli sviluppatori indipendenti possono utilizzare un'unica piattaforma per creare giochi online completi sia per la parte real-time sia per la persistenza delle sessioni di gioco e dei progressi del giocatore.

Photon Network è probabilmente la soluzione più diffusa quando si parla di Networking su Unity. Non solo per i vantaggi già esposti, ma anche perché si è affermata sin dal 2005 come una soluzione stabile e fidata ottenendo una grande community capace di fornire supporto e tutorial.

È proprio grazie a questo grande supporto che risulta molto facile e immediato addentrarsi nel mondo online, riuscendo con semplici passi a trasformare un gioco singleplayer in multiplayer.

Il concetto di funzionamento di Photon è spiegato dall'immagine sottostante:

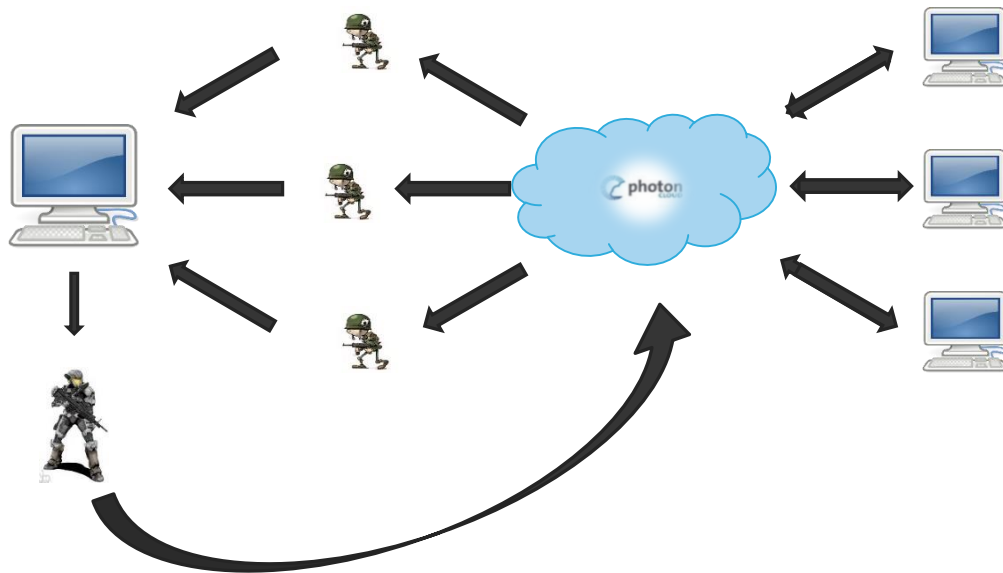


Figura 8 - Funzionamento Photon Cloud

Ogni computer rappresenta un utente e di conseguenza un'istanza di gioco. Ogni utente si connette al cloud tramite la sua applicazione di gioco e, una volta stabilita la connessione, istanzia il suo giocatore.

Il messaggio di istanza del giocatore viene inviato, tramite il Cloud, a tutti i giocatori connessi. Ogni giocatore, di conseguenza, troverà all'interno della sua applicazione sia il suo personaggio che personaggi "zombie", proprietà degli altri utenti, che verranno animati e controllati dai pacchetti inviati dal cloud e, ancora prima, dagli utenti proprietari.

Questo funzionamento si può ben vedere nelle prime fasi di sviluppo. Infatti, senza inviare i pacchetti, l'istanza del personaggio di un giocatore rimarrà ferma, in quella dell'altro utente, fin tanto che non si implementa l'invio di pacchetti contenenti le informazioni necessarie per far "muovere" il personaggio.

Per gestire determinati eventi quali connessioni, disconnessioni, join della lobby o delle varie stanze di gioco, Photon possiede una serie di **Callbacks** che vengono richiamati non appena si verifica l'evento corrispondente. Tutti i Callback sono descritti nella documentazione ufficiale e, per implementarli, basterà riscrivere il nome del metodo trattandoli come delle vere e proprie funzioni.

Tra i Callback più importanti abbiamo:

- **OnJoinedLobby():** richiamato all'ingresso della lobby del cloud di appartenenza
- **OnJoinedRoom():** richiamato all'ingresso di una determinata stanza di gioco.
- **OnPhotonPlayerConnected():** richiamato non appena un utente si connette alla stessa stanza di gioco. Spesso all'interno di questo callback si inseriscono quelli che sono gli aggiornamenti del conteggio dei giocatori.
- **OnPhotonPlayerDisconnected():** richiamato quando un giocatore si disconnette.
- **OnPhotonRandomJoinFailed():** nel caso in cui si è effettuato il metodo di Join randomico ad una stanza, questo metodo viene richiamato nel caso in cui il join fallisce.

La lista dei Callback implementabili è molto ampia e copre la maggior parte degli eventi verificabili all'interno di un gioco multiplayer. Grazie ai Callback, si può quindi creare un sistema di gioco solido che consente di gestire errori o disconnessioni in maniera rapida e ordinata ma permettere anche di utilizzare le funzionalità necessarie quali, per esempio, l'evocazione dell'istanza del proprio giocatore. Per istanziare un oggetto nella rete, si utilizza il codice **PhotonNetwork.Istantiate(..)**. Questa funzione avrà il compito di notificare a tutti gli utenti, connessi in quell'istante, di istanziare l'oggetto specificato come parametro. Ogni oggetto, da istanziare tramite questa funzione, dovrà possedere la componente **PhotonView** che permette di poter dare un identificativo univoco, in tutta la

rete, ad un determinato oggetto. Per quanto riguarda, invece, la sincronizzazione di parametri, valori e movimento degli oggetti è possibile utilizzare due metodi proposti da ExitGames:

- **State Synchronization:** consiste di un metodo basato su serializzazione e deserializzazione dei dati, che avviene all'interno di una funzione chiamata **OnPhotonSerializeView(..)**. Questo metodo viene utilizzato nel caso di parametri che cambiano spesso il loro valore e che necessitano quindi di aggiornamenti continui. Un esempio possono essere le coordinate di un giocatore, le animazioni o i suoi parametri vitali.
- **Remote Procedure Calls (RPC):** le RPC sono funzioni che vengono dichiarate, con un prefisso [**PunRPC**], all'interno di un determinato oggetto dotato di **PhotonView**. Se viene avviata una funzione RPC, questa verrà eseguita, allo stesso e identico modo, da ogni utente e sullo stesso oggetto. Il metodo delle RPC viene spesso usato per determinate funzionalità più complesse che richiedono più calcoli o che semplicemente hanno una frequenza di aggiornamento minore.

Questi metodi possono essere usati sui personaggi direttamente proprietari dei giocatori, ma anche su quelli che sono gli oggetti di scena. Infatti, se si associa la componente **PhotonView** ad un **gameObject** esistente nella scena, quest'ultimo diventerà uno **Scene Object** e potrà essere sincronizzato nella rete con gli stessi metodi di un oggetto proprietario.

Grazie agli **Scene Object** è possibile sincronizzare, tra le varie istanze utente, tutto quel genere di eventi di contorno che spesso notiamo all'interno dei giochi: vedasi esplosioni, NPC, IA etc.

Con un po' d'inventiva, gli elementi mostrati permettono di creare qualsiasi tipologia di gioco multiplayer, in modo abbastanza rapido. Tuttavia, vi sono alcuni problemi legati, soprattutto nei giochi realtime, che necessitano una sincronizzazione più precisa e rapida.

Esaminiamo per un attimo la posizione, nella rete, di un determinato oggetto appartenente ad un giocatore. Questa verrà sincronizzata grazie a dei pacchetti, continuamente inviati da parte del giocatore sulla base dei suoi input inviati in locale tramite le periferiche di gioco. Per definizione, questi pacchetti non saranno inviati sempre ma saranno spediti a determinati intervalli temporali. A causa di questo, implementando una sincronizzazione semplice, si noterà come l'istanza del giocatore si muoverà a scatti, generando un brutto effetto ottico.

Per ovviare a questo problema bisogna effettuare una **interpolazione** tra i pacchetti ricevuti dal proprietario.



Figura 9 – Player e ricezione dei pacchetti di aggiornamento

L'interpolazione può essere di qualunque tipo, ma su Unity è effettuata spesso tramite il metodo **Lerp**. Apparentemente l'interpolazione risolve il problema, ma ad un'analisi più attenta introduce un ritardo sull'effettiva posizione del giocatore. Per questo viene utilizzata un'altra tecnica che viene chiamata **Predizione**. Questa consiste nell'effettuare una sorta di scommessa su quella che potrebbe essere la successiva posizione del giocatore.

La scommessa viene fatta analizzando l'ultimo pacchetto; se all'ultimo pacchetto il giocatore si sposta in una determinata posizione, posso supporre che anche per gli ulteriori istanti di tempo lo farà e di conseguenza spostarlo ad una posizione maggiorata di un determinato delta. A tutto questo va inserita la latenza che influisce sul numero di pacchetti inviati e anche sul ritardo di ricezione. Anche questa va valutata assieme alla predizione sfruttando le tecniche di **Lag Compensation**. Tra queste tecniche vi è quella di inviare, oltre al dato, anche il ping del giocatore. Il ping verrà sfruttato dal ricevente del pacchetto per migliorare la tecnica di predizione.

Fino a 6 mesi fa, gli utilizzatori di PhotonNetwork dovevano creare da se le varie tecniche illustrate, partendo dall'interpolazione per finire alla lag compensation. Tuttavia, grazie all'ultimo aggiornamento, sono stati ideati dei componenti da installare direttamente ai GameObject che permettono di utilizzare tecniche già scritte e variarne i parametri a seconda dell'utilizzo, riducendo ulteriormente le barriere di creazione di giochi Multiplayer.

Photon è, dunque, un ottimo servizio, con un ottimo supporto e un downtime praticamente nullo. Se non si hanno grosse pretese, si può utilizzare tranquillamente il servizio Cloud creando rapidamente tutto ciò che si desidera. Tuttavia il passaggio alla componente Server diventa obbligato quando la complessità di gioco si fa un po' più viva. Un po' troppo spesso si cerca invano di stabilire il concetto di autorità all'interno di ogni partita, sia per evitare i cheaters, ma anche per avere un oggetto o utente fondamentale che possa facilitare le operazioni di sincronizzazione. Inoltre, nonostante serializzazione e RPC si prestano bene per le operazioni più semplici, manca un modo per evocare eventi e inviare messaggi senza dover passare da un determinato GameObject che debba essere obbligatoriamente sincronizzato.

Infine, è impossibile effettuare una sorta di classificazione dei giocatori: sarebbe meglio se il sistema offrisse un modo nativo per classificare i giocatori in determinati gruppi, per filtrare le sincronizzazioni a seconda delle necessità. Tuttavia, per l'obiettivo della tesi, Photon si è dimostrato un ottimo alleato che ha permesso di soddisfare tutti i requisiti prefissati.

3.2.4 Estimote Beacons

Gli Estimote Beacons sono piccoli sensori wireless che possono essere posizionati in qualunque luogo. Inviano segnali radio che possono essere captati dai dispositivi smartphone circostanti e essere interpretati per avere una sensibilità ambientale.



Figura 10 - Beacons Estimote

Apparentemente sembrano delle piccole pietre colorate ma, internamente, sono dei piccoli computer. Sono dotati di un processore 32-bit ARM Cortex M0, accompagnato da memoria integrata, accelerometro, sensore di temperatura e radio da 2.4Ghz che usa Bluetooth 4.0, anche conosciuto col nome di Bluetooth Low Energy. Il segnale radio viene inviato tramite l'antenna integrata e garantisce mesi di funzionamento senza dover cambiare la batteria. Tuttavia, la durata della batteria (che non è ricaricabile) dipende esclusivamente dalle impostazioni di ogni singolo device.

Tramite l'applicazione Estimote, è possibile connettersi a ogni Beacon disponibile e influire sulle caratteristiche di funzionamento quali raggio di azione, intervallo di

advertising e potenza di segnale. Di conseguenza, utilizzando features al minimo indispensabile, si può arrivare ad una durata di 4 anni (dichiarati) ma per ottenere un risultato è necessario utilizzare il massimo delle potenzialità arrivando quindi ad una durata massima di 10 mesi (dichiarati).

I beacon possono essere acquistati ad un prezzo molto competitivo (circa 30 euro per 3 beacons) e dispongono di una SDK, creata dal team di sviluppo, abbastanza completa. Il problema risiede sul fatto che la compagnia Estimote si è dedicata maggiormente a sviluppare API, esempi e Tutorial per i dispositivi Apple, tralasciando le API Android. Il motivo di questa scelta è dovuto al fatto che i dispositivi Android, più diffusi e prodotti da marche diverse, utilizzano moduli bluetooth e hardware che reagiscono in maniera diversa, negando la possibilità di utilizzare un'unica API funzionante. L'eterogeneità dei dispositivi Android ha penalizzato, quindi, lo sviluppo software e il supporto degli Estimote Beacons. Questa differenza, tra Android e Apple, si può notare andando a verificare la documentazione e soprattutto l'implementazione dell'API.

La controparte Apple presenta addirittura l'SDK Indoor che consente, a detta del team di sviluppo, di individuare con 3 beacons la posizione di un soggetto all'interno di una stanza. Sono stati fatti numerosi tentativi da parte della community per creare una soluzione analoga per Android. Tuttavia, i risultati sono stati totalmente inefficienti generando parecchio malcontento tra gli utenti del droide verde.

Anche nel caso in cui le richieste di progetto sono abbastanza esigue, si può notare come i Beacons rappresentano una soluzione poco professionale che conferma l'economicità del prodotto. Utilizzando un dispositivo Android, anche di ultima generazione, il segnale rilevato, direttamente dall'applicazione Estimote, varia in maniera esagerata corrompendo e rendendo impossibile qualsiasi sviluppo.

Nonostante i problemi con Android, sembrerebbe che utilizzando iOS si possano ottenere dei buoni risultati anche per quello che riguarda la localizzazione indoor. In ogni caso l'SDK di estimote è utilizzabile sia su Apple tramite Xcode che su Android Studio sfruttando il protocollo iBeacon o Eddystone. Ogni Beacon, è comunque riconoscibile

dagli altri dispositivi sia tramite nome (se si utilizza l'SDK proprietaria) o tramite indirizzo MAC.

I numerosi test effettuati su Android, hanno permesso di affermare come i Beacons non sono sicuramente un buon acquisto, almeno al momento. Anche cambiando protocollo di comunicazione, il risultato non cambia.

Utilizzando l'applicazione proprietaria si è notato come lo stesso segnale di ricezione sia troppo aleatorio e variabile. Si intravede, quindi, come la presenza ambientale sia ancora qualcosa da superare. Vi sono le basi ma ancora manca un sistema che permette all'eterogeneità dei dispositivi mobile di poter comunicare in maniera omogenea.

Nonostante i vari problemi di utilizzo, si è cercato di sfruttare al meglio i dati acquisibili dai beacon. Tramite vari test, si è cercato di trovare impostazioni e un raggio d'azione che potesse permettere l'individuazione di un individuo.

Tramite misurazioni effettuate con diversi dispositivi cellulari, si è notato che il segnale RSSI riusciva a mantenersi stabile nel raggio di un metro di distanza. Di conseguenza si è sfruttato il modulo Bluetooth per accedere ai beacon e ottenere il segnale RSSI che è stato successivamente analizzato e elaborato sotto forma di presenza fisica dell'utente in una determinata zona.

3.2.5 Blender e Stampa3D

Blender è un programma di modellazione 3D che è stato creato da centinaia di volontari attivi da tutto il mondo: studios e singoli artisti, professionisti e hobbisti, scienziati e studenti, esperti di effetti speciali, animatori, e così via. Tutti accomunati dal desiderio di avere accesso a una catena di lavoro di creazione 3D completamente gratuita e open source.

Blender Foundation sostiene e facilita questi obiettivi impiegando un piccolo staff sovvenzionato dalla comunità online. Questo software è un progetto open source rilasciato sotto licenza GNU GPL. Tutto il codice è scritto in C, C++ e Python.

Volontari e professionisti contribuiscono al rilascio ufficiale di Blender. Questo include sviluppatori, scripter, traduttori, progettisti e utenti che testano il software e ne danno un feedback.

