ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

IDEAZIONE DI SISTEMI DISTRIBUITI COLLABORATIVI BASATI SU REALTÀ AUMENTATA MOBILE: UN CASO DI STUDIO, CON APPROFONDIMENTO RELATIVO ALL'INTERFACCIA UTENTE

Relazione finale in
PROGRAMMAZIONE DI SISTEMI EMBEDDED

Relatore
Prof. ALESSANDRO RICCI

Presentata da
BRANDO MORDENTI

Co-relatori
Ing. ANGELO CROATTI
Ing. PIETRO BRUNETTI

Seconda Sessione di Laurea
Anno Accademico 2014 – 2015

PAROLE CHIAVE

augmented reality

distribuited systems

location awareness

cooperation

communication

I consider augmented reality to be a medium, as opposed to a technology. By medium, I mean that it mediates ideas between humans and computers, humans and humans, and computers and humans.

Alan B. Craig on Augmented Reality

Indice

Introduzione

Nell'ambito dei sistemi software distribuiti è, oggigiorno, più che mai evidente l'esigenza di supportare contesti applicativi in cui forte enfasi è data agli aspetti collaborativi tra gli utenti del sistema stesso. Le recenti tecnologie in ambito Wearable e le sempre più sofisticate tecniche in ambito di realtà aumentata hanno portato ad una profonda riflessione in merito ai meccanismi di collaborazione attuabili in tali sistemi e alla rivisitazione dei principi di progettazione del software stesso. Avvalendosi di dispositivi Wearable come smart-glasses, ad esempio, risulta possibile per un utente interagire con il sistema (per scambiare informazioni e cooperare) in modalità hands-free e contemporaneamente operare in contesti di totale immersione con la realtà fisica, in cui reale e virtuale si fondono nella cosiddetta Mixed Reality.

Obiettivo di questa tesi è esplorare l'ideazione di sistemi software collaborativi innovativi basati su smart-glasses e forme di realtà aumentata mobile. In particolare, è stato formulato un caso di studio che cattura alcuni aspetti essenziali di questi sistemi: un gioco nel quale più utenti dotati di smart glasses si muovono in una zona precisa cercando di raggiungere tutti i punti d'interesse preimpostati in fase di inizializzazione del gioco e ottendendo le ricompense contenute dentro agli scrigni situati nei suddetti punti. Questo caso di studio è stato formulato in cooperazione con i colleghi Filippo Berlini e Matteo Aldini, che nelle rispettive tesi hanno approfondito gli aspetti di collaborazione e di comunicazione. In questa tesi sono stati approfonditi gli aspetti relativi all'interfaccia utente - basata su smart-glasses - e al suo interfacciamento con il resto del sistema.

Lo studio di questa applicazione si propone di mettere in luce molti aspetti innovativi che ancora oggi non sono stati approfonditi né sviluppati da piattaforme a supporto della realtà aumentata. Nello specifico, l'interfaccia grafica che l'utente visualizzerà sullo schermo degli smart-glasses non sarà basata su tecniche di riconoscimento di markers, ma verrà implementata con messaggi

testuali e figure geometriche semplici, disegnate dinamicamente in base ai dati forniti dalle API di geolocalizzazione e alle interazioni dell'utente.

Complessivamente, il sistema progettato è *modulare*, ovvero è composto di unità separate dette moduli, che consentono di capire il sistema in funzione delle sue parti, è *dinamico* grazie alla presenza di oggetti aumentati con uno stato che può variare a seconda delle interazioni che l'utente avrà con esso. È *location-based*, poiché utilizza in maniera rilevante tecniche di geolocalizzazione GPS che lo rendono un sistema location-aware ed è *hands-free* in quanto l'interfaccia grafica presentata all'utente risiede sui glasses, opzione che gli consente maggiore libertà nei movimenti disimpegnandone le mani. Un'altra caratteristica importante dell'applicativo che andrò a presentare è il fatto di essere un sistema distribuito e condiviso, nel quale sono presenti oggetti aumentati non solo in qualità di immagini renderizzate, ma come entità computazionali vere e proprie, in grado di percepire le interazioni con l'utente e il mondo esterno e di mutare il proprio stato in base alle informazioni ricevute.

La struttura dell'elaborato sarà la seguente. Nella prima parte introduttiva verrà delineato il concetto di realtà aumentata e ne verrà approfondito lo stato attuale dell'arte, insieme a un breve excursus storico sui progressi della tecnologia attraverso gli anni e ad alcuni approfondimenti relativi alla cooperazione in sistemi AR, alla geolocalizzazione e ai componenti hardware coinvolti. In seguito verrà presentato lo specifico caso di studio, un gioco location-based di realtà aumentata strutturato in maniera tale da rendere necessaria la cooperazione tra vari utenti e l'interazione con oggetti aumentati con lo scopo di raggiungere un determinato obiettivo. Si passerà alla fase di analisi dei requisiti e di modellazione del dominio applicativo, utilizzando diagrammi dei casi d'uso e di sequenza definiti dallo standard UML. Verranno analizzati gli scenari e le situazioni di gioco che l'utente dovrà affrontare interagendo con gli oggetti aumentati. Seguirà la fase di progettazione, nella quale prima verrà illustrata l'architettura logica del sistema, poi verrà specificata la parte di dettaglio del mio lavoro, ovvero l'approfondimento relativo allo strato di Interfaccia Utente. Conseguentemente al capitolo sulla progettazione verrà mostrata la fase di sviluppo e implementazione del prototipo da noi realizzato e verranno svolte le dovute valutazioni sul lavoro svolto e sulle possibili migliorie applicabili in futuro.

Capitolo 1

Stato dell'arte

1.1 Introduzione e cenni storici

Per **realtà aumentata** (augmented reality, AR) si intende una vista del mondo reale nella quale la percezione sensoriale umana viene arricchita mediante informazioni computazionali che non sarebbero altrimenti percepibili con i cinque sensi. Un sistema basato su realtà aumentata combina oggetti fisici e oggetti virtuali in un ambiente reale, stabilendo una corrispondenza tra gli uni e gli altri sviluppandosi in maniera interattiva, in tempo reale e in tre dimensioni. La percezione degli oggetti virtuali da parte dell'individuo genera la cosiddetta **mixed reality**, ossia una realtà non più solamente fisica e palpabile, ma influenzata dalle informazioni virtuali. Nel *reality-virtuality continuum* in figura 1, la augmented reality viene considerata una parte di mixed reality. Infatti, sia nella realtà virtuale, sia nella virtualità aumentata, dove gli oggetti reali vengono aggiunti ad oggetti virtuali, l'ambiente reale viene rimpiazzato da uno virtuale. Al contrario, la realtà aumentata fornisce solo virtualizzazione locale. [1]

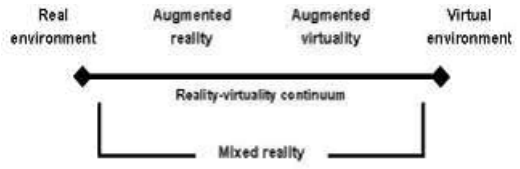

Figura 1.1: Reality-Virtuality continuum

La realtà aumentata si prefigge come obiettivo quello di rispondere all'esigenza di una tecnologia in cui l'individuo può vedere, sentire e ascoltare a un livello sovrapposto a quello della sfera sensoriale umana, rivoluzionando la percezione comune della realtà. Le informazioni relative a un oggetto potrebbero

essere accessibili solamente tramite il riconoscimento dello stesso da parte del supporto per la realtà aumentata o semplicemente per il fatto che ci si trova in quel determinato luogo.

Premesso ciò, è opportuno specificare che quando si parla di realtà aumentata non si fa riferimento a una tecnologia in sé e per sé, ma a un *medium* che concilia le idee tra umani e computer, umani e umani, computer e umani. Ovviamente, per implementare la realtà aumentata, è necessario il ricorso a una o più tecnologie (che verranno approfondite più avanti), ma è necessario chiarire che con la definizione di realtà aumentata non ci si riferisce a una tecnologia vera e propria, ma a un ambito di ricerca ampio e a sé stante, che può essere applicato su diverse aree: educazione, intrattenimento e medicina sono solo alcune. [2]

Un interessante elenco delle principali caratteristiche della realtà aumentata ci viene fornito da Alan B Craig in 'Understanding Augmented Reality'

- Il mondo fisico è aumentato da informazioni digitali che sono *sovrapposte* alla realtà fisica. Questo comporta la presenza di un elaboratore in grado di processare i dati e le informazioni (statiche o dinamiche), ma rimanendo nel mondo fisico, senza il tentativo di dare l'impressione all'individuo di non essere più nella sua posizione nel mondo reale (contrariamente alla realtà virtuale). Le informazioni digitali sono quindi solamente sovrapposte alla realtà fisica, senza il bisogno di dover modificare o occultare quest'ultima

- L'informazione è mostrata *in raccordo* al mondo reale. Infatti questa ha un proprio spazio e un'ubicazione esattamente come li avrebbe la sua controparte nel mondo fisico. Il raccordo con la natura deve essere sia spaziale, sia temporale. Spaziale poichè la rappresentazione di un oggetto aumentato deve essere soggetta a restrizioni, che possono variare in termini di tolleranza a seconda del campo di applicazione: si pensi alla precisione richiesta nell'ambito della chirurgia. Temporale poiché per rendere credibile e realistica la visione di un oggetto aumentato è necessario ridurre per quanto possibile ritardi e lag dovuti all'elaborazione dell'informazione

- L'informazione mostrata dipende dalla *locazione* geografica nel mondo reale e dalla *prospettiva* fisica degli osservatori. Altro aspetto chiave della realtà aumentata è quello di associare il punto di vista fisico dell'individuo a quello del suo profilo di realtà aumentata. Come è facile intuire non è semplice e immediato - in questo genere di applicazioni

– provvedere alla rappresentazione di oggetti aumentati che si raccordino alla natura spazialmente e temporalmente e che vengano percepiti in maniera diversa da prospettive differenti. Quello di cui si deve tener conto è che gli oggetti sono aggiunti al mondo reale e talvolta è necessario semplificare il più possibile l'informazione rappresentata e nascondere gli aspetti non strettamente necessari dalla vista

- L'esperienza di realtà aumentata è *interattiva*, ciò significa che una persona può percepire le informazioni e apporre delle modifiche se lo desidera. Il livello di interattività può variare dal semplice cambiamento di prospettiva (cioè vedere l'oggetto da un altro punto di vista) alla manipolazione e creazione vera e propria di una nuova informazione

1.1.1 Cenni Storici

La prima apparizione della realtà aumentata si colloca negli anni '50, quando il cinematografo Morton Helig pensò al cinema come un'attività che proiettasse lo spettatore nelle azioni sullo schermo coinvolgendo tutti i sensi in maniera effettiva, progettando un prototipo chiamato Sensorama, che anticipò gli elaboratori digitali.

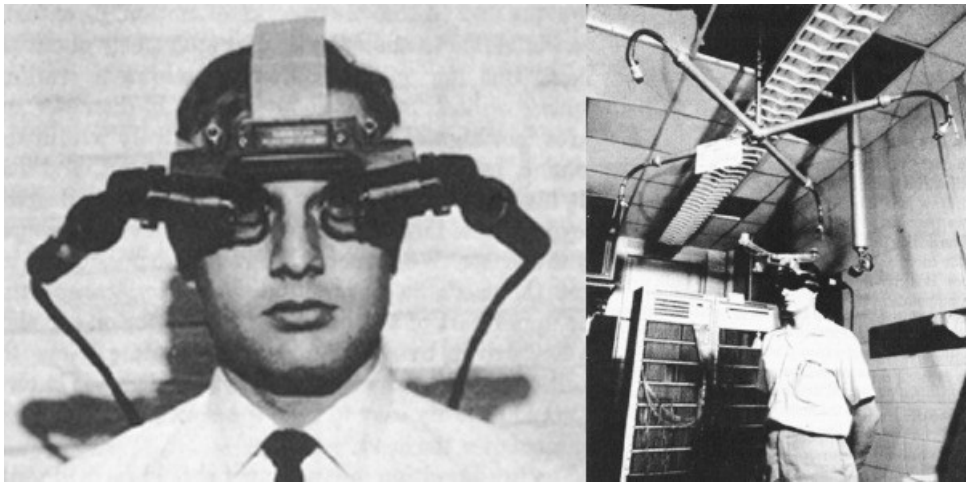

Figura 1.2: Head-mounted display

Poi fu Ivan Sutherland, pioniere della Computer Graphics, nel 1966, a costruire il primo prototipo a supporto della realtà aumentata. Negli anni '70 e '80 vennero approfondite ricerche sulla materia da diversi istituti, tra cui la NASA e il Massachusets Institute of Tecnology, mentre sul mercato si diffondevano su larga scala i primi dispositivi mobile come il Sony Walkman (1979),

gli orologi e le agende digitali. Questa diffusione aprì la strada al *wearable computing* negli anni '90, quando gli elaboratori divennero abbastanza piccoli da poter essere indossati.

Ma la prima apparizione del termine *realtà aumentata* si colloca nei primi anni '90, per opera di Claudell e Mizell, scienziati che al tempo lavoravano su un sistema sperimentale che fornisse supporto ai lavoratori di una ditta di cavi elettrici. Nonostante la scarsa potenza delle tecnologie di quel periodo, venne anche realizzato un prototipo di un sistema mobile basato su realtà aumentata che proiettasse informazioni sui monumenti che il visitatore incontra nel suo tour in 3D della città.

Verso la fine del decennio, contestualmente allo sviluppo della realtà aumentata come campo di ricerca vero e proprio, iniziarono a tenersi diverse conferenze come International Workshop and Symposium on Augmented Reality, International Symposium of Mixed Reality e Designing Augmented Reality Environments workshop. Vennero fondate le prime organizzazioni come il MRLab a Nottingham e Arvika consortium in Germania e divenne possibile sviluppare in breve tempo applicazioni basate su realtà aumentata grazie a toolkit gratuiti come ARToolKit. [1]

Nel 1997, Ronald Azuma scrisse il primo trattato sulla AR identificandola come una combinazione di ambiente reale e virtuale. Nel 2000 venne pubblicato ARQuake, primo gioco mobile AR outdoor, sviluppato da Bruce Thomas. Nel 2005 venne previsto che la tecnologia emergesse completamente nei 4-5 anni successivi, sfruttando anche lo sviluppo delle telecamere. [3]

1.2 Geolocalizzazione e tecniche di riconoscimento

La realtà aumentata può essere fonte di numerosi approfondimenti e spunti di riflessione su tematiche e tecnologie ad essa correlate. La **geolocalizzazione**, ad esempio, è un aspetto che risulta molto spesso cruciale in applicazioni e giochi AR, in particolar modo se questi ultimi sono ambientati *outdoor*. Tali applicazioni si dicono location-based o situate. La posizione di utenti, oggetti reali e oggetti *aumentati* è un aspetto imprescindibile di questi sistemi, che di norma richiedono anche un alto raggio di precisione delle informazioni.

Il **GPS** (Global Positioning System) è il sistema di posizionamento attualmente più diffuso, liberamente accessibile da qualsiasi device dotato di ricevitore GPS. Il suo grado di accuratezza è nell'ordine dei metri, ma la precisione è soggetta a variazioni di diverso genere. Il segnale infatti può risultare troppo debole e quindi è necessario considerare alternative valide. Esistono al giorno d'oggi due varianti del GPS, ovvero D-GPS e A-GPS.

D-GPS (Differential-GPS) permette un'ottima approssimazione della localizzazione utilizzando una rete di postazioni fisse in grado di migliorare la precisione delle coordinate fornite dal sistema GPS tramite il confronto con la posizione relativa rilevata localmente dalla rete.

A-GPS (Assisted-GPS) riduce notevolmente i tempi iniziali di ricerca della posizione, sfruttando i dati provenienti dalla rete quando il segnale GPS è debole.

Nel caso in cui, però, tali tecniche non fossero utilizzabili, è necessario il ricorso ad altre opzioni, come il riconoscimento diretto di oggetti, che può essere *marker-based* o *markerless* e aggiunge informazioni agli oggetti *aumentandoli*. Il riconoscimento marker-based è basato sull'utilizzo di **markers**, immagini predefinite che vengono riconosciute e identificate dalla telecamera. Il riconoscimento markerless invece non fa uso di markers, ma fa leva sul riconoscimento di forme, spigoli e colori. Queste tecniche però introducono il problema della scalabilità, in quanto l'aggiunta di ciascun oggetto aumentato all'interno dell'ambiente di riferimento è necessariamente da associare a un preciso marker, con forma o colore ben stabiliti.

1.3 Cooperazione e realtà aumentata in CSCW

La **collaborazione** è un aspetto della realtà aumentata che merita particolare attenzione. Le tecnologie di collaborazione attuali spesso creano una separazione artificiale tra il mondo reale e il contenuto digitale condiviso. Si pensi all'esempio delle video-conferenze. Risulta difficile per i partecipanti condividere documenti reali o interagire con contenuti 2D sullo schermo con la stessa naturalezza che ci sarebbe in un contesto fisico. La realtà aumentata invece si comporta come medium e non come tecnologia vera e propria, con lo scopo di fondere il mondo fisico con quello virtuale, superandone i limiti. Il proposito è quello di creare l'illusione che un collaboratore in remoto sia presente nell'attuale ambiente di lavoro, costruendo un collegamento più forte di quello della tradizionale videoconferenza. [2]

Il termine **CSCW** (Computer Supported Collaborative Work) fa riferimento all'ambito di ricerca volto a migliorare e rendere efficiente il lavoro in

team e la cooperazione tra i vari membri, utilizzando le tecnologie informatiche in ogni aspetto, come hardware, software, networking e tanti altri. La realtà aumentata fornisce sicuramente una fonte di contributo per il CSCW, con tecnologie wearable e hands-free, l'interazione e manipolazione di oggetti virtuali in team tramite la collaborazione face-to-face, la collaborazione multiscala.

In *Studierstube*, un progetto su un ambiente per la collaborazione in sistemi basati su realtà aumentata, Schmalstieg delineò cinque proprietà chiave alla base di questi sistemi, che, se seguite e presentate in maniera efficace, possono milgiorare notevolmente l'affidabilità di questi sistemi

Virtuality, vale a dire la possibilità di vedere ed esaminare oggetti che non sono accessibili direttamente e che non esistono nel mondo reale

Augmentation, cioè la possibilità di 'aumentare' gli oggetti del mondo reale con informazioni allineate nello spazio, descrizioni o proprietà virtuali.

Multi-user Support, ovvero la base del CSCW, situazione nella quale più utenti si aggregano per discutere, progettare o svolgere altre tipologie di lavori collaborativi.

Independence, ossia il concetto per cui non esiste una persona che detiene il controllo sul sistema mentre gli altri svolgono il ruolo di osservatori passivi. Ogni utente ha quindi la possibilità di muoversi liberamente e indipendentemente, effettuando scelte autonome su cosa vedere e come interagire con il sistema

Individuality, ovvero la possibilità per ogni utente di vedere informazioni diverse da quelle mostrate agli altri utenti, in base alle necessità dell'applicazione e le scelte di ogni individuo

Più tardi, altri sistemi "discendenti" di Studierstube, esplorarono le possibilità di un ambiente mobile 3D per la collaborazione nella realtà aumentata. L'accento venne posto in particolar modo sulla naturalezza con la quale più utenti - che indossano la tecnologia specifica pensata per il sistema – collaborano istantaneamente. Il sistema distribuito sottostante ha il compito di sincronizzare in maniera 'trasparente' la grafica e i dati dell'applicazione anche nei momenti in cui vengono processati notevoli flussi di dati 3D o quando nuovi utenti entrano a far parte del gioco. [5]

1.4 Componenti hardware principali a supporto della realtà aumentata

Nonostante si possa comunemente pensare che i sistemi di realtà aumentata siano del tutto *virtuali*, bisogna precisare che questa non è la realtà ad oggi, ma solo una speranza. I sistemi AR di oggi richiedono infatti sia una parte **software** che una **hardware** per implementare un'esperienza di realtà aumentata

soddisfacente. A seconda della struttura e dei requisiti di un sistema, variano le esigenze delle tecnologie a supporto di sistemi AR. È importante sottolineare che le possibilità di questi tipi di hardware cambiano rapidamente. Per esempio, un'applicazione potrebbe richiedere un processore di una potenza non disponibile attualmente sul mercato ma che probabilmente in futuro - anche prossimo - potrebbe esserci.

I tre componenti hardware di base per ogni sistema AR sono sensori, processori e display. Esistono tante forme o ruoli diversi che ciascuno di questi componenti può assumere, ma è necessario combinare efficacemente questi componenti per dare vita a un sistema AR compatto.

Sensori I sensori hanno come funzionalità principale quella di acquisire informazioni riguardo l'ambiente fisico e di comunicare queste informazioni all'applicazione AR. Il ruolo dei sensori è fornire all'applicazione un **tracking** che può essere di varia natura: può riguardare l'orientazione o la localizzazione di un individuo, fornire informazioni riguardo alla temperatura, al pH, alla luminosità di un ambiente e tante altre. Esiste una grande varietà di sensori. I sensori **ottici** – come una telecamera – offrono la possibilità di tener traccia di più oggetti simultaneamente, di evitare l'uso di cavi o oggetti aggiuntivi e di non avere una connessione fisica tra l'oggetto considerato e il mondo fisico. I sensori **acustici** (microfoni) offrono il vantaggio di non essere soggetti alle condizioni di luce dell'ambiente, ma non risultano efficaci in ambienti rumorosi. I sensori **elettromagnetici** sono molto diffusi, specialmente i trasmettitori, e possono essere molto precisi ed accurati. Non dipendono dalle condizioni di luce, ma sono molto sensibili ai metalli e vanno attentamente calibrati. I sensori **meccanici** – come i potenziometri - operano ponendo dei collegamenti con dei sensori all'oggetto da tracciare. Sono veloci e molto precisi ma presentano problemi di scalabilità in quanto non applicabili a molti oggetti. I sensori di **profondità** infine misurano quanto è distante un oggetto dal sensore. Possono essere ottici o acustici (ultrasuoni) e sono economici ma abbastanza limitati. E' frequente anche l'uso di sensori multipli per il tracking – come per esempio nel Kinect, espansione della Xbox - rispondendo in modo specifico a ogni esigenza della propria applicazione, anche se è opportuno integrarli.

