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 AL LIVELLO DELLE COMUNICAZIONI

*Relazione finale in*
PROGRAMMAZIONE DI SISTEMI EMBEDDED

*Relatore*
Prof. ALESSANDRO RICCI

*Presentata da*
MATTEO ALDINI

*Co-relatori*
Ing. PIETRO BRUNETTI
Ing. ANGELO CROATTI

Prima Sessione di Laurea
Anno Accademico 2014 – 2015

# PAROLE CHIAVE

augmented reality

distributed systems

location awareness

cooperation

communication

*"When you think of any aspect of life or work, augmented reality is completely going to change how we do it." – Ori Inbar*

# Indice

# Introduzione

Il grande sviluppo a cui stiamo assistendo negli ultimi anni nell'ambito delle tecnologie mobile e del wearable computing apre le porte a scenari innovativi e interessanti per quanto riguarda sistemi distribuiti collaborativi. Le persone possono sempre più facilmente cooperare e scambiarsi informazioni grazie a queste nuove tecnologie e si può pensare allo sviluppo di sistemi che permettano forme molto avanzate di collaborazione facendo leva sull'aspetto hands-free, ovvero sulla possibilità di utilizzare dispositivi che liberino le mani, come i moderni smart-glasses.

Per lo sviluppo di tali sistemi è necessario però studiare nuove tecniche e architetture, in quanto gli strumenti ad oggi a disposizione a supporto della realtà aumentata non sembrano essere del tutto adeguati per tale scopo. Infatti piattaforme come Wikitude o Layar, seppure offrano potenti tecniche di riconoscimento di markers e immagini e di rendering, non offrono quella dinamicità fondamentale per un sistema distribuito collaborativo.

Questo scritto ha lo scopo di esplorare questi aspetti mediante l'ideazione, l'analisi, la progettazione e la prototipazione di un semplice caso di studio ispirato a sistemi collaborativi distribuiti basati su realtà aumentata. In particolare in questo lavoro si porrà l'attenzione sul livello delle comunicazioni e delle infrastrutture di rete. Tale caso di studio offrirà spunti interessanti e innovativi riguardo molteplici aspetti.

Innanzitutto si porrà l'attenzione sull'aspetto "hands-free" delle tecnologie utilizzate, ovvero sulla possibilità di cooperare e muoversi in un ambiente "aumentato" senza l'ausilio delle mani, grazie a strumenti come i moderni smart-glasses.

In secondo luogo, il sistema richiederà la progettazione di componenti in grado di gestire dati totalmente distribuiti tra vari utenti, per permetterne la collaborazione e la condivisione di informazioni. Tale aspetto fa emergere la problematica fondamentale del'applicazione, ovvero la necessità di gestire informazioni totalmente condivise e necessariamente replicate in maniera invariata per ogni utente. Proprio questo punto differenzia il sistema che si andrà a realizzare dai più classici sistemi basati su realtà aumentata.

Infatti tipicamente applicazioni AR (Augmented Reality) fondono l'utilizzo di markers e pois a avanzate tecniche di rendering per creare un livello "aumentato" al di sopra di quello reale e tangibile. Framework come Wikitude e Layar forniscono potenti strumenti per questo tipo di realtà aumentata. Al contrario, nella nostra applicazione non si pone il focus su riconoscimento di markers e rendering, bensì sulla condivisione di dati riguardanti oggetti aumentati e dinamici, il cui stato e valore possono variare significativamente in seguito alle interazioni con i vari utenti e sulla collaborazione tra utenti. Sarà così il concetto stesso di realtà aumentata ad essere rivisitato. Difatti l'obiettivo è quello di fornire esperienze significative di realtà aumentata anche attraverso l'utilizzo di semplici messaggi testuali sullo schermo di smart-glasses, per esempio mostrando la presenza di un determinato oggetto attraverso una semplice notifica. L'attributo che renderà "aumentato" un qualche oggetto (non necessariamente presente fisicamente, in quanto potrà essere un'entità totalmente computazionale) sarà proprio la sua capacità di percepire le interazioni con esso e di cambiare il proprio stato.

Tutto questo apre la strada a una realtà aumentata "dinamica" e più scalabile, ovvero applicabile su più fronti senza dover necessariamente dover dipendere dall'utilizzo di tecniche di riconoscimento delle immagini.

Lo scritto sarà strutturato come segue.

In una prima parte si andrà a fare un quadro generale di quello che è lo stato attuale dell'arte relativo alla realtà aumentata, ponendo l'accento sugli aspetti più legati alla cooperazione.

Seguirà l'esplorazione di framework a supporto della realtà aumentata quali Wikitude, Metaio e Layar.

Successivamente verrà introdotto il caso di studio del sistema che è stato realizzato, ovvero un AR location-based game centrato sulla cooperazione tra utenti e sull'interazione con oggetti aumentati situati.

Si passerà quindi all'analisi dei requisiti e alla modellazione del dominio dell'applicazione. Utilizzando diagrammi di casi d'uso e di sequenza verranno sviscerati tutti gli scenari e le possibili interazioni tra utenti e oggetti aumentati.

Dopo una definizione dell'architettura logica del sistema, si descriverà la fase di progettazione di dettaglio dell'applicazione. La mia parte di dettaglio descriverà la progettazione dell'infrastruttura di rete e, più in generale, del livello di comunicazione del sistema.

Successivamente si descriveranno le tecnologie utilizzate e lo sviluppo e la prototipazione delle parti più significative dell'applicazione.

Infine si farà una valutazione sul lavoro svolto con anche uno sguardo ai possibili lavori futuri.

# Capitolo 1

# Stato dell'Arte

## 1.1 Introduzione e cenni storici

La realtà aumentata (augmented reality, AR) è una vista attraverso la quale il mondo reale viene ampliato o "aumentato" attraverso l'aggiunta di informazioni computazionali. Chi si trova in un mondo "aumentato" percepisce oltre agli oggetti fisici anche tutti gli oggetti presenti solamente in maniera virtuale intorno a sè. Questo tipo di percezione da forma alla cosiddetta "mixed reality", ovvero una realtà non più solamente fisica e tangibile, ma anche influenzata dalle informazioni dell'ambiente virtuale. La realtà aumentata è effettivamente una parte della mixed reality. Infatti, mentre l'ambiente virtuale, ovvero l'insieme di tutte le entità computazionali da aggiungere al mondo fisico, e la "virtualità aumentata", in cui gli oggetti reali si mescolano con quelli fisici, sostituiscono l'ambiente circostante con una sua controparte virtuale, la realtà aumentata fornisce solamente virtualizzazione locale. [1]

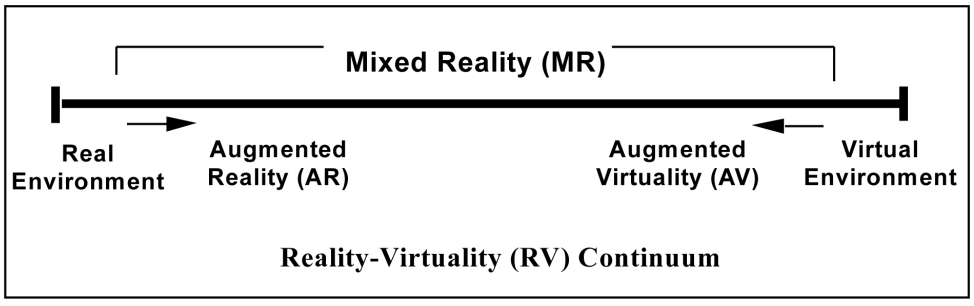


Figura 1.1: Reality-Virtuality continuum

Lo scopo della realtà aumentata è quello di rivoluzionare radicalmente il modo in cui le persone percepiscono la realtà. Le informazioni su un oggetto o

su un luogo potrebbero essere direttamente accessibili con il solo riconoscimento dell'oggetto da parte del supporto per la realtà aumentata oppure per il semplice fatto che ci si trova in quel determinato luogo. Le possibili applicazioni sono quindi molteplici: dalla cooperazione, al gaming, al learning...

La realtà aumentata nasce nel 1966 con l'invenzione da parte del ricercatore Ivan Sutherland del primo prototipo a supporto di tale tecnologia. Negli anni '70 e '80 la ricerca in questo campo porta allo sviluppo nel 1975 da parte di Myron Krueger di Videoplace, una stanza che per la prima volta permetteva di interagire con oggetti virtuali. [2] Nel contempo il grande sviluppo tecnologico porta alla creazione di dispositivi come il Sony Walkman e gli orologi digitali, che aprono la strada al cosiddetto "wearable computing".

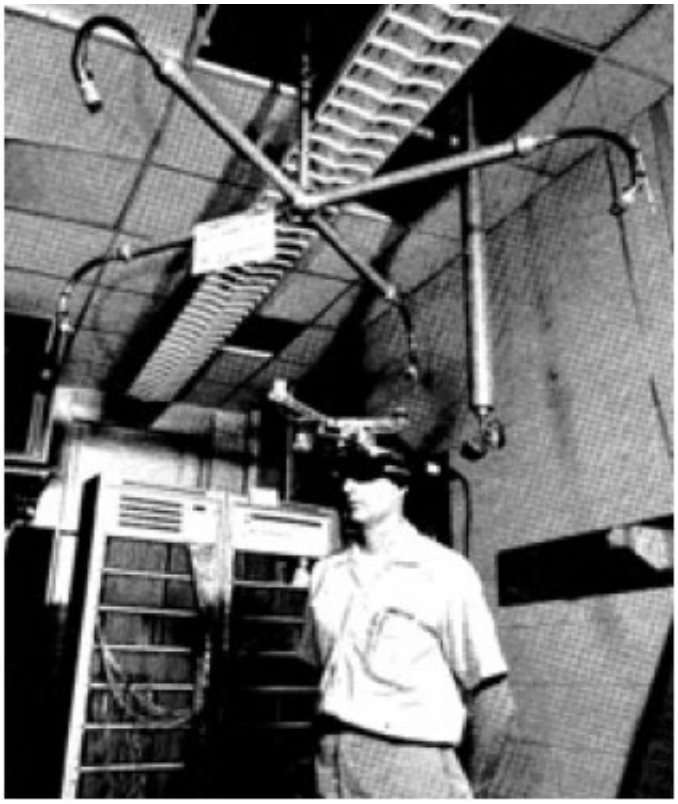

Figura 1.2: Primo prototipo di smart glasses creato da Sutherland

Il termine "realtà aumentata" viene però coniato solo negli anni '90 da Caudell e Mizell, scienziati che al tempo lavoravano su un sistema in grado di offrire supporto ai lavoratori di una catena di montaggio di cavi elettrici. Negli anni lo sviluppo delle tecnologie mobili porta lo sviluppo dell'AR in tale direzione.

Nonostante la scarsa potenza delle nuove tecnologie Feiner sviluppa un prototipo di sistema di realtà aumentata che proietta informazioni e tour in 3D in base ai monumenti che l'utente sta vedendo. Nel frattempo porta avanti insieme a Blair MacIntyre e Doree Seligmann la prima pubblicazione su un sistema AR denominato KARMA. A partire dagli anni '90 la realtà aumentata diviene un campo ben distinto di ricerca e vengono organizzate molteplici conferenze sul tema come l'International Workshop and Symposium on Augmented Reality, l'International Symposium on Mixed Reality, il Designing Augmented Reality Environments workshop. Nascono anche le prime organizzazioni interamente dedite alla realtà aumentata e i primi kit per sviluppatori come ARToolKit, ancora oggi molto diffuso[1].

Il continuum tra realtà e virtualità che definisce la mixed reality, descritto in breve nell'introduzione, viene definito nel 1994 da Paul Milgram e Fumio Kishino. Nel 1997 Ronald Azuma scrive la prima indagine sull'AR identificandola come la combinazione tra un ambiente reale e virtuale. Il primo gioco mobile outdoor basato su AR viene rilasciato nel 2000 da Bruce Thomas, ed esposto all'International Symposium on Wearable Computers. Nel 2005 viene effettuata nell'Horizon Report la previsione secondo la quale le tecnologie sulla realtà aumentata si sarebbero sviluppate largamente negli anni successivi. Proprio in quell'anno vengono sviluppate telecamere di nuova generazione in grado di analizzare gli ambienti e stimare la posizione relativa degli oggetti, innovazione che risulterà fondamentale negli anni a venire. Successivamente si assiste ad un notevole incremento delle applicazioni basate su realtà aumentata, anche con l'avvento di nuovi tools come Wikitude e Metaio. Oggigiorno l'AR sta sempre più prendendo piede e si prepara ad entrare nella vita quotidiana di milioni di persone[2].

## 1.2 Applicazioni Location Based e riconoscimento

La realtà aumentata può racchiudere molteplici aspetti e spunti interessanti per indagare su tutte le tecnologie e le tecniche ad essa collegate. Uno di questi è sicuramente la geolocalizzazione in tutte le sue sfaccettature.

Nello sviluppo di applicazioni o giochi AR l'aspetto della localizzazione risulta spesso cruciale. Basti pensare a tutte le applicazioni outdoor che offrono all'utente un'esperienza aumentata; è evidente che le informazioni circa la posizione degli utenti e degli oggetti reali e aumentati sono cruciali. Oltretutto tali informazioni devono spesso avere una grande precisione (anche dell'ordine del centimetro talvolta). Questo fa sì che in fase di progettazione le scelte che si effettuano riguardo l'utilizzo delle tecnologie e delle tecniche di localizzazio-

ne risultano fondamentali, e da esse può dipendere il funzionamento dell'intero sistema. Applicazioni che sfruttano la localizzazione sono dette location-based.

La localizzazione GPS è probabilmente la più diffusa tecnica di geolocalizzazione oggigiorno. Il segnale GPS può però talvolta essere troppo debole , per cui risulta necessario trovare alternative valide. Attualmente esistono due varianti del GPS, ovvero D-GPS e A-GPS. D-GPS (Differential-GPS) migliora notevolmente la precisione del segnale utilizzando una rete di postazioni fisse in grado di aumentare la precisione delle coordinate fornite dal sistema GPS grazie al confronto con la posizione relativa rilevata localmente dalla rete. A-GPS (Assisted-GPS) abbatte i tempi di ricerca iniziale della posizione, utilizzando dati provenienti dalla rete quando il segnale GPS è debole.