Blender usa il linguaggio di programmazione Python per lo scripting delle sue API. Le Blender Python API sono basate su Python 3. Esse si integrano profondamente e vengono utilizzate per la scrittura di add-ons, la generazione di layout dell'interfaccia utente e l'importazione ed esportazione di molti formati di file.

Blender fornisce una gamma completa di strumenti di modellazione che lo rendono un software completo per tutte le esigenze che riguardano il mondo 3D. Questi strumenti di modellazione includono: scorciatoie da tastiera per un flusso di lavoro veloce, movimento, collasso e dissolvenza dei bordi e, infine, l'utilizzo e la creazione di script in Python per strumenti personalizzati e componenti aggiuntivi.

Anche per la fase di rendering, Blender è molto attrezzato, infatti, è dotato di un potente motore di rendering chiamato Cycles che offre splendida resa ultra-realistica, permettendo anteprime in tempo reale e supporto all'illuminazione HDR.

Con questo motore di rendering, le possibilità di creare materiali sono infinite grazie all'utilizzo del sistema di nodi che consente di utilizzare tutte le tecniche disponibili in cascata e in serie, mantenendo comunque realistiche le apparenti proprietà fisiche dei materiali.

La grande profondità di Blender si estende anche agli strumenti di supporto all'animazione. Infatti, sia lo skinning che il rigging diventano molto immediati grazie all'utilizzo di componenti di creazione automatica di gestione dello scheletro e dei pesi.

L'animazione viene gestita sempre tramite l'utilizzo dei keyframes. Questi permettono di creare animazioni professionali andando a sfruttare diverse tecniche e concetti di interpolazione. Ogni animazione potrà essere ulteriormente ampliata sfruttando la possibilità di muovere oggetti lungo le curve, tramite l'animazione non lineare(NLA) o la sincronizzazione audio.

Ritornando alle mesh, all'interno di Blender è possibile effettuare l'unwrapping seguendo diversi schemi automatici o andando a lavorare direttamente i bordi. Sarà proprio attraverso la finestra di unwrapping che potranno essere applicate textures oppure effettuare il painting direttamente sul modello.

Grazie agli strumenti di simulazione, è possibile migliorare il risultato generale delle scene ricreando qualsiasi necessità quali pioggia, fuoco, fumo, liquido, tessuto o di una distruzione completa sfruttando vari tools tra cui:

- **Fluidi** - acqua realistica e simulazioni di fluidi.
- **Fumo** - fumo con eventualmente fiamme e interazione con la scena.

- **Capelli** - banchi di capelli che reagiscono al soffio del vento e interagiscono con le collisioni.
- **Tessuti** - simulazioni di tessuto realistico per l'abbigliamento e gli ambienti
- **Fisica dei Corpi Rigidi** - Fra qualsiasi oggetto distruttibile e collidibile
- **Fisica delle Particelle** - Per la creazione di cose come pioggia, scintille o schegge.

Blender è anche dotato di un compositore interno, ciò significa che non è più necessario l'utilizzo di programmi di terze parti, poiché è possibile terminare un lavoro senza lasciare mai il software.

Considerando che il software è in grado di creare vere e proprie animazioni, viene fornito anche editor video. Il Video Editor permette di eseguire operazioni di base, come i tagli video ed il montaggio, nonché i compiti più complessi come il video masking.

Tutte queste operazioni saranno monitorabili e gestibili sfruttando l'anteprima dal vivo, istogrammi, mixaggio audio, sincronizzazione e visualizzazione della forma d'onda, fino a 32 slot per l'aggiunta di video, immagini, audio, scene, maschere ed effetti, controllo della velocità, dei livelli di regolazione, inserimento di transizioni, di fotogrammi chiave, filtri e altro ancora.

Con una grande comunità di appassionati e sviluppatori, Blender viene ampliato e migliorato tramite una vasta gamma di estensioni che è possibile attivare o disattivare facilmente. Alcune estensioni esistenti includono: generatori per alberi, terreni, edera e le nuvole, 3D Printing Toolbox, Rigify, sistema di meta-rigging e infine supporto e guida per l'esportazione e importazione di file provenienti da software esterni.

Per quanto riguarda i formati possibili di esportazione e salvataggio si hanno varie possibilità, tra cui:

- **Immagine**
JPEG, JPEG2000, PNG, TARGA, OpenEXR, DPX, Cineon, Radiance HDR, SGI Iris, TIFF
- **Video**
AVI, MPEG e Quicktime (su OSX).
- **3D**
Studio 3D (3DS), COLLADA (DAE), Filmbox (FBX), Autodesk (DXF), Wavefront (OBJ), DirectX (x), Lightwave (LWO), Motion Capture (BVH), SVG, Stanford PLY, STL, VRML, VRML97, X3D.

Nel caso di questa tesi, Blender è stato sfruttando per creare i modelli da utilizzare per la creazione dell'equipaggiamento in dotazione ai giocatori. Si è quindi esportato il lavoro nel formato utilizzato dalla stampante 3D MakerBot Replicator 2X, che utilizza il filamento in ABS, un comune polimero termoplastico sfruttato per creare oggetti leggeri e rigidi.

Capitolo 4

Progettazione e architettura

4.1 Panoramica generale

Mentre la fase di ricerca ha permesso di individuare, all'interno di un grande bacino di tecnologie e step disponibili, gli strumenti da utilizzare, la fase architettonica ha lo scopo di sviluppare un prototipo teorico realizzabile, tramite l'utilizzo delle tecnologie selezionate. Lo studio degli strumenti descritti è stato curato e approfondito, infatti, si è cercato di capire sia le funzionalità offerte, per applicarle in modo coerente al contesto, sia il modo da far cooperare all'unisono device e tecnologie apparentemente differenti.

Da una rapida occhiata a quelle che sono le features del game concept, s'intravede come la componente di comunicazione ricopra un ruolo fondamentale, ovvero, tutti i partecipanti alla partita devono avere la possibilità di inviare messaggi o segnalare situazioni di pericolo. A questo va ad aggiungersi la varietà e differenziazione del Team Leader, che non si trova fisicamente nello stesso luogo di azione dei membri della sua squadra, ma allo stesso tempo ha bisogno di avere dei feedback chiari e istantanei per una rapida scelta di azione strategica. Per questo, è obbligatorio che i dispositivi di tutti i giocatori, di qualsiasi ruolo e squadra, siano collegati a uno stesso server o rete. A questa complicazione, va anche aggiunta la flessibilità dei compiti richiesti dal device dei

giocatori nel campo di gioco.

L'Applicazione Mobile dovrà essere in grado di comunicare sia con il proprio Team Leader che con le schede Arduino installate nell'equipaggiamento, per fornire feedback al munizionamento e alla vita rimasta, ma anche rilevare, tramite il Bluetooth, i beacons sparsi nell'arena di gioco.

La grande complessità apparente del progetto è stata, per buona parte, superata dall'eccellente flessibilità e adattabilità di Unity. Con la stessa piattaforma, è stata realizzata sia l'applicazione Web utilizzata dai Team Leader, che l'applicazione Mobile, includendo tutte le features richieste dalla fase di concept.

Un vantaggio essenziale è stato dato anche dall'Asset Store di Unity e dalla sua crescente community che ha permesso di utilizzare svariati pacchetti e assets pronti per l'utilizzo, tra questi si ricorda Photon Network, che rappresenta il server Cloud al quale i device sono connessi durante ogni partita, e i Bluetooth Manager che sono una serie di script che permettono di accedere in maniera abbastanza rapida al sensore Bluetooth del dispositivo mobile.

Per stabilire la composizione architeturale del progetto, si è partiti dal Cloud che Photon Network mette a disposizione. Il cloud, garante di flessibilità e del bilanciamento del carico in caso di più utenti (e dopo averlo testato, anche di rapidità di invio dei messaggi di sincronizzazione), deve servire come punto di incontro di tutti i giocatori presenti nell'arena di gioco e non.

Team Leader e membri di ogni squadra, effettuano un'operazione di connessione non appena la partita inizia e viene assegnato, ad ogni partecipante, un numerico identificativo del device, per facilitare la comunicazione multicast e unicast all'interno del cloud stesso.

Al fine di facilitare la comprensione dell'architettura ideata, è stata creata l'immagine sottostante.

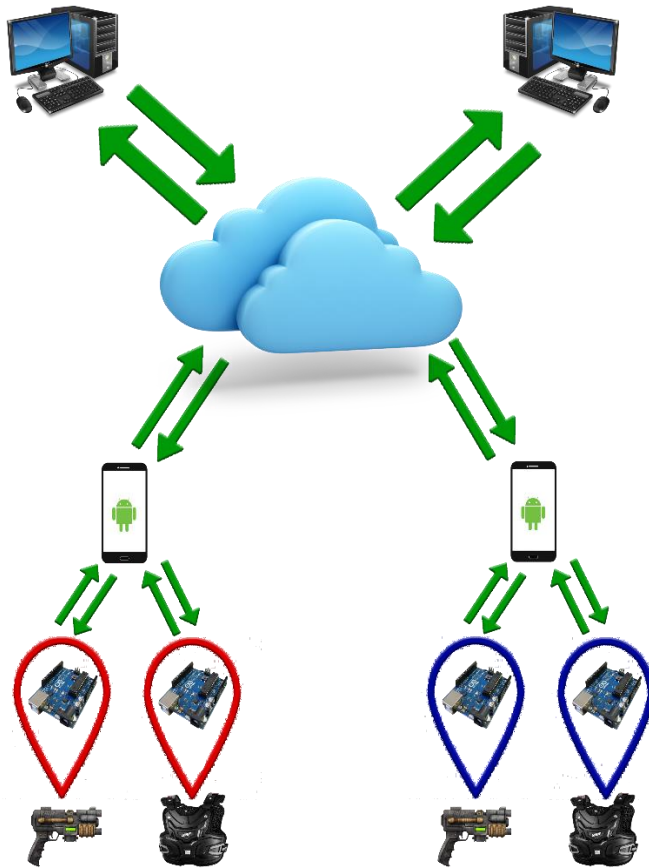


Figura 11 - Architettura

Come è possibile notare, il Cloud ricopre una parte centrale poiché svolge il ruolo di smistamento dei vari messaggi tra i soggetti del gioco. È di fondamentale importanza che ogni device, soprattutto quelli mobile, abbia una connessione internet al fine di poter collegarsi in maniera corretta al Cloud. Grazie alla duttilità di Photon Network, è possibile suddividere la rete disponibile in varie **Stanze** o **Rooms** che rappresentano istanze di gioco separate. In questo modo, è possibile utilizzare un unico Cloud per ospitare partite svolte in diverse parti del mondo. Le stanze vengono differenziate per nome ed è possibile richiedere anche l'inserimento di una password a chi tenterà di connettersi.

L'immagine mostra solo due device mobile connessi, ma permette di capire bene in quale punti avviene la comunicazione e quali sono i soggetti coinvolti. Il Pc rappresenta il Team Leader che, godendo di una postazione fissa, si collega da qualsiasi punto alla stanza di gioco. Il telefono Android rappresenta il dispositivo personale dell'utente che si trova fisicamente nell'arena di gioco. Invece, le schede Arduino rappresentano, rispettivamente, la pistola a infrarossi e la pettorina sensibile. Compito delle schede è quello di effettuare un check continuo sui dati di input, sia interni che esterni. In particolare dovranno:

- Rilevare la pressione dei tasti di input
- Inviare un codice Infrarosso utilizzando la periferica installata
- Rilevare, nel caso della pettorina, segnali infrarossi
- Modificare il conteggio di vita e munizionamento
- Inviare i nuovi dati al device mobile collegato

In merito all'ultimo punto, è stato fondamentale fare in modo di connettere, in maniera diretta, smartphone e schede Arduino. Per questo ogni scheda comunica, in maniera isolata dall'altra, con l'applicazione mobile tramite socket TCP, inviando stringhe di testo come identificativo dei messaggi scambiati. Questo metodo di comunicazione si è rivelato il più efficace e il più adatto visto che Unity permette di utilizzare le api del framework.NET.

Considerato l'alto numero di punti di comunicazione nell'architettura sviluppata, è stato necessario stabilire dove risiede la logica di gioco e qual è il device di riferimento da interrogare. Di fondamentale importanza, infatti, è fissare il corretto valore di vita e munizionamento iniziale per ogni giocatore. A tal proposito, si è deciso di dare al Team Leader di ogni squadra il ruolo di "Inizializzatore". Grazie a una serie di pacchetti inviati, esso potrà fornire al giocatore, appena connesso alla sua squadra, il corretto valore iniziale di munizioni e punti vita previsto per quella partita. Inoltre, data l'incapacità di gestire

thread da parte degli Arduino, il Team Leader ha lo scopo di sancire l'avvio del funzionamento delle periferiche infrarossi. Terminata questa fase di inizializzazione, il controllo della logica di gioco si sposterà direttamente sul device di ogni singolo giocatore. Il team Leader resta soltanto in "ascolto" su quelli che sono i pacchetti di sincronizzazione e sulle informazioni legate al giocatore, al solo scopo di aggiornarle sull'interfaccia grafica. Il Device del giocatore rappresenta, dunque, una sorta di ponte tra il Team Leader e le schede Arduino. Il device serve, dunque, a svolgere le seguenti funzioni:

- **Ottiene informazioni dalle schede Arduino in merito a vita e munizionamento:** ogni volta che il giocatore preme il pulsante di sparo o verrà colpito, le schede invieranno un messaggio testuale contenente le informazioni sull'evento avvenuto.
- **Elaborazione delle informazioni ricevute:** ad ogni evento ricevuto, il device effettuerà il calcolo di vita e munizionamento, direttamente in locale.
- **Invio del dato elaborato:** dopo aver controllato se vi è stata una variazione tra il dato precedentemente immagazzinato e quello ricevuto, invierà la nuova situazione al Team Leader, aggiornando le informazioni.
- **Interrogare device Bluetooth e invio informazioni al Team Leader:** l'applicazione Unity tramite l'utilizzo del modulo Bluetooth, interroga i Beacons rilevabili al fine di inviare l'informazione di prossimità al Team Leader.
- **Ricevere Bonus e Potenzamenti dal Team Leader:** anche nel caso in cui il Team Leader voglia inviare un potenziamento all'utente, lo farà inviando un determinato messaggio che viene interpretato dal Device stesso come un determinato evento da elaborare.

Di conseguenza, è possibile affermare che la vera logica di gioco, nonché elaborazione degli eventi di input, risiede nell'applicazione mobile. Questa scelta ha permesso di suddividere il lavoro richiesto ed effettuare l'elaborazione degli eventi direttamente sul device di ogni singolo giocatore, al fine di evitare un sovraccarico su quelli che erano i Team Leader di ogni squadra e migliorare la scalabilità del codice. In questo modo si tenta, anche, la protezione da eventuali disconnessioni o malfunzionamenti.

Il malfunzionamento di uno dei soggetti presenti nella partita, causa la disconnessione soltanto di quest'ultimo, evitando la reazione di un effetto a catena che potrebbe bloccare totalmente la partita.

Ogni utente è, dunque, responsabile per sé stesso e il Team Leader ha il ruolo di inizializzatore e supervisore dei parametri degli utenti.

Per quanto riguarda, invece, la gestione del timer di gioco e la sua relativa scadenza, quest'ultima sarà gestita da uno dei due Team Leader presenti durante la partita. Al termine del timer invieranno un pacchetto di termine partita a tutti gli utenti presenti nella stanza. Anche in questo caso, il messaggio di termine partita verrà elaborato e successivamente diffuso agli Arduino.

4.2 Implementazione



Figura 12 - Architettura di riferimento

Osservando l'architettura di riferimento è possibile individuare i punti principali dell'implementazione del game concept, che sono:

- **Applicazione Mobile** che nell'immagine viene identificata dallo smartphone Android

- **Cloud per la sincronizzazione** identificata dalla nuvola interposta tra i pc e gli smartphone
- **Rilevazione punti di interesse** tramite Bluetooth
- **Applicazione Team Leader** identificato dai computer connessi al Cloud
- **Connessione all'equipaggiamento** indentificato dagli Arduino

Tutti questi punti verranno analizzati nel dettaglio nei paragrafi successivi.