Processori I processori sono componenti essenziali in un sistema basato su realtà aumentata. Col termine processore si intende una singola unità oltre che componenti multipli che lavorano insieme per fornire un sistema di elaborazione. Oltre all'unità centrale, il processore copre un vasto numero di ruoli in un sistema AR: elabora i dati provenienti dai sensori, esegue le istruzioni dell'applicazione e crea i segnali che guidano i display. In generale, un sistema

AR comprende un microprocessore primario (la *CPU*) 'general-purpose' e uno o più processori grafici (GPU) 'special-purpose', cioè orientati a un compito specifico. La **GPU** è un componente hardware ottimizzato per processare grafica in 3D, particolarmente adatto in sistemi AR.

Numerose configurazioni e architetture diverse vengono utilizzate per applicazioni AR. Tra le più comuni troviamo le architetture per sistemi portatili come gli smartphone o i tablet, che hanno una discreta potenza computazionale di base ma non sufficiente – in generale, ad oggi – per supportare task che richiedono elaborazioni di grande intensità. Per ovviare a tali limitazioni si può pensare a un'architettura mobile connessa a un server, ma saranno presenti comunque delle limitazioni in termini di rappresentazione delle informazioni scambiate sul device. Le architetture su laptop o computer dispongono di ottime risorse computazionali ma presentano chiari limiti di portabilità, mentre esistono anche applicazioni web basate su browser che presentano vantaggi in quanto accessibili da ogni computer ma anche svantaggi in quanto il browser necessita di essere correttamente configurato con i sensori e potrebbero insorgere problemi di rete in determinati luoghi. Scegliere adeguatamente l'architettura hardware per la propria specifica applicazione è quindi un aspetto fondamentale nella progettazione e nell'analisi di un progetto.

Display Il display è il device che produce il segnale che i nostri sensi percepiscono. Esistono varie tipologie di display, visuali, acustici, tattili, stereo. I display visuali creano segnali di luce che percepiamo tramite la vista, sono i display con cui le persone hanno più familiarità perché molto diffusi (si pensi allo schermo di un computer). Per la realtà aumentata si distinguono tre categorie di display visuali: stazionari, display che si muovono con la testa di chi li controlla (head-mounted), display che si muovono con le mani di chi li controlla o tramite altre parti del corpo. Tutte queste categorie sono comunemente utilizzate in sistemi AR. [2]

1.5 Mirror worlds

La realtà aumentata offre anche un interessante spunto di riflessione sui Mirror Worlds. Il mondo, non più solo reale e fisico, viene ampliato dal software, che aumenta anche ciò che l'utente percepisce, ma allo stesso tempo anche il mondo percepisce il reale, e così anche l'utente. In questo modo non è solo il mondo reale ad essere esteso dal virtuale, ma anche il reale diventa estensione del virtuale, facendo sì che possa variare il suo stato in base ai cambiamenti

e alle variazioni del mondo reale, come ad esempio gli sbalzi di luminosità, rumori, spostamenti d'aria.

La realtà aumentata si inserisce nel contesto dei Mirror Worlds come tecnologia che rende possibile lo sviluppo di questo tipo di mondo: gli oggetti del mondo fisico devono avere – implicitamente o esplicitamente – un'estensione digitale nel Mirror World che rappresenti l'oggetto stesso in termini di software agent o di parte dell'ambiente degli agenti. L'estensione degli oggetti può anche includere un 'augmentation' come nel caso della realtà aumentata mobile, che può essere percepita come una manifestazione – statica o dinamica – dagli abitanti del mondo fisico tramite device come smartphones o smart-glasses.

Un esempio di gioco mobile AR modellato sul concetto di Mirror World è Ghost Game in the City. Il Mirror World è composto da una serie di tesori e fantasmi distribuiti in varie parti della città. Ci sono due team di giocatori umani, dotati di AR glasses e smartphone (che funge da bacchetta magica) . L'obiettivo è quello di raccogliere quanti più tesori possibile camminando per la città senza essere catturati dai fantasmi. I fantasmi sono agenti che si muovono autonomamente nel Mirror World e nella città, con l'obiettivo di catturare i giocatori, inseguendoli non appena avvertono la loro presenza.

I giocatori percepiscono i fantasmi grazie ai glasses quando si trovano nella stessa posizione; i fantasmi – allo stesso modo – avvertono la presenza dei giocatori appena si trovano a una certa distanza, ma divengono anche vulnerabili non appena aumenta lo stato di luminosità del mondo reale e valutano strategie di inseguimento diverse a seconda di variabili del mondo reale. Questi sono tutti aspetti che introducono una differenza sostanziale rispetto ai classici giochi AR.

Capitolo 2

Un Caso di Studio: location-based mobile AR game collaborativo

2.1 Introduzione

L'idea alla base del progetto è quella di fornire una rappresentazione grafica delle informazioni e degli oggetti attraverso la realtà aumentata all'interno di un sistema nel quale un team formato da più individui collabora al raggiungimento e all'analisi di un numero predefinito di punti di interesse all'interno di una determinata zona. Ogni punto di interesse presente nel sistema corrisponde a determinate coordinate geografiche e funge da 'scrigno' nel quale è contenuto un oggetto. L'oggetto può essere un valore monetario oppure una chiave per aprire un altro scrigno. Lo scopo dei giocatori è quello di visitare i punti di interesse per raccogliere tutte le chiavi e tutte le monete presenti sul campo di gioco. Quando un giocatore raccoglie una chiave in un punto di interesse, solo lui potrà aprire lo scrigno a cui è associata quella chiave. Il gioco termina quando tutti gli scrigni saranno stati aperti.

Gli scrigni sono rappresentati da una figura geometrica precisa, un quadrato, mentre la colorazione di questa indica lo stato in cui si trovano. Informazioni aggiuntive – come ad esempio il modo in cui poter 'sbloccare' il punto ed accedere alle informazioni – possono essere mostrate attraverso appositi campi di testo.

- **Bianco**: non visitato, ancora nessuno conosce le informazioni sul contenuto del punto d'interesse

- **Verde**: il punto è stato aperto, la chiave o il denaro in esso contenuto sono stati già raccolti da un altro giocatore

- **Rosso**: il punto necessita di una chiave contenuta in un altro punto per essere aperto

- **Blu**: per raccogliere il contenuto del punto è necessario che almeno due membri del team siano presenti simultaneamente nel punto

La visualizzazione della figura geometrica - posta in corrispondenza del punto di interesse – varia in base alla distanza dell'osservatore, aumentando in dimensioni quando il giocatore si avvicina. Quando il giocatore raggiunge il punto di interesse, lo stato dello scrigno – se apribile senza bisogno di chiave - cambia in 'aperto' e ne viene specificato il contenuto (chiave o monete). Nel caso in cui invece lo stato cambi in 'non apribile', viene specificato il metodo da utilizzare per aprire lo scrigno (con chiave o con cooperazione).

Se il giocatore è in grado di aprire lo scrigno, può visualizzare il contenuto di esso. Il contenuto può consistere in una chiave, la quale sarà aggiunta alla lista delle chiavi già in possesso del giocatore; oppure in monete, che andranno ad aggiornare il totale di quelle possedute dal soggetto.

Se uno scrigno è 'bloccato', l'informazione sul tipo di apertura che richiede è condivisa tra tutti gli utenti, ovvero tutti possono conoscere lo stato dei punti in ogni momento tramite una schermata apposita e ogni cambiamento di stato viene notificato agli altri. Ogni giocatore visualizza sullo schermo un elenco con tutti i punti di interesse presenti sul campo di gioco, la posizione dell'altro giocatore e lo stato dei punti di interesse (se aperti, non visitati oppure ancora da sbloccare).

Sul campo di gioco agiscono anche dei 'ladri fantasma', posizionati in determinate zone, che agiscono assalendo il giocatore che si trova nella loro stessa posizione in quel momento. Il giocatore visualizza un segnale - un cerchio bianco – quando raggiunge l'effettiva posizione del fantasma. Il ladro fantasma, una volta raggiunto, deruba il giocatore di una quantità predefinita di monete, se ne possiede, che viene detratta dal totale. Il giocatore a questo punto può rilasciare un segnale di pericolo sul campo, simboleggiato da un triangolo, per evitare che i compagni passino dal punto in cui è presente il fantasma.

2.2 Considerazioni

Il caso di studio offre diversi spunti per quanto riguarda la realtà aumentata. In primo luogo, si vuole realizzare una presentazione delle informazioni nel sistema che sia coerente con la realtà e possa dare l'idea – a uno sguardo esterno – di un progetto di realtà aumentata che fa uso di elementi geometrici allo stesso tempo semplici ma realistici. Gli oggetti 'aumentati' presenti nel sistema non pretendono di fornire avanguardistici modelli tridimensionali, ma al contrario sono pensati in modo tale da essere chiari e lineari, fornendo all'utente un'idea della realtà aumentata. Questo può essere reso tramite figure geometriche più o meno semplici, ma anche grazie ad altre forme di 'augmentation', come l'aggiunta di informazioni testuali vicino a una figura. Per rappresentare le informazioni nelle modalità sopra specificate, sarà necessario servirsi di algoritmi e tecniche di disegno di figure geometriche a supporto della realtà aumentata.

L'applicazione è location-based, cioè la localizzazione sarà un punto cruciale nel sistema. La posizione degli utenti deve essere nota in ogni momento e confrontata con quella dei punti di interesse e dei fantasmi. Ciò rende necessario uno studio adeguato delle tecniche di geolocalizzazione e di un sistema di condivisione delle posizione tra i vari elementi del gioco. Un altro aspetto da analizzare nella realizzazione del sistema è quello della comunicazione, fondamentale per un sistema dove l'informazione è condivisa tra tutti gli utenti e distribuita su diversi devices. In ultimo, anche la cooperazione tra i vari giocatori è presente nel gioco: gli utenti infatti sono costretti – implicitamente o esplicitamente – a collaborare per raggiungere l'obiettivo primario del gioco. La collaborazione, in particolare, si ha quando entrambi gli utenti devono presenziare simultaneamente in un punto di interesse per sbloccare lo scrigno e quando un utente rilascia la notifica di pericolo in prossimità di un fantasma.

Capitolo 3

Analisi e Modellazione

3.1 Casi d'Uso

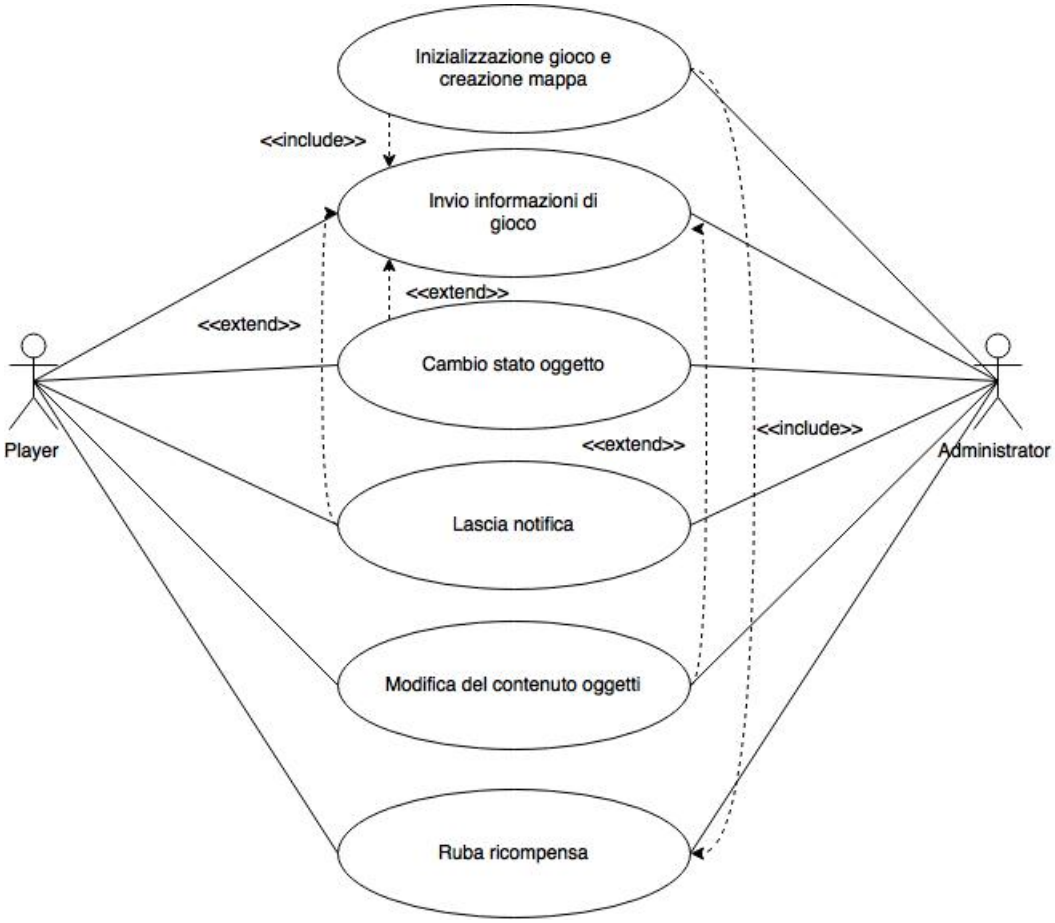

Figura 3.1: Casi d'uso del nostro sistema

3.1.1 Inizializzazione del gioco e creazione della mappa

Il server, una volta avviato, crea gli oggetti e fornisce loro delle coordinate, andando a definire così la mappa di gioco. Le informazioni contenute negli oggetti saranno il fulcro del sistema e verranno modificate e condivise con gli utenti.

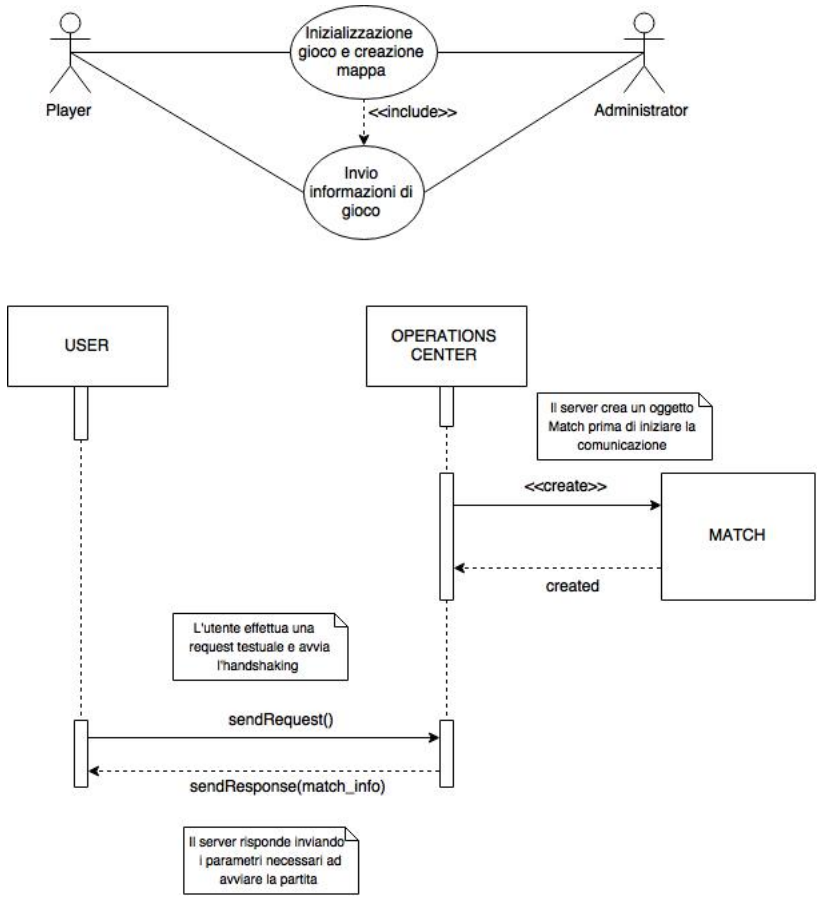

Figura 3.2: Diagramma di sequenza relativo all'inizializzazione del gioco

3.1.2 Invio delle informazioni di gioco

Il server viene contattato da un utente e risponde inviando le informazioni relative a tutti i punti di interesse precedentemente inizializzati. Una volta completato il download delle informazioni da entrambi i giocatori il gioco ha inizio. Si noti che inizialmente le informazioni inviate agli utenti danno indicazioni solo sulla posizione dei punti ma non sul loro contenuto.

3.1.3 Cambio stato oggetto

L'oggetto è localizzato attraverso due zone circolari, il cui centro è dato dalle coordinate dell'oggetto. La zona più ampia indica la vicinanza all'oggetto, per cui quando l'utente vi entra è avvisato di essere in prossimità dell'oggetto. La zona più piccola indica l'oggetto stesso, per cui quando l'utente entrerà in tale zona andrà ad agire direttamente sull'oggetto, causandone eventualmente il cambio di stato.

Gli stati che caratterizzano un oggetto sono i seguenti:

- UNVISITED: oggetto che si trova in un punto non ancora visitato
- OPEN: oggetto aperto (il suo contenuto è stato ottenuto)
- LOCKEDKEY: oggetto visitato ma chiuso e da aprire con la chiave specificata
- LOCKEDCOOPERATION: oggetto visitato ma chiuso e da aprire con cooperazione attraverso gli utenti
- FINAL: oggetto finale, visitato ma da aprire solamente con la cooperazione di entrambi gli utenti e dopo aver aperto tutti gli altri oggetti

L'arrivo di un utente in prossimità di un oggetto ne determina l'eventuale cambio di stato.

- boolean key_needed
- boolean cooperation_needed
- boolean final
- boolean visited
- boolean have_key
- int number_of_user
- boolean all_open

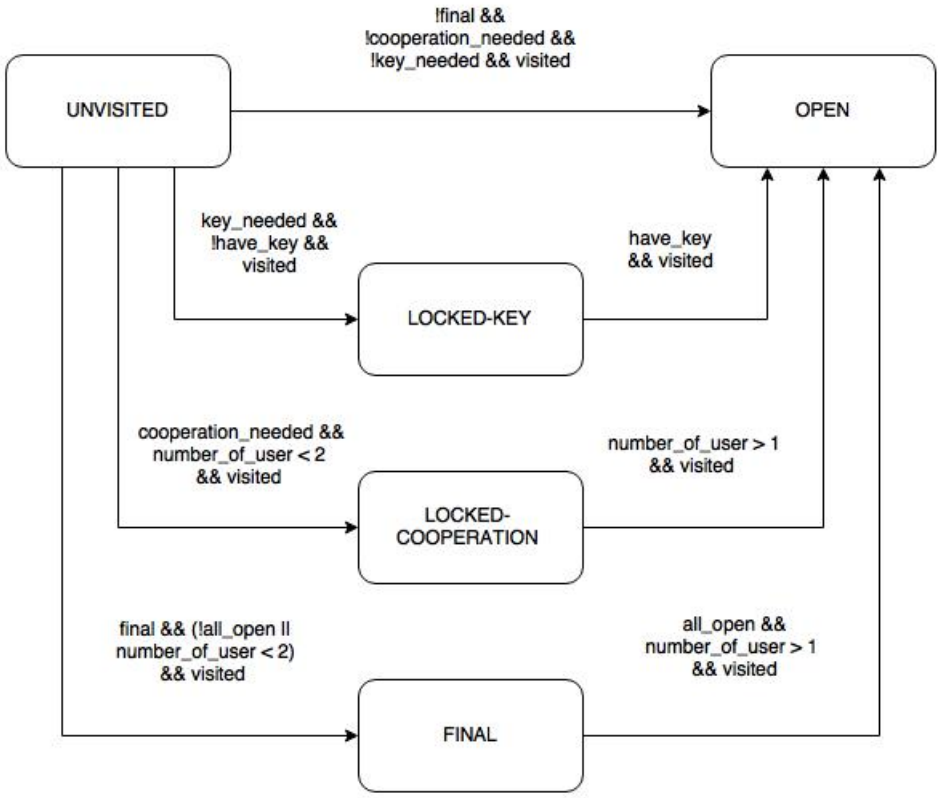

Figura 3.3: Diagramma a stati del Treasure Chest

Ecco il diagramma di sequenza che mostra le interazioni che avvengono al momento del cambio di stato di un oggetto:

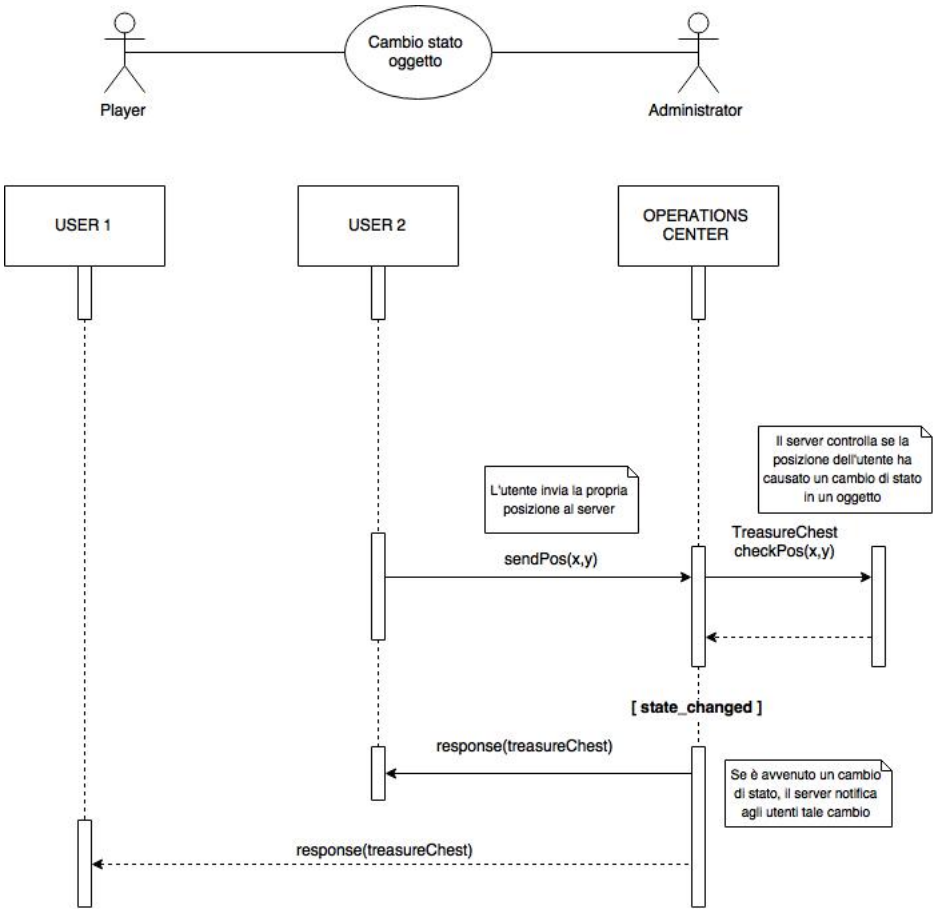

Figura 3.4: Diagramma di sequenza relativo al cambio di stato di un oggetto

Ecco i vari scenari:

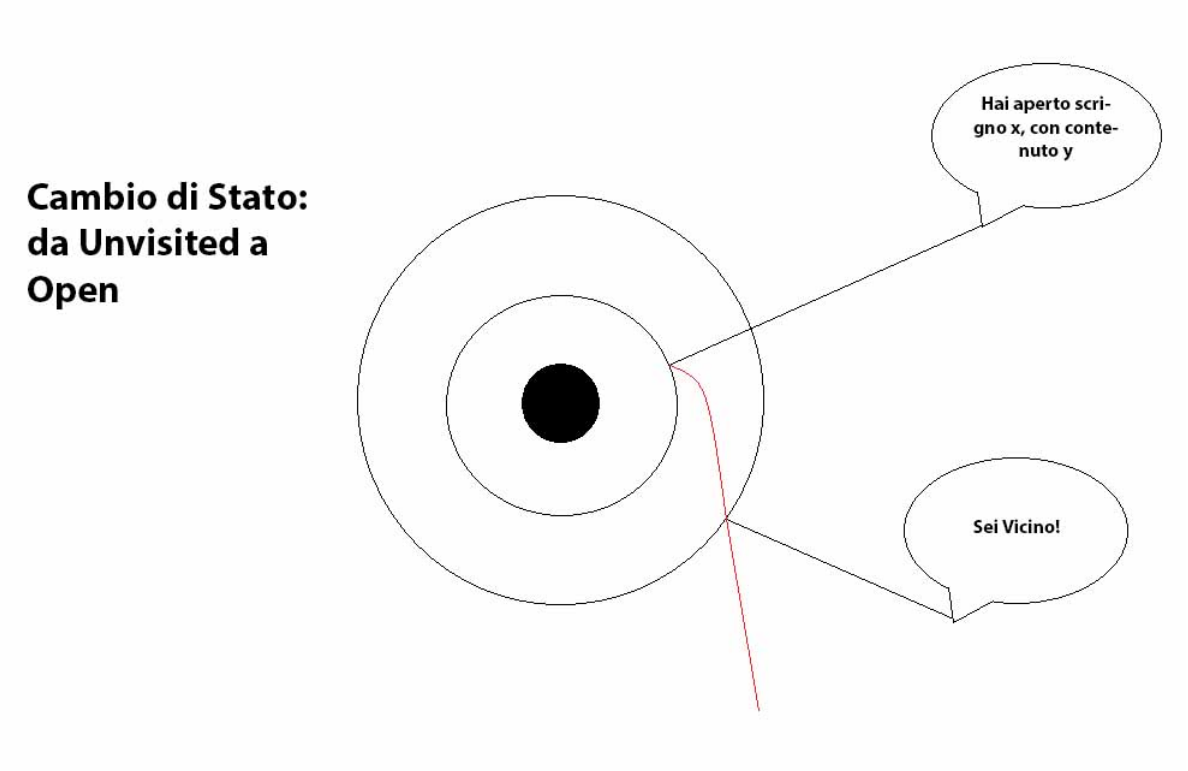

Figura 3.5: Caso in cui lo scrigno può essere aperto

Caso in cui lo scrigno può essere aperto Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato di essere in prossimità dell'oggetto. Tale notifica non è generata dal server ma direttamente lato utente, siccome la posizione dell'oggetto è nota a priori grazie all'handshaking iniziale. Una volta arrivato entro il raggio più ristretto, l'utente viene notificato dal server (che conosce la posizione dell'utente in ogni istante) che ha aperto con successo l'oggetto e ne ha ottenuto il contenuto. Il cambio di stato dell'oggetto da UNVISITED a OPEN viene notificato ad entrambi gli utenti insieme all'informazione relativa al suo contenuto (monete o chiavi).

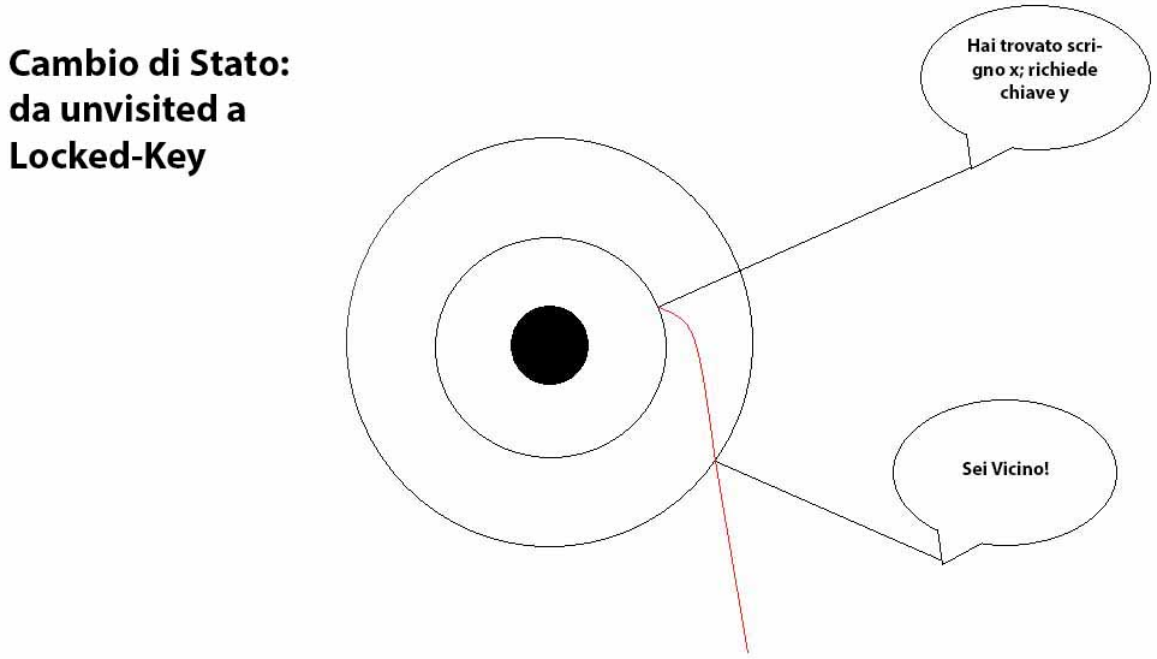

Figura 3.6: Caso in cui l'apertura dell'oggetto richiede una chiave Y, NON POSSEDUTA dall'utente

Caso in cui l'apertura dell'oggetto richiede una chiave Y, NON POSSEDUTA dall'utente Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato di essere in prossimità dell'oggetto. Tale notifica non è generata dal server ma direttamente lato utente. Una volta arrivato entro il raggio più ristretto, l'utente viene notificato dal server che è richiesta la chiave Y per poter aprire l'oggetto. Il cambio di stato dell'oggetto da UNVISITED a LOCKEDKEY viene notificato ad entrambi gli utenti insieme all'informazione relativa alla chiave necessaria per aprirlo.

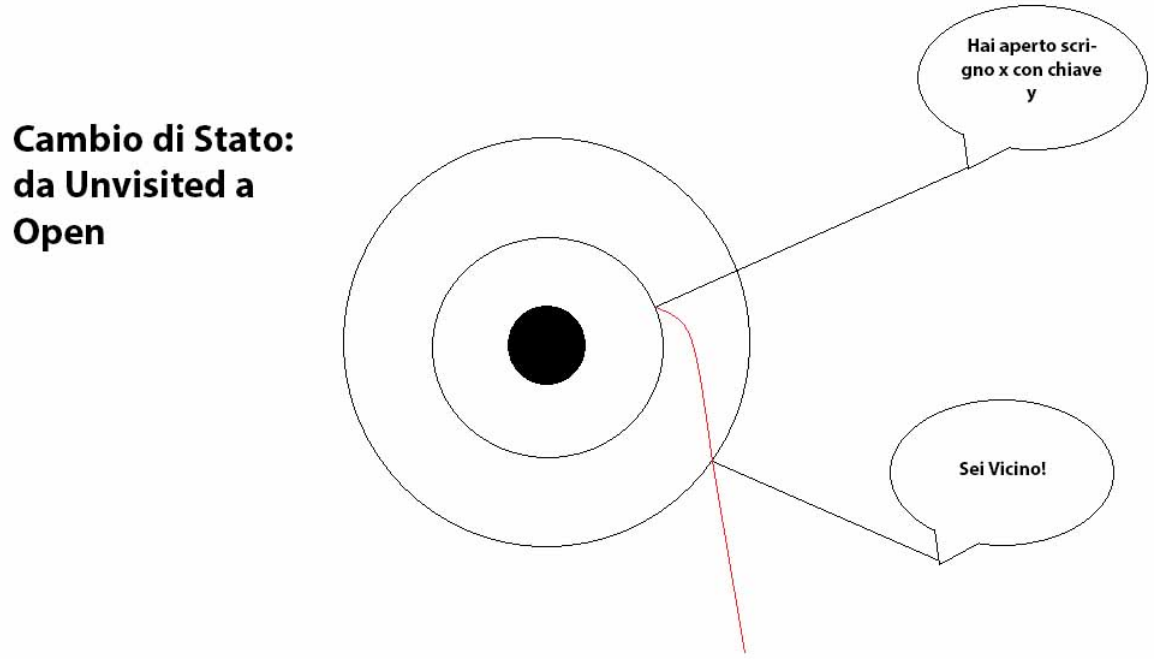

Figura 3.7: Caso in cui l'apertura dell'oggetto richiede una chiave Y, POSSEDUTA dall'utente

Caso in cui l'apertura dell'oggetto richiede una chiave Y, POSSEDUTA dall'utente Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato di essere in prossimità dell'oggetto. Tale notifica non è generata dal server ma direttamente lato utente. Una volta arrivato entro il raggio più ristretto, l'utente viene notificato dal server che ha aperto con successo l'oggetto grazie all'ausilio della chiave Y. Il cambio di stato dell'oggetto da UNVISITED a OPEN viene notificato ad entrambi gli utenti insieme all'informazione relativa relativa al suo contenuto (monete o chiavi).

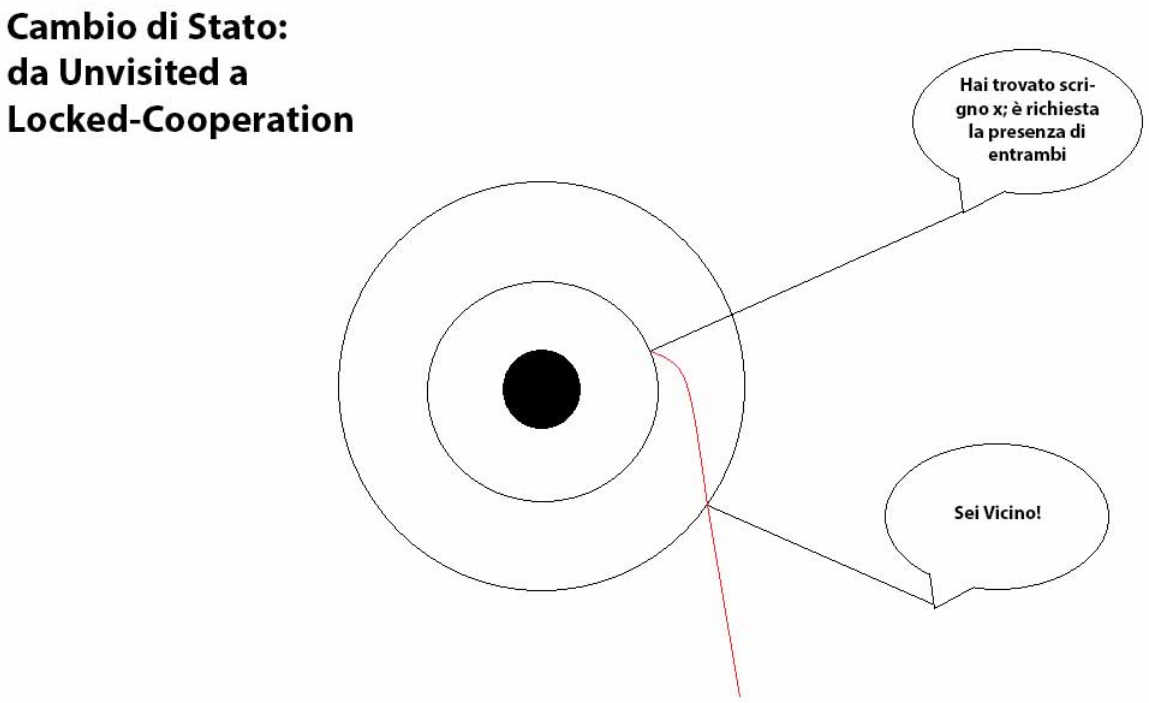

Figura 3.8: Caso in cui l'apertura dell'oggetto richiede cooperazione

Caso in cui l'apertura dell'oggetto richiede cooperazione Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato di essere in prossimità dell'oggetto. Tale notifica non è generata dal server ma direttamente lato utente. Una volta arrivato entro il raggio più ristretto, l'utente viene notificato dal server che per aprire l'oggetto è necessaria la cooperazione tra i due utenti. Il cambio di stato dell'oggetto da UNVISITED a LOCKED-COOPERATION viene notificato ad entrambi gli utenti. All'utente che non si trova nel punto viene richiesto di esplicitare se è intenzionato a recarsi nel punto o meno. In base alla risposta l'utente che si trova nel punto saprà se attendere o meno.

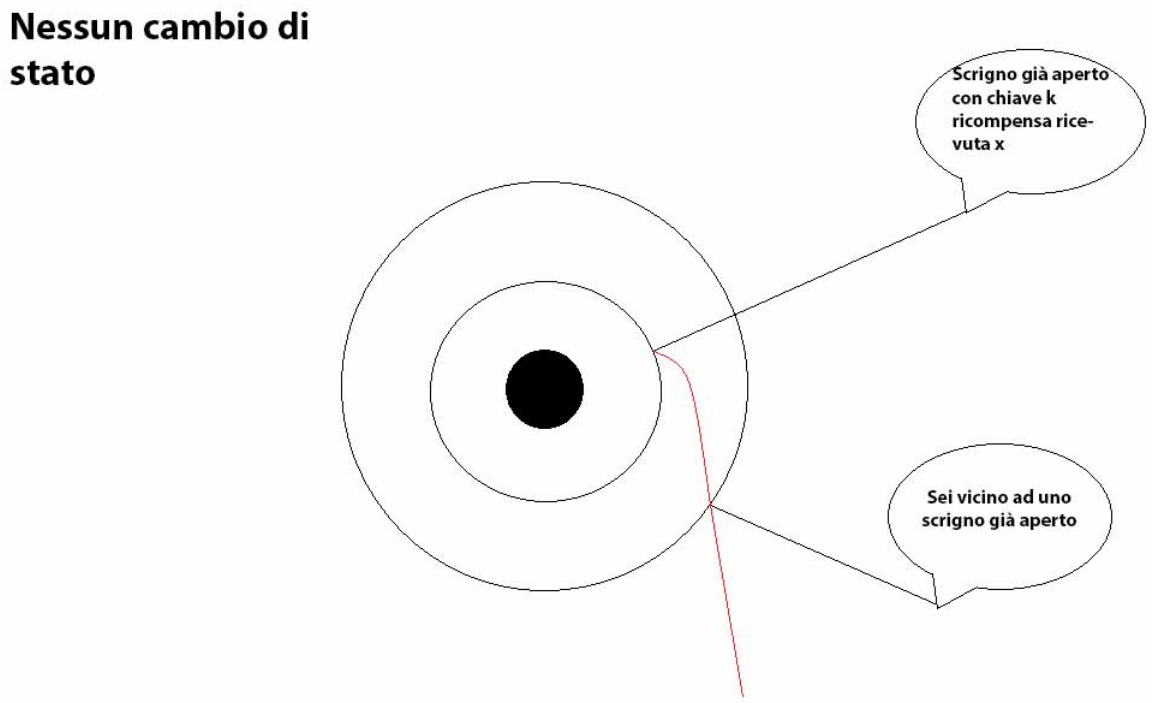

Figura 3.9: Caso in cui l'oggetto è stato già aperto

Caso in cui l'oggetto è stato già aperto Interazioni: in tale caso non avviene alcuna interazione tra utente e server: l'utente era già stato notificato dell'apertura dell'oggetto per cui possiede già tale informazione, quindi una volta giunto entro il raggio più ampio viene notificato che si trova in prossimità di un oggetto che è già stato aperto. Una volta arrivato entro il raggio più ristretto viene notificato che l'oggetto a cui si trova di fronte è già stato aperto precedentemente con conseguente guadagno di X monete. Non avviene alcun cambio di stato dell'oggetto.

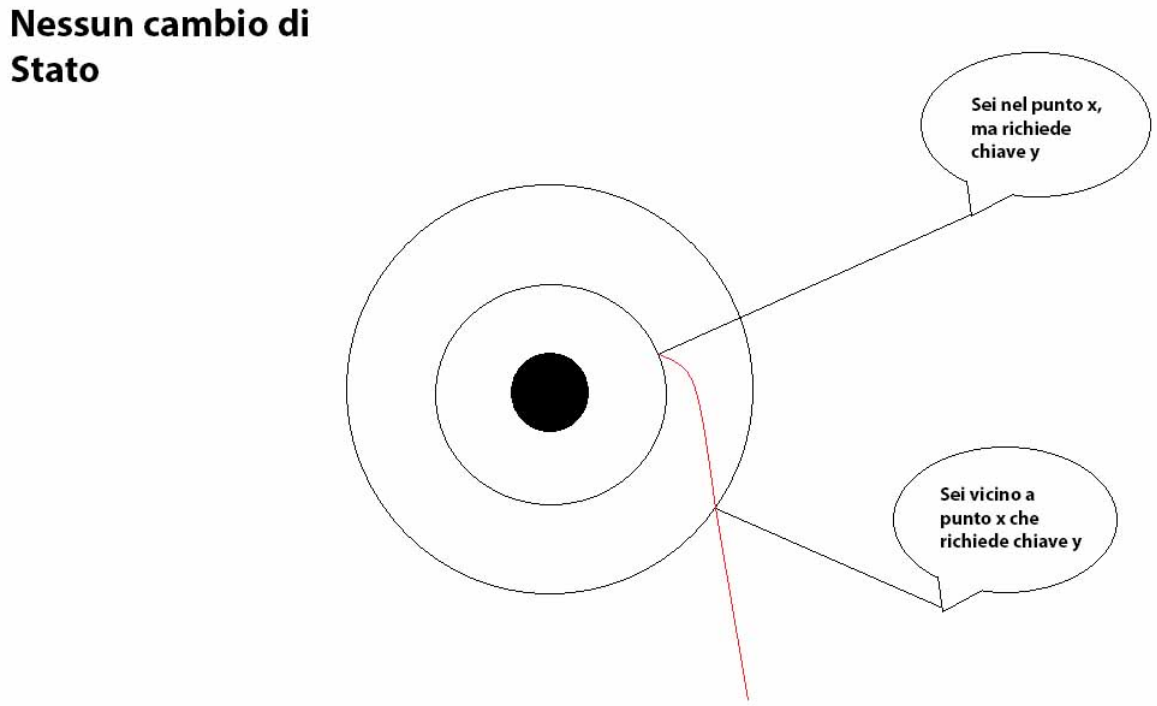

Figura 3.10: Caso in cui l'oggetto richiede una chiave Y, NON POSSEDUTA dall'utente, ed è già stato precedentemente visitato

Caso in cui l'oggetto richiede una chiave Y, NON POSSEDUTA dall'utente, ed è già stato precedentemente visitato Interazioni: anche in tale caso non avviene alcuna interazione tra utente e server. Infatti l'utente era già stato precedentemente notificato dell'avvenuta visita all'oggetto (da parte sua o del compagno) e dell'impossibilità nell'aprirlo se non attraverso la chiave Y. Per cui in locale l'utente verrà prima notificato di essere in prossimità di un punto che richiede la chiave Y, successivamente verrà notificato (sempre in locale) di trovarsi di fronte ad un oggetto che attualmente non può aprire poichè non possiede la chiave Y. Non avviene alcun cambio di stato dell'oggetto.