Nel caso in cui nemmeno questi accorgimenti fossero possibili, diventa necessario utilizzare altre tecniche. Le tecniche in questione si basano sul riconoscimento diretto degli oggetti per poterli "aumentare" aggiungendovi informazioni. Il riconoscimento può essere marker-based oppure markerless. Per quanto riguarda il riconoscimento marker-based, esso consiste nell'utilizzo di markers, ovvero immagini predefinite che vengono identificate dalla telecamera. Il riconoscimento markerless invece non fa uso di markers ma fa leva sul riconoscimento di colori, forme, spigoli. Risulta evidente che l'utilizzo delle tecniche di riconoscimento diretto (quindi non basato sul posizionamento GPS) pone problemi di scalabilità, in quanto l'aggiunta di oggetti "aumentati" nell'ambiente deve essere necessariamente associata o a un preciso marker o a una forma o colore ben predefiniti.

## 1.3 Pervasive software e tecniche di comunicazione

Un ulteriore aspetto dei sistemi AR (tipicamente di quelli distribuiti) è quello della comunicazione. Gli utenti, potendo avere la possibilità di agire sul mondo aumentato, devono avere la possibilità di cambiare lo stato degli oggetti, per cui sono necessarie avanzate tecniche di comunicazione per poter agire a seguito di tali eventi per modificare le informazioni relative agli oggetti nel luogo in cui risiedono (su server, su Cloud...). Inoltre, tutti gli utenti che partecipano ad un'esperienza comune di realtà aumentata devono percepire i cambiamenti di stato degli oggetti in maniera simultanea e in hard real-time, per evitare incongruenze in quello che vedono nello stesso momento e nello stesso punto. Oltretutto può essere determinante la possibilità di far comunicare gli utenti direttamente tra di loro, in ottica di collaborazione e cooperazione.

Tutti questi aspetti rendono il problema della comunicazione cruciale in applicazioni e giochi basati su realtà aumentata. Un'applicazione o gioco che richieda un sistema tale di condivisione di informazioni può essere definito pervasive, poichè lo stato e il contenuto degli oggetti aumentati devono essere distribuiti a tutti gli agenti in campo, dando forma appunto a software "pervasivo", ovvero presente in ogni angolo del gioco o applicazione.

La compagnia di giochi mobile "It's Alive" descrisse i pervasive games come "games that surround you" [3], ovvero "giochi che ti circondano", mentre il libro di Montola, Stenros e Waern, "Pervasive Games" afferma che hanno "one or more salient features that expand the contractual magic circle of play spatially, temporally, or socially" [4], ovvero "una o più caratteristiche salienti che espandono il magico cerchio contrattuale del gioco spazialmente, temporalmente, socialmente".

In giochi pervasivi l'infrastruttura di comunicazione va studiata con lo scopo di renderla perlopiù immune ai problemi causati dalla temporanea assenza di rete e dalle disconnessioni da essa. Per cui è necessario che tutti i componenti del gioco possano gestire tali problemi e non debbano affidarsi a connessioni continue con gli altri partecipanti. Un metodo per risolvere il problema può essere quello di nascondere le disconnessioni all'utente, per poi sfruttarle nel game design. Questo approccio fa sì che l'utente veda i periodi di disconnessione come situazione particolari di gioco ma non come malfunzionamenti (per esempio, durante quel periodo potrebbe essere "nascosto" agli altri utenti)[5].

Un'altra soluzione, studiata nell'ambito del lavoro collaborativo, potrebbe essere quella di utilizzare una MANET, ovvero una Mobile Ad hoc Network. Con l'utilizzo delle MANET gli utenti non devono preoccuparsi delle disconnessioni, e viene fornita una rete dinamica per lo scambio di messaggi e di informazioni. I dispositivi degli utenti possono cambiare stato e interazioni e sono liberi di muoversi. La natura delle MANET, che tipicamente forniscono un raggio ristretto di comunicazione con alto tasso di disconnessione, porta a dover progettare sistemi totalmente distribuiti, in quanto l'accesso ad un server centrale potrebbe non essere sempre garantito. [6] Risulta evidente che nell'ottica di un sistema basato su realtà aumentata, che quindi potrebbe richiedere la persistenza dello stato degli oggetti in un server centralizzato, questo comporta un rischio notevole.

Una soluzione alternativa può essere ritrovata nello sviluppo del pervasive game NetAttack [7], che racchiude aspetti sia dei giochi location-based, sia dei giochi pervasive augmented reality. Il gioco consiste nel trovare degli artefatti all'interno di uno scenario per poter ottenere una password che permetta di distruggere il database centrale di una fantomatica organizzazione malvagia. Sono presenti due squadre che competono tra loro, ognuna composta da due giocatori, uno sul campo e uno davanti a un pc che comunica al compagno

come agire. Il framework utilizzato per questo progetto è MORGAN[8], che fa uso di CORBA e di design patterns come il publish-subsrcibe. CORBA (Common Object Request Broker Architecture) è uno standard che facilita la comunicazione tra sistemi che sono sviluppati su piattaforme diverse. Il publish-subscribe pattern permette a processi interessati a dati provenienti da particolari componenti di "abbonarsi" (subscribe) al componente per poi ricevere i suddetti dati da lui "pubblicati" (publish). MORGAN integra questi meccanismi e ne risolve i problemi principali, legati soprattutto alle difficoltà dovute all'eventuale caduta della rete e alla pesantezza di CORBA. Tali difficoltà vengono "by-passate" attraverso l'introduzione di un protocollo più snello, che permette di inviare/ricevere solo le informazioni necessarie, ad esempio latitudine e longitudine di un giocatore[5].

NetAttack fa quindi uso del framework appena descritto, e sfrutta la tecnologia WiFi per avere copertura dell'area in cui si svolge il gioco. Tale approccio si è rivelato efficace in alcuni casi, infatti in assenza di disconnessioni il gioco si è sviluppato senza intoppi, mentre si è rivelato fragile in caso di fail del sistema di rete, poichè non essendo presenti metodi di recovery il giocatore vittima della disconnessione ha dovuto inesorabilmente abbandonare il gioco[5]. Questo dimostra come il modo in cui si progetta un sistema di comunicazione e in particolare un'infrastruttura di rete sia fondamentale in ottica di pervasive augmented reality gaming.

## 1.4 Cooperazione e applicazioni CSCW (cenni)

Una delle applicazioni pratiche della realtà aumentata più interessanti è sicuramente quella in ottica di cooperazione. Fin dagli albori dell'AR i ricercatori si sono resi conto che il suo utilizzo nell'ambito della cooperazione avrebbe potuto portare a scenari molto interessanti. Già nel 1996 i ricercatori del progetto Studierstube identificavano i cinque punti fondamentali delle applicazioni collaborative:

**Virtuality:** gli oggetti che non esistono nel mondo reale possono essere visualizzati e esaminati.

**Augmentation:** gli oggetti reali possono essere aumentati tramite annotazioni virtuali.

**Cooperation:** più utenti possono vedersi e cooperare tra loro.

**Indipendence:** utenti individuali controllano punti di vista differenti

**Individuality:** i dati possono essere visualizzati in maniera differente per ogni utente in base alle proprie esigenze[10].

Tali punti specificano tutte le caratteristiche che, se soddisfatte, possono rendere un'applicazione AR collaborativa affidabile.

La cooperazione viene studiata da un ramo molto importante, ovvero il CSCW (Computer-supported cooperative work). La realtà aumentata offre un supporto fondamentale per tale ramo, in quanto le possibilità che offre ben si prestano per supportare la cooperazione.

## 1.5 Mirror World

La realtà aumentata offre anche lo spunto per introdurre un altro fondamentale concetto, quello di Mirror World. Il mondo non è più solo fisico e tangibile, ma c'è dell'altro. Il software, sempre più "pervasivo" racchiude informazioni sul mondo che ne ampliano il significato. L'obiettivo è quello di fornire agli utenti (che possono essere gli abitanti di una città) una percezione nuova di quello che li circonda, aiutandoli così a comprendere aspetti del mondo nuovi. Oltretutto, non solo gli utenti hanno una percezione diversa del mondo intorno a loro, ma è il mondo stesso a percepire gli utenti e anche il mondo fisico. Così un oggetto facente parte del Mirror World può percepire cambamenti di luminosità, rumori, spostamenti d'aria e agire di conseguenza.

La realtà aumentata si inserisce nel concetto di Mirror World come tecnologia abilitante. Infatti parte dell'estensione degli oggetti reali nel Mirror World può essere proprio vista come "augmentation", ovvero come aggiunta di informazioni legate all'AR con lo scopo, ad esempio, di visualizzare qualcosa tramite glasses o smartphone. Un esempio per comprendere al meglio tali concetti è quello del "Ghost game in a City". Due team di persone dotate di AR smart glasses devono individuare dei tesori mentre sono inseguiti da fantasmi. Sia i tesori sia i fantasmi fanno parte del Mirror World, e la differenza da un comune gioco in realtà aumentata è che gli oggetti del MW (in questo caso i fantasmi), percepiscono il mondo reale e si comportano in base a tale percezione. Per esempio, i fantasmi possono percepire la conformazione del mondo reale, per cui possono studiare strategie in modo da riuscire nel loro intento di catturare i giocatori. Oppure, ad esempio, percependo la luminosità di un ambiente potrebbero esserne vulnerabili, ed essere neutralizzati quando è superiore a un certo limite.

Tutti questi aspetti rendono chiaro quali possono essere eventuali sviluppi futuri delle tecnologie AR, che possono arrivare a rivoluzionare il modo di vedere la realtà[9].

## 1.6 Tools a supporto della realtà aumentata

In questa sezione si descrivono i principali tools a supporto della realtà aumentata, ovvero Wikitude, Metaio e Layar.

Wikitude è un tool che fornisce tecnologie utili alla realtà aumentata nato in Austria nel 2008. Inizialmente nato come supporto ad applicazioni AR location-based, dal 2012 è stato indirizzato anche verso il riconoscimento di immagini e markers, con il lancio del Wikitude SDK, che sfrutta anche tecnologie di geolocalizzazione. Per quanto riguarda le applicazioni location-based, Wikitude fornisce esperienze di realtà aumentata utilizzando la localizzazione GPS e sensori quali giroscopio e accelerometro per permettere il calcolo della posizione degli oggetti da visualizzare sullo schermo di smartphone o smart-glasses.

Metaio, compagnia fondata nel 2003 a Monaco di Baviera, si occupa dello sviluppo di software che supportino la realtà aumentata. Ha prodotto il Metaio SDK, che permette di sviluppare applicazioni AR in svariati ambienti come, ad esempio, iOS, Android e Windows. Nel 2009 rilascia Junaio, un browser studiato per la realtà aumentata, che ancora oggi è ampiamente diffuso.

Sempre nell'ottica dei browser si inserisce Layar, compagnia olandese fondata nel 2009 ad Amsterdam. Layar si è imposto fin da subito come uno dei browser per realtà aumentata dominanti nel mercato, offrendo esperienze di augmented reality accessibili a un numero molto elevato di utenti.

Tutti questi strumenti offrono esperienze significative nell'ambito dell'AR, ma sono limitati dalla necessità di utilizzare markers o tecniche avanzate di riconoscimento delle immagini per risultare efficaci.

# Capitolo 2

# Caso di Studio

## 2.1 Caso di Studio

L'idea di base è quella di realizzare un sistema che permetta ad un team formato da più elementi di cooperare al fine del raggiungimento e dell'analisi di determinati punti d'interesse, sfruttando la realtà aumentata. Un punto può essere visto come uno "scrigno" che contiene un determinato oggetto (può anche essere vuoto). L'obiettivo del team è quello di raggiungere i punti d'interesse per trovare tutte le monete che compongono il "tesoro" finale. Uno scrigno può contenere delle monete oppure una chiave per un altro scrigno, per cui solo il giocatore che ha raggiunto il punto in cui vi è una chiave può ottenere l'oggetto dello scrigno aperto da tale chiave. Una volta aperti tutti gli scrigni il gioco è concluso. Tutte le informazioni relative ai punti già raggiunti e allo stato di ogni punto dovranno essere condivise tra i membri del team.

A seconda delle informazioni in esso contenute, lo scrigno può essere aperto secondo diverse modalità:

- può essere aperto direttamente giungendo al punto di interesse
- può essere aperto solo se si è in possesso della chiave dello scrigno
- può essere aperto solo se almeno due membri del team sono presenti nel punto d'interesse
- può essere aperto solo se tutti gli altri scrigni sono stati aperti e almeno due membri del team sono presenti nel punto d'interesse (in questo caso si tratta dello scrigno finale)

Inizialmente non si conoscono le informazioni contenute all'interno dei punti d'interesse e il loro stato è "non visitato". Quando avviene l'arrivo di un

giocatore in un punto, lo stato può cambiare in "scrigno aperto", specificando anche il tipo di informazione ivi ottenuta (monete, chiave, nessuna informazione), o "scrigno non apribile", specificando la causa, ovvero mancanza di chiave o bisogno della presenza di un altro giocatore. Ogni giocatore visualizza sullo schermo i dieci punti di interesse che rappresentano gli "scrigni" da aprire (attraverso una mappa, un radar o un elenco) e la sua posizione.

Viene anche visualizzata la quantità totale di "monete" raggiungibili, data dalla somma dei contenuti di ogni scrigno (la quantità di ogni scrigno diminuisce col passare del tempo o con l'azione di una "centrale di controllo").