4.2.1 Applicazione Mobile

Dopo aver stabilito il funzionamento generico, tramite la fase progettuale e architettuale, si è passati alla fase implementativa al fine di realizzare, sotto forma di prototipo, quanto ricercato. È chiaro che lo smartphone deve essere dotato di un'applicazione capace di fornire feedback e funzionalità, ma è necessario capire quanto può essere sfruttato Unity per raggiungere l'obiettivo di progetto. L'applicazione mobile deve, infatti, svolgere quattro grandi compiti:

- Connettersi al Cloud di Photon Network
- Gestire l'interfaccia, parametri vitali e movimento
- Abilitare la connessione socket con le schede Arduino
- Rilevare la posizione stimata tramite la ricerca Bluetooth

Inizialmente, Unity si è mostrato come un ottimo software in grado di sviluppare applicazioni 3D per qualsiasi piattaforma, in maniera rapida e immediata. Tutto questo fino a quando non si è iniziato a utilizzare software di terze parti o script esterni che non riguardano l'architettura interna. In questo caso, Unity si è dimostrato parecchio ostico, soprattutto con le ultime due funzionalità, poichè è richiesto l'utilizzo di codice e API non facenti parte della propria pubblicazione. Tuttavia, grazie anche all'utilizzo di add-ons esterni, creati ad hoc, si è riusciti comunque a realizzare quanto richiesto.

4.2.1.1 Connessione al cloud e sincronizzazione

PhotonNetwork si è rivelato un framework di grande intuitività e facilità di utilizzo. Dopo essersi registrati, basta installare il pacchetto in formato unity3d per iniziare. Un rapido menu guiderà l'utente verso i passi necessari per l'impostazione iniziale. I settaggi principali potranno essere accessi tramite il file. **PhotonServerSettings** :

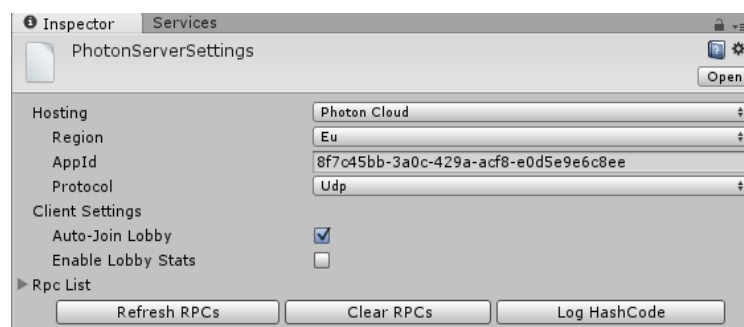


Figura 13 - Parametri Photon Server

Dall'immagine è possibile vedere come è possibile cambiare i parametri di funzionamento principali e anche il protocollo scelto per la comunicazione. In questo caso, si è scelto di utilizzare il Cloud, per evitare la programmazione della componente server del gioco e, vista la frequenza con la quale sarebbero avvenuti gli aggiornamenti, il protocollo UDP.

La gestione del PhotonCloud è stata affidata ad una classe chiamata **NetworkManager**. Questa classe risulta la prima ad essere inizializzata e eseguita dall'applicazione Unity ed è legata al ciclo vitale di quest'ultima. Gli scopi principali sono:

- **Preparare i settaggi iniziali:** la connessione al server di gioco avviene per step diversi. Il primo step da superare è quello della connessione al Cloud, successivamente al Lobby e infine alla stanza di gioco. Per facilitare le operazioni di connessione è possibile stabilire di connettersi direttamente al Lobby, senza invocare altri comandi. Questo settaggio avviene tramite la modifica della variabile **autojoinLobby** della classe **PhotonNetwork**.
- **Connessione:** la connessione avviene tramite l'invocazione di un metodo, sempre dalla classe **PhotonNetwork**. Nel caso del prototipo, per evitare connessioni inutili, quest'ultima viene avviata tramite la pressione di un tasto, da parte dell'utente.
- **Gestire le disconnessioni:** nel caso di disconnessione del giocatore, l'applicazione si sposterà nuovamente nel menu principale.

A connessione avvenuta, la classe **NetworkManager** ha lo scopo di istanziare il **GameObject Player**. L'istanziamento avviene sempre tramite una delle classi di **PhotonNetwork**, più nello specifico, la funzione **Istantiate**. Questa funzione, che prende come parametri di input il **GameObject** e la posizione, ha lo scopo di notificare a tutti i giocatori, presenti nella stanza di gioco, e a quelli futuri, di istanziare lo stesso oggetto.

Questo è, dunque, il primo messaggio di sincronizzazione che viene inviato al Cloud e successivamente a tutti i membri connessi.

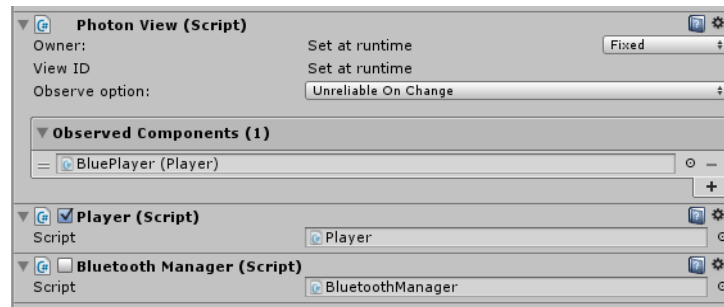


Figura 14 - Componenti utilizzate per un Player

La Figura 12 rappresenta la schermata delle componenti utilizzate all'interno dell'oggetto Player. Di fondamentale importanza, per garantire l'istanziamento dell'oggetto nella rete, è la componente **PhotonView**. Questa rappresenta lo script al quale viene legato l'identificativo univoco del giocatore e anche le informazioni sulle classi da sincronizzare.

All'interno dell'opzione **Observed Components** è possibile inserire la classe che ospiterà i metodi di sincronizzazione tra i giocatori della stanza. In questo caso, la classe sincronizzata è **Player.cs** che contiene al suo interno il metodo di sincronizzazione, per la serializzazione dei dati.

Il metodo di sincronizzazione è **OnPhotonSerializeView(...)** ed è richiesto da tutti gli oggetti PhotonView che "osservano" una determinata componente. Negli aggiornamenti per la serializzazione è consigliato inserire tutte quelle variabili che necessitano di updates continui, quali il movimento o i parametri vitali.

In ogni caso, il metodo si suddivide in due parti principali:

- **Invio parametri:** questa parte è segnalata dal booleano **stream.isWriting** e rappresenta la parte di codice che verrà eseguita nel caso in cui si è in possesso dell'oggetto in esame. Tramite la funzione **stream.SendNext(..)** è possibile inviare, nel flusso di dati, variabili base tra cui interi, decimali, float, stringhe e booleani. Non è possibile inviare strutture dati o intere istanze di classi a causa della serializzazione e deserializzazione.
- **Ricezione parametri:** questa parte è eseguita da tutti gli utenti che non sono in possesso dell'oggetto in esame. Come tutte le operazioni di deserializzazione è necessario seguire lo stesso ordine di invio, avvenuto in prima fase da parte del proprietario dell'oggetto.

Nella Figura 13 è possibile vedere una schermata del metodo `OnPhotonSerializeView` della classe `Player.cs`.

```
void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.isWriting)
    {
        stream.SendNext(health);
        stream.SendNext(ammo);
        stream.SendNext(status);
        stream.SendNext(blueDistance);
        stream.SendNext(purpleDistance);
        stream.SendNext(greenDistance);
        stream.SendNext(isMoving);
        stream.SendNext(isDead);
        stream.SendNext(inDanger);
    }
    else
    {
        this.health = (int)stream.ReceiveNext();
        this.ammo = (int)stream.ReceiveNext();
        this.status = (string)stream.ReceiveNext();
        this.blueDistance = (float)stream.ReceiveNext();
        this.purpleDistance = (float)stream.ReceiveNext();
        this.greenDistance = (float)stream.ReceiveNext();
        this.isMoving = (bool)stream.ReceiveNext();
        this.isDead = (bool)stream.ReceiveNext();
        this.inDanger = (bool)stream.ReceiveNext();
    }
}
```

Figura 15 - Metodo `OnPhotonSerializeView`

In particolare vengono inviati e ricevuti:

- **Parametri del giocatore:** parametri vitali e munizionamento attuale.
- **Informazioni sulla posizione ambientale:** dopo aver elaborato le informazioni Bluetooth, vengono inviati agli altri giocatori le informazioni inerente alla distanza dai Beacons.
- **Parametri di Movimento:** elaborando le informazioni dell'accelerometro, viene inviato un booleano per notificare se il giocatore si sta muovendo o è fermo.
- **Segnalazione di pericolo:** nel caso di pressione del bottone di segnalazione da parte dell'utente.

Oltre al metodo di serializzazione, vengono utilizzate numerose funzioni **RPC** con frequenza minore rispetto al metodo precedente, tra cui:

- **Inizializzazione degli Arduino:** invocata dal Team Leader per inviare la stringa di avvio delle periferiche.
- **Invio ordini:** invocata sempre dal Team Leader per inviare messaggi testuali al giocatore
- **Aggiornamento parametri a seguito di upgrade:** prerogativa del Team Leader.

L'oggetto Player permane nella scena di gioco per tutta la durata della partita e finché non avviene una disconnessione. Quest'ultima causerà la distruzione dell'oggetto e la seguente notifica verrà inviata a tutti gli utenti connessi in quel momento.

4.2.1.2 Connessione Bluetooth e dati ambientali

Per poter stabilire una presenza fisica all'interno dell'arena di gioco e fornire feedback al Team Leader di ogni squadra, l'oggetto Player è stato dotato della classe **BluetoothManager** con lo scopo di interrogare e ricercare i Beacons sparsi. Questa classe sfrutta l'addon Bluetooth, creato dalla compagnia Shatalmic, disponibile nell'Asset Store di Unity sotto il nome di **Bluetooth LE for iOS and Android**. L'addon permette di effettuare l'inizializzazione e la de-inizializzazione della componente Bluetooth del device, effettuare lo scan dei dispositivi, ottenere RSSI e informazioni o sottoscrivere.

Tra tutti i metodi elencati, all'interno della classe BluetoothManager sono stati utilizzati l'inizializzazione, la de-inizializzazione e lo scan dei periferici. L'immagine sottostante mostra uno schema di come le funzioni Bluetooth vengono utilizzate all'interno della logica di gioco:

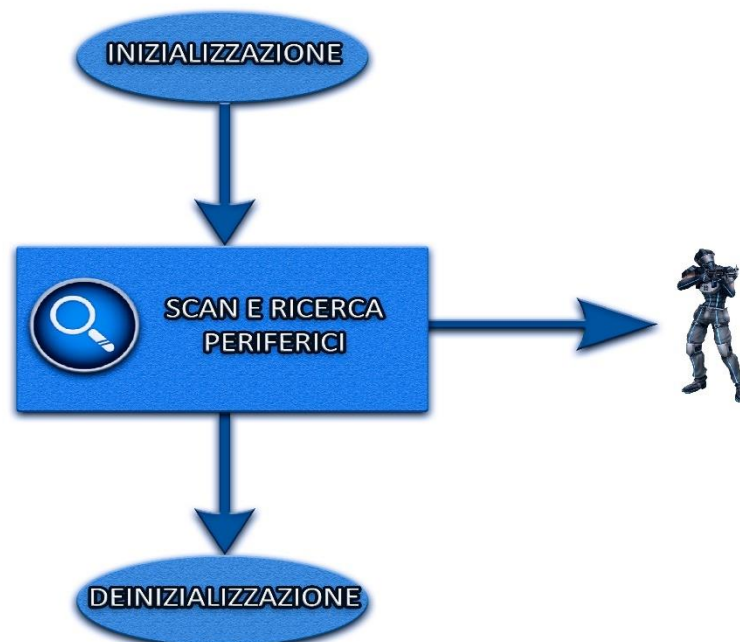


Figura 16 - Funzionamento comunicazione Bluetooth

La fase di inizializzazione viene effettuata non appena il GameObject Player viene istanziato, per evitare che questa operazione blocchi le successive azioni. Una volta inizializzato, la classe metterà in ascolto il device tramite l'utilizzo della funzione di Scan. All'interno dell'avvio della funzione di scan va dichiarato un'azione di Callback che viene richiamata tutte le volte che accade il ritrovamento di un dispositivo Bluetooth. Nel caso attuale, la funzione Callback è stata nutrita con quelli che erano gli indirizzi MAC degli estimate in possesso e inseriti dei controlli per ottenere le informazioni di RSSI corrispondenti.

```
void SearchForEstimate()
{
    BluetoothLEHardwareInterface.ScanForPeripheralsWithServices(null, null, (address, name, rssi, advertisingInfo) =>
    {
        if (advertisingInfo != null)
        {
            if (address == blueEstimateMAC)
            {
                bDistance = rssi;
                bArrived = true;
            }
            if (address == greenEstimateMAC)
            {
                gDistance = rssi;
                gArrived = true;
            }
            if (address == purpleEstimateMAC)
            {
                pDistance = rssi;
                pArrived = true;
            }
        }
    });
}
```

Figura 17 - Metodo SearchForEstimate

Nel caso in cui il device rilevato corrisponda a uno degli indirizzi MAC memorizzati, l'informazione RSSI è immagazzinata all'interno di variabili ad hoc. Quest'operazione viene effettuata nel modo più veloce possibile per evitare sovraccarichi. Infatti, è stato notato come la ricerca Bluetooth rappresenti una sorta di collo di bottiglia per le applicazioni Unity e il richiamo continuo dell'azione di Callback si è rivelato poco

efficiente ma necessario. Nel metodo `Update()` della classe, queste variabili vengono elaborate. Considerando che il valore RSSI è in scala decibel, è stato necessario effettuare un'operazione di conversione per ottenere la corretta distanza in metri. Una volta ottenuta la distanza in metri, questa è stata inviata alla classe `Player` del `GameObject`. Nel caso in cui il giocatore si disconnette dalla partita, il Bluetooth viene deinizializzato per evitare conflitti con le successive partite e inizializzazioni.

I dati ottenuti, dallo scan dei periferici, sono elaborati e ricevuti dalla classe `Player` tramite delle funzioni *setter*. Il dato è sovrascritto in quelli che sono i booleani della classe e inviati agli altri giocatori tramite la serializzazione. Inoltre, la vicinanza con uno dei Beacons è utilizzata anche in caso di morte del giocatore.

Solo nel caso in cui il giocatore si trovi vicino al Beacon Madre potrebbe resuscitare e tornare nuovamente a combattere nell'arena di gioco. Questo requisito è gestito tramite un timer che diminuisce secondo per secondo fin tanto che il giocatore si trova nelle vicinanze del Beacon della sua squadra.

4.2.1.3 Connessione Arduino e Sockets

A differenza del Bluetooth, per gestire la connessione Socket, si è deciso di realizzare una classe manager chiamata **ArduinoCommunication** il cui ciclo vitale non è legato al `GameObject Player` ma, bensì, all'istanza di gioco. Questa scelta è avvenuta perché il device del giocatore deve essere sempre collegato agli Arduino fin tanto che l'applicazione è aperta. In questo modo si facilita anche la duttilità dell'applicazione nel caso di partite successive.

La classe `ArduinoCommunication` presenta alcuni metodi per gestire l'apertura, la chiusura e l'invio/ricezione dei messaggi. L'api utilizzata è quella propria del framework .NET di C#. Per ogni scheda Arduino, è creato un oggetto **TcpClient**, **NetworkStream**, **StreamWriter** e **StreamReader** per facilitare le operazioni di connessione, chiusura e comunicazione tra i peer.

L'operazione di setup del socket e degli stream avviene prima di connettersi al Cloud tramite la pressione dei tasti di connessione alle varie schede. Ogni tasto prevede una scheda di feedback per verificare se la connessione è avvenuta in maniera corretta o meno.

Da questo momento, le schede Arduino, che hanno rilevato un Client corretto, si possono porre in condizione di attesa della stringa `"startgame"` che è inviata dal Player non appena la partita viene avviata. Nel corso della partita vengono utilizzati i metodi di scrittura e lettura del flusso dati per ottenere informazioni degli eventi di sparo, colpito o potenziamento.