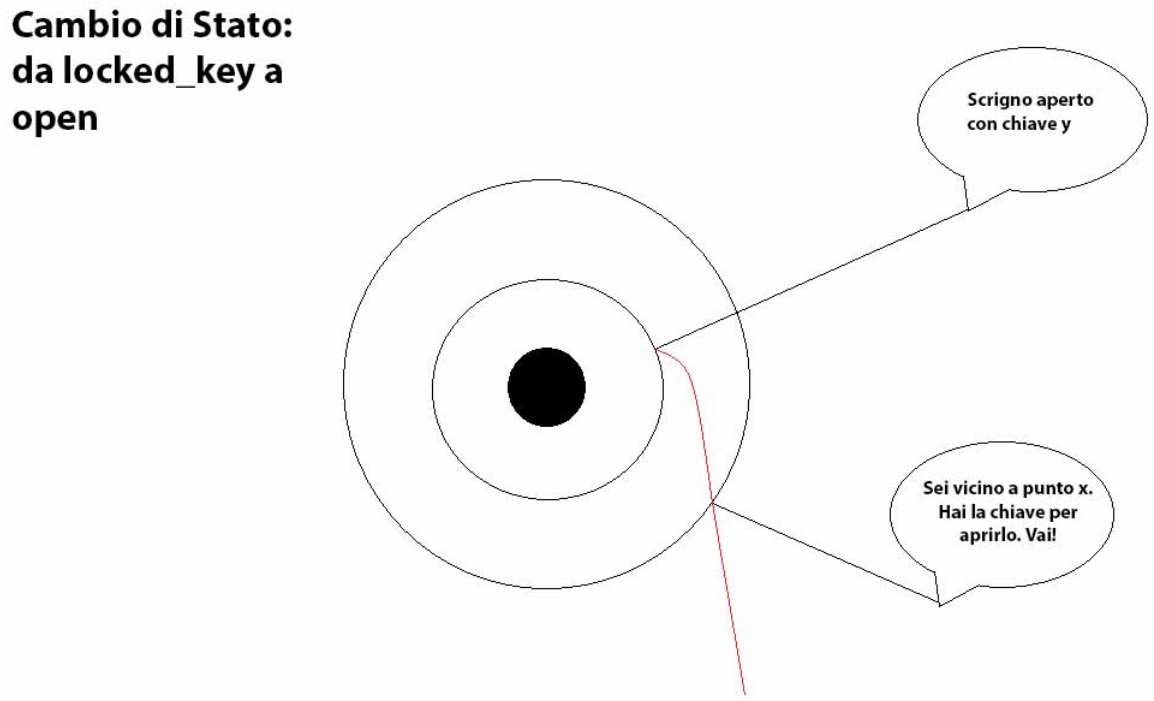

Figura 3.11: Caso in cui l'oggetto richiede una chiave Y, POSSEDUTA dall'utente, ed è già stato precedentemente visitato

Caso in cui l'oggetto richiede una chiave Y, POSSEDUTA dall'utente, ed è già stato precedentemente visitato Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato di essere in prossimità dell'oggetto e di possedere la chiave Y necessaria ad aprirlo. Tale notifica non è generata dal server ma direttamente lato utente, poichè l'utente aveva già ricevuto la notifica della visita a tale oggetto (da parte sua o del compagno). Una volta arrivato entro il raggio più ristretto, l'utente viene notificato dal server che ha aperto l'oggetto grazie all'ausilio della chiave Y, ottenendo il suo contenuto (monete o chiave). Il cambio di stato dell'oggetto da LOCKED-KEY a OPEN viene notificato ad entrambi gli utenti insieme all'informazione relativa relativa al suo contenuto (monete o chiavi).

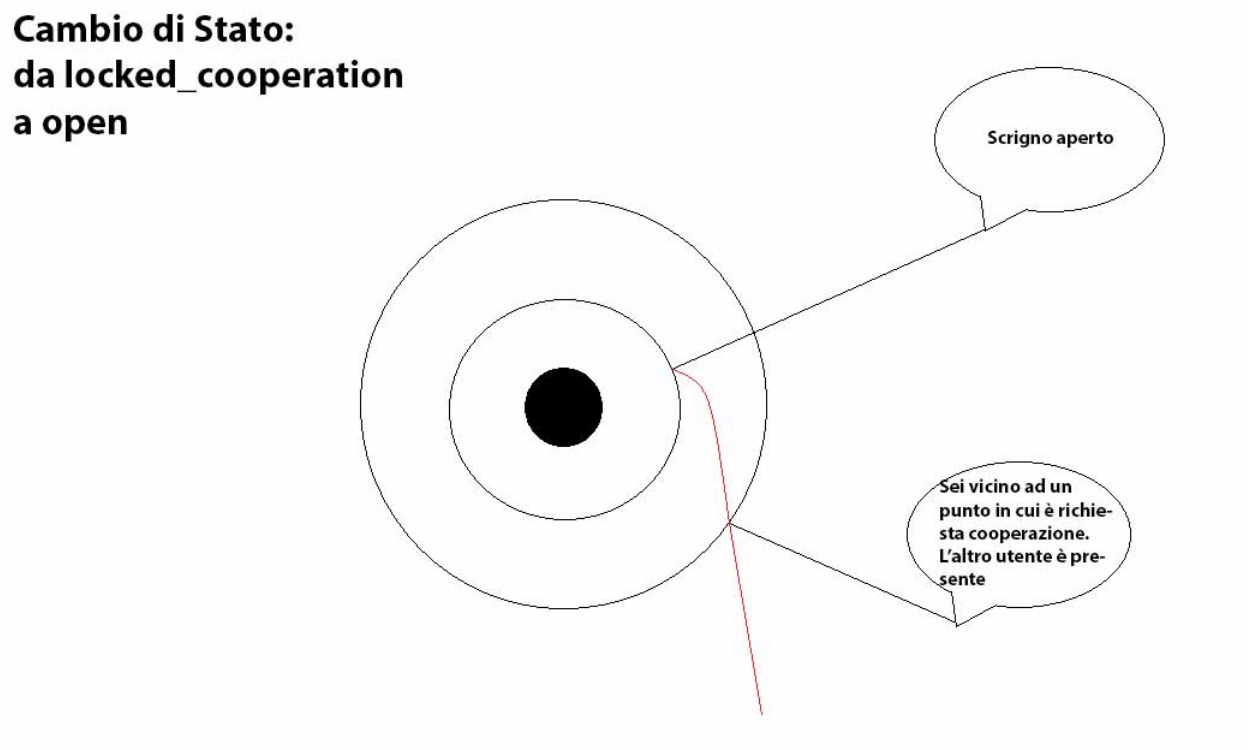

Figura 3.12: Caso in cui l'oggetto richiede cooperazione ed è già stato precedentemente visitato (l'altro utente è PRESENTE entro il raggio più ristretto dell'oggetto)

Caso in cui l'oggetto richiede cooperazione ed è già stato precedentemente visitato (l'altro utente è PRESENTE entro il raggio più ristretto dell'oggetto) Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato di essere in prossimità dell'oggetto e che il proprio compagno è già in corrispondenza di tale oggetto. Tale notifica proviene dal server, che possiede le informazioni circa la posizione dei due utenti ad ogni loro spostamento. Una volta giunto entro il raggio più ristretto, l'utente viene notificato dell'avvenuta apertura dell'oggetto insieme al proprio compagno. Se è presente una chiave, verrà assegnata ad uno solo dei due utenti, mentre all'altro verrà notificata la sua scoperta. Il cambio di stato dell'oggetto da LOCKEDCOOPERATION a OPEN viene notificato ad entrambi gli utenti insieme all'informazione relativa relativa al suo contenuto monetario.

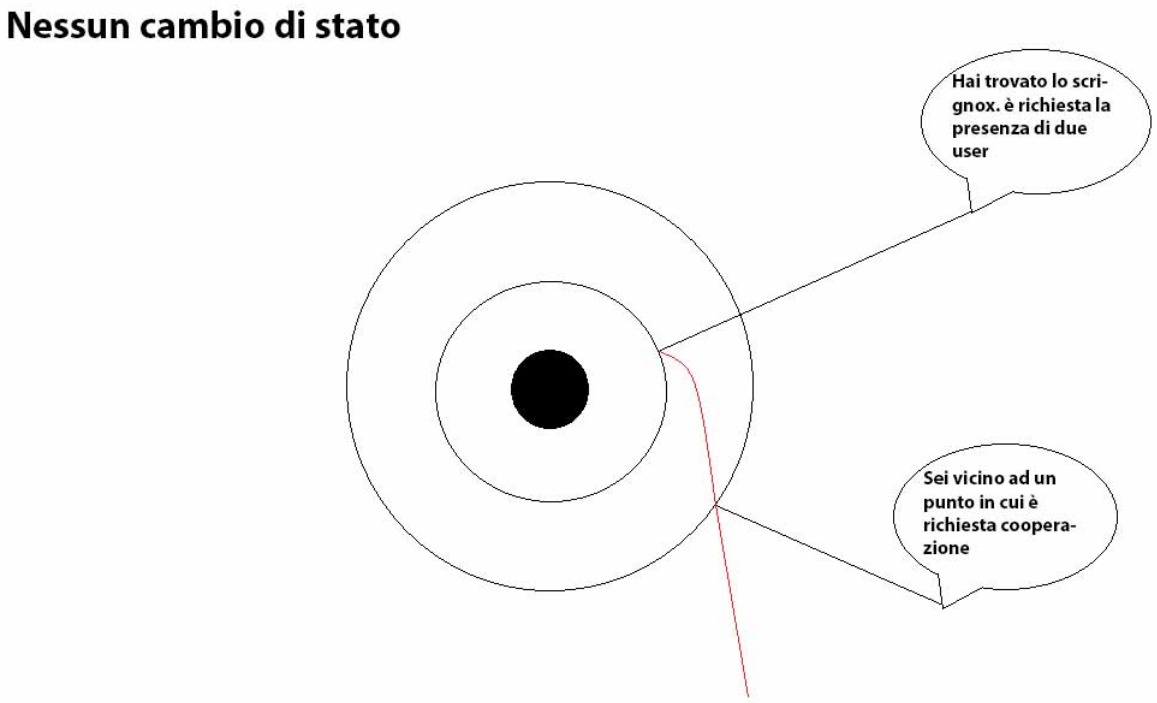

Figura 3.13: Caso in cui l'oggetto richiede cooperazione ed è già stato precedentemente visitato (l'altro utente NON è PRESENTE entro il raggio più ristretto dell'oggetto)

Caso in cui l'oggetto richiede cooperazione ed è già stato precedentemente visitato (l'altro utente NON è PRESENTE entro il raggio più ristretto dell'oggetto) Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato di essere in prossimità di un oggetto che richiede cooperazione. Tale notifica non è generata dal server ma direttamente lato utente. Si ripresenta lo scenario della scoperta di un oggetto che richiede cooperazione, per cui una volta arrivato entro il raggio più ristretto, l'utente viene notificato dal server che per aprire l'oggetto è necessaria la cooperazione tra i due utenti. All'utente che non si trova nel punto viene richiesto di esplicitare se è intenzionato a recarsi nel punto o meno. In base alla risposta l'utente che si trova nel punto saprà se attendere o meno. Non avviene alcun cambio dui stato dell'oggetto.

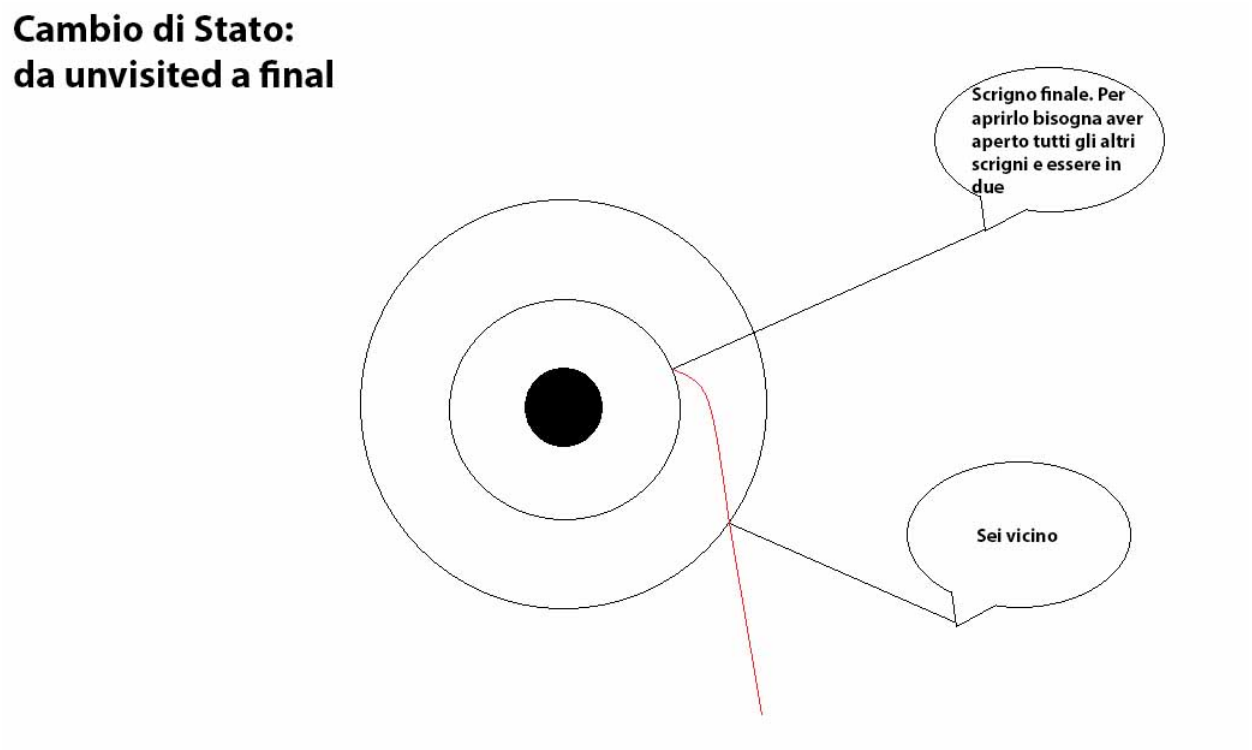

Figura 3.14: Caso in cui l'oggetto finale viene scoperto ma non può essere aperto

Caso in cui l'oggetto finale viene scoperto ma non può essere aperto
Interazioni: una volta arrivato entro il raggio più ampio, l'utente viene notificato in locale di essere in prossimità di un punto. Una volta arrivato entro il raggio più ristretto, avviene il cambio di stato da UNVISITED a FINAL che viene notificato agli utenti.

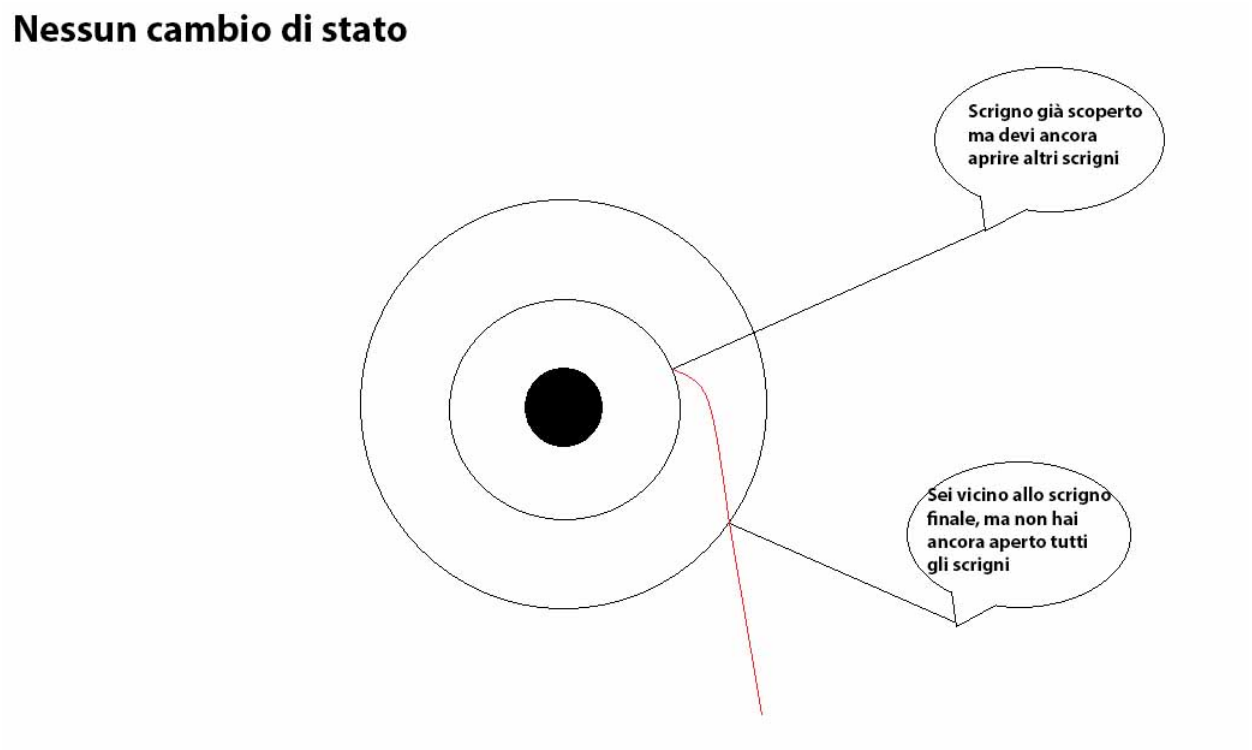

Figura 3.15: Caso in cui l'oggetto finale è già stato visitato ma non può essere aperto perchè non tutti gli altri oggetti sono già stati aperti

Caso in cui l'oggetto finale è già stato visitato ma non può essere aperto perchè non tutti gli altri oggetti sono già stati aperti Interazioni: non avvengono interazioni, siccome gli utenti possiedono già l'informazione relativa al numero di scrigni già aperti, per cui le notifiche riguardo la prossimità all'oggetto finale e il bisogno di aprire altri oggetti vengono generate in locale. Non avviene alcun cambio di stato.

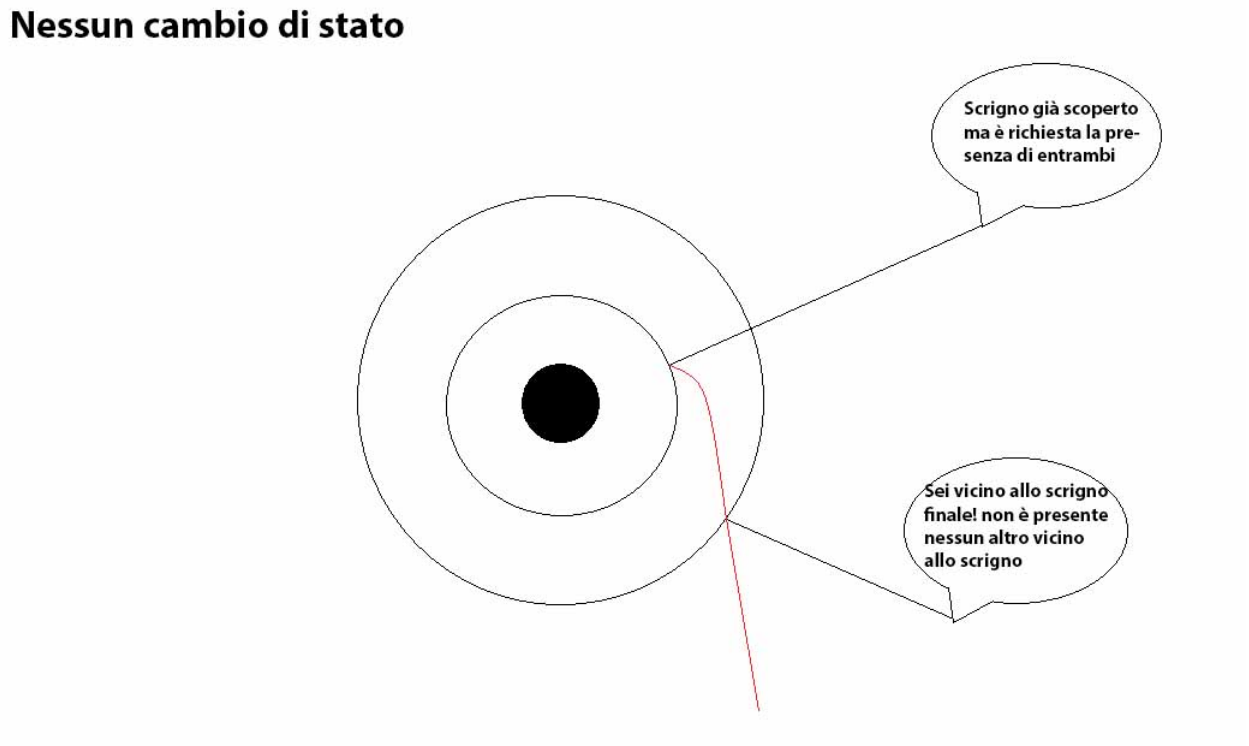

Figura 3.16: Caso in cui l'oggetto finale è già stato visitato ma non può essere aperto perchè il compagno non è presente (tutti gli altri oggetti sono già stati aperti)

Caso in cui l'oggetto finale è già stato visitato ma non può essere aperto perchè il compagno non è presente (tutti gli altri oggetti sono già stati aperti) Interazioni: in questo caso, siccome tutti gli altri oggetti sono già stati aperti, la situazione è analoga al caso in cui sia richiesta cooperazone per l'apertura dello scrigno (per cui avviene la richiesta di disponibilità all'altro utente). Non avviene alcun cambio di stato.