Nel momento in cui un giocatore raggiunge uno dei punti, se il giocatore è autorizzato ad aprire lo scrigno (possiede la eventuale chiave o è in compagnia dell'altro giocatore se è necessaria la cooperazione) può visualizzare il contenuto dello scrigno il quale viene aggiunto al totale delle monete e anche scalato dal totale disponibile. Se lo scrigno contiene anche una chiave, questa viene visualizzata e aggiunta alle chiavi già possedute dal giocatore. Se lo scrigno non è apribile, l'eventuale esigenza di cooperare o di possedere una determinata chiave viene condivisa tra i due giocatori, ovvero entrambi possono conoscere lo stato di ogni punto in ogni momento tramite un'apposita schermata e vengono notificati di ogni loro cambiamento di stato.

Sullo scenario di gioco agiscono anche dei "ladri fantasma", posizionati in delle zone predefinite. All'arrivo di un giocatore in quelle zone, il ladro assale il giocatore. Il ladro ruba parte del suo denaro (se ne possiede) che viene scalato dal totale. Al termine dell'aggressione, il giocatore può lasciare un segnale di pericolo sul campo, in modo tale che quando il compagno arriverà in tale punto sarà conscio del pericolo.

Il gioco termina quando tutti gli scrigni sono stati aperti.

L'idea è quella di inserire dieci scrigni nello scenario (dieci punti d'interesse). Quattro di questi sono apribili solo possedendo la chiave (per cui ci saranno quattro scrigni che possiedono una chiave al loro interno). Quattro sono apribili semplicemente raggiungendo il punto. Uno è apribile solo con la presenza simultanea di entrambi i giocatori. Uno (il punto finale) è apribile solo con la presenza di entrambi i giocatori e solo dopo aver aperto tutti gli altri.

## 2.2 Considerazioni

Tale caso di studio offre spunti interessanti riguardo i principali aspetti della realtà aumentata.

Innanzitutto, il sistema da realizzare può essere definito come un gioco location-based. Difatti la localizzazione è cruciale per il sistema. La posizione

dei giocatori deve essere nota in ogni momento e deve essere confrontata in continuazione con la posizione degli scrigni. Inoltre, anche i ladri sono dotati di una propria posizione. Sarà quindi necessario individuare tecniche di localizzazione adeguate e studiare un sistema di condivisione delle posizioni.

Anche la comunicazione giocherà un ruolo fondamentale. Infatti il sistema da sviluppare sarà a tutti gli effetti distribuito, in quanto informazioni sugli oggetti (come ad esempio il loro stato) dovranno essere necessariamente condivise tra tutti gli utenti. Per cui sarà necessaria un'infrastruttura di rete robusta e in grado di reagire a eventuali malfunzionamenti.

Un altro aspetto interessante e cruciale riguarda la cooperazione. Molti degli scenari di gioco prevedono l'interazione, sotto forma di collaborazione, di diversi utenti. Infatti il fatto che alcuni scrigni possano essere aperti solamente con la presenza di due giocatori, o anche che un giocatore possa lasciare una notifica di pericolo per il compagno, implica che vengano sviluppati meccanismi più o meno espliciti di cooperazione. Più in generale, è evidente che il sistema dovrà soddisfare tutti i cinque punti ipotizzati nel progetto Studierstube sulle applicazioni AR collaborative. Dovrà far leva sulla virtuality, in quanto i giocatori dovranno poter interagire con gli scrigni e i ladri, non presenti "fisicamente" sul campo. Dovrà far uso dell'augmentation, per aggiungere informazioni al reale (in questo caso il reale può essere visto come una posizione ben precisa nel mondo fisico data da latitudine e longitudine). Dovrà, come detto precedentemente, basarsi sulla cooperation. Dovrà rispettare l'indipendence, poichè i giocatori vedranno il mondo aumentato da diverse prospettive. Infine, dovrà integrare aspetti dell'individuality, in quanto i giocatori potranno vedere gli stessi oggetti in maniera differente (ad esempio, il giocatore in possesso della chiave per aprire un determinato scrigno lo vedrà come "apribile", un altro giocatore non in possesso della chiave no).

Tutte queste caratteristiche rendono il gioco che verrà sviluppato un'applicazione AR collaborativa a tutti gli effetti.

# Capitolo 3

# Analisi e Modellazione

## 3.1 Casi d'Uso

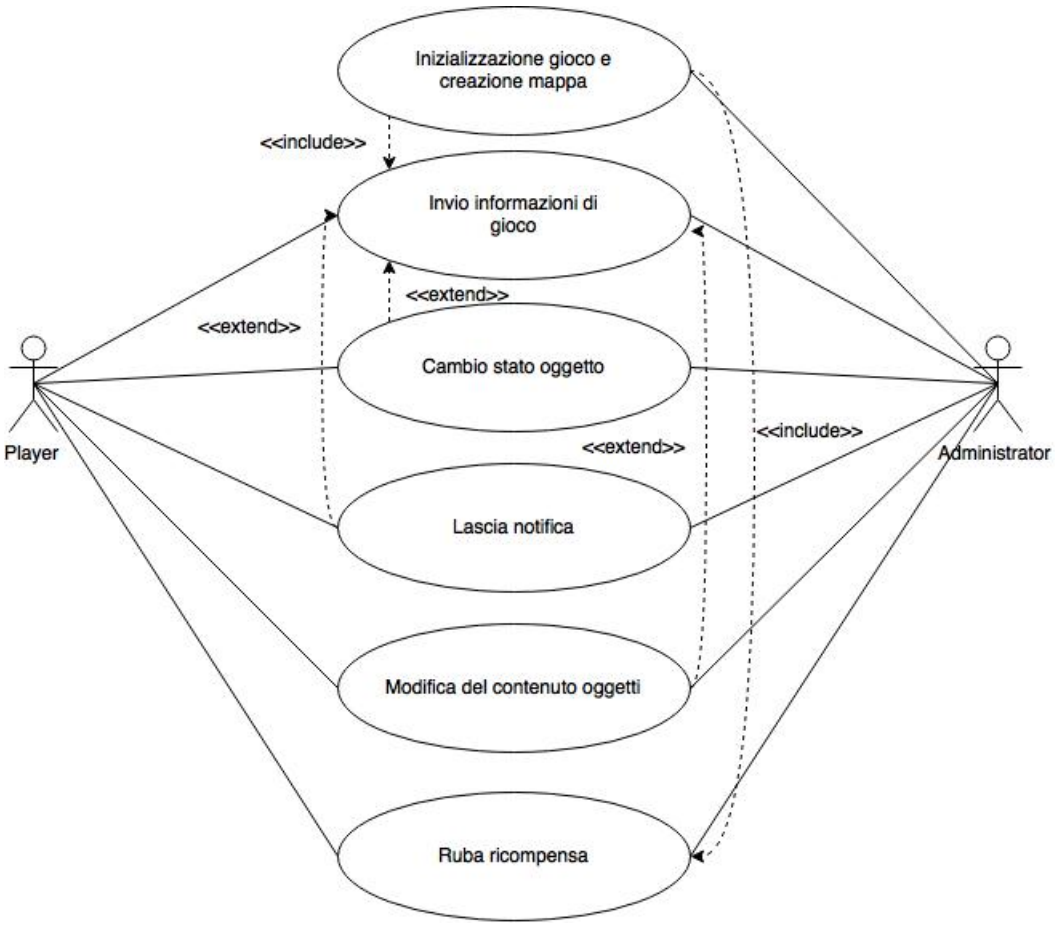


Figura 3.1: Casi d'uso del nostro sistema

### 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.

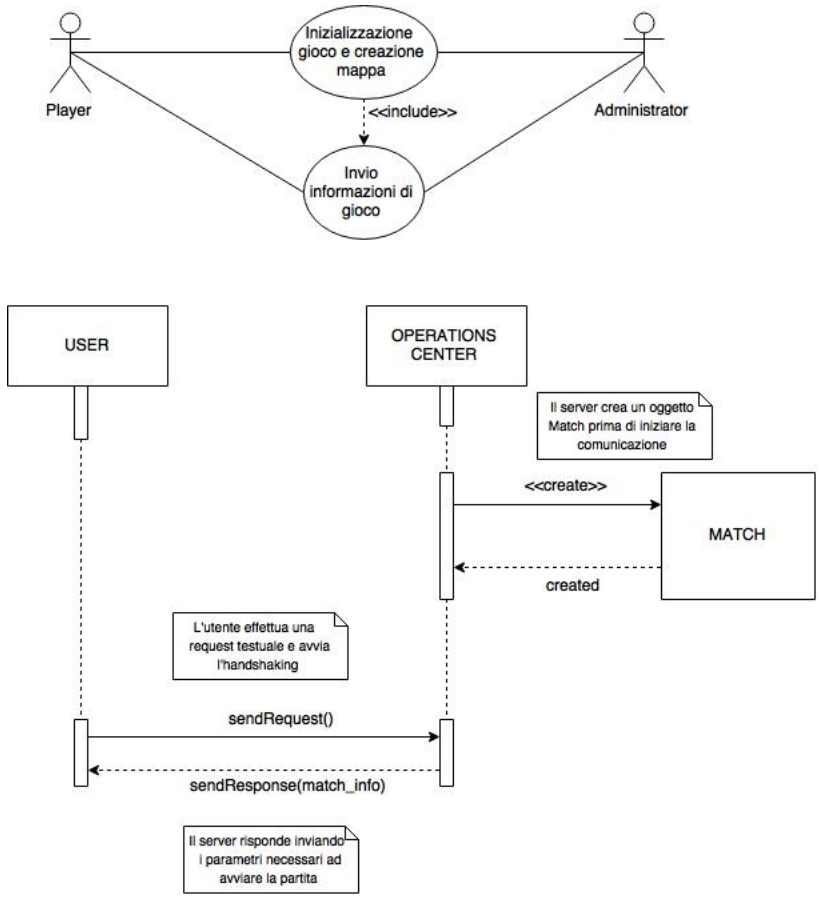


Figura 3.2: Diagramma di sequenza relativo all'inizializzazione del gioco

### 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

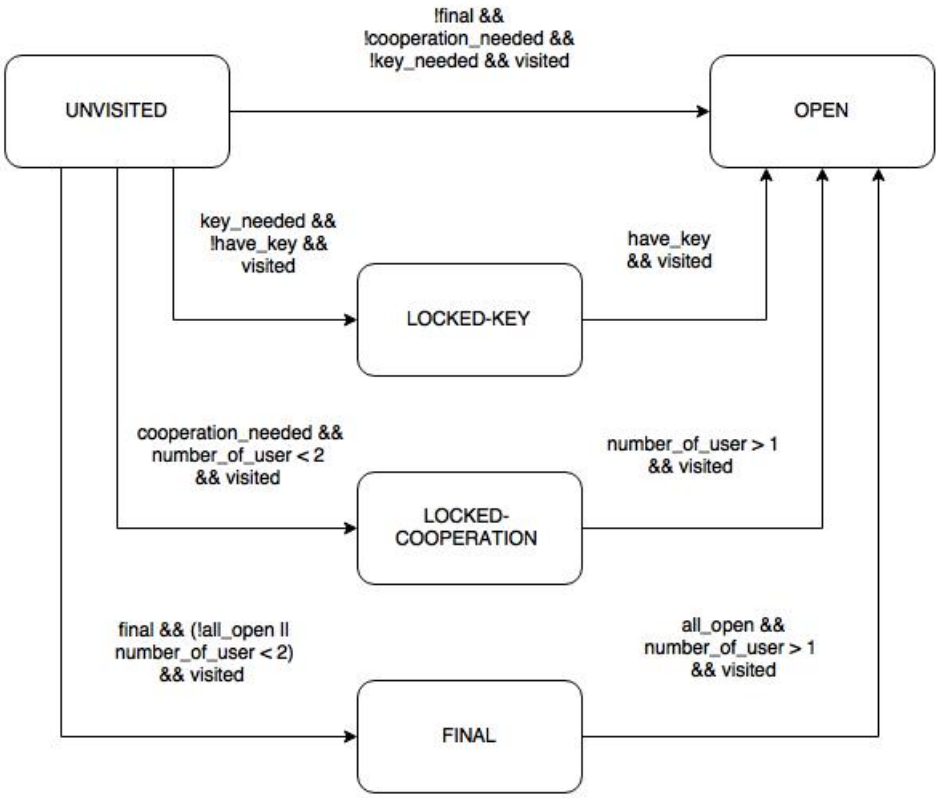


Figura 3.3: Diagramma a stati del Treasure Chest

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

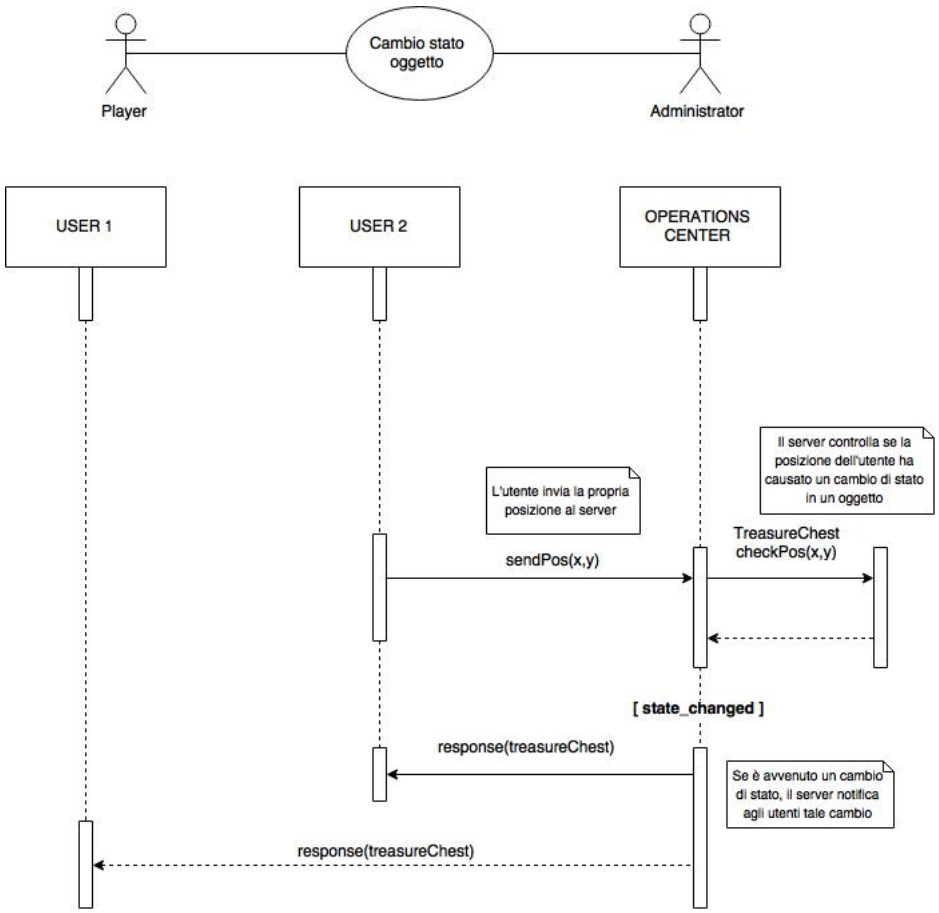


Figura 3.4: Diagramma di sequenza relativo al cambio di stato di un oggetto

Ecco i vari scenari:

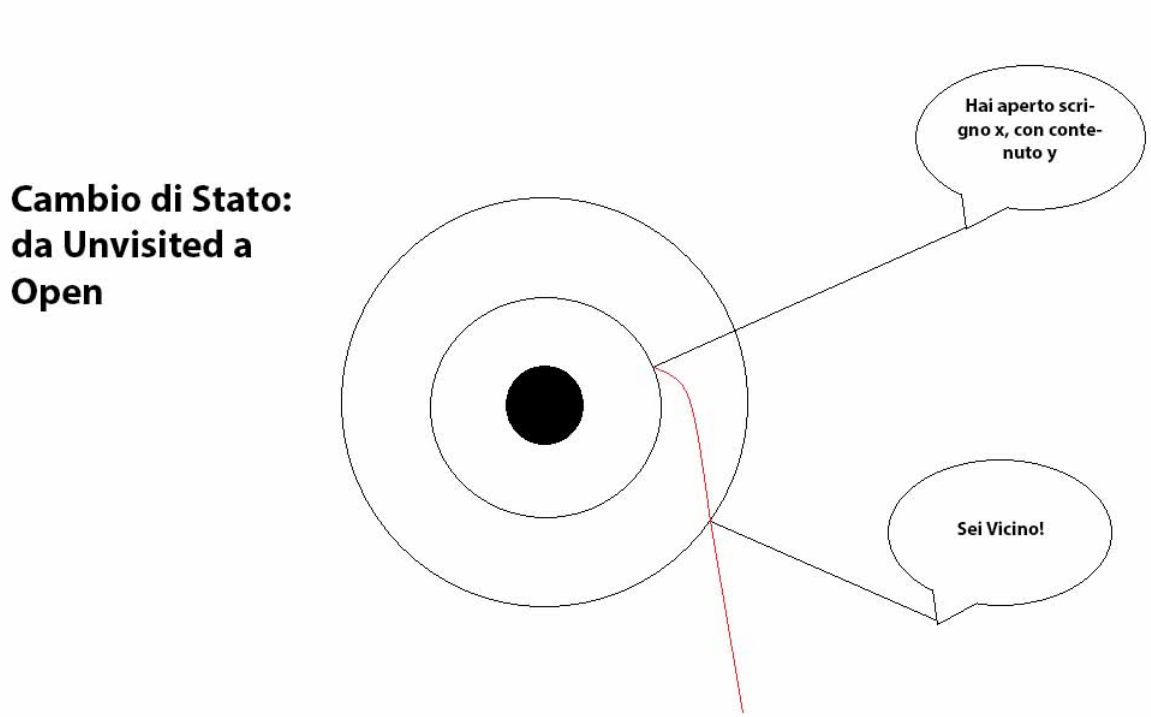


Figura 3.5: Caso in cui lo scrigno può essere aperto

**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).

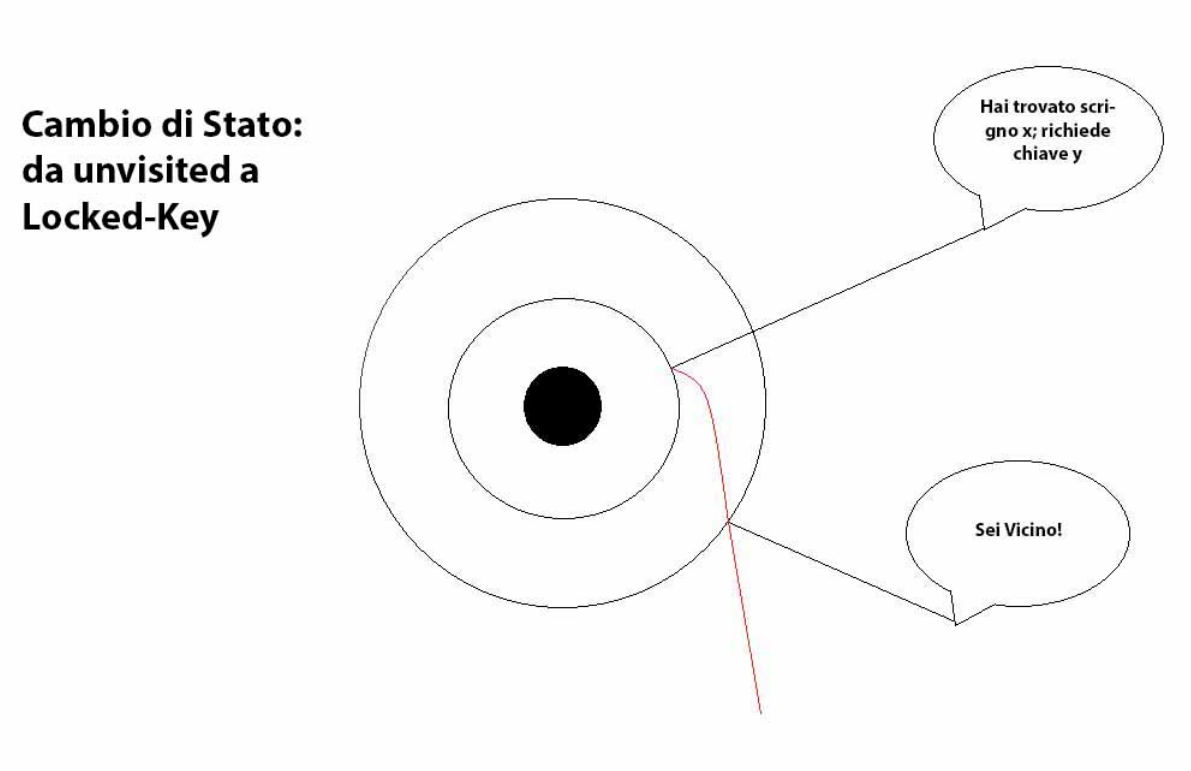


Figura 3.6: Caso in cui l'apertura dell'oggetto richiede una chiave Y, NON POSSEDUTA dall'utente

**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.

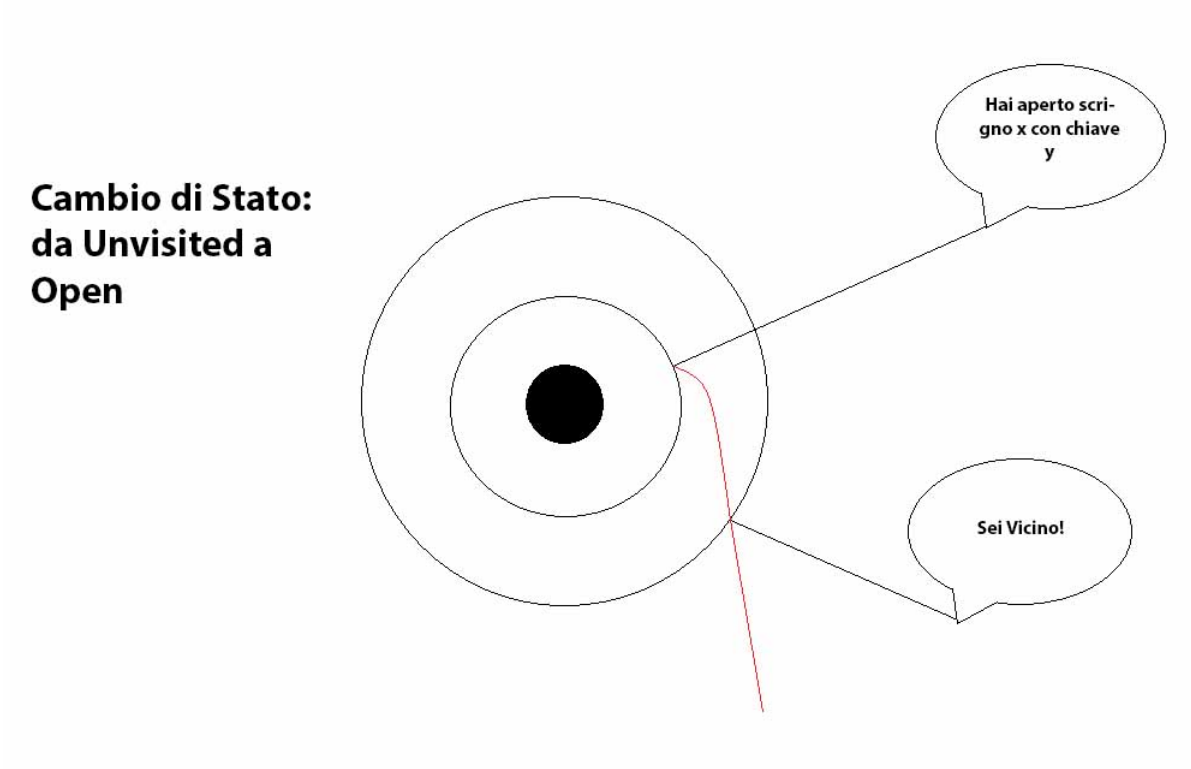


Figura 3.7: Caso in cui l'apertura dell'oggetto richiede una chiave Y, POSSEDUTA dall'utente