L'immagine sottostante rappresenta le funzioni di scrittura e lettura per il socket legato all'Arduino con il compito di pistola.

```
void writeGunSocket(string theLine)
{
    // function to write data out
    if (!gunSocketReady)
        return;
    String tmpString = theLine;
    writerGun.Write(tmpString);
    writerGun.Flush();
}
String readGunSocket()
{
    // function to read data in
    if (!gunSocketReady)
        return "";
    if (gunStream.DataAvailable)
        return readerGun.ReadLine();
    return "NoData";
}
```

Figura 18 – Metodo di invio e ricezione stringhe

In ogni caso, qualsiasi informazione è inviata sotto forma di stringa. Per evitare malfunzionamenti ogni stringa è seguita da una serie di parole identificative che permettono di riconoscere l'evento legato al pacchetto inviato. Ogni parola identificativa è seguita da una virgola e successivamente, se necessario, dal valore del parametro.

Tra queste abbiamo:

- *Life*: indica un pacchetto d'invio del parametro vita del giocatore
- *Ammo*: indica un pacchetto contenente le informazioni delle munizioni disponibili in quel momento.
- *Startgame*: stabilisce l'inizializzazione delle periferiche a infrarosso
- *Godmode*: corrispondente all'inizializzazione del potenziamento di Invulnerabilità, disabilita per un periodo la possibilità di essere colpiti.
- *Dead*: comando inviato non appena la vita è a 0
- *Alive*: comando inviato dopo essere ritornati in vita essendo in prossimità del Beacon Madre.

4.2.1.4 Interfaccia utente e feedback di movimento

È possibile accedere alle funzionalità sopra descritte, tramite l'interfaccia utente pensata per il giocatore. L'interfaccia utente è stata realizzata utilizzando Photoshop per creare i file immagine necessari. Contrariamente a quella del Team Leader, l'applicazione mobile non necessita di una navigazione del menu, bensì di pannelli semplici per permettere la connessione alle schede Arduino, al Cloud e infine alla visualizzazione dei parametri.

Il primo pannello, che apparirà all'utente, è quello iniziale contenente svariati InputFields per inserire:

- il nome del giocatore
- il nome della stanza di gioco

Oltre agli InputField saranno presenti alcuni bottoni che permetteranno di:

- Scegliere la squadra in cui giocare
- Connettersi al cloud e alla partita di gioco.
- Passare al menu Setup per impostare la connessione con gli Arduino

Per ricordare, in maniera più evidente, quale squadra è stata selezionata dal giocatore una fascia semi-trasparente di colore rosso o blu, ricoprirà la parte bassa di ogni pannello dell'interfaccia.

L'immagine sottostante offre una replica del pannello iniziale mostrato al giocatore.

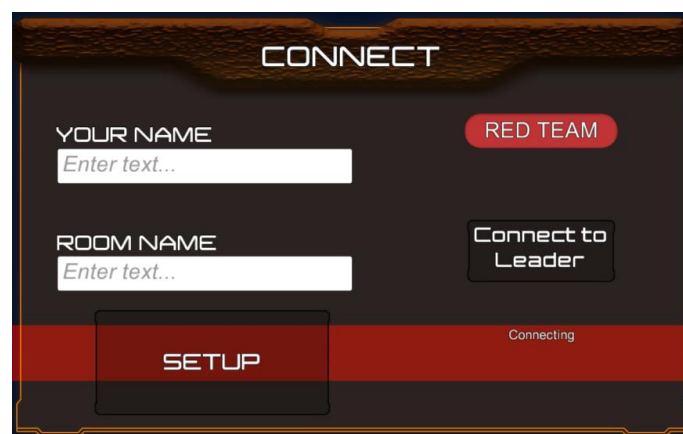


Figura 19 - Interfaccia iniziale applicazione mobile

Il bottone Connect to Leader, eseguirà la connessione al Cloud, utilizzando la stringa Connecting per dare un feedback sulla riuscita dell'operazione.

Per offrire solidità al codice, sono stati inseriti alcuni check sulla validità delle stringhe inseribili, sia nel nome (non inferiore a 2 caratteri) sia la stanza (senza caratteri speciali e non inferiore a 2 caratteri). Inoltre, risulterà obbligatorio effettuare prima la connessione agli Arduino, cliccando il tasto Setup.

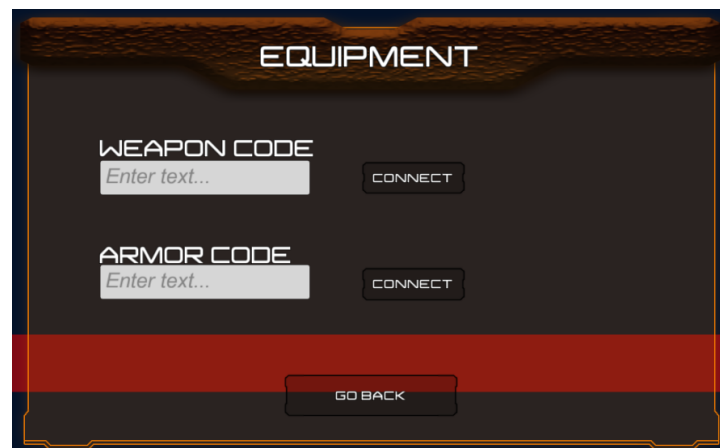


Figura 20 - Schermata inserimento codice dell'equipaggiamento

Il Tasto Setup mostrerà il pannello EQUIPMENT, in cui è possibile inserire nei due InputFields il codice identificativo degli Arduino.

I tasti di connessione agli Arduino, sono stati separati per fornire un migliore feedback di riuscita dell'operazione e capire, in caso di problemi, a quale dispositivo non ci si è collegati correttamente.

Il feedback dell'operazione viene fornito grazie a due icone rappresentative, che appariranno alla destra dei tasti connect:



Figura 21 - Simboli connessione con equipaggiamento riuscita/fallita

L'immagine sulla destra, indica l'insuccesso dell'operazione di connessione, mentre l'immagine sulla sinistra ne rappresenta il successo.

Quando si effettua la connessione agli Arduino, ottenendo la prima immagine di feedback, è possibile tornare al pannello iniziale, tramite il tasto Go Back, e connettersi al Cloud e quindi alla partita tramite il tasto Connect to Leader.

Una volta premuto il bottone di connessione alla partita, verrà visualizzato il pannello di gioco.



Figura 22 - Interfaccia di gioco smartphone

Il pannello di gioco mostrerà continuamente la situazione aggiornata dei punti vita e delle munizioni disponibili al giocatore. La stringa di Testo gialla, presente al centro della schermata, mostra l'ultimo ordine inviato dal Team Leader. Inoltre, sono presenti due bottoni:

- **Bottone di Emergenza:** in basso a sinistra, punto esclamativo su sfondo giallo. Questo bottone, se premuto, invia una segnalazione di emergenza al Team Leader che è segnalata, ad entrambi gli utenti, sotto forma di un'animazione rossa lampeggiante.
- **Bottone di Uscita:** nel caso in cui il giocatore voglia abbandonare prematuramente la partita o nel caso in cui ha effettuato un'errata impostazione. La pressione di questo bottone apre un popup in cui viene confermata la scelta intrapresa.

Nel caso in cui i punti vita saranno 0, apparirà il pannello di Morte:

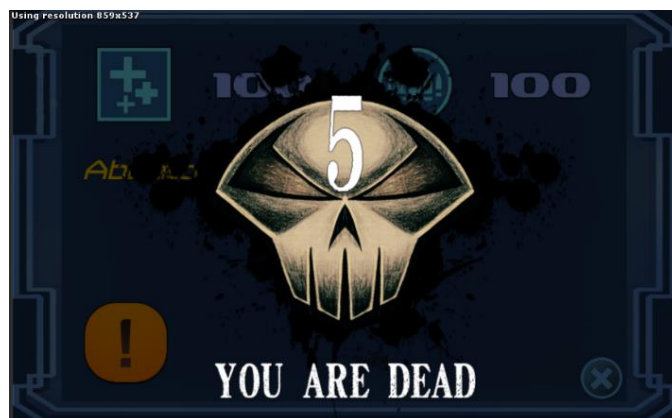


Figura 23 - Pannello che segnala al giocatore di essere stato ucciso

All'apparizione di questo pannello, il giocatore deve indirizzarsi verso la sua base e, più nello specifico, verso il Beacon Madre. Una volta arrivato nelle vicinanze della base, appare un numero indicante i secondi rimanenti al Respawn. Non appena il timer è esaurito, il pannello di morte svanirà lasciando nuovamente spazio al pannello di gioco.

Durante tutto lo svolgimento della partita, il device dell'utente utilizza l'accelerometro per fornire un ulteriore feedback al suo Team Leader. Lo scopo di questo feedback è quello di far capire al Team Leader se l'utente si sta muovendo o sta correndo lungo l'arena di gioco.

Unity permette di accedere egregiamente a quelle che sono le componenti del vettore movimento rilevato dall'accelerometro in maniera molto rapida e non invasiva. È stata creata, quindi, una funzione ad hoc che differenzia, con un controllo basato su una soglia, l'ultimo dato rilevato e il precedente.

```
void CheckAccelerometer()
{
    if (Mathf.Abs(Input.acceleration.x - lastAccelerometerX) > 0.6f)
    {
        lastAccelerometerX = Input.acceleration.x;
        isMoving = true;
    }
    else
    {
        if (!couroutineRunning)
            StartCoroutine(ChangeMovingStatusAfterTime());
    }
}
```

Figura 24 - Funzione per la gestione dell'accelerometro dello smartphone

Se la soglia è soddisfatta il booleano `isMoving` viene settato a `true`. In tutti gli altri casi, viene avviata una `Coroutine` per far cambiare il valore `isMoving` in maniera graduale, dopo un determinato intervallo di tempo.

4.2.2 Applicazione Team Leader

Per innalzare il livello di giocabilità ai massimi livelli, si è deciso di rilasciare la piattaforma Team Leader direttamente su Web, sfruttando Unity. Applicazione Mobile e Applicazione Web sono simili; molte delle classi condividono lo stesso funzionamento, mentre le altre hanno una diversa implementazione.

Questa disambiguità di funzionamento, tra l'applicazione Mobile e Web, è dovuta al modo di operare di Unity e di Photon Network. Unity incapsula gli oggetti di ogni scena all'interno dei **GameObject** mentre PhotonNetwork fa di questi **GameObject** la materia prima per la sincronizzazione in rete. Ogni qual volta si effettua un Istantiate tramite PhotonNetwork, viene richiesto un determinato GameObject che appare in tutte le scene dei giocatori presenti. Di conseguenza, nel caso di progetti diversi, il GameObject da sincronizzare deve avere necessariamente lo stesso nome e condividere le stesse componenti di base.

Questo comportamento, che apparentemente rappresenta uno svantaggio, si è rivelato un ottimo metodo per diversificare il funzionamento tra Mobile e Web, andando a creare GameObject apparentemente uguali, ma profondamente diversi. Un esempio calzante è quello dell'oggetto **Player**, anch'esso presente nell'applicazione Mobile e che rappresenta il fulcro dell'azione di gioco. Lo stesso script **Player** è presente nella versione web ma ha una diversa implementazione che serve a fornire ciò che è strettamente necessario al Team Leader.

L'applicazione del Team Leader risulta più complessa di quella Mobile. Il Team Leader, per definizione, non stanzierà all'interno della stessa arena di gioco degli altri membri della sua squadra, bensì si trova a casa o in qualsiasi altro luogo senza ricevere alcun supporto da parte di altri giocatori o supervisori. Di conseguenza è fondamentale rendere l'applicazione il più possibile navigabile e completa per rendere l'azione di gioco fluida ed esente da problemi.

Nel complesso, l'applicazione si suddivide in una serie di pannelli utente ognuno con uno scopo specifico:

- Pannello Iniziale
- Selezione Squadra
- Lobby
- Pannello di Gioco

I pannelli si susseguono nell'ordine scritto e vengono invocati ogni qual volta che il giocatore preme un determinato bottone. Da ogni pannello è possibile tornare a quello precedente tramite la pressione del tasto ESC o del bottone di uscita.

4.2.2.1 Pannello iniziale

Il pannello iniziale apparirà per primo al giocatore che decide di svolgere il ruolo di Team Leader. Lo scopo del pannello è quello di permettere al giocatore l'inserimento del suo Nickname e il nome della stanza di gioco tramite gli appositi InputField.

In background, il pannello effettuerà alcune impostazioni iniziali tra cui:

- **Connessione a PhotonNetwork:** con i metodi illustrati precedentemente per l'applicazione mobile
- **Inizializzazione Variabili:** in questa fase vengono letteralmente trovati tutti gli oggetti della scena necessari a garantire il funzionamento del gioco.
- **Gestione errori di inizializzazione:** Nel caso in cui l'inizializzazione non ha avuto successo, viene notificato il tutto all'utente tramite una stringa di testo
- **Gestione Pressione tasto Connect:** il tasto Connect è ciò che avvierà la connessione alla stanza di Gioco e ha anche lo scopo di verificare che i dati immessi siano validi

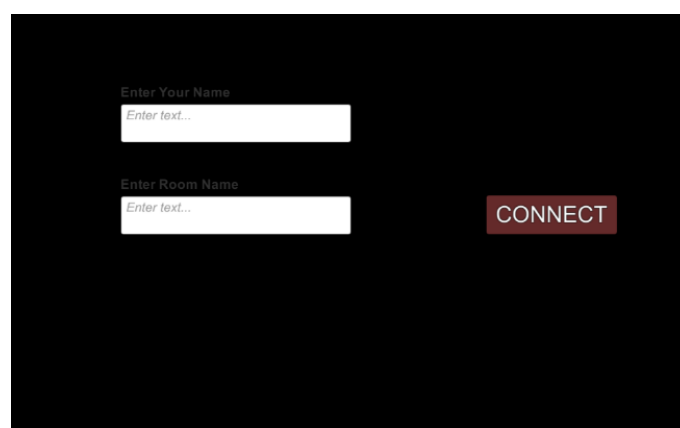


Figura 25 - Creazione della stanza di gioco

4.2.2.2 Selezione squadre



Figura 26 - Pannello di scelta del team di appartenenza

Una volta connessi alla stanza di gioco, l'utente dovrà selezionare la sua squadra di appartenenza. La pressione di uno dei due tasti di scelta scatena l'istanziamento di uno dei GameObject legati al TeamLeader: **MasterBlue** o **MasterRed**. Questi due oggetti sono identici in termini di componenti e funzionamento ma hanno un **TAG** differente che specifica la squadra di appartenenza. Il tag è uno strumento messo a disposizione da Unity per identificare una determinata categoria di appartenenza di un GameObject all'interno della scena. Il tag, assieme ai metodi di ricerca dei GameObject, ha permesso di effettuare un controllo sulla squadra selezionata dall'utente ed evitare che si potessero selezionare squadre già occupate.

Il fulcro del funzionamento di questo controllo appartiene alle funzione **FindGameObjectsWithTag()** che permette di ottenere tutti i GameObject dotati di un determinato tag. Il controllo viene effettuato sul tag **RedTeam** e **BlueTeam** e successivamente viene messo a sistema ciò che è stato trovato con ciò che ha premuto l'utente per stabilire la squadra di appartenenza.

Una volta stabilita la squadra verrà istanziato il corretto oggetto **Master** all'interno della stanza e, quindi, a tutti gli utenti connessi. L'istanziamento dell'oggetto Master, richiama un'altra serie di funzioni **RPC** che aumenta i conteggi dei giocatori totali connessi e imposta il nome del giocatore. È necessario utilizzare le funzioni RPC per avere la stessa istanza di gioco tra tutti gli utenti connessi nella stessa stanza.