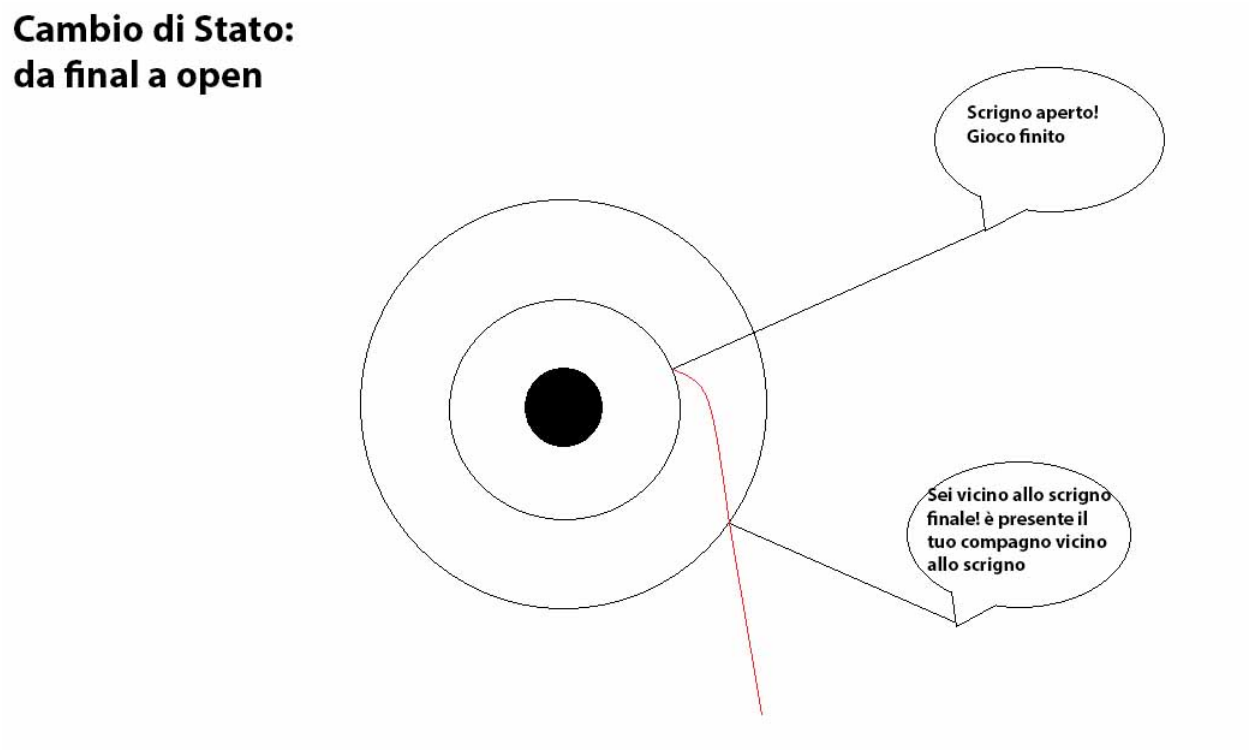

Figura 3.17: Caso in cui l'oggetto finale può essere aperto (è presente il compagno e tutti gli oggetti sono stati aperti)

Caso in cui l'oggetto finale può essere aperto (è presente il compagno e tutti gli oggetti sono stati aperti) Interazioni: in questo caso, siccome tutti gli altri oggetti sono già stati aperti e il compagno è presente nel punto dell'oggetto, la situazione è analoga a quella della cooperazione richiesta con presenza del compagno in loco. Una volta raggiunto l'oggetto il gioco è concluso.

3.1.4 Lascia Notifica

L'utente ha la possibilità, nell'eventualità che ne abbia bisogno, di lasciare una notifica, di cui tiene traccia il server, in una zona dove è svolto il gioco. Il caso che verrà studiato è la presenza di un ladro virtuale in una certa zona della mappa.

Nel caso in cui uno degli utenti incappi nel ladro può lasciare un segnale in quella zona in modo tale che se l'altro utente si avvicini a quella zona il server gli notifichi il pericolo.

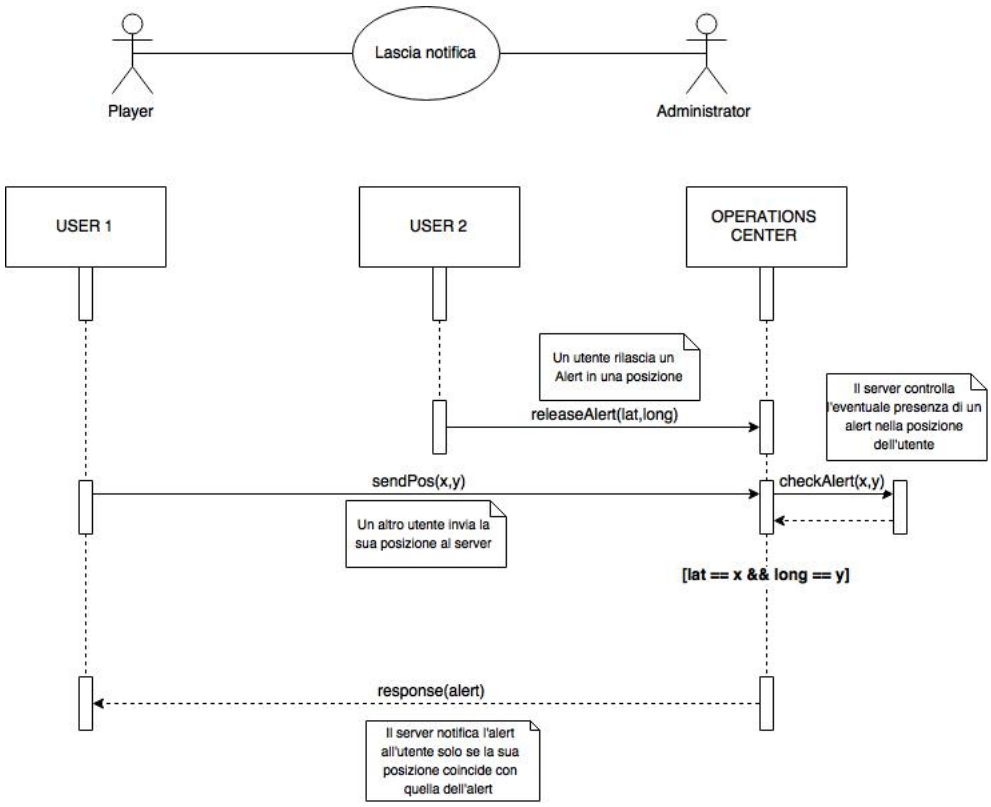

Figura 3.18: Diagramma di sequenza relativo al rilascio di una notifica

3.1.5 Modifica Contenuto

La centrale operativa può modificare ed in particolare diminuire il valore della ricompensa contenuta all'interno degli oggetti.

Nel nostro caso di studio verrà gestita la diminuzione delle ricompense con un certo timer, ovvero dopo intervalli di tempo prefissati verrà diminuito il valore del contenuto degli oggetti.

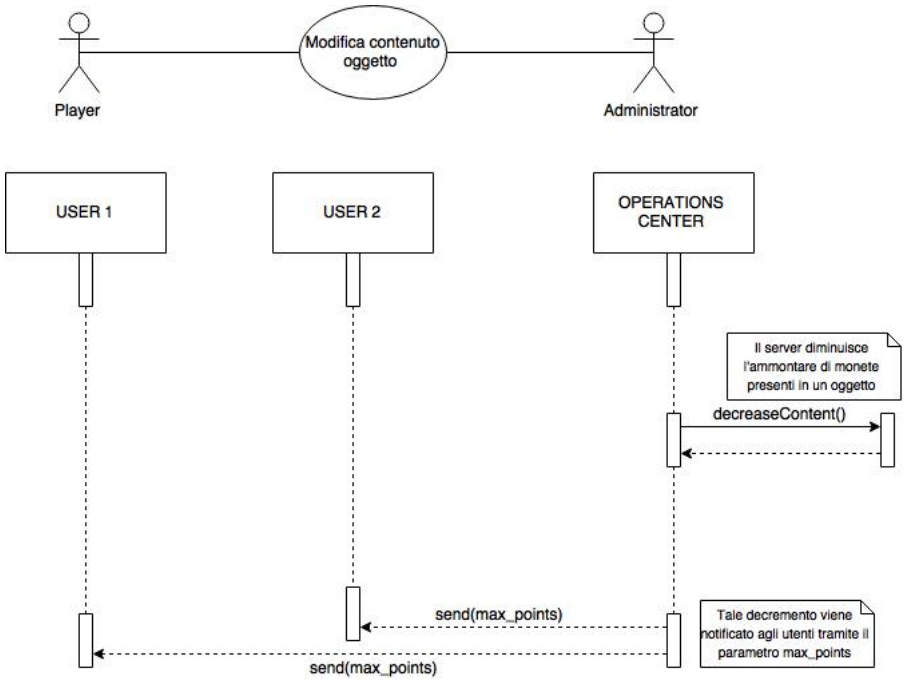

Figura 3.19: Diagramma di sequenza relativo alla modifica del contenuto degli oggetti

3.1.6 Ruba Ricompensa

L'attore ladro nel momento in cui l'utente entra nel suo raggio d'azione ha la possibilità di rubare una quantità prestabilita di ricompensa, e questo chiaramente solo se gli utenti hanno già aperto qualche scrigno .

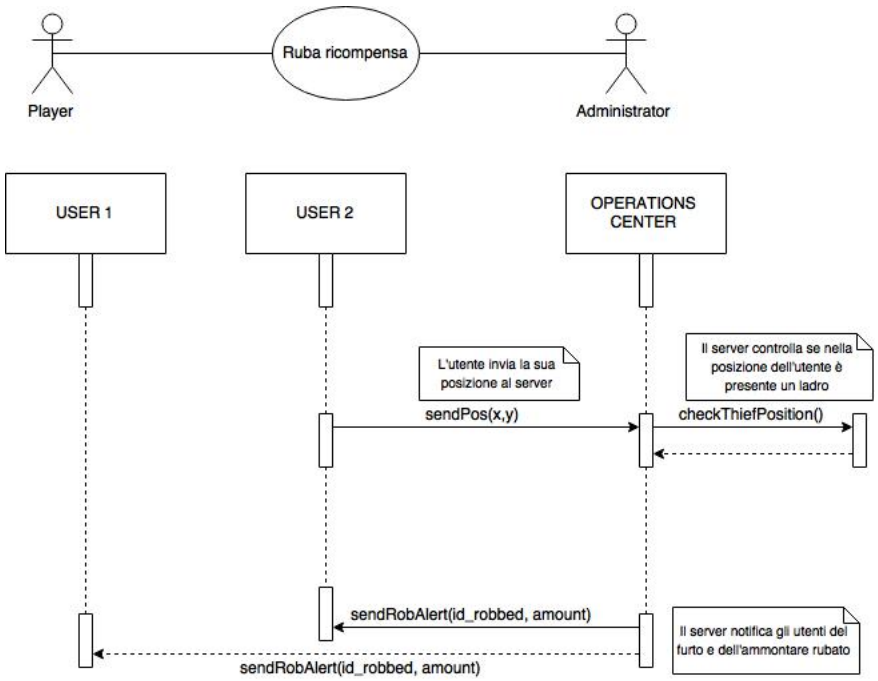

Figura 3.20: Diagramma di sequenza relativo all'azione del ladro

3.2 Dominio Applicativo

3.2.1 Lato Server

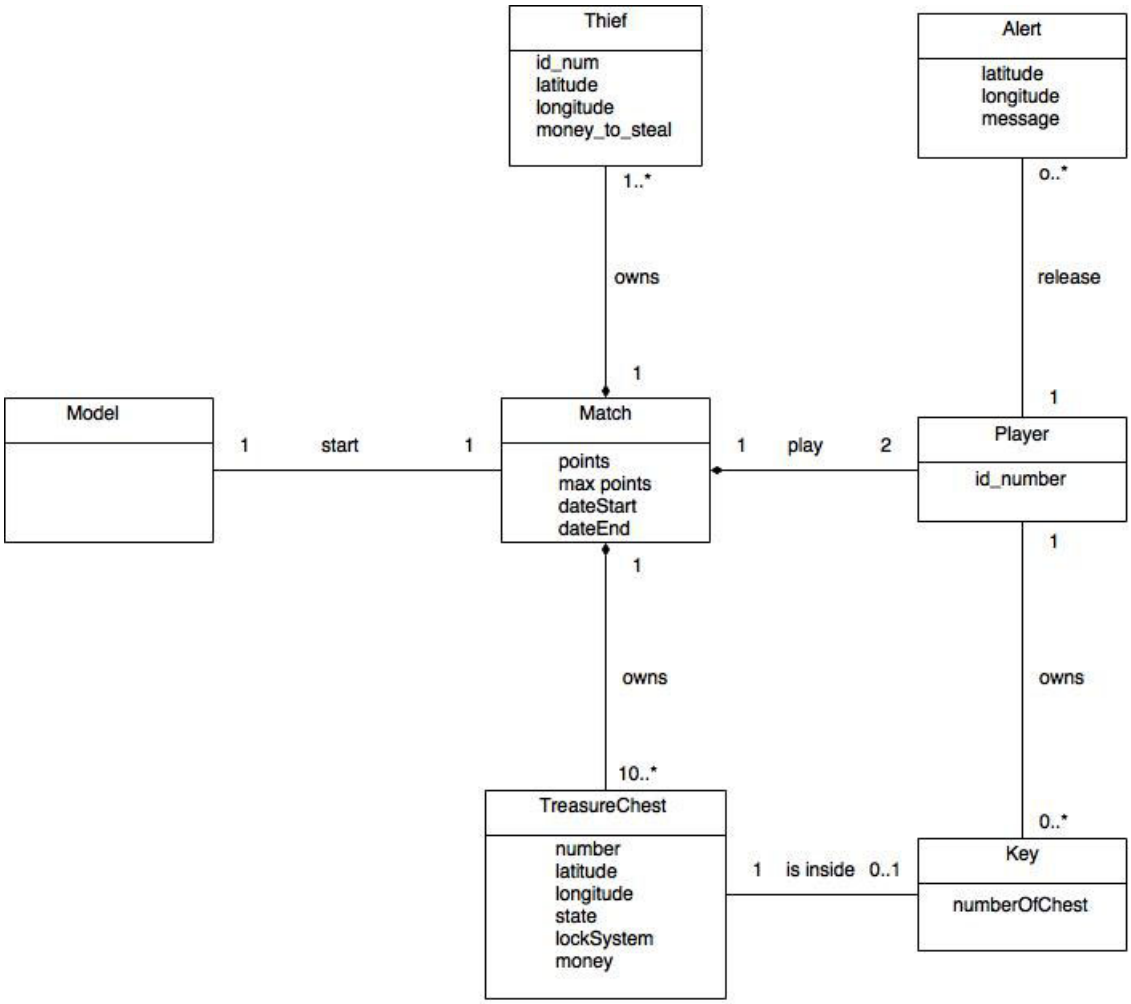

Figura 3.21: Diagramma delle classi del dominio applicativo lato server

Model E' il container del dominio applicativo, ovvero è la classe da cui verrà inizializzata la partita e gestito tutto il dominio.

Match E' la modellazione della partita. Al suo interno si tiene traccia del punteggio attuale, che all'inizio è zero, e del massimo punteggio raggiungibile che viene aggiornato dopo i furti e dopo le diminuzioni da parte della centrale operativa. In più tiene traccia del giorno di inizio e di fine.

TreasureChest E' la modellazione dello scrigno al cui interno si tiene traccia del numero dello scrigno, della posizione, del suo stato, del sistema di blocco che influenza poi il cambio di stato e l'ammontare di monete al suo interno. In più può contenere una chiave di cui tiene una referenza.

Key E' la modellazione della chiave e ha come unico campo il numero dello scrigno che può aprire.

Player E' la modellazione dei giocatori e tiene traccia dell'id, che lo identifica in tutti i suoi messaggi mandati, degli alert che rilascia ,delle chiavi che possiede e della sua posizione.

Alert E' la modellazione della notifica che un giocatore può lasciare in un certo punto della mappa. E' caratterizzato dalla posizione e dal messaggio lasciato dall'utente.

Thief E' la modellazione dell'entità nemica. Anche questa viene situata in un punto sulla mappa, per cui si tiene traccia della sua posizione, è identificata dal suo id e ha una quantità che può rubare quando "assale" un giocatore.

3.3 Lato Client

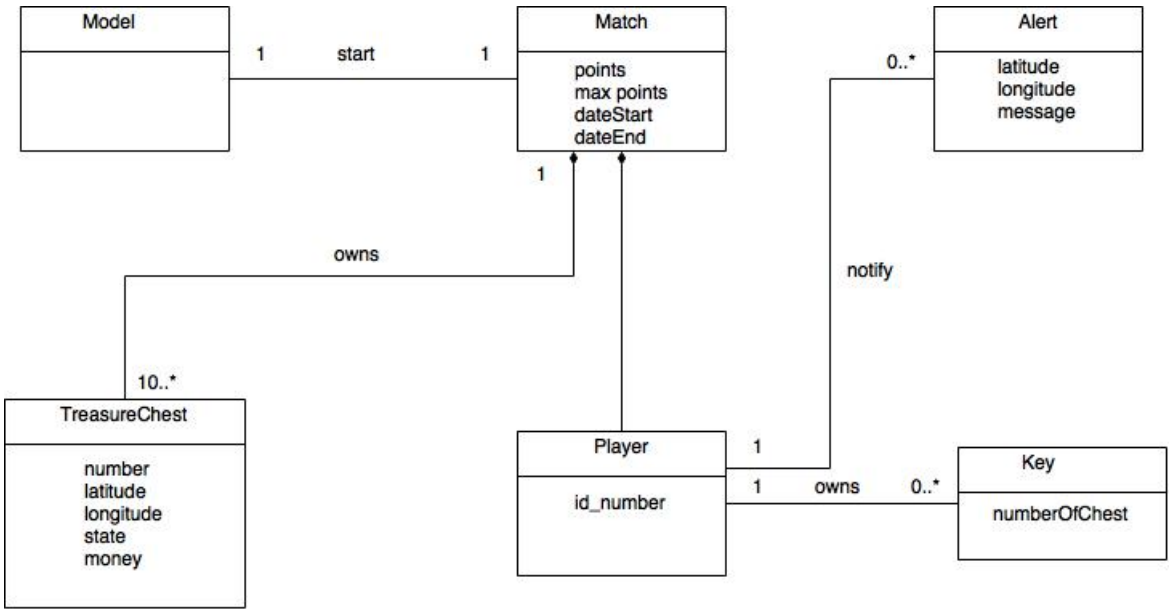

Figura 3.22: Diagramma delle classi del dominio applicativo lato client

TreasureChest E' l'unica modellazione che ha delle differenze dal lato server in quanto non tiene traccia del sistema di blocco. In più quando il gioco viene creato sono inizializzate solamente la posizione e lo stato, mentre le altre variabili vengono inizializzate solo nelle successive modifiche.

3.4 Dominio applicativo dei messaggi scambiati e tecniche di comunicazione e interazione

Di seguito vengono elencati tutti i tipi di messaggi scambiati tra server e utente, specificando il tipo di informazione inviata e la tecnica di invio per ognuno.

3.4.1 Inizializzazione

La comunicazione ha inizio tramite un "handshaking" che da il via all'invio dei dati relativi al mondo condiviso da server a user. L'utente effettua una **request**, composta da un semplice messaggio testuale destinato al server. Il server, dopo aver assegnato un ID all'utente, reagisce con una **response** contenente i seguenti dati:

- Tipo messaggio
- ID Utente
- Monete totali disponibili
- Lista contenente i TreasureChest in questa forma:

TreasureChest
longitude
latitude

Tali oggetti verranno poi mappati in oggetti di questo tipo, lasciando inizialmente vuoti i campi number e money.

TreasureChest
longitude
latitude
state
number
money

In questo modo l'utente ha ricevuto le informazioni necessarie ad iniziare il gioco, ovvero la posizione GPS degli oggetti (senza ancora conoscerne il contenuto), il totale di monete che può ottenere insieme al compagno e il proprio ID.

3.4.2 Invio posizione (da Client a Server)

L'utente comunica la propria posizione GPS ad ogni suo spostamento, inviando i seguenti dati:

- Tipo messaggio
- ID Utente
- Latitudine
- Longitudine

3.4.3 Cambio di stato dell'oggetto (da S a C)

Quando l'utente arriva in un punto d'interesse, il server invia informazioni riguardanti il cambiamento di stato del relativo oggetto (se è avvenuto un cambio di stato). Si distinguono due casi:

L'oggetto viene aperto In questo caso vengono inviati i seguenti dati:

- Tipo messaggio (object open)
- ID Utente
- Chiave contenuta (campo a zero se non contiene chiavi)
- TreasureChest in questa forma:

TreasureChest
longitude
latitude
state
number
money

In questo modo l'utente riceve tutte le informazioni contenute nell'oggetto.

L'oggetto non viene aperto In questo caso vengono inviati i seguenti dati:

- Tipo messaggio (object not open)
- ID Utente
- Chiave contenuta (campo a zero se non contiene chiavi)
- TreasureChest in questa forma:

TreasureChest
longitude
latitude
state
number

In questo modo l'utente non riceve le informazioni relative al contenuto dell'oggetto, ma solo sul suo stato e sul numero dell'oggetto.

3.4.4 Invio di conferma/rifiuto cooperazione (da C a S):

Messaggio generato dal cliente che riceve la richiesta di cooperazione. E' un semplice messaggio testuale con cui l'utente può accettare/rifiutare la richiesta di cooperazione.

Cooperazione accettata

- Tipo messaggio
- ID Utente
- Messaggio accetto

Cooperazione rifiutata

- Tipo messaggio
- ID Utente
- Messaggio rifiuto

3.4.5 Notifica di conferma/rifiuto cooperazione (da C a S):

Il server, una volta ricevuta la conferma/rifiuto di cooperazione da parte di U2, deve notificarla a U1. Viene "inoltrato" lo stesso messaggio precedentemente ricevuto.

Cooperazione accettata

- Tipo messaggio
- ID Utente
- Messaggio accetta cooperazione

U1 resterà nel punto in attesa del compagno.

Cooperazione non accettata

- Tipo messaggio
- ID Utente
- Messaggio rifiuta cooperazione

U1 sarà libero di proseguire, poiché il compagno non è intenzionato a recarsi lì.

3.4.6 Notifica di allerta (da C a S):

L'utente può segnalare degli Alert messages con cui aiutare il compagno a non incappare in zone pericolose. Può lasciare un messaggio che verrà visualizzato dall'altro utente quando entrerà nella zona in cui il messaggio è stato rilasciato.