**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).

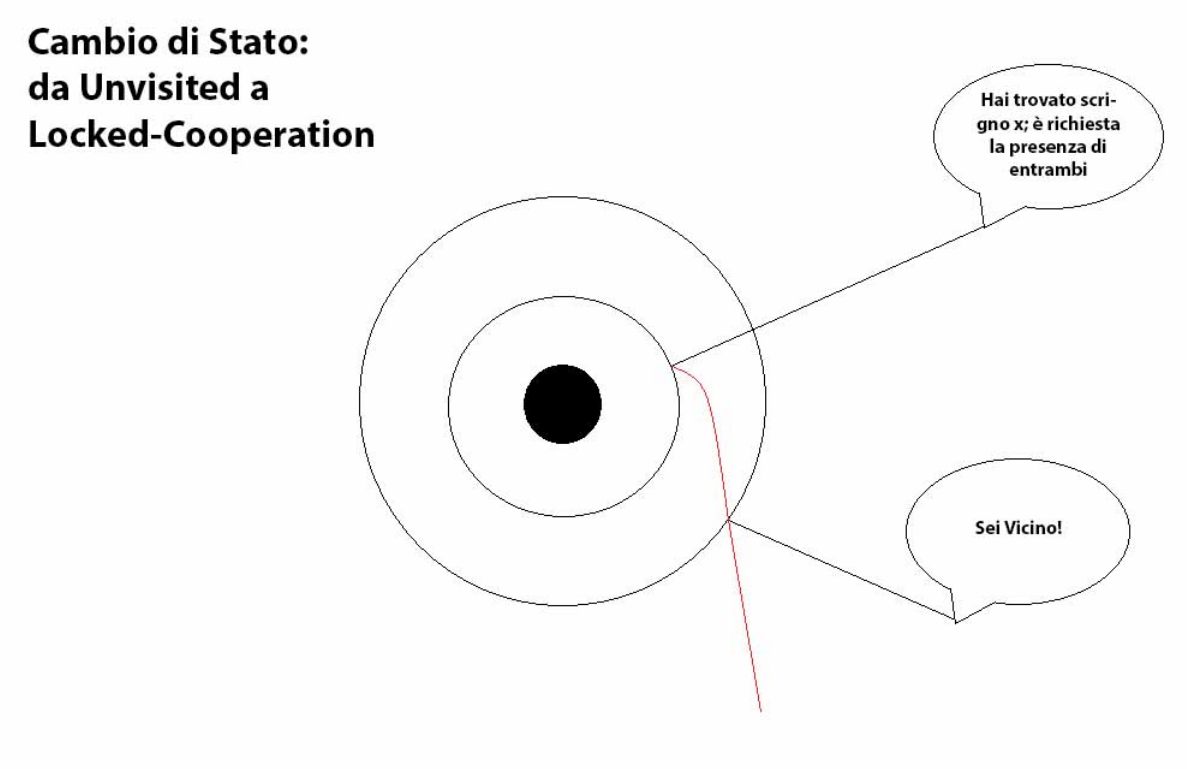


Figura 3.8: Caso in cui l'apertura dell'oggetto richiede cooperazione

**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.

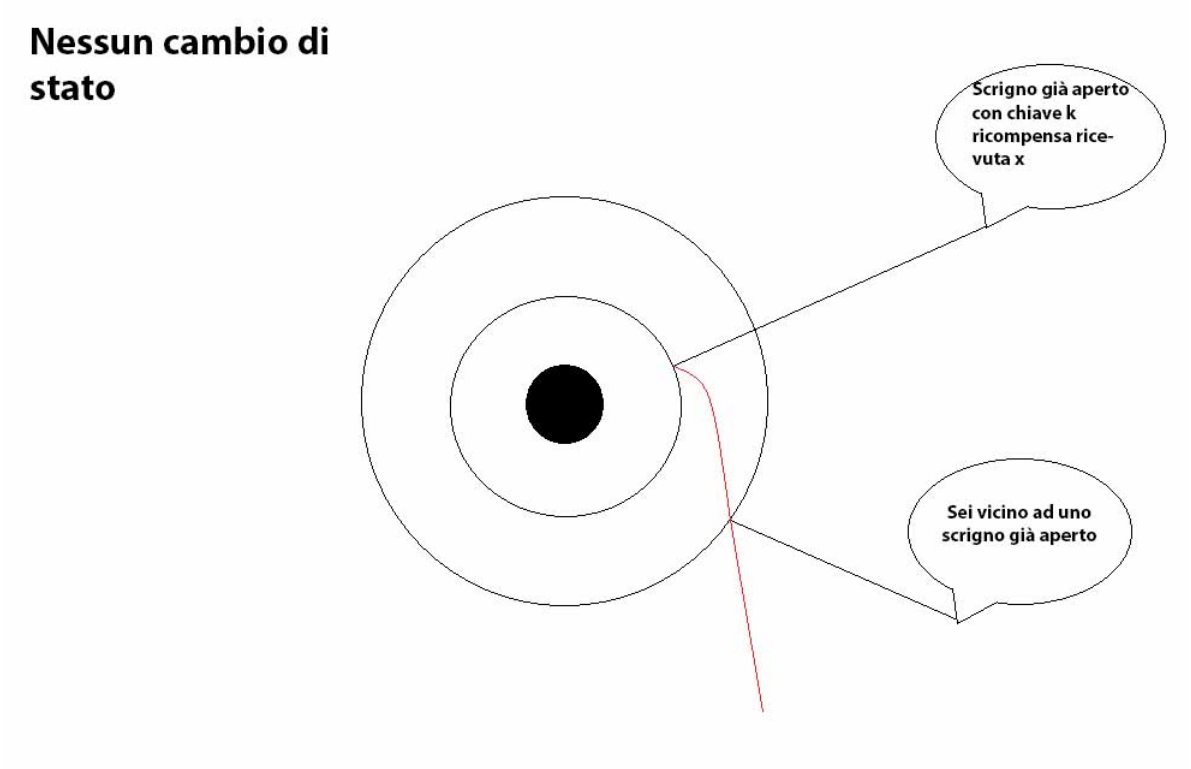


Figura 3.9: Caso in cui l'oggetto è stato già aperto

**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.

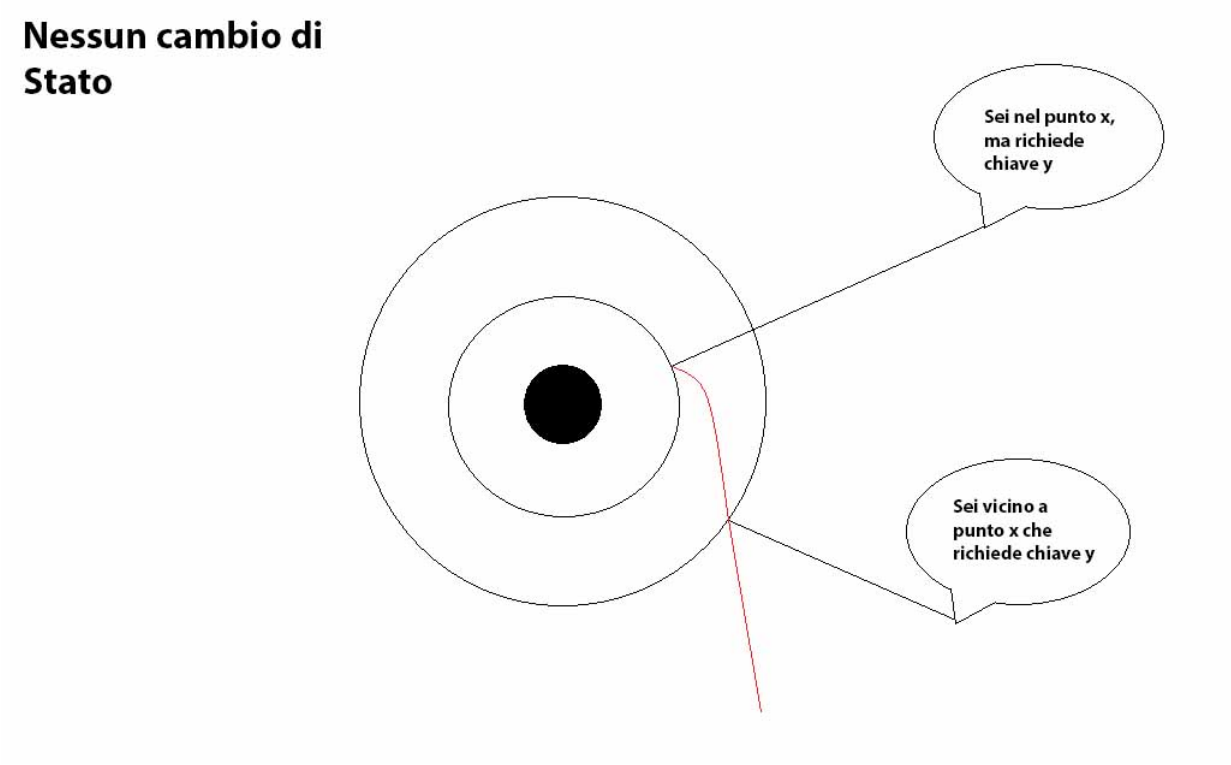


Figura 3.10: Caso in cui l'oggetto richiede una chiave Y, NON POSSEDUTA dall'utente, ed è già stato precedentemente visitato

**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.

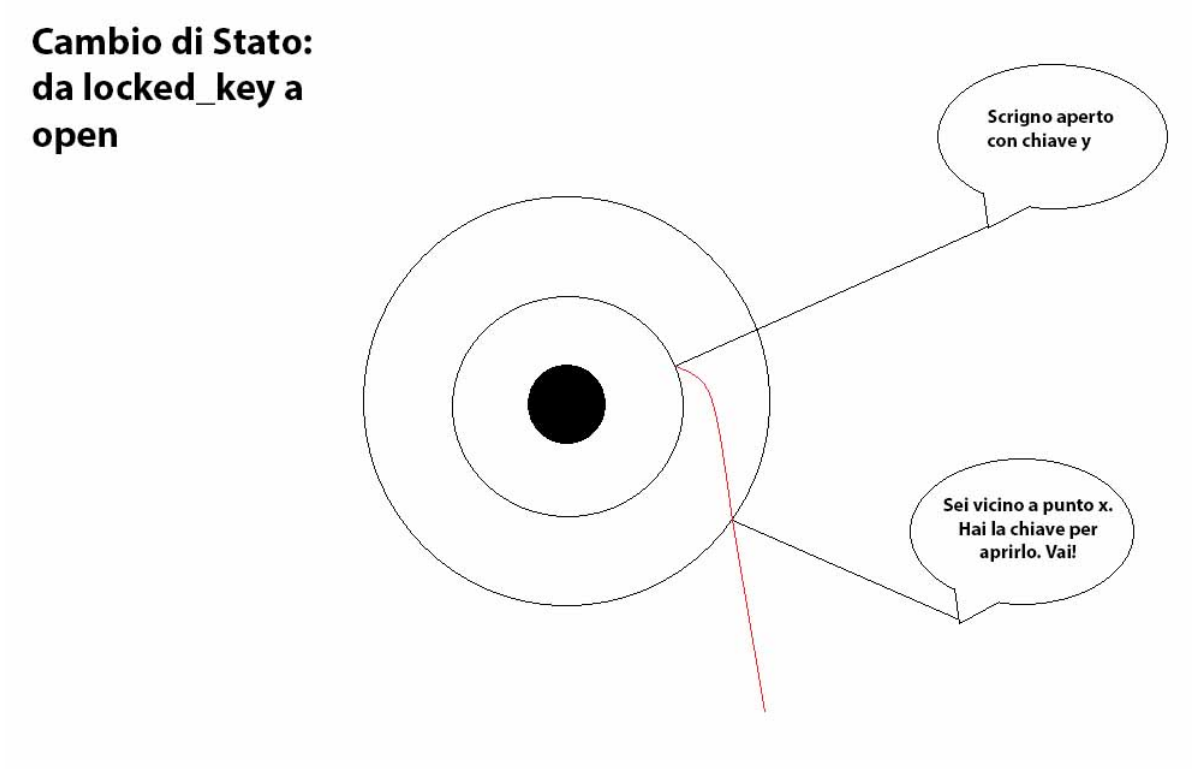


Figura 3.11: Caso in cui l'oggetto richiede una chiave Y, POSSEDUTA dall'utente, ed è già stato precedentemente visitato

**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).

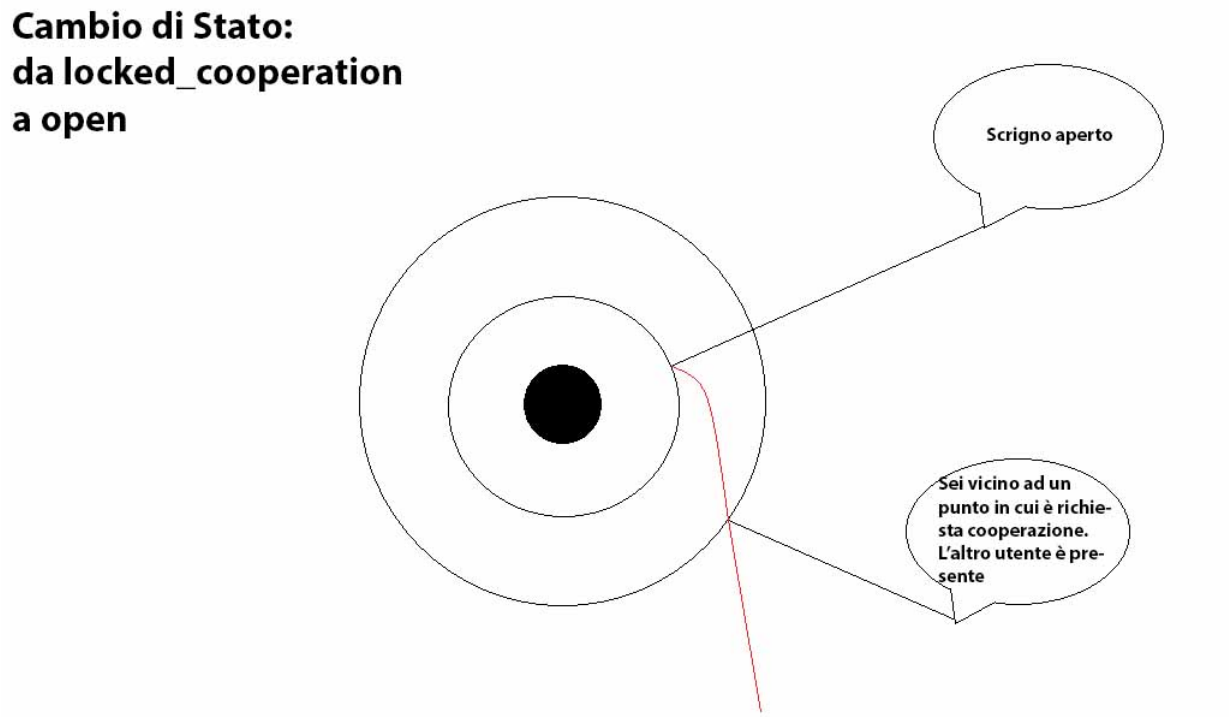


Figura 3.12: Caso in cui l'oggetto richiede cooperazione ed è già stato precedentemente visitato (l'altro utente è PRESENTE entro il raggio più ristretto dell'oggetto)

**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.

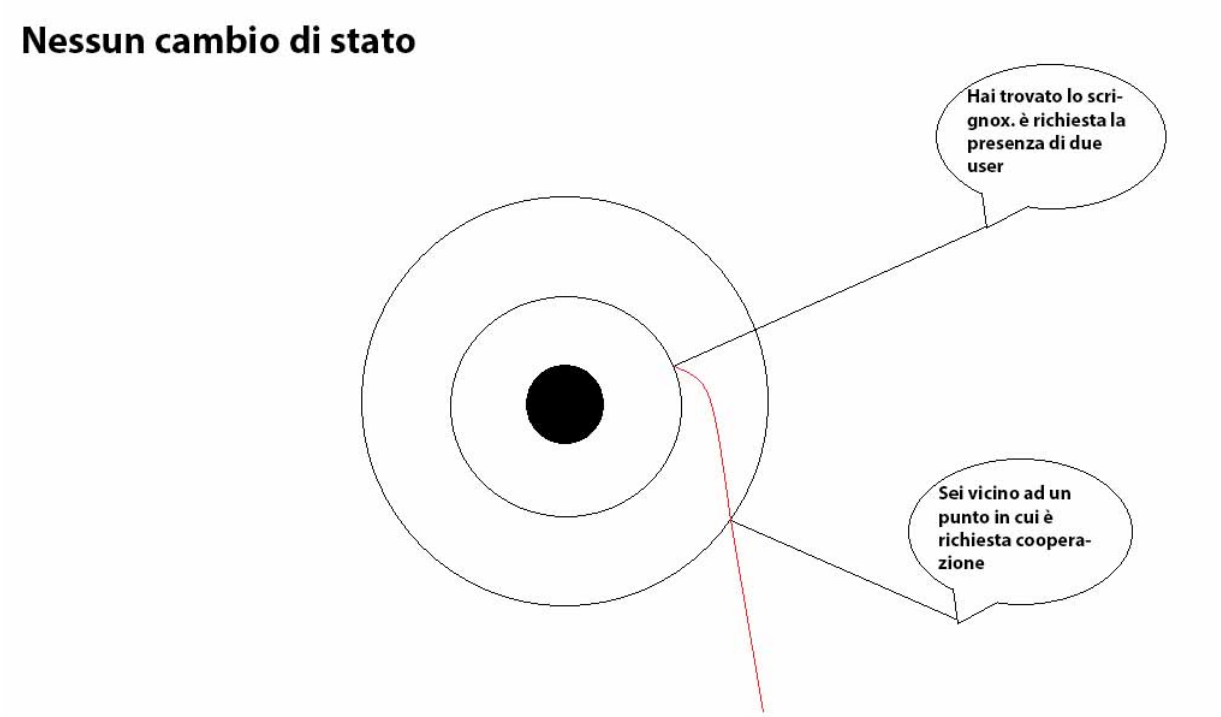


Figura 3.13: Caso in cui l'oggetto richiede cooperazione ed è già stato precedentemente visitato (l'altro utente NON è PRESENTE entro il raggio più ristretto dell'oggetto)