4.2.2.3 Lobby

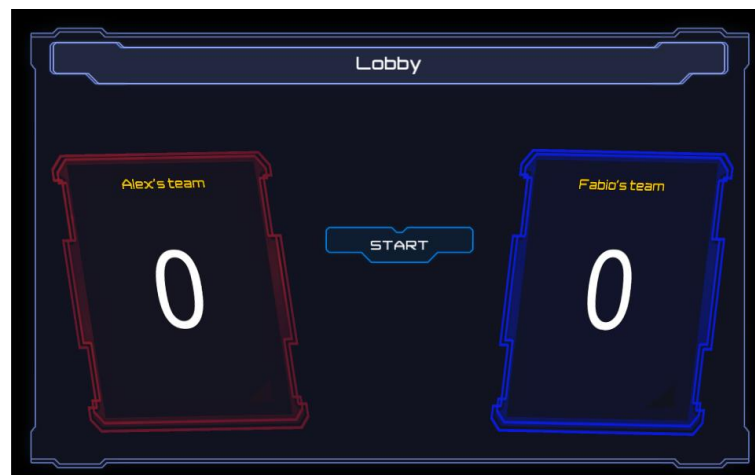


Figura 27 - Pannello indicativo dei giocatori connessi divisi per squadre

Il pannello Lobby è quello che permette di capire, a entrambi i Team Leader, la composizione della loro squadra in termini di giocatori connessi e anche il nome del giocatore avversario. Il numerico dei giocatori connessi è indicato dal numero in bianco contenuto all'interno del frame di squadra e quest'ultimo viene aggiornato tramite una funzione RPC. Questa funzione, viene richiamata in maniera indipendente da tutti gli

oggetti **Player** che vengono istanziati nella rete sfruttando gli eventi **OnPhotonPlayerConnected()** e **OnPhotonPlayerDisconnected()** che sono richiamati rispettivamente al join e all'abbandono della partita. Il menu Lobby prevede, inoltre, un tasto **Start** posto al centro della schermata. Lo scopo del tasto start è notificare all'altro Team Leader che si è pronti per iniziare la partita. La pressione del tasto è accompagnata da un'animazione che farà lampeggiare il frame di squadra e dall'apparizione di una stringa **READY**.

Quando entrambi i giocatori premono il tasto Start, il pannello Lobby viene disabilitato per lasciare spazio al pannello di gioco.

4.2.2.4 Pannello di gioco

La visualizzazione dei dati critici, dei membri della propria squadra, punteggi e azioni strategiche vengono gestite dal pannello di gioco. Per facilitarne l'utilizzo, il pannello di gioco è stato suddiviso in vari sottopannelli forniti di un bordo di navigazione posto nella parte superiore della schermata. La navigazione tra i vari sottopannelli avviene tramite la pressione dei vari bottoni che li identificano.

Nel bordo superiore viene anche mostrato il tempo rimanente prima del termine della partita. La totalità dell'interfaccia utente, viene personalizzata con un colore differente a seconda della squadra di appartenenza.

L'interfaccia è rossa per i giocatori della squadra rossa e blu per i giocatori della squadra blu. Il primo sottopannello è quello del Sommario.

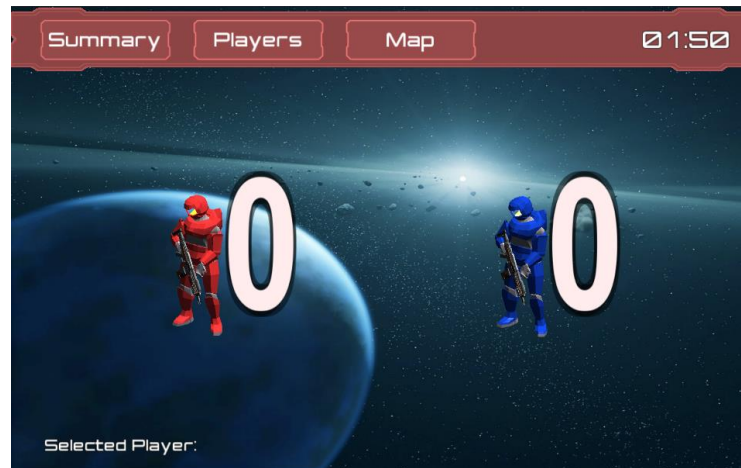


Figura 28 - Pannello dei punteggi

Nel sommario vengono mostrati i punteggi acquisiti dalla propria squadra fino a quel momento. I punteggi sono aggiornati dall'oggetto **Player** di ogni giocatore, non appena la variabile "punti vita" arriva a 0. A questo punto, sarà inviata una funzione RPC, che viene, quindi, eseguita da tutti i giocatori online, per aumentare di 1 il punteggio della squadra avversaria. Anche da questo, si può notare come il gioco sia essenzialmente **Player**-centrico, nel senso che l'oggetto Player gestisce quasi la maggior parte dei sistemi di funzionamento.

Con le funzioni RPC si vede, anche, la differenza implementativa tra mobile e web. PhotonNetwork obbliga a dichiarare, in ogni istanza di gioco, le RPC utilizzate all'interno di un'apposita lista. Tuttavia, in questo caso, alcune di queste funzioni sono prive di implementazione nella parte mobile o nella parte Web, oppure, hanno un'implementazione diversa a seconda dello scope nel quale si trova.

Un esempio è il caso dell'RPC **StartGame**. Questa RPC è inviata dal Team Leader non appena entrambi i giocatori hanno premuto il tasto Start.

La parte web l'RPC in questione ha lo scopo di inizializzare alcune variabili e avviare il pannello di gioco, mentre nella parte mobile ha lo scopo di far inviare ad ogni Arduino appartenente ai giocatori il messaggio di Inizio gioco per le periferiche infrarosso.

Per una questione puramente estetica, i punteggi vengono accompagnati da un modello 3d di un soldato. Cliccando il pulsante Players, si passerà al pannello dedicato alla gestione dei membri della propria squadra.

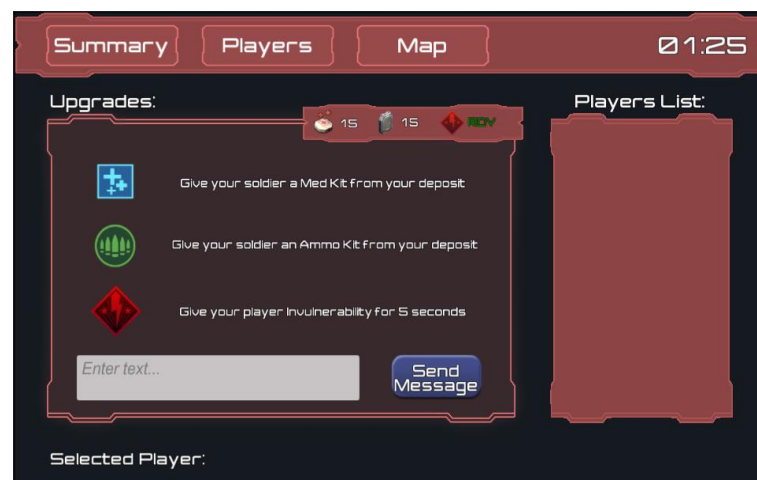


Figura 29 – Pannello Players dell'applicazione web

Questo pannello è suddiviso in due parti:

- **Upgrades:** all'interno di questo riquadro vengono mostrati e descritti tutti i bonus e i potenziamenti che è possibile inviare al giocatore
- **Players List:** in questo riquadro vengono mostrati i giocatori appartenenti alla propria squadra.

La `playerList` viene popolata degli oggetti grafici chiamati **PlayerElement**. Ogni `PlayerElement` è dotato di un bottone, una stringa testuale e un modello 3D rappresentante il giocatore corrispondente.



Figura 30 - Dati visualizzati dal Team Leader appartenenti al componente di squadra

Il `PlayerElement` non viene sincronizzato in rete ma viene istanziato dall'oggetto **Player**. Infatti, non appena viene effettuata l'inizializzazione della classe appartenente all'oggetto, viene trovato il riquadro `PlayersList` e aggiunto l'oggetto `PlayerElement` che

modifica quelli che sono i parametri che lo riguardano. Il modello 3D viene invece “posizionato” direttamente sopra l’oggetto `PlayerElement`.

Durante lo svolgimento della partita, il `PlayerElement` viene continuamente aggiornato sulla base degli update ricevuti dal giocatore tra cui: punti vita, munizionamento disponibile e anche animazioni del modello 3D. Quest’ultimo, invece, viene animato grazie ad una macchina a stati creata all’interno della componente **AnimatorController**.

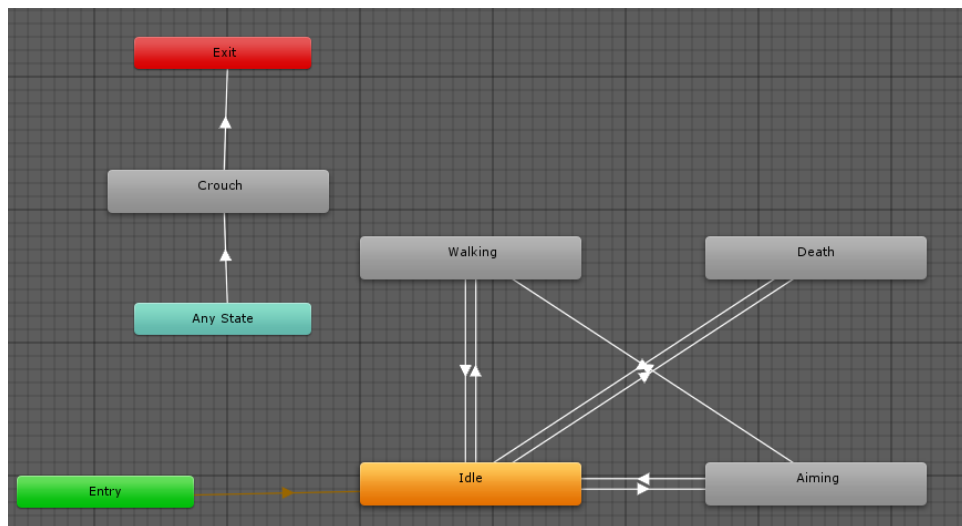


Figura 31 - Macchina a stati dell'animazione del player

Questa macchina utilizza variabili booleane per muoversi tra i vari stati. Ogni stato rappresenta una diversa animazione:

- **Idle**: animazione default del modello 3D. Questa animazione viene avviata nel caso in cui non vi sono dati di input da parte del giocatore corrispondente

- **Aiming:** nel caso in cui il giocatore sta sparando, il modello 3D si posizionerà in modalità di puntamento avviando anche un sistema particellare che simulerà l'esplosione del colpo.
- **Walking:** sulla base dei dati ricevuti dall'accelerometro del dispositivo, verrà impostata questa animazione per notificare che il giocatore si sta spostando.
- **Death:** nel caso in cui il giocatore viene ucciso.
- **Crouch:** questa animazione è diversa dalle altre perchè può essere avviata a partire da qualsiasi stato. Ad ogni modo, viene lanciata nel caso in cui il giocatore clicca il pulsante d'emergenza dalla sua interfaccia. L'animazione Crouch è seguita anche da un cambio di colore continuo, dal rosso a quello iniziale, per segnalare meglio la situazione di emergenza al Team Leader.

I dati di navigazione della macchina a stati vengono ottenuti tramite il metodo `OnPhotonSerializeView` che è presente anche nell'implementazione Mobile. Per inviare un potenziamento ad un determinato giocatore, basterà selezionarlo cliccando sul pulsante corrispondente e, infine, cliccare sul potenziamento o kit desiderato. Il nome del giocatore selezionato, apparirà in basso accanto alla stringa "Selected Player".

In alto a destra al pannello Upgrades è sempre presente un riquadro riepilogativo di quelli che sono i kit disponibili e lo stato del potenziamento **GodMode**. Il giocatore farà bene a dosare l'utilizzo dei kit poichè non sarà possibile ottenerne altri nel corso della partita. Il potenziamento **GodMode** può essere inviato ogni 2 minuti dal suo ultimo utilizzo.

Tutti i potenziamenti e kit inviabili, vengono gestiti dalla classe **UpgradeManager.cs**. Questa classe non fa altro che memorizzare il giocatore selezionato e utilizzare dei metodi pubblici contenuti all'interno della classe del giocatore. Ogni metodo, evocherà al suo interno una funzione RPC per poter aggiornare in modo esatto il nuovo valore di punti vita o munizionamento a tutti i giocatori connessi.

Oltre ai potenziamenti è possibile inviare anche dei messaggi testuali utilizzando l'`InputField` e il bottone residenti nella parte inferiore del riquadro Upgrades. Il messaggio

sarà inviato esclusivamente al giocatore selezionato e apparirà come stringa gialla nel device del giocatore. Anche in questo caso, il messaggio sarà inviato tramite RPC che mira a cambiare una variabile locale stringa all'interno della classe del giocatore.

Infine vi è il Pannello Map selezionabile dall'ultimo bottone del bordo superiore.



Figura 32 - Mappa dislocazione Beacons Estimote

All'interno di questo pannello sono inseriti 3 oggetti di colore diverso che rappresentano i Beacons sparsi per la mappa. Questi oggetti, inizieranno ad animarsi cambiando il colore in rosso, nel caso in cui il giocatore selezionato si trovi in prossimità di uno di essi. Anche in questo caso, questa informazione è ottenuta tramite il metodo `OnPhotonSerializeView` e gli oggetti Beacon vengono animati grazie ad una macchina a stati con due stati complessivi.

A causa dei problemi descritti nei capitoli precedenti, non è stato possibile fornire un feedback più completo, inerente alla posizione del giocatore, al Team Leader. Tuttavia

grazie al modello 3D, alle sue animazioni e all'informazione di prossimità la componente strategica rimane comunque valida e intatta.

In un'ipotesi di partita, ogni Team Leader navigherà spesso tra i vari pannelli verificando ogni volta i dati di interesse, inviando upgrades ai giocatori selezionati, ordini sulla base di quelle che sono i feedback ricevuti dal modello 3D, punti vita e munizionamento e rilevazione della posizione approssimata.

Non appena la partita è terminata, allo scadere del timer, si abiliterà il pannello **EndGame** che mostrerà con una stringa di colore rosso o blu a seconda della squadra selezionata se il giocatore ha vinto o perso la partita.

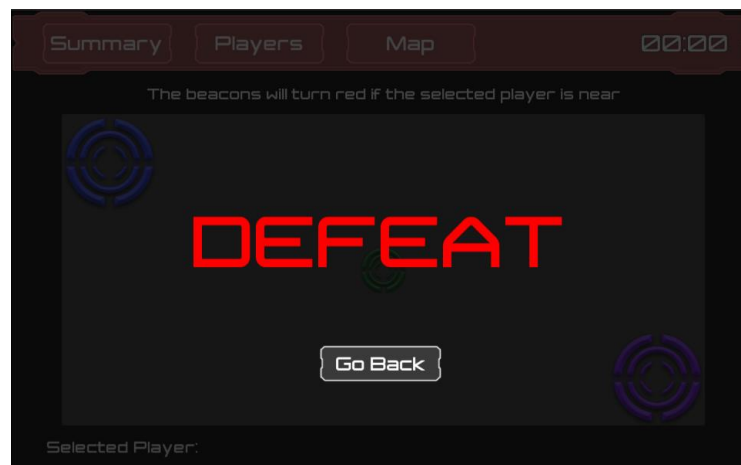


Figura 33 - Schermata che segnala la sconfitta

Sotto la stringa, vi è un bottone che permetterà di ritornare al pannello iniziale e quindi di poter avviare una nuova partita.

In termini di funzionamento, la pressione del tasto **Go Back** effettua un reset di tutte le variabili statiche, label e stringhe dell'interfaccia utilizzate nel corso della precedente partita al fine di azzerare l'istanza di gioco e riportarlo allo stato iniziale.