Il messaggio ha la seguente struttura:

- Tipo messaggio
- ID Utente
- Alert in questa forma:

Alert
longitude
latitude
message

3.4.7 Notifica di allerta (da S a C):

Quando un utente raggiunge una zona in cui il compagno (o anche se stesso) ha lasciato un Alert message, il server gli notifica il messaggio in questo modo:

- Tipo messaggio
- ID Utente
- Alert in questa forma:

Alert
longitude
latitude
message

Si noti che in base all'ID ricevuto l'utente saprà se l'Alert message era stato rilasciato da lui stesso o dal compagno.

3.4.8 Messaggio Ladro (da S a C):

Quando un utente giunge in un'area in cui è presente un ladro, il server notifica l'avvenuta rapina, tramite un messaggio di questo tipo:

- Tipo messaggio
- ID Utente
- Quantità derubata

Si noti che in base all'ID ricevuto l'utente saprà se la vittima è stata lui stesso o il compagno. Entrambi gli utenti sapranno quale importo è stato derubato per cui l'importo verrà scalato dal totale delle monete fino a quel momento ottenute e dal totale di monete ottenibili.

3.4.9 Messaggio diminuzione ricompensa (da S a C):

Quando il server diminuisce la quantità di monete presenti negli oggetti notifica l'avvenimento inviando il nuovo importo totale disponibile:

- Tipo messaggio
- Nuovo importo massimo

Capitolo 4

Progettazione

4.1 Compito svolto all'interno del progetto

Il mio compito nella progettazione e sviluppo del progetto è stato in un primo luogo quello di collaborare nell'ideazione e nella realizzazione dell'infrastruttura di rete e delle funzionalità di localizzazione, cooperazione e collaborazione tra diversi utenti. Questo lavoro preliminare mi ha permesso di ottenere le API a livello applicativo per sviluppare lo strato dell'interfaccia utente in modo da rendere efficacemente l'idea di un'applicazione basata su realtà aumentata, inserendosi nel contesto di un sistema location-based nel quale la cooperazione tra più utenti al fine del raggiungimento di un obiettivo predefinito gioca un ruolo di primaria importanza.

4.2 Problematiche principali

La progettazione e lo sviluppo di un sistema distribuito sono fasi che devono tenere conto di diverse problematiche, per avere una consapevolezza più generale e per studiare un approccio che consenta di ovviare alle possibili difficoltà

Un primo aspetto da considerare è quello della coerenza dei dati. Il sistema – come già detto in precedenza – è distribuito e multi-user, quindi è molto importante che le informazioni condivise siano in ogni momento le stesse per tutti gli utenti e che corrispondano a quelle mantenute nel server. Il caso più evidente di incoerenza dei dati è quello in cui un oggetto cambia stato ma per qualche motivo tale cambio non viene notificato a un utente: quest'ultimo, quando giungerà nel determinato punto, realizzerà che l'informazione in suo posses-

so non è la stessa mantenuta nel server, e questo aspetto pregiudicherebbe il funzionamento dell'intero sistema.

È importante anche tenere presente delle latenze e dei ritardi che potrebbero portare a situazioni di incoerenza all'interno del sistema: un esempio è quello di un utente che – arrivato in un punto in cui serve la presenza simultanea di un altro utente per essere sbloccato – si muove dal punto e riceve la conferma o il rifiuto da parte di un compagno in ritardo, rendendo praticamente inutile la comunicazione.

Altra tematica da affrontare in un sistema location-aware è quello della frequenza con cui avviene uno scambio di informazioni riguardo alla posizione tra user e server. È chiaro il fatto che l'utilizzo di frequenze molto elevate di invio aumenta la velocità e la reattività del sistema, ma questo non è sempre possibile. Per esempio, un utente potrebbe non essere in grado di vedere l'oggetto aumentato situato in un punto di interesse per il semplice fatto che ci transita troppo velocemente perché il sistema possa accorgersene.

Uno scenario outdoor abbastanza ampio come quello che ospita il sistema è inoltre una fonte di problemi per quanto riguarda il tema delle disconnessioni. Questa problematica, forse la più critica dell'intero sistema, va approcciata sapendo che l'eventuale disconnessione di un utente non deve compromettere l'intero gioco, ma al tempo stesso comporta la perdita di tutte le informazioni al suo riguardo (inclusa la sua posizione). Per questo vanno approfondite tecniche di recovery adeguate.

4.3 Architettura logica generale

Il sistema è basato su un'architettura Client-Server in cui il Server è una piattaforma con funzione di gestione generale dei dati e delle informazioni di gioco mentre il client è costituito da un sottosistema costituito da uno smartphone e un paio di smart-glasses, di cui ogni utente dovrà disporre per partecipare. Lo smartphone ha la funzione di localizzatore GPS e si occuperà di comunicare col server. Gli smart-glasses invece forniranno l'interfaccia utente dove l'utente visualizzerà ed interagirà con gli elementi grafici che rappresentano le informazioni necessarie allo svolgimento del gioco.

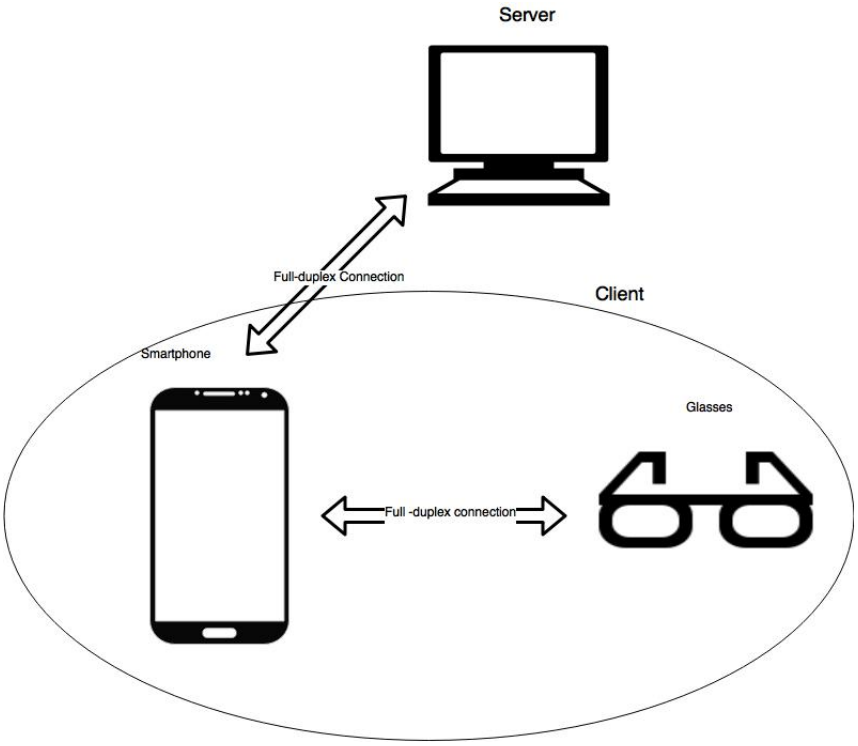

Figura 4.1: Architettura generale del sistema

I software presenti su Client e Server saranno strutturati su tre livelli come segue:

- **User interface layer**, strato applicativo di presentazione grafica delle informazioni e degli elementi di gioco sulla vista degli smart glasses (lato client) e su un applicazione desktop (lato server)
- **Cooperation layer**, strato di cooperazione e di collaborazione in grado di gestire il dominio condiviso tra i vari client e il server e allo stesso tempo capace di supportare le interazioni tra utente e oggetto e tra più utenti
- **Communication layer**, strato in cui viene studiata e realizzata l'infrastruttura di rete e più in generale le tecniche di comunicazione del sistema

4.3.1 Architettura logica del sistema lato Server

L'architettura logica lato Server rispecchia fedelmente la suddivisione in strati dell'intero sistema. Il livello più basso di Comunicazione fornisce le API al

livello di Cooperazione, che a sua volta le fornisce allo strato applicativo. Le API fornite dallo strato di Comunicazione a quello successivo sono divise sostanzialmente in API in invio e API in ricezione. Quelle fornite dallo strato di Cooperazione allo strato di UI sono relative alla cooperazione e alle interazioni tra i vari utenti

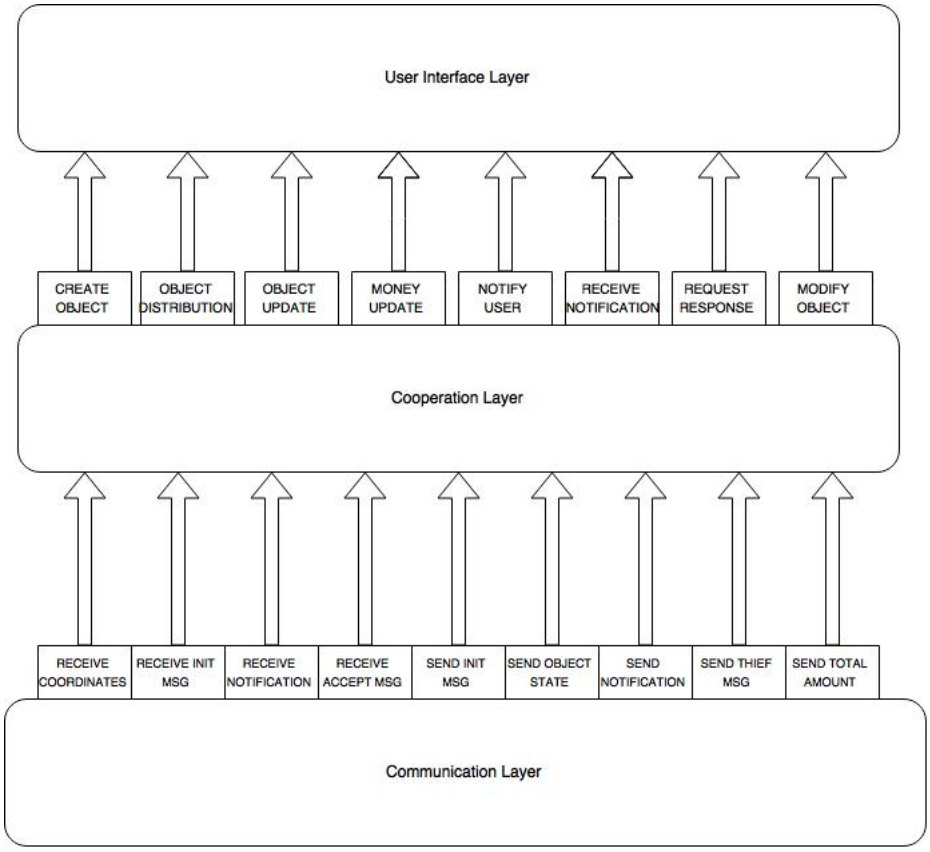

Figura 4.2: Architettura logica del sistema lato server

4.3.2 Architettura logica del sistema lato Client

L'architettura logica lato Client è anch'essa strutturata secondo i tre livelli principali. Le API fornite dallo strato di Comunicazione sono divise in Send API e Receive API e riguardano lo scambio di messaggi, notifiche e cambi di stato degli oggetti. Le API che lo strato di Cooperazione fornisce all'interfaccia sono relative alle dinamiche di gioco e guidano gli eventi sulla vista dell'utente.

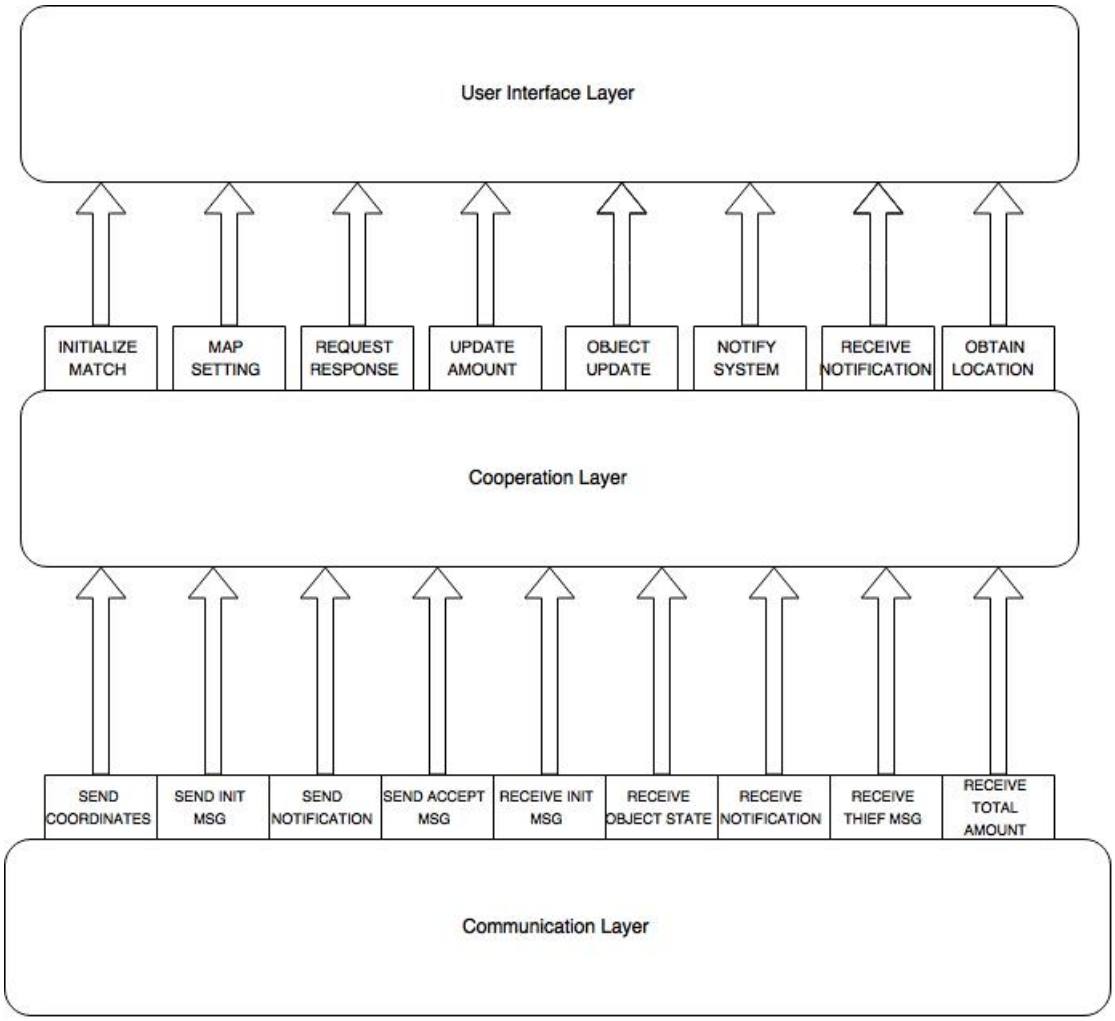

Figura 4.3: Architettura logica del sistema lato client

4.3.3 API ricevute dallo strato inferiore

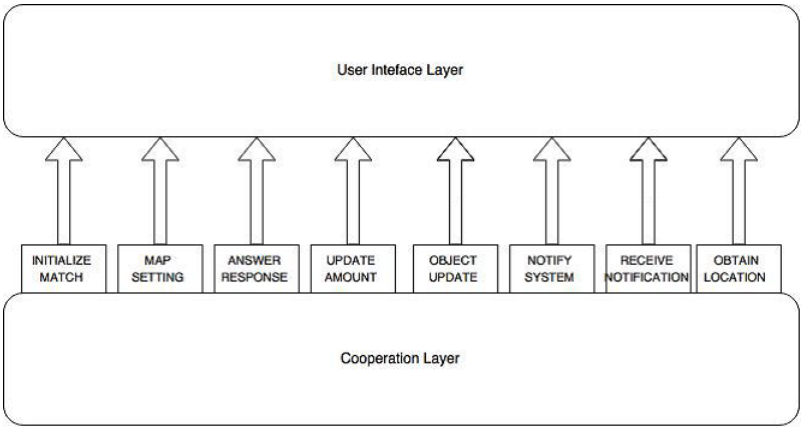

Figura 4.4: API fornite allo strato di applicazione

INITIALIZE MATCH : la richiesta di inizializzazione del gioco viene fatta partire dall'utente nel momento in cui decide di far partire la sessione di gioco, dando l'input al server nel quale verranno inizializzati tutti gli oggetti e la mappa e verrà avviato il gioco

MAP SETTING : l'utente effettua questa chiamata per inizializzare la mappa e tutti gli scrigni presenti in essa

ANSWER RESPONSE : chiamata effettuata nel momento in cui il server ha fatto una richiesta all'utente

UPDATE AMOUNT : chiamata effettuata quando il server notifica l'utente dell'aggiornamento dell'ammontare del denaro posseduto o ancora da raccogliere

OBJECT UPDATE : chiamata effettuata quando è appena stata fatta un'interazione con un oggetto e il server invia un aggiornamento riguardo al suo stato

NOTIFY SYSTEM : chiamata effettuata quando l'utente intende rilasciare una notifica nel sistema

RECEIVE NOTIFICATION : questa chiamata viene effettuata quando il server manda una notifica all'utente

OBTAIN LOCATION : chiamata effettuata ogni qual volta l'utente ha necessità di conoscere la propria posizione

4.4 User Interface Layer

Lo strato di interfaccia utente si occupa di fornire al giocatore una rappresentazione grafica e visiva di ogni elemento e ogni interazione possibile nel gioco. Lo scopo della progettazione di questo sistema è quindi quello di modellare l'interfaccia grafica dell'applicazione sui glasses in modo tale che risponda ad ogni evento in maniera specifica, sfruttando tecniche algoritmiche di disegno di oggetti con il supporto delle librerie OpenGL. Il modulo principale di cui si compone questo strato è il modulo Draw, nel quale sono convogliate le procedure di definizione, rendering e disegno vero e proprio delle figure geometriche.

Questo strato processa le informazioni fornitegli dal substrato di connessione e in base al loro effettivo contenuto mostra all'utente l'opportuna interfaccia grafica. Nella sottosezione seguente verranno affrontate le principali modalità di presentazione delle informazioni all'user, in base ai principali casi d'uso del sistema mostrati nella sezione 3.1 del documento.

4.4.1 Inizializzazione e invio informazioni di gioco

La fase di inizializzazione del gioco e della creazione della mappa è un'operazione svolta dal server che viene passata all'user in quanto meccanismo abilitante per l'inizio del gioco. Come già detto in precedenza le informazioni generali sono condivise su tutti i device del sistema distribuito, quindi l'utente riceve tutte i dettagli utili alla sessione: il totale delle monete raggiungibili, la lista dei treasure chest sparsi per la mappa, l'elenco delle chiavi disponibili. Il diagramma di sequenza che segue mostra lo scenario di inizializzazione tra client e server nel quale viene invocata la creazione della connessione client-server una volta iniziato il gioco.

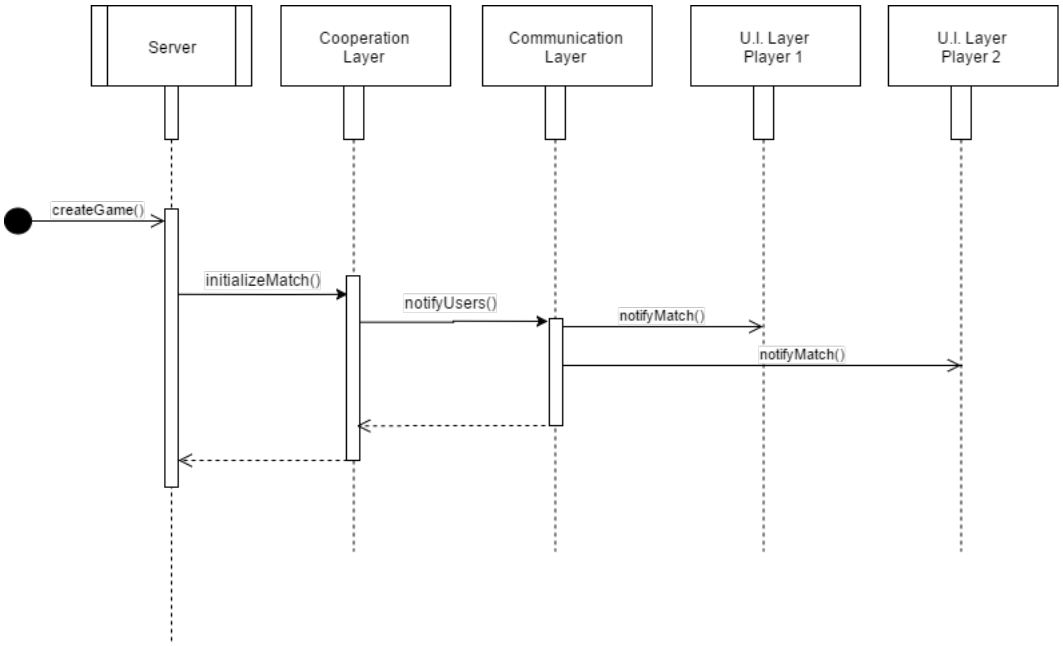

Figura 4.5: Diagramma di sequenza dell'inizializzazione e dell'invio delle informazioni di gioco

4.4.2 Cambio stato

L'evento del cambio stato di un oggetto è molto frequente nel sistema e una delle interazioni principali all'interno del gioco. Ogni oggetto è caratterizzato da due zone circolari situate tramite GPS, il cui centro è costituito dalle coordinate geografiche dell'utente. Il cambio di stato di un oggetto avviene ogni qualvolta un utente entra dentro alla zona più interna delle due, che rappresenta l'oggetto stesso. L'idea più semplice ma al contempo più efficace per rendere graficamente il cambio di stato di un oggetto è quella di mutarne il colore, in base allo stato in cui si trova nel determinato momento. Per rendere visivamente la differenza tra i due cerchi concentrici (ovvero quando si è vicini ad un Treasure Chest e quando effettivamente si è dentro al punto) si è fatto ricorso a due forme geometriche affini ma di dimensioni diverse, lasciando intuire all'user la distanza dal punto. Il diagramma delle interazioni che segue mostra come l'interfaccia di entrambi i giocatori partecipanti al gioco venga aggiornata in base ai controlli sulla posizione del giocatore presenti nel centro operazioni che risiede nel server.