**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.

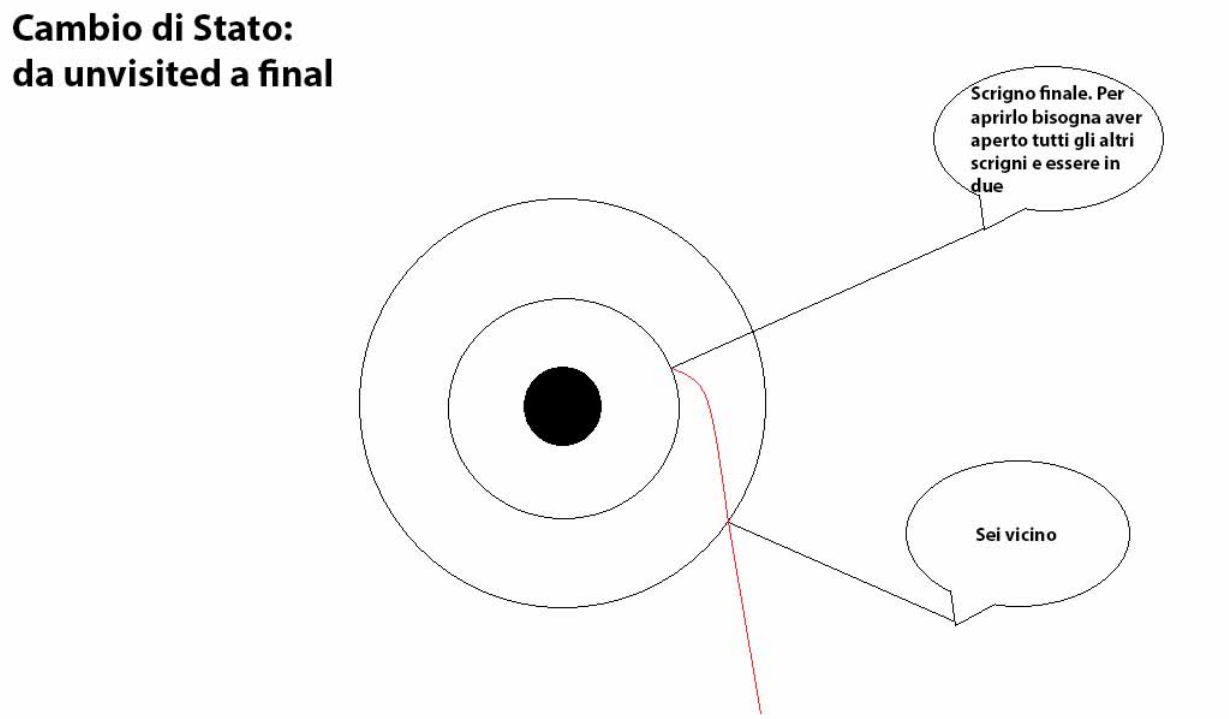


Figura 3.14: Caso in cui l'oggetto finale viene scoperto ma non può essere aperto

**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.

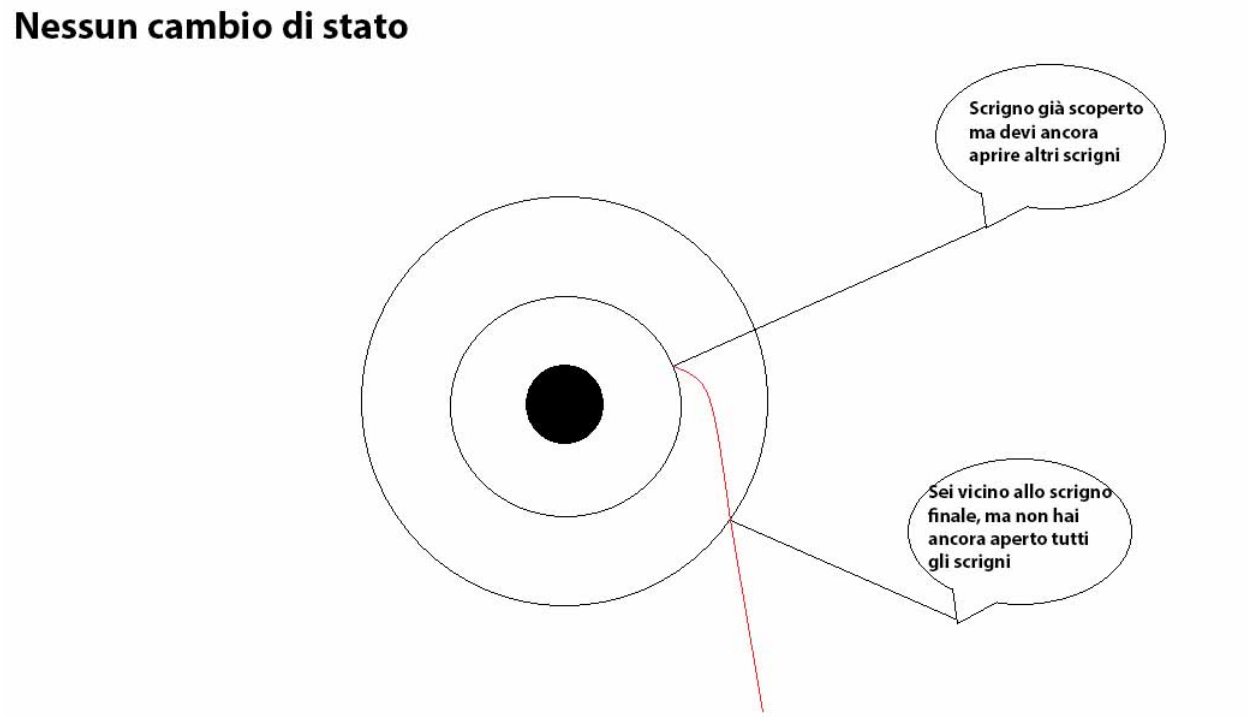


Figura 3.15: Caso in cui l'oggetto finale è già stato visitato ma non può essere aperto perchè non tutti gli altri oggetti sono già stati aperti

**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.

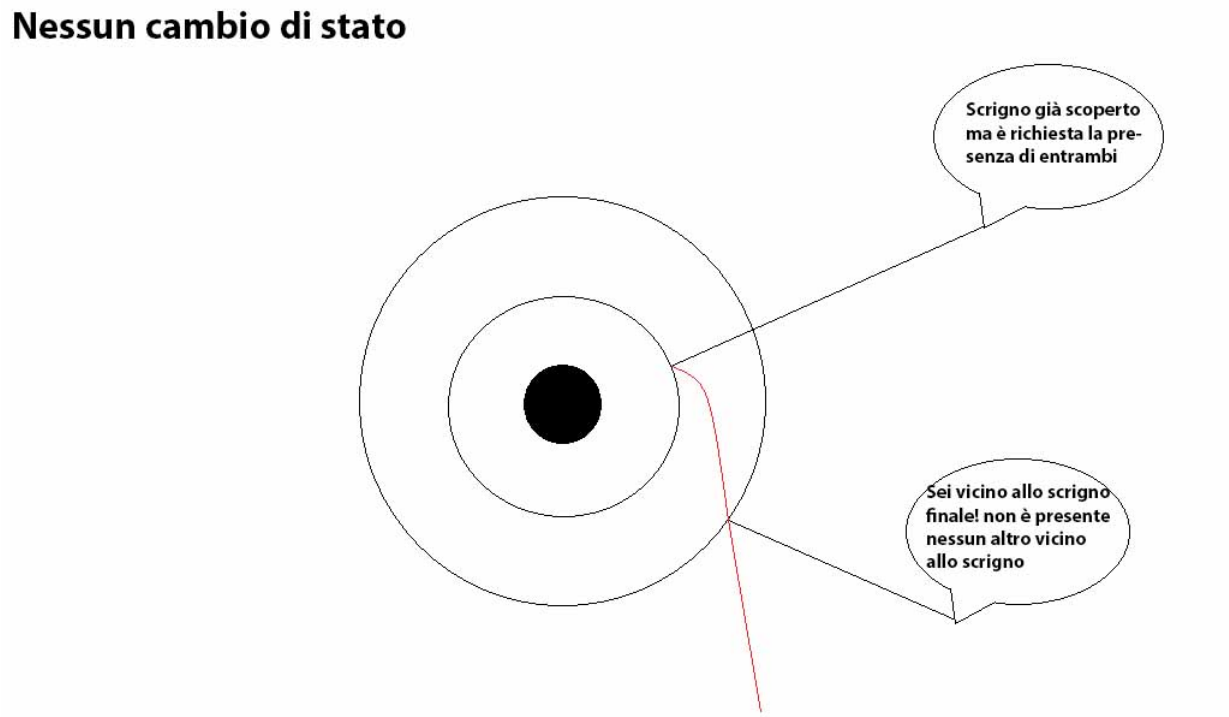


Figura 3.16: Caso in cui l'oggetto finale è già stato visitato ma non può essere aperto perchè il compagno non è presente (tutti gli altri oggetti sono già stati aperti)

**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.

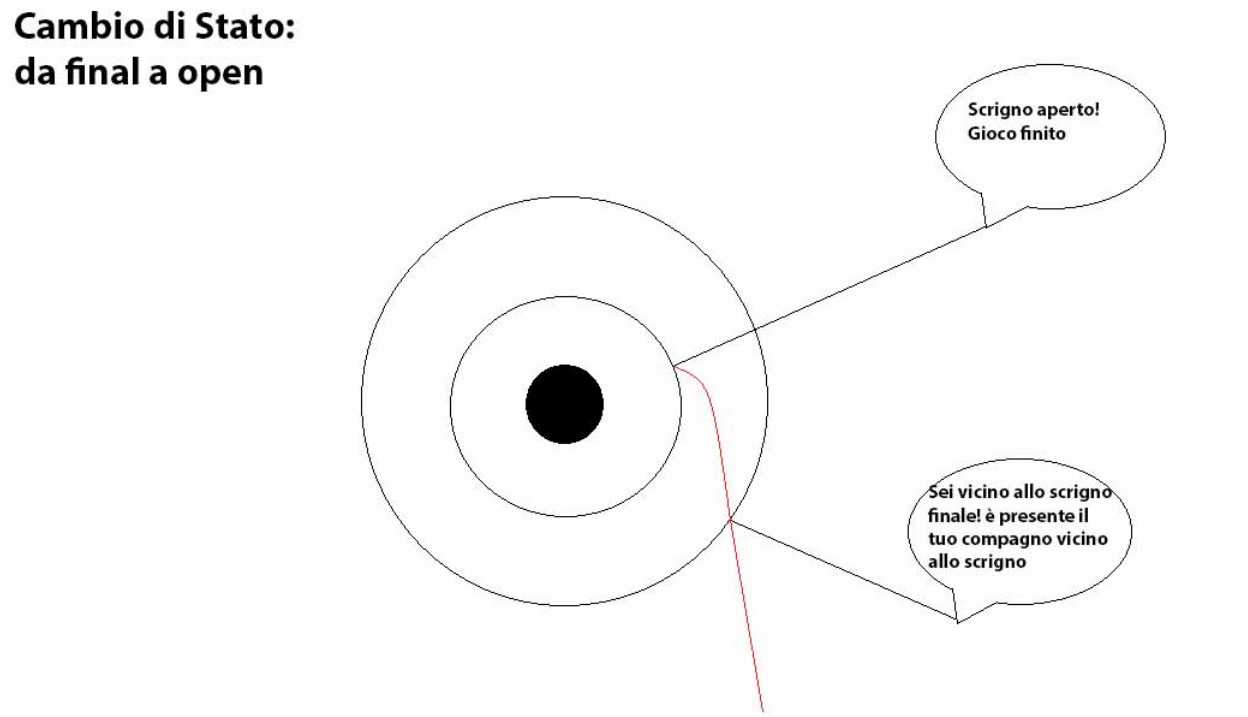


Figura 3.17: Caso in cui l'oggetto finale può essere aperto (è presente il compagno e tutti gli oggetti sono stati aperti)