4.2.3 Arduino e sketch

Per realizzare l'equipaggiamento, in dotazione a ciascuno giocatore del Laser game, si sono utilizzati i seguenti componenti:

- Emittitore Infrarossi da 0.5 mm, lunghezza d'onda da 940 nm e angolo di emissione da 20°
- Sensore ricevitore IR TSOP38238
- Transistor NPN PN2222A
- Resistenza da 470 ohm e 120 ohm
- Push button

Analizziamo i circuiti realizzati partendo dall'emettitore infrarossi:

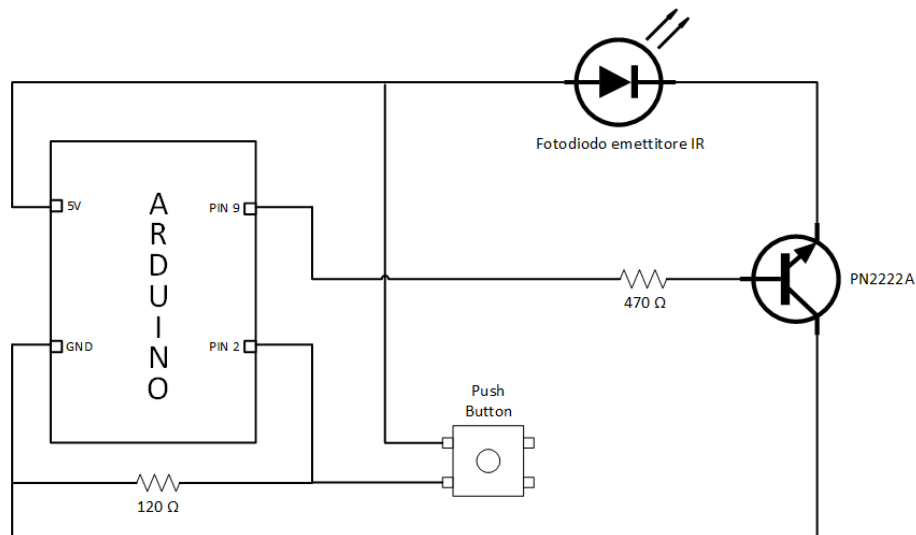


Figura 34 - Schematico pistola infrarossi

Il pulsante è composto da quattro connettori esattamente simmetrici rispetto al centro. Questi quattro piedini sono cortocircuitati a due a due e, per permettere il funzionamento, bisogna collegare i fili in due differenti connettori non cortocircuitati.

Il pulsante ha due principali funzioni: la prima è quella di simulare il gesto dello sparo, la seconda è quello di pilotare al livello logico alto il pin 2 dell'Arduino.

La resistenza da 120 ohm funziona da resistenza di Pull Down. Quando il circuito è aperto, quindi, non c'è connessione tra i due pin del Push Button, il valore letto dal pin 2 dell'Arduino risulta, a livello logico, basso per via della resistenza di Pull Down. Invece, quando il pulsante viene premuto, si crea un cortocircuito tra i due piedini del bottone portando il valore letto dal pin 2 dell'Arduino al valore logico alto di circa 5 volt.

Il fotodiode infrarossi, il quale possiede una polarità ben precisa, è composto da due piedini dove uno è l'anodo (polo positivo) e l'altro è il catodo (polo negativo).

L'anodo viene collegato alla VCC di 5 volt, mentre il catodo viene collegato all'emettitore del transistor NPN PN2222A.

Poiché l'Arduino non riesce a pilotare il fotodiode direttamente dal pin, serve un circuito di amplificazione della corrente attivato attraverso un amplificatore a emettitore comune realizzato grazie al transistor NPN su menzionato.

Il transistor è composto da tre piedini che sono: l'emettitore, il collettore e la base. All'emettitore viene collegato il catodo del fotodiode, il collettore viene collegato a massa e la base viene collegata al pin 9 dell'Arduino per mezzo di una resistenza da 470 ohm

Per quanto riguarda il ricevitore infrarossi, il circuito realizzato risulta molto semplice:

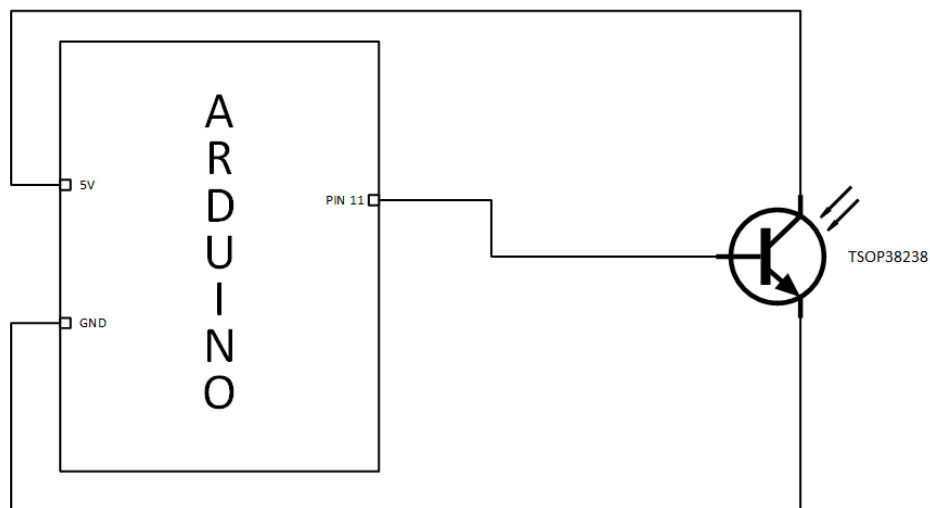


Figura 35 - Schematico pettorina infrarossi

Esso è composto da un fototransistor TSOP 38238 di tipo NPN composto da tre piedini. Il collettore viene collegato a VCC con un valore di 5 volt, l'emettitore viene commesso a massa mentre la base viene connessa al pin 11 dell'Arduino. Quest'ultima, quando il ricevitore riceve una forma d'onda infrarossi, trasmette al pin 11 il valore ricevuto in modo che, in base alle specifiche fatte nel codice caricato all'interno dell'Arduino, il dispositivo reagisca alle sollecitazioni esterne.

Tutto quanto detto fino ad ora in Arduino viene pilotato attraverso uno sketch, cioè un frammento di codice caricato all'interno della memoria della scheda che ha il compito di inizializzare le variabili e indicare il comportamento che la scheda deve assumere durante il funzionamento.

Nello sketch vi sono due funzioni fondamentali, la funzione `setup()` che viene eseguita una sola volta all'avvio della scheda e ha principalmente il compito di inizializzare le variabili, e la funzione di `loop()` che è quella che viene eseguita ciclicamente sul dispositivo e indica il comportamento da assumere.

Di seguito andiamo ad analizzare l'implementazione del dispositivo di puntamento e di ricezione.

4.2.3.1 Pistola Infrarossi

Nel funzionamento di questo dispositivo bisogna distinguere due macro aree, il pilotaggio del sensore infrarossi e la comunicazione tramite socket con il dispositivo mobile per rendere visibile la dotazione di munizione del giocatore.

Per quanto riguarda il sensore infrarossi, è stata utilizzata la libreria `IRLib.h` che è in grado di gestire la comunicazione infrarossi tra dispositivi. Questa libreria viene utilizzata per

diverse applicazioni, la più comune è la creazione custom di un telecomando per diversi dispositivi presenti in ogni casa e delle diverse marche, come televisori, impianti Hi-Fi, condizionatori, ecc.

Questa libreria è stata adattata alle esigenze del progetto sfruttando le funzioni messe a disposizione per la comunicazione IR.

Tra i diversi protocolli di comunicazione IR disponibile all'interno della libreria è stato scelto lo stesso utilizzato dai dispositivi JVC in commercio, in quanto risulta quello più adatto alle nostre esigenze in particolare, per esempio il protocollo utilizzato da SONY comporta l'invio di un segnale ripetuto tre volte consecutive e questo provoca una ridondanza nella ricezione che si traduce come se l'avversario viene colpito tre volta.



```
ArmaBlue_Pulito
button = digitalRead(buttonPin);
if (button==1 && count==0 && magazine > 0) {
    count++;
    magazine--;
    client.print("ammo,");
    client.println(String(magazine));
    My_Sender.send(JVC, 3333, 0);
}else{
    if(button==0){
        count=0;
    }
}
}
```

Figura 36 - Codice funzionamento emettitore infrarossi

Analizzando la linea di codice `My_Sender.send(JVC,8888,0)` abbiamo notato che `My_Sender` è un oggetto della classe `IRsend`. Attraverso esso viene chiamata la funzione `send()` nella quale vanno specificati tre parametri. Il primo il tipo di codifica utilizzata,

nel nostro caso la codifica JVC, il secondo parametro un valore da inviare che deve essere un unsigned long, il terzo ed ultimo parametro è un booleano che può assumere il valore di 0 = false o 1 = true. Questo booleano sta ad indicare, in base alla codifica JVC, se questo è la prima volta se si invia un segnale o una delle vote successive. Questo viene fatto perché in base questo protocollo, creato appunto dalla JVC, ogni qualvolta un valore viene inviato per la prima volta esso viene replicato due volte. Nel nostro caso studio lasciando il valore settato sempre a false esso viene sempre inviato una volta sola e questo permette di simulare lo sparo di un'arma.

Questa funzione è inserita all'interno di un costrutto if() in quanto viene chiamata ogni qualvolta viene premuto il bottone che simula il grilletto dell'arma.

Ad ogni pressione del bottone viene inviato il segnale IR e scalato il caricatore di uno in modo da consumare i colpi a disposizione del giocatore.

Oltre a quanto detto fino ad ora, entra in gioco la seconda macro area. Viene inviato al socket una stringa in modo da comunicare al dispositivo mobile collegato che un colpo è stato sparato.

Perciò per quanto riguarda la comunicazione con lo smartphone, e quindi la comunicazione dell'Arduino con l'esterno, vengono utilizzate le librerie messe a disposizione dal Bridge dello Yun. Queste librerie consentono la comunicazione tra i due processori, quello dell'Arduino e quello che gestisce il Wi-Fi, e quindi tramite le funzioni disponibili è possibile inviare al socket e leggere da socket delle stringe permettono di implementare la comunicazione.

```
ArmaBlue_Pulito
/*
 * Codice Arma Blu
 */
#include <IRLib.h>
#include <Bridge.h>
#include <YunServer.h>
#include <YunClient.h>

IRsend My_Sender;
YunServer server;
YunClient client;

void loop() {
    // try to accept a new connection
    if(!client){
        client = server.accept();
    }

    if(startGame){
        currentTime=millis();
    } else {

        if(client) {
            // read the command and trim spaces
            String command = client.readString();
        }
    }
}
```

Figura 37 - Libreria bridge yun

Viene gestito pure il caso in cui un giocatore, durante la partita, muore. In questa situazione l'applicazione sullo smartphone invia alla pistola una stringa con scritto "dead", ciò permette di bloccare il funzionamento della pistola in modo ad evitare di sparare e quindi colpire un avversario.

Il controllo del socket avviene con un sistema di temporizzazione ad intervalli di tempo di 5 secondi.

```
ArmaBlue_Pulito
.

if(currentTime-previousTime > interval){

    previousTime=currentTime;
    upgrade ();

}

< >
```

Figura 38 - Codice per temporizzazione

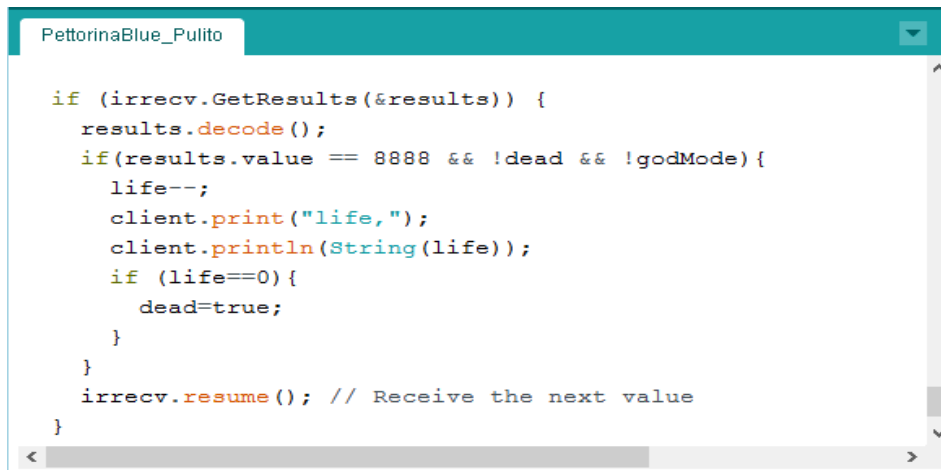
Questo perché l'Arduino, non essendo dotato di processore multi core, non è in grado di gestire thread che lavorano in parallelo. Controllando il socket ad ogni ciclo di funzionamento non permette di avere una giocabilità accettabile in quanto, durante il lasso di tempo in cui viene controllato se dei dati sono arrivati dall'esterno, la pistola non è in grado di sparare ed inviare il segnale infrarossi. Ciò ha obbligato la definizione di un intervallo temporale accettabile alla quale, nel peggiore dei casi, un giocatore riceve un potenziamento dal Team Leader con un ritardo di 5 secondi. Ritardo che, in un gioco con queste dinamiche, non risulta limitante per il suo corretto funzionamento.

Nel controllo del socket si è dovuta implementare una sorta di sincronizzazione in quanto, il problema del multi thread si ha pure in ricezione e, se emettitore e ricevitore non risultano sincronizzati, può capitare che quando la pistola spara il ricevitore è in ascolto sul socket quindi non riesce a rilevare il colpo ricevuto. Questo implicherebbe che nei 5 secondi ci siano due intervalli di tempo in cui i dispositivi non sono reattivi. Con la sincronizzazione i due intervalli di inattività coincidono, evitando di creare problemi durante lo svolgimento della partita.

4.2.3.2 Pettorina Infrarossi

Il sistema di ricezione infrarossi gestisce, anch'esso, sia la comunicazione IR sia la comunicazione Wi-Fi con l'esterno per comunicare e mantenere sempre aggiornati i punti vita.

Per quanto riguarda la comunicazione IR, il ricevitore risulta sempre reattivo e in attesa di ricevere qualche segnale.



```
if (irrecv.GetResults(&results)) {
    results.decode();
    if(results.value == 8888 && !dead && !godMode){
        life--;
        client.print("life,");
        client.println(String(life));
        if (life==0){
            dead=true;
        }
    }
    irrecv.resume(); // Receive the next value
}
```

Figura 39 - Codice funzionamento ricevitore infrarossi

Tramite la funzione `irrecv.GetResults(&results)` viene inserito all'interno della variabile `results` il valore ricevuto dal sensore IR. Per accedere a questo valore, viene utilizzata la funzione `results.value()` che come valore di ritorno fornisce la stringa ricevuta dal sensore.

Viene controllato se il valore ricevuto risulta quello effettivamente aspettato, in quanto per scelte progettuali, si è voluto implementare una modalità di gioco nella quale il fuoco amico non provoca danno ai punti vita. Quindi ogni giocatore, appartenete ad una squadra avversaria, possiede una pistola che invia un determinato codice identificato della squadra. Se il codice ricevuto dal sensore non risulta essere quello della squadra avversaria i punti vita non vengono intaccati.

Ogni volta che si viene colpiti, viene comunicato allo smartphone (attraverso il comando `client.print()`), tramite socket, una stringa che indica i punti vita a disposizione del giocatore.

Come per il caso della pistola, anche la pettorina è in continua comunicazione con l'esterno in quanto deve essere in grado di ricevere eventuali potenziamenti inviati dal comandante di squadra.

Il controllo di dati provenienti dall'esterno avviene ad intervalli di tempo regolari di 5 secondi per le motivazioni sopra illustrate e con lo stesso metodo di funzionamento.

Oltre ad eventuali kit vita, è possibile ricevere dal Team Leader il potenziamento chiamato "God Mode". Questo pacchetto permette, al giocatore che lo riceve, di attivare una barriera d'invulnerabilità consentendo di non essere colpito dal fuoco nemico. Ciò si attua sempre tramite la ricezione di una particolare stringa, "godmode", che permette di settare a true un booleano che blocca la sottrazione dei punti vita.