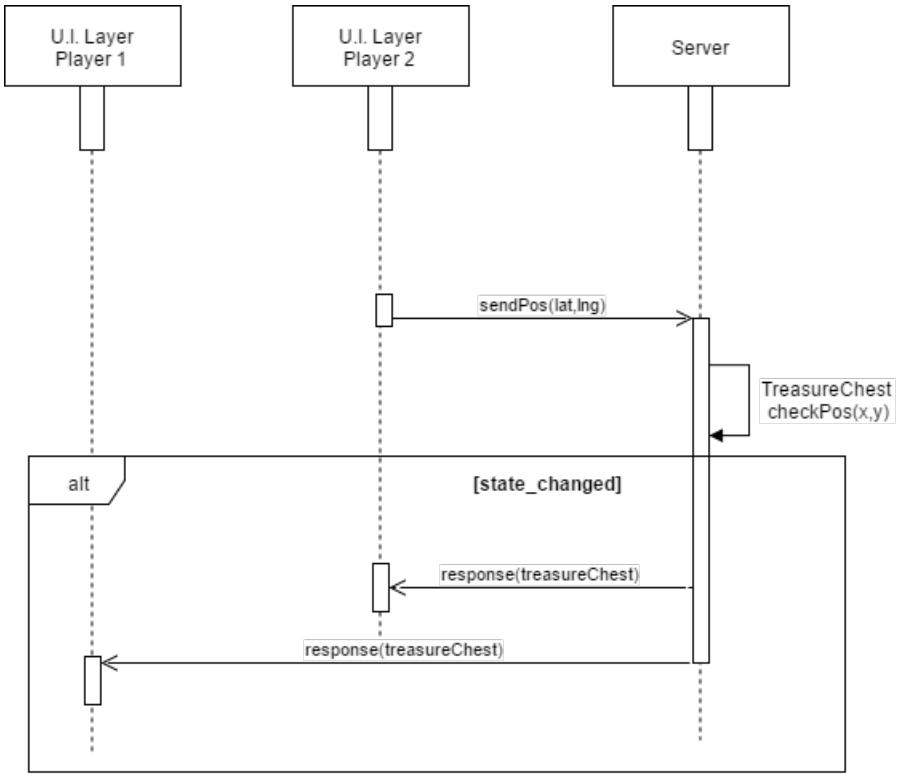

Figura 4.6: Diagramma di sequenza del cambio di stato di un oggetto

4.4.3 Lascia notifica

Il caso d'uso dell'utente che rilascia una notifica si verifica subito dopo che l'entità nemica (il ladro fantasma) lo ha derubato. L'utente ha la possibilità di rilasciare un segnale nella determinata zona in cui è stato derubato per fare in modo che il compagno possa accorgersi – una volta giunto in quel punto – del pericolo imminente costituito dal ladro e quindi valuti eventuali alternative. Intuitivamente si è pensato di raffigurare sui glasses il segnale di pericolo con un triangolo.

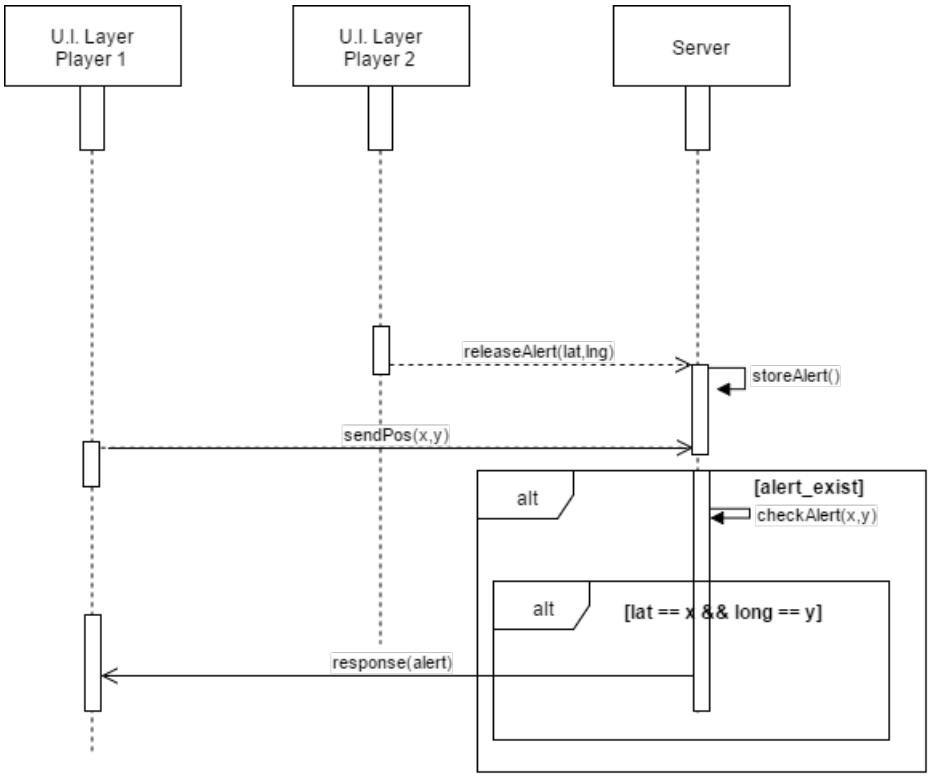

Figura 4.7: Diagramma di sequenza del rilascio di una notifica

4.4.4 Modifica contenuto oggetto

La centrale operativa situata nel server prevede anche la modifica e in particolare la diminuzione dell'ammontare totale delle monete raggiungibili dai giocatori. Nello specifico, verrà effettuata la diminuzione della ricompensa contenuta all'interno degli oggetti secondo intervalli di tempo prefissati. L'utente visualizzerà un segnale di colore rosso che gli mostrerà che la ricompensa è stata ridotta dal server.

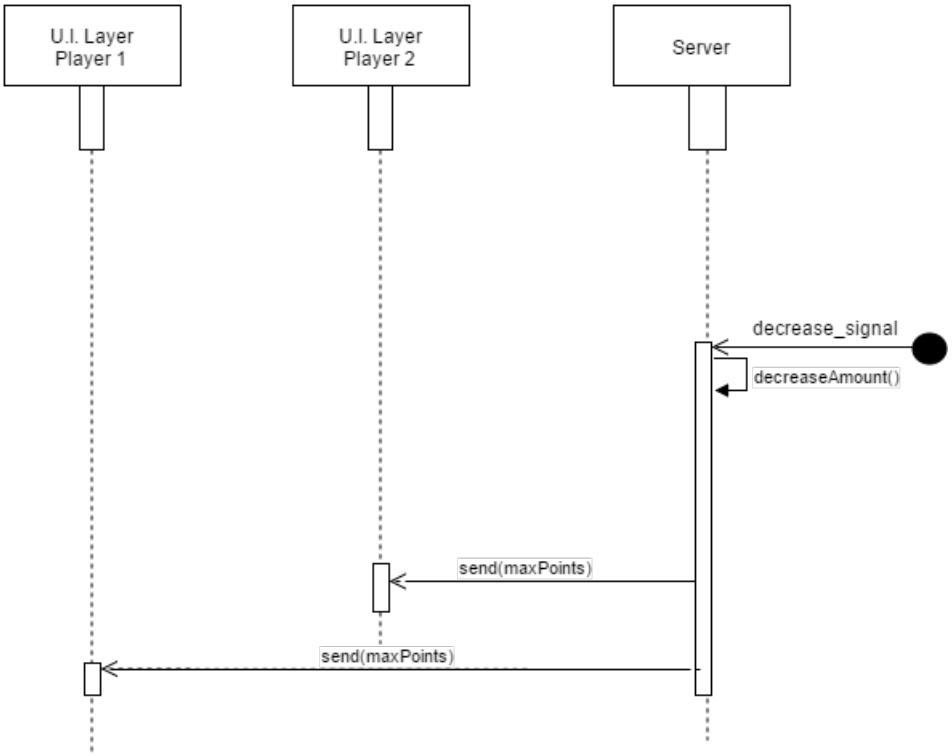

Figura 4.8: Diagramma di sequenza della modifica del contenuto di un oggetto

4.4.5 Ruba ricompensa

Il ladro – situato attraverso coordinate geografiche in uno o più punti della mappa - può rubare parte delle monete all'utente nel caso quest'ultimo transiti nel suo raggio d'azione. Logicamente l'utente deve aver già aperto dei Treasure Chest contenenti monete per poter essere derubato dal ladro. Conseguentemente all'azione del ladro, il centro operazioni notificherà agli utenti l'ammontare di monete rubato al giocatore, come illustrato nel diagramma sottostante.

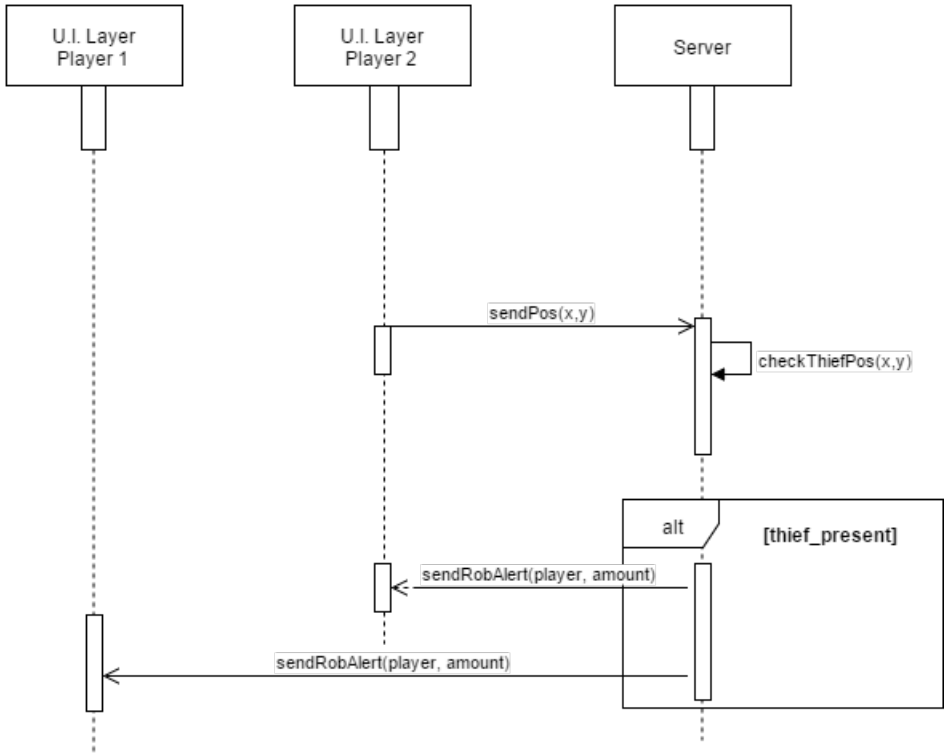

Figura 4.9: Diagramma di sequenza del furto della ricompensa da parte del ladro

4.5 Modulo principale dello strato di interfaccia utente

4.5.1 Modulo Presentation

Lo strato di interfaccia utente si compone essenzialmente di un modulo principale che si occupa della presentazione delle informazioni all'utente tramite l'interfaccia grafica dei glasses. In questo modulo risiede ciò che concerne la definizione e le tecniche di disegno delle principali figure geometriche che troviamo nello strato di interfaccia utente. Il modulo Presentation viene "rifornito" dai dati e dalle istruzioni che l'applicazione a sua volta riceve dal sottostante livello di Connessione.

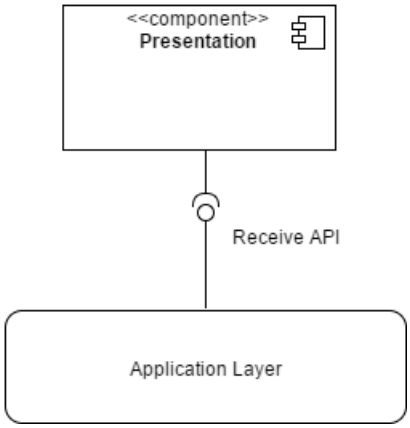

Figura 4.10: Diagramma dei componenti del modulo Presentation in relazione allo strato di U.I.

Il modulo Presentation si forma a sua volta da tre componenti principali, che interagiscono tra loro per garantire l'interfaccia all'utente.

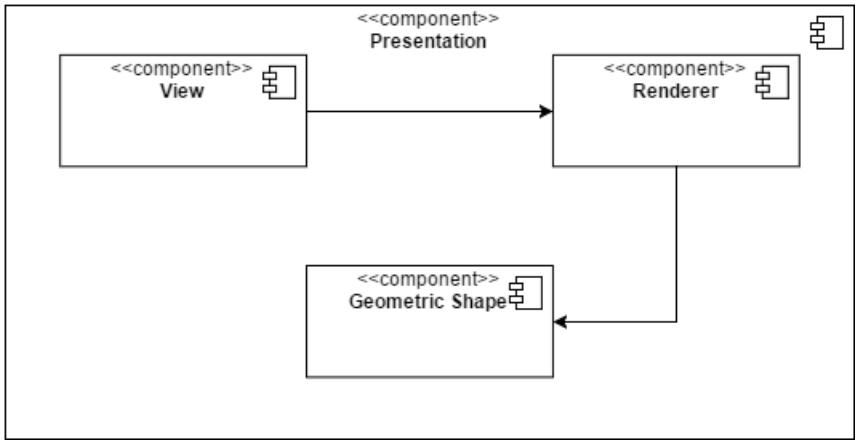

Figura 4.11: Diagramma dei componenti relativo al modulo Presentation

View è il componente che si inserisce nel layout dell'applicazione fornendo la vista all'utente di una superficie. Nel mio caso specifico, è stata utilizzata una `GLSurfaceView` (`http://developer.android.com/reference/android/opengl/GLSurfaceView.html`), view personalizzata che accetta un Renderer modellato dall'user con il compito di *renderizzare*

Renderer è il componente che permette di effettuare le chiamate di libreria OpenGL per effettuare il rendering di un frame. Il `GLSurfaceView.Renderer`

è un'interfaccia che viene generalmente implementata per poter essere passata alla `GLSurfaceView`. I metodi dichiarati nell'interfaccia permettono di modellare il comportamento della view.

- `onDrawFrame(GL10)` permette di disegnare il frame attuale
- `onSurfaceChanged(GL10, int, int)` viene chiamato quando la superficie cambia dimensioni
- `onSurfaceCreated(GL10, EGLConfig)` viene chiamato quando la superficie viene creata o ricreata

Geometric Shape è il componente che rappresenta la definizione della forma geometrica che il Renderer andrà a processare. La libreria a supporto OpenGL permette di definire gli oggetti da disegnare in uno spazio tridimensionale. La definizione degli oggetti si svolge tipicamente inizializzando un array di vertici di `float` per le coordinate, che – per questioni di efficienza – vengono scritte in un `ByteBuffer`, da passare poi alla pipeline di rendering.

Capitolo 5

Sviluppo

5.1 Tecnologie utilizzate

Per lo sviluppo è stato fatto uso di tecnologie nell'ambito mobile e gli smart-glasses Moverio BT-200 prodotti dalla Epson. I Moverio BT-200 smart-glasses sono una delle tecnologie a supporto della realtà aumentata più recenti e valide sul mercato. Il visore binoculare con lenti trasparenti è studiato appositamente per la realtà aumentata e permette di sovrapporre i contenuti (visualizzati da entrambi gli occhi) alla realtà circostante, consentendo di *renderizzare* oggetti 3D in maniera efficace per l'utente. I glasses – una volta indossati – permettono di vedere uno schermo virtuale di 50" con la view di un comune tablet Android 4.0.4 in modalità landscape a una distanza di circa 5 metri. Essendo un display head-mounted lo schermo si muove con il movimento della testa di chi controlla i glasses. Al visore è connessa un'unità di controllo tascabile con trackpad touch (un telecomando connesso ai glasses) che facilita il controllo e permette all'utente di muovere un cursore sullo schermo per intraprendere le azioni. Nel corpo dei glasses sono presenti diversi sensori, tra cui fotocamera, giroscopio, accelerometro e GPS. Il sistema operativo è Android API 15 e i glasses sono programmabili come un qualsiasi device con tale sistema operativo.

5.2 Linguaggi utilizzati

Le fasi di sviluppo e implementazione del progetto sono state svolte utilizzando Java e Android come linguaggi di programmazione.

Lato server è stata sviluppata un'applicazione Java-based tramite l'IDE Eclipse.

Lato client sono state sviluppate due applicazioni Android con il supporto dell'IDE Android Studio, una progettata per smartphone e una per gli smartglasses Moverio BT-200.

5.3 Librerie OpenGL

In fase di sviluppo è risultato molto utile il supporto grafico delle librerie Open Graphics Library (OpenGL) nello specifico la OpenGL ES API. OpenGL è una API grafica multi-piattaforma che specifica un'interfaccia software standard per l'elaborazione di grafica 2D e 3D da parte dell'hardware. OpenGL ES è un genere di OpenGL pensato appositamente per sistemi embedded. Android supporta OpenGL sia tramite il suo framework API sia tramite il Native Development Kit (NDK). Le librerie forniscono anche il supporto per la compressione delle texture. Ci sono due classi fondamentali nel framework (già introdotte nella sezione 4.7) che permettono di creare e manipolare componenti grafici con l'API OpenGL ES:

- `GLSurfaceView` è una View dove è possibile disegnare e manipolare oggetti usando le chiamate di libreria OpenGL e può essere paragonata a una SurfaceView. Per essere utlizzata è necessario aggiungere all'istanza della classe il Renderer

- `GLSurfaceView.Renderer` è un interfaccia che definisce i metodi utili per disegnare graficamente oggetti nella `GLSurfaceView`. Viene lasciato allo sviluppatore il compito di implementare l'interfaccia tramite una specifica classe e di collegarla a `GLSurfaceView` con il metodo `GLSurfaceView.setRenderer()`. I metodi che richiedono l'implementazione sono stati già descritti nella sezione 4.7

5.3.1 Mappare le coordinate degli oggetti disegnati

Uno dei problemi principali nella rappresentazione grafica nei device Android è che i display possono differire notevolmente sia in forma che in dimensioni. OpenGL assume di default un quadrato come sistema di coordinate uniforme ma quando poi disegna le coordinate su uno schermo (tipicamente non quadrato) come se fosse un quadrato perfetto.

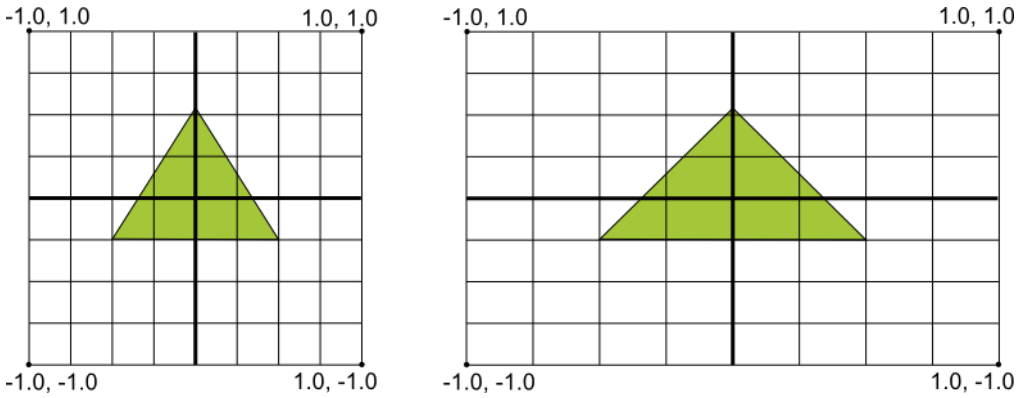

Figura 5.1: Sistema di coordinate OpenGL di default (sinistra) mappato su un generico schermo Android in modalità landscape (destra)

Al fine di applicare la proiezione, si crea una matrice di proiezione e la si applica alla pipeline di rendering di OpenGL. La matrice di proiezione ricalcola le coordinate dell'oggetto da disegnare mappandolo correttamente nello schermo del device. Lo stesso ragionamento viene effettuato per la vista della telecamera, la cui matrice permette la trasformazione che renderizza gli oggetti in base alla specifica posizione degli occhi dell'osservatore. Di seguito vedremo i passaggi effettuati per rendere tale proiezione. In primo luogo bisogna creare una variabile e aggiungere la matrice (nell'esempio uMVPMatrix) agli shader del vertice. Gli shader contengono codice in OpenGL Shading Language (GLSL) che va compilato prima di essere usato nell'ambiente OpenGL ES.

```
package graphics;

import android.opengl.GLES20;
import android.util.Log;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;

public class Circle {

    [..]

    private final String vertexShaderCode =
            "uniform mat4 uMVPMatrix;" +
                    "attribute vec4 vPosition;" +
                    "void main() {" +
                    "  gl_Position = uMVPMatrix * vPosition;" +
                    "}";
```

```
    [..]

}
```

Dopo aver agganciato gli shaders per applicare la proiezione e la camera view, possiamo accedere alla variabile per utilizzare la matrice dal Renderer e creare le matrici di proiezione e vista, modificando i metodi `onSurfaceCreated()` e `onSurfaceChanged()` del `GLSurfaceView.Renderer` basandosi sulla risoluzione del device

```
package graphics;

import android.opengl.GLES20;
import android.opengl.GLSurfaceView;
import android.opengl.Matrix;
import android.os.SystemClock;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    ...

    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        ...
   //access the shader matrix
   muMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram,
       "uMVPMatrix");

   //create a camera view matrix
        Matrix.setLookAtM(mVMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 1.0f,
            0.0f);
    }

    public void onSurfaceChanged(GL10 unused, int width, int height)
        {
   GLES20.glViewport(0, 0, width, height);

     float ratio = (float) width / height;

     // create a projection matrix from device screen geometry
     Matrix.frustumM(mProjMatrix, 0, -ratio, ratio, -1, 1, 3, 7);
```

```
    }

    public static int loadShader(int type, String shaderCode){

        // create a vertex shader type (GLES20.GL_VERTEX_SHADER)
        // or a fragment shader type (GLES20.GL_FRAGMENT_SHADER)
        int shader = GLES20.glCreateShader(type);

        // add the source code to the shader and compile it
        GLES20.glShaderSource(shader, shaderCode);
        GLES20.glCompileShader(shader);

        return shader;
    }
}
```