**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.

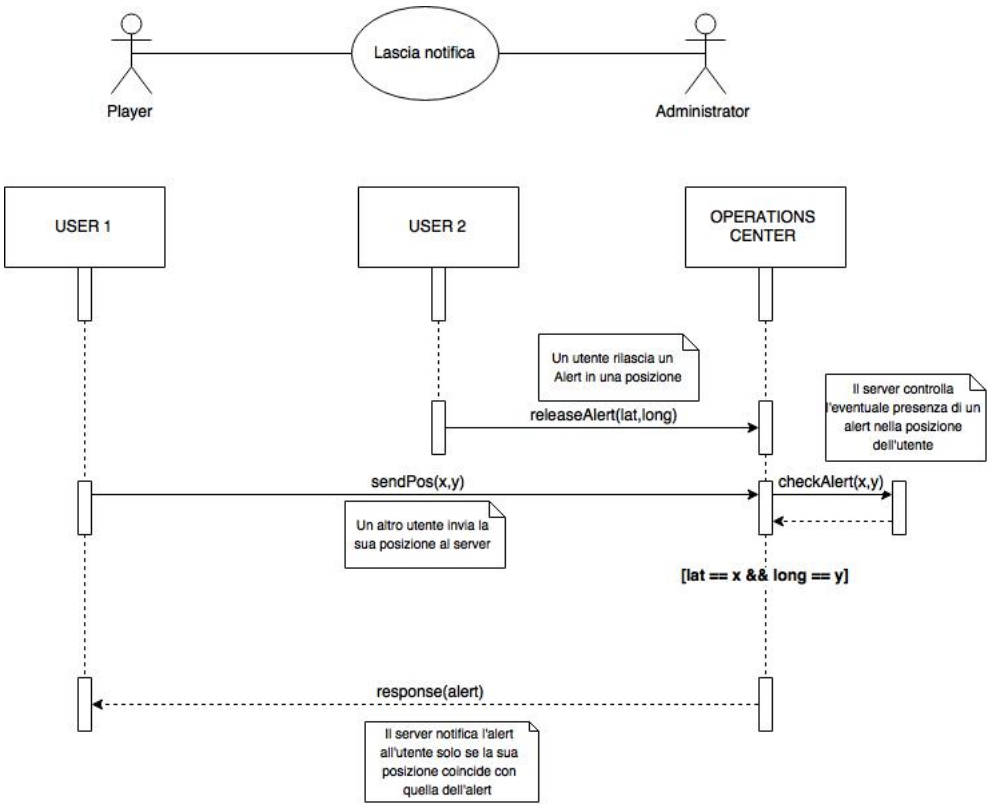


Figura 3.18: Diagramma di sequenza relativo al rilascio di una notifica

### 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.

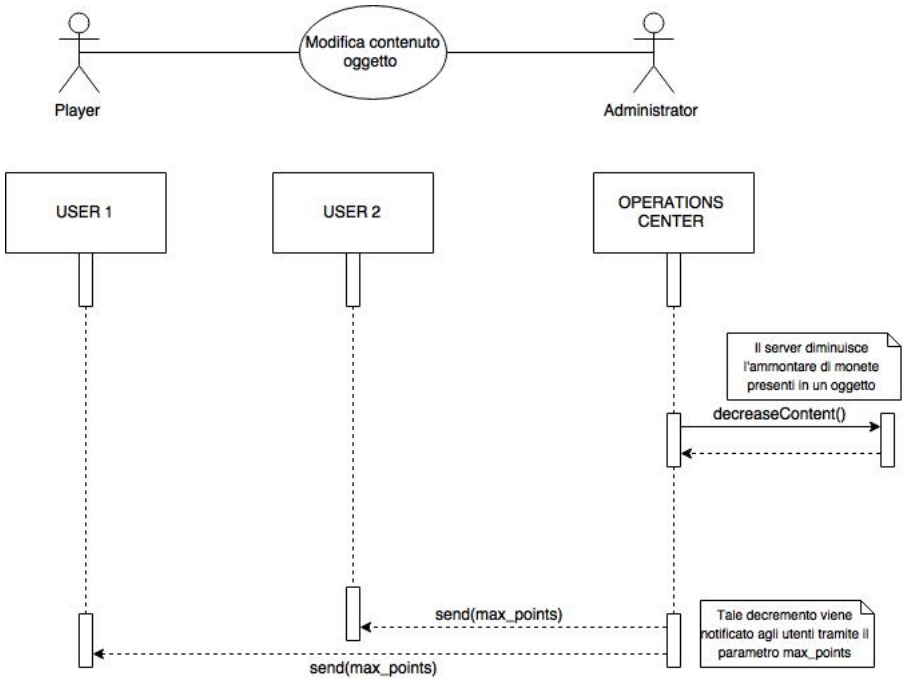


Figura 3.19: Diagramma di sequenza relativo alla modifica del contenuto degli oggetti

### 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 .

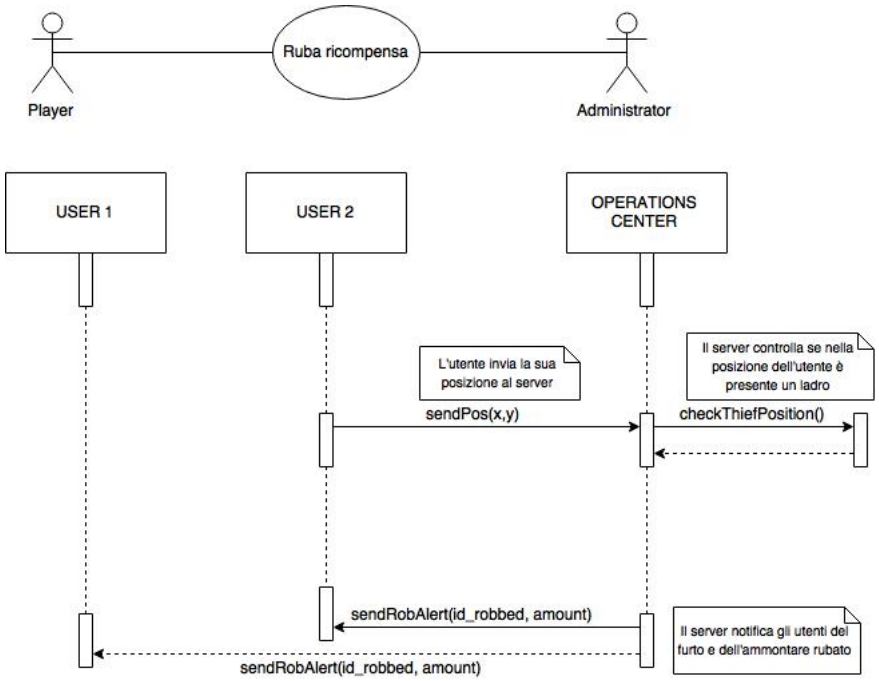


Figura 3.20: Diagramma di sequenza relativo all'azione del ladro

## 3.2 Dominio Applicativo

### 3.2.1 Lato Server

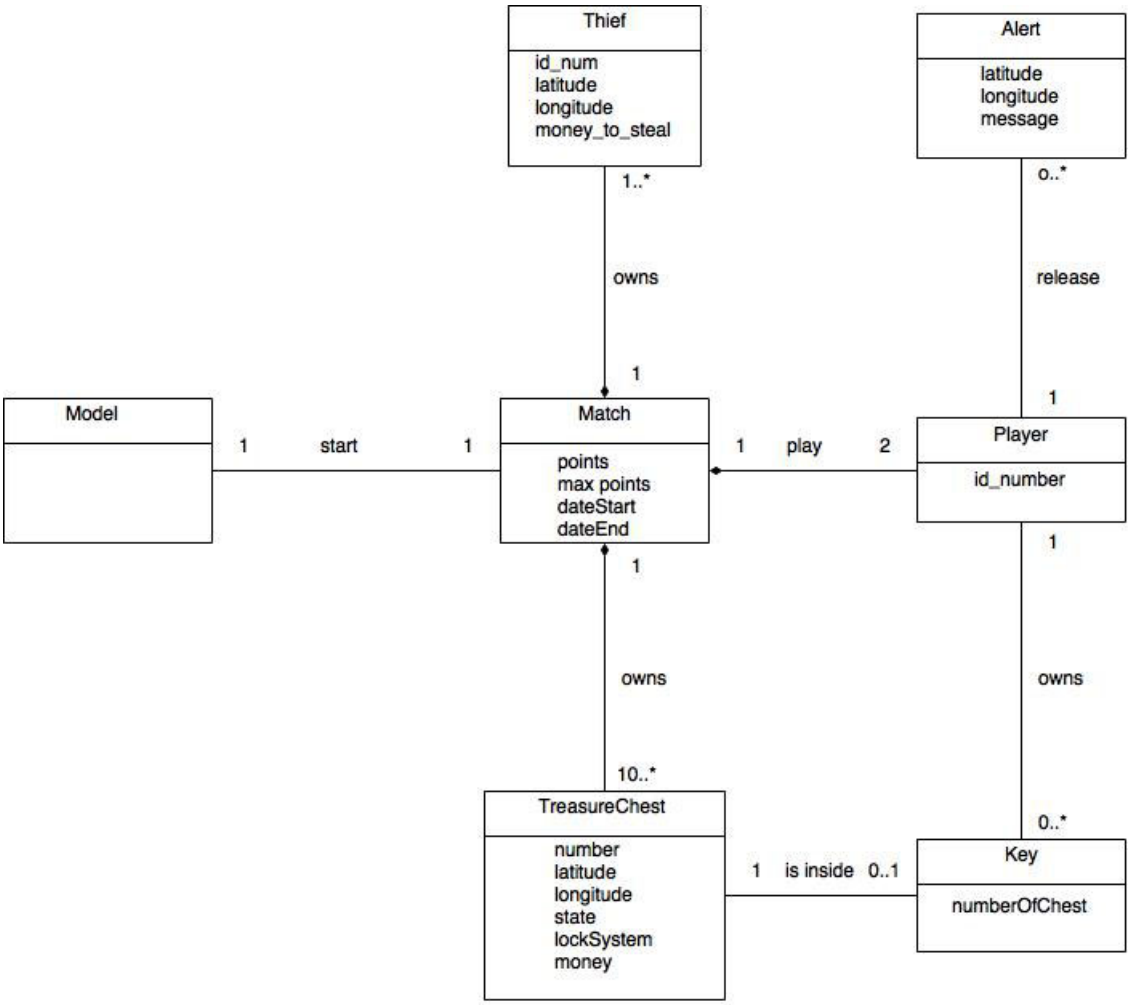


Figura 3.21: Diagramma delle classi del dominio applicativo lato server

**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

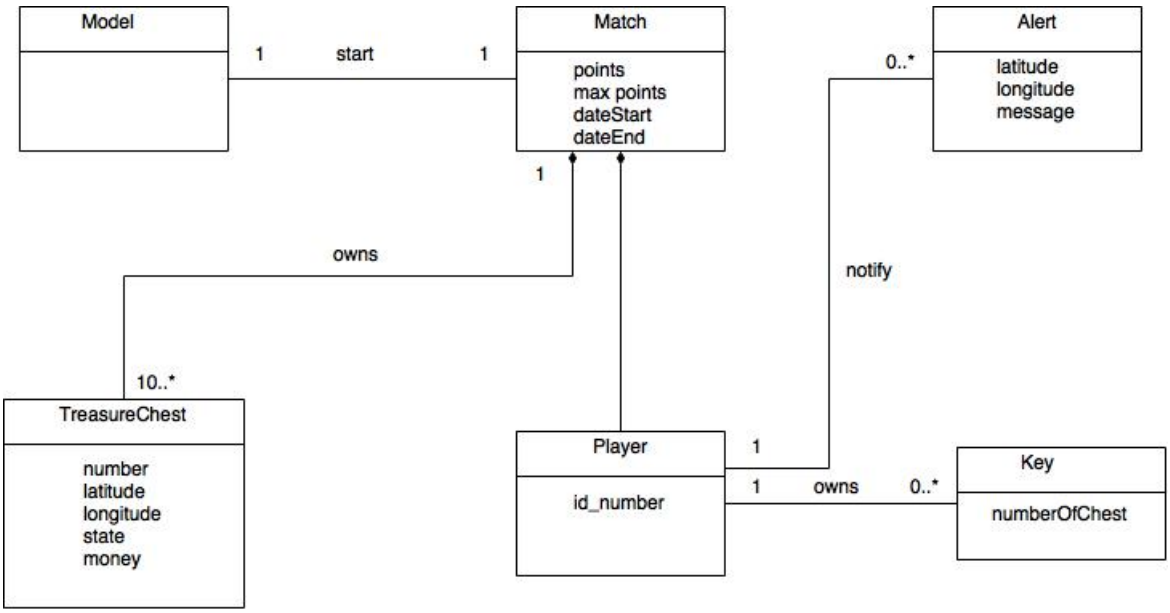


Figura 3.22: Diagramma delle classi del dominio applicativo lato client

**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 quello di studiare e realizzare l'infrastruttura di rete e, più in generale, le tecniche di comunicazione del sistema. Questo significa, oltre a mettere a disposizione le API che verranno descritte in questo capitolo, implementare dei metodi di comunicazione adeguati a sostenere la complessità del sistema e tutte le criticità di esso, come disconnessioni e latenza.

## 4.2 Problematiche principali

Nel progettare un sistema fondato sulla condivisione di informazioni risulta fondamentale studiare prima le eventuali problematiche a cui si può andare incontro.

**Le informazioni, in ogni momento, devono essere le stesse per tutti gli utenti e corrispondere a quelle mantenute dal server.** Questo è il primo aspetto cruciale di un sistema di questo tipo, ovvero la coerenza delle informazioni. Difatti, anche la perdita di una sola informazione può compromettere l'intero sistema. Si pensi allo scenario in cui avviene il cambio di stato di un oggetto. Se, per qualche motivo, l'avvenuto cambiamento di stato dell'oggetto viene rilevato dal server ma non riesce ad essere notificato a uno (o addirittura a più di uno) degli utenti, è evidente che l'intero sistema viene compromesso. Infatti, al successivo arrivo degli utenti in quel punto, l'informazione da loro posseduta sull'oggetto ivi presente sarà diversa, per cui la coerenza delle informazioni non sarà rispettata.

**La latenza, per evitare situazioni di incoerenza, deve essere ridotta al minimo.** Anche l'aspetto della latenza è cruciale. Infatti un sistema di questo tipo non può tollerare latenza troppo elevata, in quanto la collaborazione tra più utenti necessita di alta responsiveness, poichè qualsiasi ritardo può compromettere lo svolgimento del gioco.

Si consideri lo scenario della richiesta di cooperazione da parte di un giocatore per poter aprire un oggetto. Se la richiesta (o la risposta) arrivasse con una latenza troppo elevata, l'utente potrebbe ricevere la conferma/rifiuto di collaborazione quando ormai si è allontanato molto dal punto che richiede cooperazione, rendendo la comunicazione pressochè inutile.

**Il sistema deve reagire a eventuali disconnessioni senza che l'intero gioco sia compromesso.** Questo forse è il punto più critico dell'intero sistema di comunicazione. Le disconnessioni dalla rete sono inevitabili, soprattutto in uno scenario di gioco ampio in cui la connessione è affidata per lo più alle reti mobili.

Risulta evidente che in seguito alla disconnessione di un giocatore il rischio è quello di compromettere tutto il gioco e l'intero sistema. Un utente non connesso significa perdita immediata di tutte le informazioni su di esso (compreso la sua posizione) e impossibilità di raggiungerlo per eventuale condivisione di informazioni. Per cui vanno studiate tecniche di recovery adeguate per ovviare al problema.