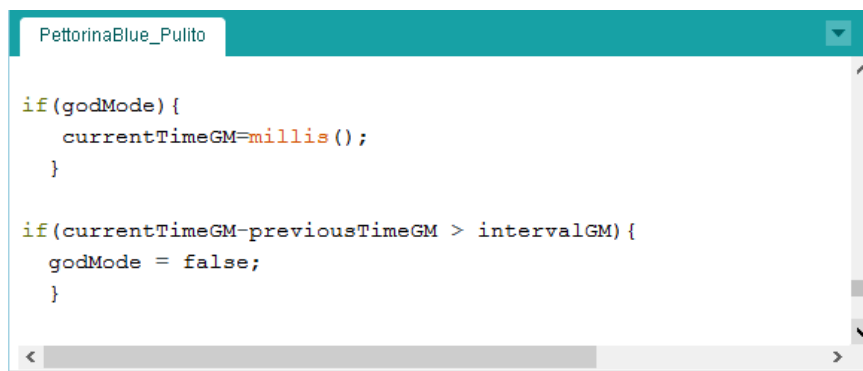
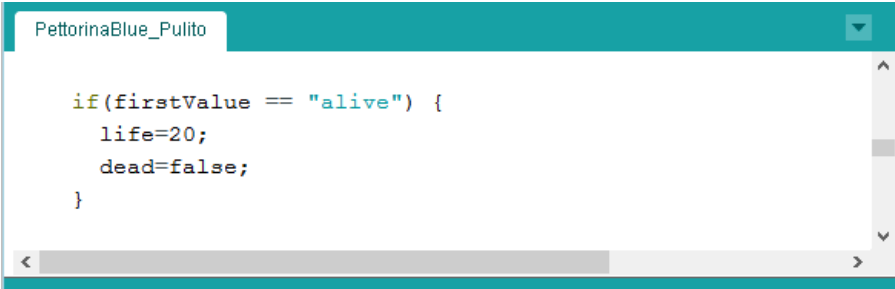
The image shows a code editor window with a teal header bar containing the text "PettorinaBlue_Pulito". The code is written in a light-colored font on a white background. It consists of two main conditional blocks. The first block is an if-statement: "if(godMode) {" followed by an indented line "currentTimeGM=millis();" and a closing brace "}". The second block is another if-statement: "if(currentTimeGM-previousTimeGM > intervalGM) {" followed by an indented line "godMode = false;" and a closing brace "}". The code is enclosed in a scrollable container with a vertical scrollbar on the right and a horizontal scrollbar at the bottom.

Figura 40 - Codice God Mode

Quando, durante una partita, un giocatore muore, deve recarsi alla base madre per poter rientrare in gioco. Ciò viene ottenuto tramite la stringa "alive", inviata dallo smartphone, che segnala al dispositivo di essere rientrato in gioco e ripristina tutti i punti vita.

A screenshot of a code editor window titled "PettorinaBlue_Pulito". The editor contains a JavaScript code snippet:

```
if(firstValue == "alive") {  
  life=20;  
  dead=false;  
}
```

 The code is displayed in a light blue font on a white background. The editor has a teal header bar and a scroll bar on the right side.

Figura 41 - Codice ritorno in vita di un giocatore

La sincronizzazione dei dispositivi, che permette di controllare il socket tutti allo stesso intervallo di tempo contemporaneamente, viene effettuata tramite una stringa ricevuta tramite socket, la stringa “startgame”.

Questa stringa viene invitata a tutti i dispositivi contemporaneamente dal Team Leader, non appena viene creata la stanza di gioco. Connessi tutti i giocatori delle squadre e si è pronti ad iniziare la partita.

4.2.4 Stampa 3D

Per rendere chiara la possibilità di realizzazione del progetto e per aiutare la fase di testing e dimostrazione di quanto realizzato, si è scelto di creare i dispositivi portatili con una stampante 3D.

Per poter effettuare la stampa è importante dare forma all'oggetto 3D con un programma di modellazione. Il programma da noi utilizzato è Blender.

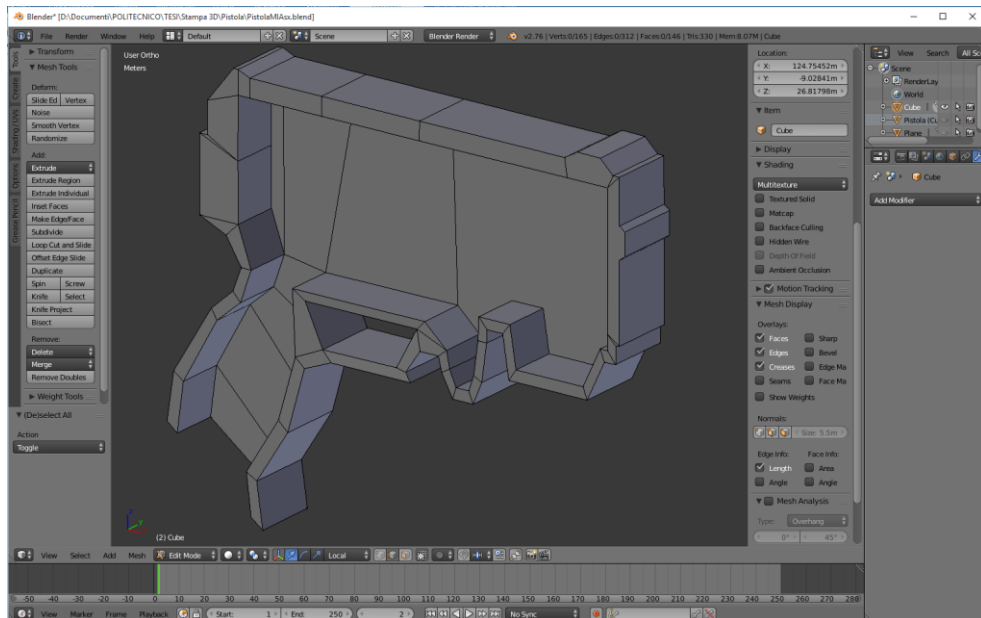


Figura 42 - Modello su Blender

L'oggetto modellato viene esportato in formato .stl (StereoLithography) e questo file viene caricato all'interno di un'applicazione in grado di generare il file .gcode che la stampante 3D è in grado di leggere per effettuare la stampa.

La stampante usata da noi è la MakerBot Replicator 2X e di conseguenza il software utilizzato per la generazione del file .gcode è quello messo a disposizione dalla casa produttrice della macchina.

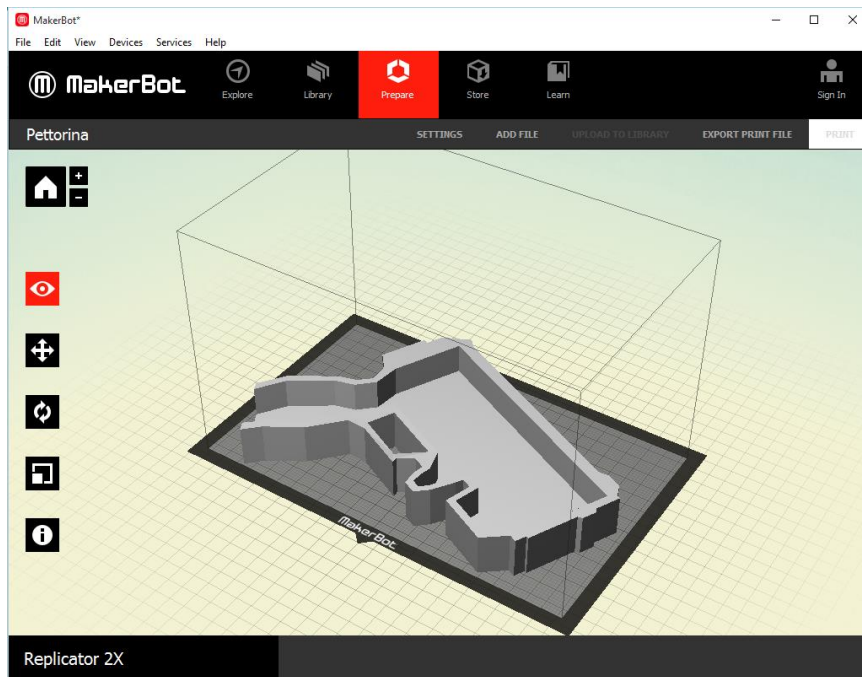


Figura 43 - Modello su MakerBot Desktop

La stampante utilizzata da noi è dotata di filamento ABS, e ciò ha permesso di ottenere oggetti molto resistenti anche se leggeri.

Una volta realizzati i componenti dell'equipaggiamento in dotazione ad ogni giocatore, sono stati realizzati tutti i circuiti necessari a dare vita a questi dispositivi inserendoli all'interno di essi.

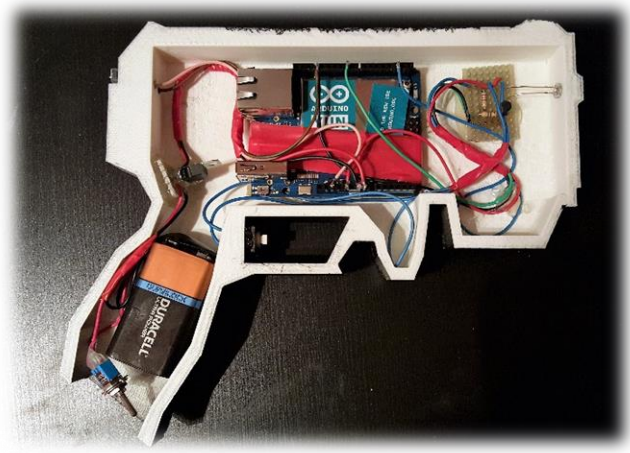


Figura 44 - Pistola 3D con circuiteria interna



Figura 45 - Pettorina 3D con circuiteria interna

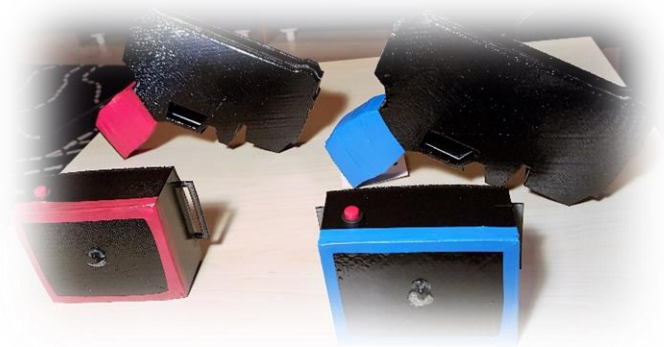


Figura 46 - Kit infrarosso

Capitolo 5

Valutazioni

La struttura architettuale del prototipo ha permesso di superare, in maniera completa, alcune limitazioni presenti nei Laser Game di oggi. Infatti, sia nei primi mesi di sviluppo che ultimamente, grazie al WTT di Torino, si è avuta la possibilità di parlare con i gestori di una delle arene di Laser Game della città. Essenzialmente, il Laser Game, attualmente diffuso, si basa su un concetto di ricarica forzata per aggiornare i punteggi e stabilire i vincitori. L'addetto incaricato, tramite un software dedicato, effettuerà una semplice associazione del nome del giocatore all'ID dell'equipaggiamento (pistola e pettorina). Iniziata la partita, ogni giocatore diventa un micro-cosmo a sé stante e, il suo equipaggiamento, immagazzina al suo interno i dati relativi a munizionamento, vita e da quali pistole IR si è stato colpito. Ogni qual volta effettua le operazioni di ricarica alle colonnine, l'equipaggiamento "scarica" questi dati che vengono successivamente trasmessi al software, in modo tale da tracciare per ogni giocatore i punteggi. Ecco spiegato il perché, prima di lasciare l'arena, bisogna obbligatoriamente passare per una delle colonnine. Si può quindi affermare che il progetto di tesi, oltre a includere forti elementi di Ambient Intelligence, presenta grosse innovazioni anche nelle più semplici meccaniche che riguardano il Laser Game. Le partite, infatti, non saranno limitate da

forzate meccaniche dovute all'hardware grazie alla componente di comunicazione tra tutti i device messi in gioco.

Adesso si cercherà di effettuare una valutazione oggettiva del prototipo creato, andando a coprire tutte le fasi illustrate nei precedenti capitoli che hanno ricoperto i sei mesi di sviluppo.

5.1 Costo

Il funzionamento base di ogni partita, richiede di avere un equipaggiamento necessario per poter svolgere le funzionalità di base. Ogni soldato, appartenente ad una determinata squadra, dovrà dunque essere in possesso di 3 elementi fondamentali, due dei quali andranno forniti:

- **Pistola/Arma da fuoco:** che dovrà avere al suo interno dispositivi e intelligenza per poter inviare segnale infrarosso e comunicare con l'esterno.
- **Pettorina sensibile:** che dovrà avere al suo interno dispositivi e intelligenza per potere ricevere segnale infrarosso e comunicare con l'esterno.

Sarà necessario aggiungere, agli elementi elencati, uno smartphone Android (almeno al momento) in cui installare l'applicazione del gioco. Questo set di elementi compone, quindi, un kit di equipaggiamento obbligatorio per il corretto svolgimento della partita. Per la realizzazione del prototipo di funzionamento, sono stati utilizzati e scelti

determinati componenti e accessori già presenti in commercio al solo scopo di raggiungere l'obiettivo ed effettuare una dimostrazione. Questi componenti presentano un costo d'acquisto non indifferente che ha influito sulla quantità di kit disponibili all'interno del prototipo.

Volendo tralasciare la difficoltà di realizzazione e la messa impiego, il costo per realizzare un kit è all'incirca di **150 euro** divisi in:

- **2 Schede Arduino YUN** : 68 euro ciascuna.
- **Cavistica**, Moduli Infrarossi, Resistenze, Modulatori di Tensione: 10 euro
- **Batteria 9v**: 4 euro ciascuna.

A questo costo, che riguarda esclusivamente la componente hardware, va ad aggiungersi il prezzo di realizzazione degli involucri, tramite stampa 3D. Allo stato attuale, il costo di stampa 3D si aggira attorno ai 5 euro per ora di lavorazione. Entrambi i modelli realizzati nel progetto, nonostante la semplicità, necessitavano di essere stampati alla massima qualità disponibile per evitare errori e imbarcamenti nella struttura. Il tempo di lavorazione necessario, equivale quindi a **25 ore** di cui circa 21 per la realizzazione della pistola e 4 ore circa per la realizzazione dell'involucro di sostegno della pettorina.

Il prezzo finale di realizzazione di un kit si aggira, quindi, sui **250 euro** a patto che si ottenga una stampa 3D utilizzabile al primo tentativo. Il costo di realizzazione, non essendo irrisorio, ha inciso sulla quantità finale di kit realizzabili per il prototipo. Ad ogni modo, è possibile affermare che questa spesa riguarda solo la realizzazione del prototipo che è stata effettuata utilizzando le tecnologie e le componenti elencate e non rappresenta il costo effettivo, nel caso di una realizzazione a livello industriale. Di conseguenza, realizzando hardware ad hoc sarà sicuramente possibile ridurre notevolmente i costi e realizzare un equipaggiamento che si adatta meglio alle esigenze progettuali.

Nonostante questo, grazie al Politecnico di Torino, ai laboratori e alle risorse disponibili, i costi si sono azzerati quasi totalmente riuscendo a realizzare con successo 2 kit funzionanti.

Avendo soltanto due kit disponibili, non è stato possibile effettuare un test completo delle potenzialità del gioco creato. La migliore condizione di test si sarebbe ottenuta con diversi kit disponibili in modo da far provare il gioco ad un diverso numeri di utenti. Nonostante questo, sono stati effettuati numerosi test per avere una valutazione oggettiva del prototipo e poter effettuare alcune considerazioni.

5.2 Software

Partendo dalle prime fasi del progetto, si può affermare come la parte di Game Concept sia stata una delle più importanti durante lo sviluppo. Il disporre di un documento in cui sono indicate tutte le caratteristiche finali del prototipo, i dettagli e le features di ogni utente giocante, ha permesso di poter pensare e programmare le applicazioni in modo adeguato. È stato proprio a partire dal Game Concept che si è sviluppata la struttura delle classi del progetto, prima su carta e poi su codice, andando a definirne, per ognuna, comportamento e compiti. Si sono create, quindi, delle applicazioni stabili, capaci di gestire più partite in una stessa sessione e con una serie di eventi e Callbacks per gestire eventuali errori o disconnessioni.

Tutto questo è stato possibile grazie all'utile complessità di Unity e anche alla gestione degli eventi di PhotonNetwork, sfruttati a pieno all'interno del prototipo realizzato. Come

è stato già affermato, nell'analisi degli strumenti, Unity si è rilevato ostico non appena ci si doveva allontanare da quella che era l'API nativa. È stato notato come, la mancanza del codice sorgente disponibile abbia influito sull'implementazioni di alcune features che richiedevano componenti esterne. L'esempio più lampante è quello dell'implementazione del Bluetooth all'interno dell'applicazione Mobile.