Per applicare le trasformazioni di proiezione e vista infine bisogna moltiplicare le matrici e impostarle nel vertex shader, modificando onDrawFrame()

```
package graphics;

import android.opengl.GLES20;
import android.opengl.GLSurfaceView;
import android.opengl.Matrix;
import android.os.SystemClock;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    ...

    public void onDrawFrame(GL10 unused) {
   ...

        // Combine the projection and camera view matrices
        Matrix.multiplyMM(mMVPMatrix, 0, mProjMatrix, 0, mVMatrix, 0);

        // Apply the combined projection and camera view
           transformations
        GLES20.glUniformMatrix4fv(muMVPMatrixHandle, 1, false,
           mMVPMatrix, 0);

        // Draw objects
```

```
    }
}
```

5.4 Scelte implementative

Figure geometriche Il primo step nel disegno su una `GLSurfaceView` è la definizione della forma da disegnare. Viene quindi definito un array di float che rappresenta le coordinate (in tre dimensioni) della figura. Per ottenere più efficienza queste coordinate vengono scritte su un `ByteBuffer`, che verrà passato alla pipeline. E' importante sapere inoltre che OpenGL ES assume di default come centro della GLSurfaceView il punto [0,0,0] (X,Y,Z) e che l'ordine delle coordinate va espresso in senso antiorario. Successivamente, vengono specificati alcuni dettagli utili alla pipeline di rendering. Vanno definite parti di codice per `Vertex Shader` e il `Fragment Shader`, adibiti rispettivamente al rendering di vertici e colori. Queste parti di codice sono in OpenGL Shading Language (GLSL) e devono essere compilate prima di essere usate, ragion per cui viene creato il metodo `loadShader()` nella classe del Renderer. A questo punto si può sviluppare il metodo `draw()` nel quale vengono impostati posizione e colore a `Vertex Shader` e `Fragment Shader`. Nel codice sottostante viene mostrato il codice per la classe `Triangle`

```
package graphics;

import android.opengl.GLES20;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;


public class Triangle {

    private final String vertexShaderCode =
            "uniform mat4 uMVPMatrix;" +
            "attribute vec4 vPosition;" +
                    "void main() {" +
                    "   gl_Position = uMVPMatrix * vPosition;" +
                    "}";
    private final String fragmentShaderCode =
            "precision mediump float;" +
                    "uniform vec4 vColor;" +
```

```
                "void main() {" +
                "  gl_FragColor = vColor;" +
                "}";
    private int mMVPMatrixHandle;
    private FloatBuffer vertexBuffer;

    // number of coordinates per vertex in this array
    static final int COORDS_PER_VERTEX = 3;
    static float triangleCoords[] = { // in counterclockwise order:
            0.0f, 0.622008459f, 0.0f, // top
            -0.5f, -0.311004243f, 0.0f, // bottom left
            0.5f, -0.311004243f, 0.0f // bottom right
    };

    private final int mProgram;

    //r,g,b and aplha (opacity) values
    float color[] = { 1.0f, 0.0f, 0.0f, 1.0f };

    public Triangle() {
        ByteBuffer bb = ByteBuffer.allocateDirect(
                triangleCoords.length * 4);
        bb.order(ByteOrder.nativeOrder());

        int vertexShader =
            MyGLRenderer.loadShader(GLES20.GL_VERTEX_SHADER,
                vertexShaderCode);
        int fragmentShader =
            MyGLRenderer.loadShader(GLES20.GL_FRAGMENT_SHADER,
                fragmentShaderCode);

        // create empty OpenGL ES Program
        mProgram = GLES20.glCreateProgram();

        // add the vertex shader to program
        GLES20.glAttachShader(mProgram, vertexShader);

        // add the fragment shader to program
        GLES20.glAttachShader(mProgram, fragmentShader);

        GLES20.glLinkProgram(mProgram);

        // create a floating point buffer from the ByteBuffer
        vertexBuffer = bb.asFloatBuffer();
```

```
        // add coordinates to the FloatBuffer
        vertexBuffer.put(triangleCoords);
        // set buffer to read the first coordinate
        vertexBuffer.position(0);
    }

    private int mPositionHandle;
    private int mColorHandle;

    private final int vertexCount = triangleCoords.length /
        COORDS_PER_VERTEX;
    private final int vertexStride = COORDS_PER_VERTEX * 4; // 4
        bytes per vertex

    public void draw(float[] mvpMatrix) {
        GLES20.glUseProgram(mProgram);

        mPositionHandle = GLES20.glGetAttribLocation(mProgram,
            "vPosition");

        GLES20.glEnableVertexAttribArray(mPositionHandle);

        mMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram,
            "uMVPMatrix");

        // Pass the projection and view transformation to the shader
        GLES20.glUniformMatrix4fv(mMVPMatrixHandle, 1, false,
            mvpMatrix, 0);

        GLES20.glVertexAttribPointer(mPositionHandle,
            COORDS_PER_VERTEX,
                GLES20.GL_FLOAT, false,
                vertexStride, vertexBuffer);

        mColorHandle = GLES20.glGetUniformLocation(mProgram,
            "vColor");

        // Set color
        GLES20.glUniform4fv(mColorHandle, 1, color, 0);

        // Draw the triangle
        GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

        GLES20.glDisableVertexAttribArray(mPositionHandle);
```

```
    }
}
```

Renderer `MyGLRenderer` è la classe a cui è affidato il rendering e implementa l'interfaccia `GLSurfaceView.MyGLRenderer`. Ha la funzione di controllare ciò che viene disegnato sulla `GLSurfaceView` a cui viene associato. I metodi implementati nella classe permettono di inizializzare le figure, disegnarle in base al parametro `shape` che determina la forma da disegnare e di gestire il cambio di superficie, ad esempio quando lo schermo passa dalla modalità portrait a quella landscape.

```
package graphics;

import android.opengl.GLES20;
import android.opengl.GLSurfaceView;
import android.opengl.Matrix;
import android.os.SystemClock;
import android.util.Log;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;


public class MyGLRenderer implements GLSurfaceView.Renderer {

    public int shape = 0;
    private final float[] mMVPMatrix = new float[16];
    private final float[] mProjectionMatrix = new float[16];
    private final float[] mViewMatrix = new float[16];

    private Circle mCircle;
    private Triangle mTriangle;
    private Square mSquare;

    public MyGLRenderer(int shape){
        this.shape = shape;
    }

    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        // Set the background frame color
        GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

        mTriangle = new Triangle();
```

```
        mCircle = new Circle();
        mSquare = new Square();
    }


    public void onDrawFrame(GL10 gl) {
        // Redraw background color
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);

        Matrix.setLookAtM(mViewMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f,
            1.0f, 0.0f);
        Matrix.multiplyMM(mMVPMatrix, 0, mProjectionMatrix, 0,
            mViewMatrix, 0);

        if(shape == 1){
            mTriangle.draw(mMVPMatrix);
        } else if(shape == 2){
            mCircle.draw(mMVPMatrix);
        } else if(shape == 3){
            mSquare.draw(mMVPMatrix);
        }

    }

    public void onSurfaceChanged(GL10 unused, int width, int height)
        {
        GLES20.glViewport(0, 0, width, height);
        float ratio = (float) width / height;
        Matrix.frustumM(mProjectionMatrix, 0, -ratio, ratio, -1, 1,
            3, 7);
    }

    public static int loadShader(int type, String shaderCode){

        // create a vertex shader type or a fragment shader type
        int shader = GLES20.glCreateShader(type);

        // add the source code to the shader and compile it
        GLES20.glShaderSource(shader, shaderCode);
        GLES20.glCompileShader(shader);

        return shader;
    }
```

```
}
```

View `MyGLSurfaceView` è la vista specializzata su cui il Renderer può disegnare gli elementi grafici. Il metodo `setRenderer()` verrà chiamato su un'istanza della classe nella `MainActivity` poiché deve essere dinamico in base ai diversi scenari dell'applicazione.

```
package graphics;

import android.content.Context;
import android.opengl.GLSurfaceView;
import android.util.AttributeSet;


public class MyGLSurfaceView extends GLSurfaceView {

    public MyGLSurfaceView(Context context){
        super(context);
        setEGLContextClientVersion(2);
    }

    public MyGLSurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(context);
        setEGLContextClientVersion(2);
    }

}
```

Activity Nella `MainActivity` viene creata un'istanza di `MyGLSurfaceView` e viene inizialmente chiamato il metodo `setRenderer(0)` che imposta un renderer vuoto sulla superficie. Il metodo `replaceView()` si occupa di sostituire il Renderer attualmente impostato alla `MyGLSurfaceView`. Prima viene chiamata `onPause()` sulla `MainView`, poi viene fatto ricorso a un Runnable in cui si rimuove la `MainView` dal Layout e ne si crea una nuova impostando il Renderer in base alla figura da disegnare con `setRenderer(new MyGLRenderer(shape))`. Infine l'`Handler` si occupa di chiamare la `postDelayed()` per permettere alla `MyGLSurfaceView` di cancellare file e processi interni, chiamata necessaria perché non è possibile chiamare due volte `setRenderer()` su una `MyGLSurfaceView`

```
package activity;
```

```
import android.opengl.GLSurfaceView;
import com.example.matteoaldini.bbcmoverio.model.Position;
import com.example.matteoaldini.bbcmoverio.model.TreasureChest;


public class MainActivity extends Activity {

    private MyGLSurfaceView mainView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.mainView = (MyGLSurfaceView)
            findViewById(R.id.surfaceView);
        this.mainView.setRenderer(new MyGLRenderer(0));


    private void positionReceived(Position obj) {
        ...

        for(TreasureChest t: this.match.getTreasures()){

            if(p.getDistance(obj)<0.02){
                ...
            replaceSurfaceView(3);
            }
        }

    }

    ...

    private void replaceSurfaceView(int shape){
        final GLSurfaceView.Renderer newRender = new
            MyGLRenderer(shape);
        final RelativeLayout rl = (RelativeLayout)
            findViewById(R.id.main);
        mainView.onPause();

        Handler handler = new Handler();
        class RefreshRunnable implements Runnable{
```

```
            public RefreshRunnable(){

            }

            public void run(){
                rl.removeView(findViewById(R.id.surfaceView));

                MyGLSurfaceView view = new
                    MyGLSurfaceView(getApplication());
                view.setId(R.id.surfaceView);
                view.setRenderer(newRender);

                ViewGroup.LayoutParams
                    layoutParams=mainView.getLayoutParams();
                layoutParams.width=300;
                layoutParams.height=300;
                view.setLayoutParams(layoutParams);

                rl.addView(view);

                mainView = (MyGLSurfaceView)
                    findViewById(R.id.surfaceView);
            }
        }

        RefreshRunnable r = new RefreshRunnable();
        handler.postDelayed(r, 500);

    }
}
```

Capitolo 6

Valutazioni

Il software è stato sviluppato seguendo le fasi di analisi, progettazione e prototipazione descritte nei capitoli precedenti. Le funzionalità implementate in base alle suddette fasi sono state infatti incluse nel software in maniera soddisfacente e l'applicazione, pur trattandosi di un prototipo, è conforme con quanto prefissato nel caso di studio. Esaminare approfonditamente ogni specifico scenario presente all'interno dell'applicazione ci ha permesso di adempiere ai requisiti presi in considerazione nella fase di analisi e di descrivere appieno il dominio applicativo. Le problematiche trattate nel capitolo relativo alla progettazione sono state affrontate e risolte in maniera considerevolmente accettabile.

6.0.1 Risoluzione delle problematiche

Coerenza dei dati Come accennato, in un sistema condiviso come quello trattato è possibile riscontrare problemi di coerenza delle informazioni. Al fine di poter tranquillamente operare sulle stesse informazioni è dunque necessario che ogni parte sia certa che il dominio su cui interagisce sia lo stesso per tutte le altre parti. La soluzione pensata per ovviare al problema è quella di rendere il server l'elemento garante della coerenza. Ogni variazione del dominio viene infatti registrata sul server prima di essere notificata ai client, rendendo il software solido, robusto e immune anche negli scenari più critici.

Latenza La latenza non ha rappresentato un grosso problema nel nostro caso di studio in quanto la scelta di improntare la comunicazione su uno scambio di messaggio semplice e ridotto ha permesso di affidare allo strato applicativo il compito di interpretare e presentare graficamente le informazioni che era necessario condividere nel sistema distribuito. E' chiaro che questo aspetto è prettamente vincolato alle tecnologie di cui si fa uso. In uno scenario

ampio, è possibile incorrere in zone dove il segnale di rete mobile (3G) è debole o addirittura assente, prospettiva per certi versi limitante per un sistema condiviso.

Frequenza dell'invio della posizione Si era inoltre detto che – trattandosi il nostro di un location-based game – sarebbe stato importante imbastire un sistema di localizzazione robusto e ben calibrato in termini di frequenza. Ebbene, l'invio costante e continuo della propria posizione da parte dell'user in base a un timer scaduto o a una certa quantità di spazio percorsa si è rivelata essere una buona scelta. La limitazione principale in tutto ciò è rappresentata dal fatto che le tecnologie a nostra disposizione oggigiorno sono ancora in fase di sviluppo, pur essendo notevolmente migliorate negli ultimi anni. L'utilizzo dell' Assisted-GPS rende la localizzazione ancor più precisa, limitando l'errore circa dai 3 agli 8 metri. Tuttavia nei cosiddetti canyon urbani, ovvero in strade strette con palazzi molto alti, è frequente riscontrare perdite di segnale in termini di stabilità e accuratezza. Mediante tecniche di triangolazione con sensori posti – ad esempio – nei punti d'interesse predefiniti, si potrebbe giungere a un grado di precisione più alto, ma il sistema ne perderebbe in dinamicità e indipendenza, in quanto può essere svolto ovunque e non solo in zone determinate a priori.

Disconnessioni La gestione delle disconnessioni è stata affrontata adottando la soluzione che permette a un utente che si disconnette di riottenere le informazioni aggiornate una volta riconnesso, eliminando così possibili incoerenze di dati. Uno scenario in cui la disconnessione dell'utente avviene durante una richiesta di cooperazione rappresenta un problema in quanto l'utente potrebbe rimanere prolungatamente in attesa della risposta senza poi effettivamente averla. La risoluzione a tale problema è stata trovata nell'invio di un segnale visivo e testuale sui glasses di ogni giocatore che lo notifica della disconnessione del determinato utente, in modo tale che ogni giocatore possa accorgersi di quale compagno è disponibile e quale no in ogni momento. La soluzione adottata è comunque limitata perché l'utente durante tutto il lasso di tempo in cui è disconnesso è escluso dal gioco, ma praticamente ogni opzione per ovviare a tale problema in un sistema distribuito del genere presenterà qualche limite

6.0.2 Struttura dell'applicazione

Divisione in strati La schematizzazione dell'architettura logica dell'applicazione ha portato ad individuare tre livelli principali su cui si sviluppa il

sistema: communication layer, cooperation layer e user interface layer. Il nostro caso di studio permette di modellare i meccanismi di gioco principalmente sui primi due livelli, lasciando allo strato di interfaccia utente il compito di mostrare al giocatore le informazioni e i segnali utili allo svolgimento del gioco, e lato server di inizializzare il gioco e intervenire in determinati casi. Il ruolo dei primi due livelli può essere generalizzato e astraendo dal caso di studio si può pensare di costituire una sorta di middleware che possa fornire API e strumenti a supporto di applicazioni simili alla nostra. In tale modo si permetterebbe di offrire due livelli a supporto di un livello applicativo in cui verrebbero implementate le funzionalità della specifica applicazione. Le API fornite allo strato di applicazioni sarebbero API orientate alla gestione di entità computazionali aumentate e personalizzabili in base alle esigenze.

Strato di User Interface Il nostro caso di studio mi ha permesso di sviluppare e dare una forma visuale e grafica allo strato di applicazione. È utile e interessante notare come il lavoro svolto non sia una forma prettamente autentica di realtà aumentata, ma si collochi in una posizione intermedia tra l'Augmented Reality e la cosiddetta Mixed Reality, rispecchiando i propositi del sistema in generale. Non è infatti la visualizzazione di oggetti e modelli 3D aumentati lo scopo di questa presentazione grafica, bensì il concetto di disegnare oggetti e mostrare informazioni anche in maniera testuale sullo schermo dei glasses tramite specifiche istruzioni derivanti dalla localizzazione. È infatti l'aspetto location-aware del sistema che ci ha permesso di dare forma a una vista che si pone come obiettivo quello di essere intuitiva e lineare, trasformandosi dinamicamente in base alle informazioni che giungono dagli strati inferiori di cooperazione e comunicazione.

Questo sistema da noi approfondito è in definitiva un'ottima estensione al concetto di realtà aumentata che ha l'ambizione di andare oltre ai confini dello specifico caso di studio e, tramite apposite migliorie e investigazioni su aspetti più avanzati dell'applicazione, può rappresentare un'ottima base per un progetto di sistema situato e distribuito, a supporto della cooperazione e basato su realtà aumentata

Conclusioni

Le attività svolte nel contesto di questo progetto sono state stimolanti e molto interessanti, portando a risultati soddisfacenti in base agli obiettivi prefissati in fase di ideazione e modellazione. Le tematiche affrontate nel complesso dell'applicazione hanno svariato su diverse branche della tecnologia informatica e hanno permesso a me e ai miei compagni di mettere in campo parte delle conoscenze fin qui acquisite nel percorso di studi e di ampliarle specializzandosi in determinati argomenti. Penso alle tecnologie embedded, fondamentali per un sistema basato su realtà aumentata, ma anche alla programmazione di reti che ci ha permesso di imbastire la connessione tra user e server e alle nozioni di Computer Graphics preziose per dare forma allo strato di interfaccia utente. La collaborazione con un team di colleghi nelle diverse fasi del sistema è stata sicuramente un arricchimento del mio bagaglio personale, permettendomi di confrontarmi spesso con i miei compagni e di trovare soluzioni migliori in base anche a confronti e discussioni propositive. La divisione in strati diversi del sistema è stata a mio modo di vedere una scelta azzeccata che ha permesso di rendere il sistema modulare e riusabile.

In ultimo, posso ritenermi molto appagato dal fatto di aver esplorato un campo dell'informatica nel quale le mie conoscenze pregresse erano molto superficiali e generiche e che sono state migliorate grazie agli approfondimenti su concetti come augmented reality, location-based e pervasive games e distribuited systems. Il progetto sviluppato ha quindi soddisfatto le aspettative e può rappresentare una base per eventuali accrescimenti futuri ed applicazioni pratiche diverse, spiegate nella sezione seguente

Sviluppi Futuri

Gli sviluppi futuri di questa applicazione e del mio caso di studio in particolare riguardano lo strato di User Interface che potrebbe essere raffinato inserendo rappresentazioni grafiche basate sul disegno di modelli tridimensionali avanzati in base alla posizione geografica e alle interazioni dell'utente con il sistema.

L'integrazione di framework come Wikitude, Layar o ARToolKit sarebbe stato possibile ma la possibilità di avere strumenti di sviluppo personalizzati e indirizzati più alla localizzazione che al riconoscimento di markers sarebbe l'ideale per un sistema come il nostro che vuole mantenere buoni parametri di usabilità e scalabilità.

Per quanto riguarda i substrati di cooperazione e connessione a supporto dello strato applicativo, sarebbe molto interessante generalizzare le funzionalità e le API sviluppate fino a questo momento arrivando ad avere un vero e proprio middleware a supporto di sistemi AR collaborativi. Questo può significare sia un ampliamento a livello di funzionalità dei toolkit a supporto della realtà aumentata, sia l'uso del suddetto middleware in applicazioni concrete ed avanzate di realtà aumentata. Ambiti in cui sistemi come il nostro possono trovare spazio possono essere il gaming, il learning e i soccorsi, opzioni che rendono valido e futuribile questo studio.

Ringraziamenti

Voglio innanzitutto ringraziare il prof. Alessandro Ricci e i correlatori Angelo Croatti e Pietro Brunetti, per la loro grande disponibilità e per i loro preziosissimi consigli. Ringrazio poi tutti i miei amici, la mia fidanzata Martina per avermi supportato nelle difficoltà e i compagni con cui ho condiviso questa esperienza, in particolare due ragazzi eccezionali come Matteo e Filippo con cui ho svolto in team ogni progetto in questi anni, compresa questa tesi. Ringrazio inoltre mia sorella Adria Nora, i miei zii e i miei nonni e tutti gli altri miei familiari. Ringrazio infine i miei genitori, Giorgio e Federica, per avermi sempre sostenuto ed aiutato in questo percorso.

Bibliografia

[1] D.W.F. van Krevelen, R. Poelman *A Survey of Augmented Reality Technologies, Applications and Limitations*, The International Journal of Virtual Reality, 2010.

[2] Alan B. Craig *Understanding Augmented Reality. Concepts and Applications*, Elsevier, 2013.

[3] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, Misa Ivkovic *Augmented reality technologies, systems and applications*, Springer Science+Business Media, LLC, 2010.

[4] Alessandro Ricci, Luca Tummolini, Michele Piunti, Olivier Boissier, Cristiano Castelfranchi *Mirror Worlds as Agent Societies Situated in Mixed Reality Environments*, 2011.

[5] Z. Szalavri, D. Schmalstieg, A. Fuhrmann, M. Gervautz *Studierstube. An Environment for Collaboration in Augmented Reality*, 1998.

Elenco delle figure