Sono stati effettuati numerosi tentativi prima di arrivare alla soluzione finale. Numerose le funzioni native e gli asset provati, alcuni dei quali totalmente non funzionanti. Sempre con il Bluetooth si è capito quanto potesse essere limitante l'impossibilità di gestire più thread. Sfruttando qualche exploit, è possibile invocare i Thread tramite l'utilizzo del framework .NET di C# ma risultano totalmente inutilizzabili all'interno di Unity perché si possono effettuare solo operazioni che non includono funzioni dell'API interna o che riguardano l'UI, riducendone tantissimo l'utilità. Sfruttare i thread avrebbe permesso di effettuare una ricerca Bluetooth più efficiente senza andare ad appesantire il thread principale.

Unity è essenzialmente un programma chiuso, che non mette a disposizione il codice sorgente e che si trasforma in un grande ostacolo quando si vuole creare qualcosa di complesso e articolato. Questo è probabilmente dovuto anche all'immaturità dei tempi in merito alla diffusione di alcune tecnologie e abitudini, quali per esempio Internet of Things e Ambient Intelligence; in ogni caso si è convinti che con il passare del tempo Unity riuscirà a coprire qualsiasi necessità vista la serietà e l'impegno del team di sviluppo.

Un'altra mancanza di Unity riguarda la parte mobile. Debuggare il codice è uno strumento fondamentale per ogni programmatore allo scopo di valutare l'efficienza o trovare dei banchi. Tuttavia, all'interno della parte mobile, manca la possibilità di poter visualizzare una console e di conseguenza capire cosa è andato storto durante il funzionamento.

Questo problema è ovviabile, come al solito, con un asset presente all'interno dell'Asset Store, ma risulta incompleto e dopo qualche ora di utilizzo tende a bloccarsi obbligando l'utente a riavviare la sua applicazione. Per tutto il resto, Unity ha permesso di sviluppare

tutto su Android in maniera molto rapida, programmando in un linguaggio non nativo quale C#. Anche l'accesso all'accelerometro è risultato molto semplificato ed ha permesso di ricreare una miniatura 3D del giocatore e dei suoi movimenti. Parlando, invece, di Photon e di comunicazione in generale, il prototipo ha rispettato tutti i requisiti. Ogni giocatore è in grado di comunicare rapidamente col Team Leader e viceversa. Photon ha quindi rispettato le aspettative in maniera ottimale garantendo un buon servizio esente da problemi o downtime.

Resta comunque chiaro che, nel caso di una futura implementazione, sarà necessario passare alla componente Server. Questo perché il Cloud manca di alcune possibilità e caratteristiche necessarie per far fronte ad un utilizzo corposo del gioco, per esempio la mancanza del concetto di autorità, l'impossibilità di scatenare eventi e messaggi senza l'utilizzo di GameObjects o PhotonViews e il dover dichiarare le funzioni RPC all'interno degli stessi oggetti dal quale vengono richiamate.

5.3 Hardware

Passando alla parte hardware, non si può far altro che confermare che la piattaforma Arduino è un ottimo prodotto per la prototipazione di qualsiasi esigenza, non solo in termini di funzionamento. La grande diffusione della piattaforma ha creato una grande community attorno al prodotto, garantendo una grande reperibilità delle informazioni necessarie allo sviluppo del circuito e anche del software ad esso legato. Tuttavia,

l'impossibilità di gestire più thread ha causato un grande problema nell'architettura complessiva.

Ogni scheda Arduino necessita di effettuare due compiti contemporaneamente, inviare o ricevere informazioni dallo smartphone e inviare o ricevere segnali infrarossi. La condizione ottimale sarebbe stata quella di poter avviare un thread per gestire la connessione socket e con il thread principale gestire l'infrarosso. Arduino, però, essendo un dispositivo monoprocesso, non permette l'esecuzione di più thread in contemporanea. Questo fa sì che, proprio per il funzionamento di Arduino, l'esecuzione del codice caricato nella memoria della scheda, avviene in un loop in maniera ciclica e continua, permettendo di effettuare un'operazione alla volta. Di conseguenza si è dovuto pensare a un sistema gestito da timer per temporizzare le varie operazioni. In base ad opportune valutazioni, dettate dalle dinamiche del gioco, si è scelto di temporizzare l'accesso al socket a intervalli di 5 secondi, per consentire al dispositivo di rimanere sempre reattivo all'invio e alla ricezione dei raggi infrarossi.

A causa di questa scelta, durante la partita si possono verificare dei momenti in cui la pressione del bottone non viene rilevata dal software. Trattandosi, comunque di una finestra temporale molto piccola, la probabilità che avvenga è molto bassa. Infine, si è effettuato un test anche sulla durata della batteria che alimenta la scheda Arduino.

La batteria è una 9v, facilmente reperibile in commercio, estraibile e scambiabile in caso di esaurimento. Sono stati fatti dei test sui consumi della scheda corredata dei vari dispositivi durante il suo normale funzionamento. È stato rilevato che il consumo sia di circa 180 mA e che quindi con una batteria standard di 565 mAh è possibile alimentare il dispositivo per circa 3,13 ore.

5.4 Beacons e Ambient Intelligence

Per quanto riguarda la presenza ambientale, riducendo quelle che erano le aspettative iniziali, si sono sfruttati gli Estimote Beacons al massimo delle potenzialità per includerli all'interno delle partite come dati di input fondamentali. Il giocatore, in caso di morte, deve necessariamente recarsi vicino ad un Beacon, denominato Beacon Madre, per poter resuscitare e ritornare in battaglia, mentre il Team Leader grazie al pannello Mappa riesce a visualizzare approssimativamente la posizione dei giocatori.

Tuttavia è ancora una tecnologia poco sviluppata, probabilmente influenzata dal costo irrisorio d'acquisto. Anche se si diminuisce al minimo la frequenza di advertising e si aumenta la potenza del segnale, il dato risulta spesso corrotto dando informazioni sbagliate che vanno filtrati tramite soglia o media. Per questo, nel caso di una implementazione reale del prodotto, sarà necessario utilizzare altri Beacon di qualità superiore che sfruttano più sensori, compresi quelli dell'utente, filtri complessi e segnale Wi-Fi per avere una posizione più precisa.

In generale si può affermare che il prototipo è ben riuscito ed ha permesso di dimostrare come diverse tecnologie e dispositivi possono cooperare assieme in un'unica esperienza di gioco. Grazie al prototipo creato, il Laser Game acquisisce più profondità, sia sotto forma di strategia che sotto forma di azione rendendo l'ambiente un elemento integrante di gioco.

5.5 Sviluppi futuri

Il prototipo è perfettamente funzionante e permette a 4 giocatori di combattere ricoprendo il ruolo desiderato. L'architettura si è dimostrata funzionante e capace di sostenere il carico di lavoro richiesto e risulta facile inserire nuove feature e possibilità in-game. Il gioco potrebbe variare ulteriormente grazie all'inserimento di nuove caratteristiche tra cui:

- **DIVERSE MODALITA' DI GIOCO:** L'attuale modalità implementata è il Team Deathmatch in cui, al termine del timer, verrà dichiarata vincitrice la squadra col maggior numero di uccisioni. Tuttavia, per sfruttare ulteriormente il concetto di Ambient Intelligence, si potrebbero utilizzare i Beacon per creare nuove modalità di gioco. Tra cui:
 - **Dominio:** vengono ereditate tutte le funzionalità della modalità Team Deathmatch, con l'aggiunta di un Beacon, posizionato in un determinato punto della mappa, che avrà lo scopo di fungere da area di controllo. Lo scopo di ogni squadra è quello di ottenere il controllo del Beacon andandosi a posizionare nelle vicinanze di quest'ultimo. Per assegnare il controllo sarà necessario che un membro della squadra si avvicini al Beacon (che verrà posizionato in un posto aperto e senza coperture) e utilizzare il proprio telefono per risolvere un breve puzzle matematico. Se il giocatore non viene ucciso durante questa operazione e risolve il puzzle matematico, verrà assegnato un punto alla sua squadra di appartenenza.
 - **Trova l'artefatto:** questa modalità richiederà di un numero superiore di Beacon, tanti quanta è la difficoltà della partita, e un'arena di gioco

più grande. Anche qui la modalità eredita le regole del Team DeathMatch, solo che lo scopo di ogni squadra sarà quello di trovare un artefatto. L'arena di gioco sarà, infatti, popolata da diversi beacon sparsi per la mappa, ma soltanto uno di questi è l'artefatto da trovare. I giocatori dovranno, quindi, avvicinarsi ad ogni beacon e risolvere un piccolo puzzle che apparirà nel loro dispositivo. Solo dopo aver risolto il puzzle scopriranno la vera entità del Beacon e di conseguenza se si trattava dell'artefatto.

- **NUOVI POTENZIAMENTI:** nel prototipo è stato implementato esclusivamente il potenziamento che rende invulnerabile un determinato giocatore per 5 secondi. Tramite la gestione della comunicazione tra Arduino e Android, sotto forma di stringhe, è possibile implementare diversi potenziamenti. Tutto questo è facilmente implementabile con poche linee di codice aggiuntive.
- **CONCETTO DI ARMA e ARMATURA:** allo stato attuale, ogni utente effettua lo stesso danno. Si potrebbe astrarre il concetto di armi diverse andando a influire sulla variabile del danno inflitto. Sempre sfruttando l'ambiente, si potrebbero creare dei Beacon che hanno lo scopo di fornire un'arma diversa al giocatore. L'interfaccia dell'utente potrebbe cambiare a seconda dell'arma diversa, assumendo una nuova forma o colore. Lo stesso vale per le armature.

Gli sviluppi possibili sono veramente numerosi e basta un po' di immaginazione per creare nuove features. Durante l'illustrazione di nuove, possibili features il concetto di Puzzle risolvibile era molto frequente.

Il Puzzle ha un duplice scopo:

- **Spezzare la monotonia:** si possono pensare diversi puzzle o indovinelli, di vario genere e natura. Il giocatore che si avvicinerà sarà, dunque, impossibilitato ad effettuare altre azioni, visto che sarà concentrato sullo schermo del suo dispositivo cellulare. La squadra dovrà dunque proteggerlo e il Team Leader inviare, se disponibili, i potenziamenti.
- **Ovviare all'inefficienza dei Beacons:** anche aumentando la frequenza del dato di advertising, il segnale RSSI non viene rilevato immediatamente o, a volte, risulta esageratamente alterato. Il puzzle servirà dunque a tamponare questa inefficienza cercando di sfruttare le poche rilevazioni corrette.

Grazie al concept creato, l'ambiente diventa un vero dato di Input e parte integrante di qualsiasi dinamica di gioco.

Ad ogni modo, nel caso di un effettivo sviluppo e rilascio del gioco, il primo step è sicuramente quello di realizzare tutta la componente Online del Game Concept sotto forma di browser game accessibile da qualsiasi piattaforma. Ogni giocatore dovrà registrarsi al browser game per creare un account per gestire la sua identità di giocatore virtuale. Potrà quindi iniziare a sviluppare il suo avamposto con le tecnologie scelta in base alla fazione di appartenenza ed organizzare partite amichevoli o meno.

Per l'organizzazione delle partite nei Laser Game potrà sfruttare il sistema della fazione **Bellicatio**, fazione di mercenari pronti ad essere ingaggiati per affrontare le sfide sul campo. Da qui si potrà inserire l'evento, scegliere la data e l'ora, invitare gli amici, scegliere i giocatori di sua preferenza e dichiarare la modalità di gioco.

L'utente potrà inviare feedback, scrivere recensioni di altri giocatori e partecipare attivamente in forum e discussioni. Inoltre potrà anch'egli registrarsi, creare il suo soldato Virtuale e partecipare in modalità "Mercenario" a qualsiasi evento elencato.

Per sviluppare questa parte si potrà sempre utilizzare Unity, gestendo anche persistenza dei dati e progresso dei giocatori. Quest'ultima parte potrebbe essere realizzata con **PlayFab**, un servizio BaaS (Backend as a Service), che da anni lavora con Unity per fornire un servizio rapido per rendere persistenti i dati.

Successivamente si potrebbe creare un hardware ad hoc per ovviare ai problemi riscontrati con Arduino e migliorare i consumi, andando a creare un hardware esclusivamente dedicato alle esigenze richieste. Andrà anche scelto un altro sistema per creare l'equipaggiamento.

Nonostante la stampa 3D si sia rilevata una tecnologia sfruttabile, risulta ancora troppo costosa per lavori che richiedono un quantitativo importante di dispositivi. Si potranno, quindi, sfruttare delle plastiche per creare un equipaggiamento più ergonomico e veloce da produrre industrialmente.

Il tutto sarà mirato a creare una community attiva, città per città, che permette a chiunque di poter giocare al Laser Game in qualsiasi momento e con chi desidera, all'interno di arene convenzionate oppure in qualsiasi punto della città tramite l'utilizzo di kit acquistabili.

Conclusioni

La possibilità di realizzare un lavoro di tesi, al quale si è unito la tecnologia al gaming, ha permesso di concretizzare un progetto innovativo e stimolante. Gli obiettivi prefissati all'atto della definizione di quest'ultimo sono stati tutti raggiunti con grande soddisfazione, fermo restando le limitazioni finanziarie e dei dispositivi a nostra disposizione.

Durante la fase di progettazione ci si è scontrati con diverse realtà che hanno portato ad effettuare delle scelte progettuali capaci di soddisfare tutti i requisiti imposti. Queste scelte sono state precedute da un lungo periodo di studio e analisi, in quanto si è cercato di ottenere i massimi risultati a costo zero. Tutto ciò ha portato all'implementazione di un'architettura capace di rendere realizzabile l'idea innovativa senza rinunciare agli elementi che sono la base di questa tesi, stigmatizzate nello sfruttamento dell'Ambient Intelligence, l'implementazione di dispositivi che sfruttano dell'Internet Of Things e la realizzazione di prototipi funzionanti.

L'implementazione di tutte le meccaniche di gioco ha portato alla realizzazione dei test che simulano una vera e propria partita. Le risorse limitate non hanno permesso di testare l'intera architettura con una partita completa composta da 8 giocatori e un Team Leader per squadra, che gestisce i propri giocatori, sia esso nelle vicinanze dell'arena di gioco o direttamente dal PC di casa sua.

Questo lavoro di tesi rientra nell'ottica di un progetto più grande spiegato in parte nel Game Concept, con l'intenzione di integrare due modi, quello on line e quello reale. Lo

sviluppo di questo progetto permetterà di definire un nuovo concetto di Laser Game e di creare una vera e propria comunità web in grado di alimentare, pesantemente, le arene convenzionate con questo prodotto.

Utilizzando gli opportuni dispositivi, si è in grado di incrementare notevolmente le prestazioni e le funzionalità di ciò che si è realizzato. Vedasi, per esempio, il potenziare la localizzazione indoor (per fornire un migliore feedback ai Team Leader) garantirebbe di sfruttare meglio il concetto di Ambient Intelligence. Oppure, l'utilizzo di hardware creato ad hoc per ovviare ad alcune limitazioni, come il multithread, e avere un hardware più compatto e funzionale.

Nonostante i test siano stati limitati a 4 giocatori per partita, grazie a questo progetto, si è toccato con mano come l'integrazione dell'Ambient Intelligence e di dispositivi in grado di dare supporto all'immersività aumentano di gran lunga la giocabilità e l'integrazione di un semplice gioco già noto con tecnologie evolute.

Bibliografia

- [1] Sito ufficiale Arduino, www.arduino.cc
- [2] Forum ufficiale Arduino, www.forum.arduino.cc
- [3] Articolo trasmissione IR su Arduino,
<http://www.mauroalfieri.it/elettronica/tutorial-arduino-come-trasmittitore-ir-o-come-telecomando.html>
- [4] Sito libreria per Arduino IRLib.h, <https://learn.adafruit.com/using-an-infrared-library/sending-ir-codes>
- [5] Sito ufficiale Blender, <https://www.blender.org>
- [6] Sito ufficiale MakerBot, <http://www.makerbot.com>
- [7] Sito ufficiale Unity, www.unity3d.com
- [8] Forum ufficiale Unity, <http://forum.unity3d.com/>
- [9] Unity answers, <http://answers.unity3d.com/>

- [10] Stackoverflow, <http://stackoverflow.com>

- [11] Sito ufficiale Photon Network,
<https://www.photonengine.com/en/Realtime>

- [12] Asset Store Unity, <https://www.assetstore.unity3d.com>

- [13] Sito ufficiale Bluetooth LE Asset per Unity, www.shatalmic.com