## 4.3 Possibili soluzioni

Il primo problema, quello relativo alla coerenza delle informazioni, può essere risolto con l'utilizzo di connessioni affidabili orientate alla connessione e full-duplex. Infatti è fondamentale avere la certezza assoluta che le informazioni arrivino sempre intatte al destinatario, senza errori o perdite. Può diventare necessario inoltre implementare meccanismi di acknowledgement a livello applicativo qualora fosse necessario per rafforzare ulteriormente la comunicazione. Ad esempio, nelle situazioni più critiche come il cambio di stato di un oggetto o la richiesta di cooperazione, può diventare fondamentale la conferma di ricezione a livello applicativo da parte di tutte le entità coinvolte nella comunicazione.

Per quanto riguarda il problema della latenza, la soluzione migliore è quella di ridurre al minimo le informazioni scambiate tra le varie entità. Le informazioni consisteranno dunque in singoli oggetti, la cui interpretazione spetterà allo strato superiore a quello della comunicazione, ovvero quello cooperativo.

Per quanto riguarda le disconnessioni, verrà messo a punto un sistema di recovery delle informazioni. Quando un utente subirà una caduta di rete, il suo dispositivo dovrà rendersene conto per poi ricontattare il server per ottenere nuovamente le informazioni di gioco (aggiornate a quel momento). In questo modo l'utente rimarrà "fuori dai giochi" per tutto il tempo della disconnessione, ma potrà tornare operativo con tutte le informazioni aggiornate una volta ristabilita la connessione.

## 4.4 Architettura logica generale del sistema

Il sistema sarà basato su un'architettura Client-Server, in cui il Server sarà una piattaforma che fungerà da gestore generale delle informazioni di gioco e delle interazioni, mentre il client sarà costituito dal sottosistema composto da uno smartphone e da un paio di smart-glasses per ogni utente. Lo smartphone fungerà da localizzatore GPS e si occuperà della comunicazione col server. Gli smart-glasses fungeranno da semplice interfaccia per l'utente su cui saranno visualizzate tutte le informazioni necessarie allo svolgimento del gioco.

L'architettura generale del sistema è perciò la seguente:

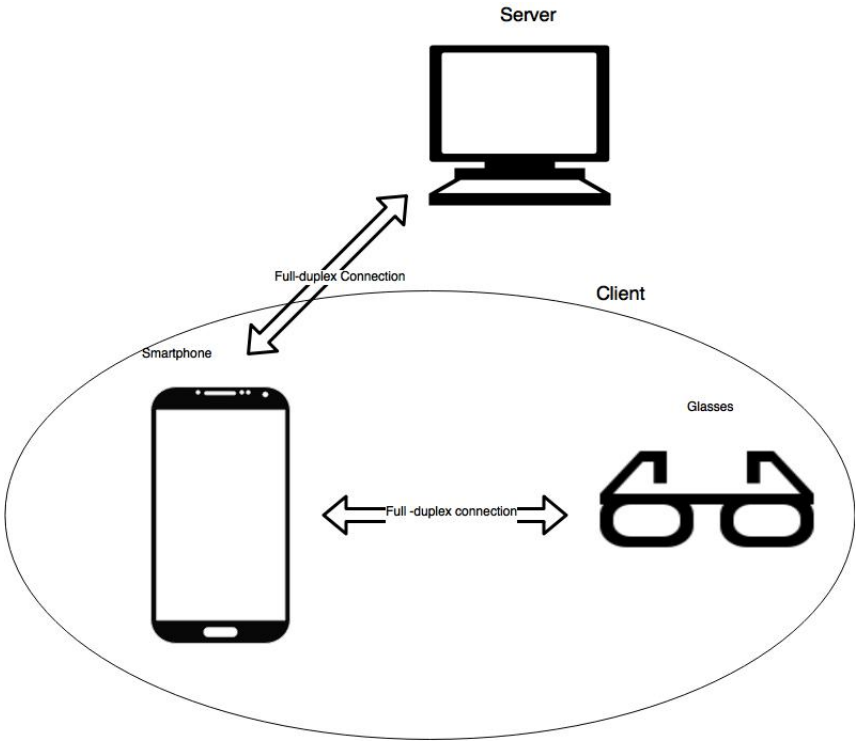


Figura 4.1: Architettura generale del sistema

I software presenti su Server e Client saranno strutturati su tre livelli, dall'alto al basso:

- User Interface Layer
- Cooperation Layer
- Communication Layer

Ognuno di tali strati fornirà API ai livelli superiori. In seguito vengono presentati i due schemi (lato Server e lato Client) che mostrano le principali API fornite dai livelli di Comunicazione e Cooperazione.

### 4.4.1 Architettura logica del sistema lato Server

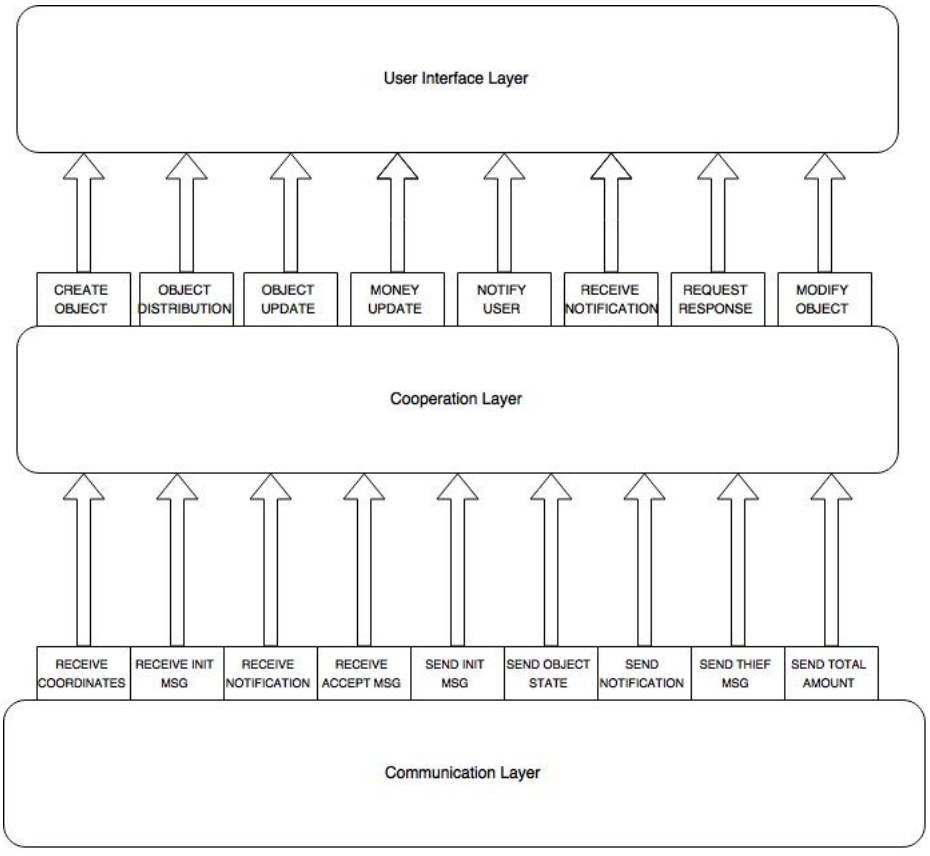


Figura 4.2: Architettura logica del sistema lato server

### 4.4.2 Architettura logica del sistema lato Client

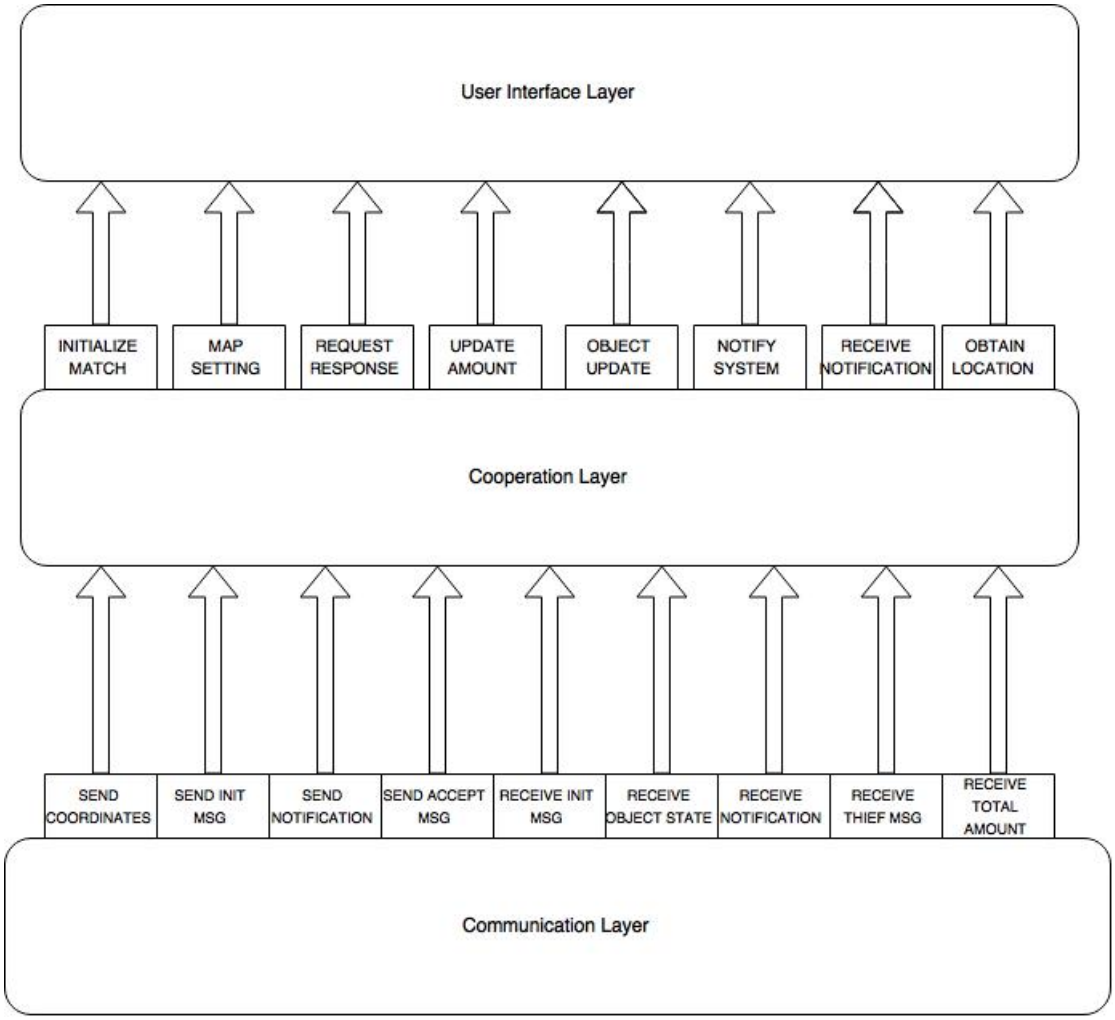


Figura 4.3: Architettura logica del sistema lato client

## 4.5 API fornite allo strato superiore (di cooperazione) lato server

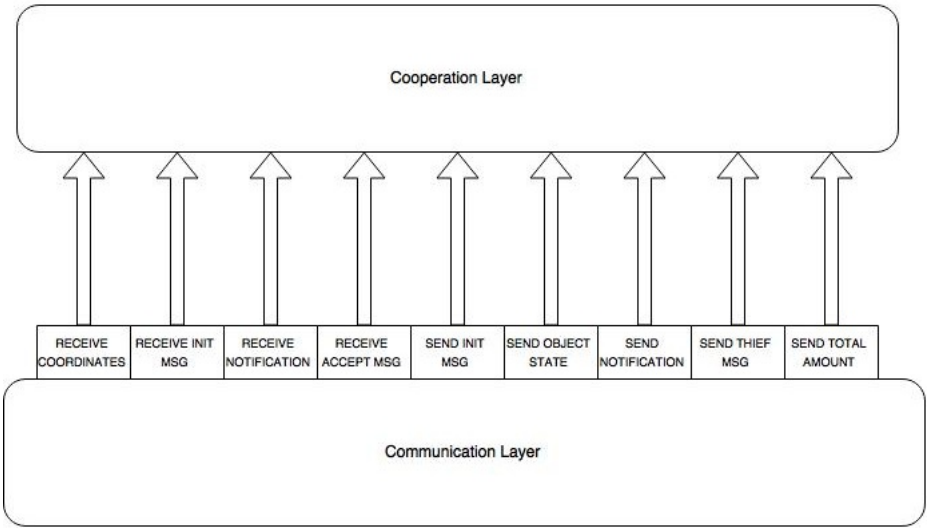


Figura 4.4: API fornite allo strato di cooperazione (SERVER)

Lo strato di cui mi sono occupato (quello di comunicazione e di rete) offre varie API utilizzabili dallo strato superiore (quello di cooperazione) qui sotto descritte:

### 4.5.1 Receive API (API in ricezione):

**RECEIVE COORDINATES:** chiamata effettuata quando il server riceve le coordinate di un giocatore.

**RECEIVE INIT MSG:** chiamata effettuata quando il server riceve la richiesta di inizializzazione da parte di un giocatore.

**RECEIVE NOTIFICATION:** chiamata effettuata quando il server riceve una notifica da parte di un giocatore.

**RECEIVE ACCEPT MSG:** chiamata effettuata quando il server riceve la conferma/rifiuto di cooperazione da parte di un giocatore.

### 4.5.2 Send API (API in invio):

**SEND INIT MSG:** attraverso questa chiamata lo strato superiore può consegnare all'utente desiderato le informazioni iniziali di gioco.

**SEND OBJECT STATE:** attraverso questa chiamata lo strato superiore può consegnare all'utente desiderato le informazioni relative a un oggetto il cui stato è cambiato.

**SEND NOTIFICATION:** attraverso questa chiamata lo strato superiore può consegnare all'utente desiderato una notifica rilasciata dal suo compagno.

**SEND THIEF MSG:** attraverso questa chiamata lo strato superiore può avvertire l'utente desiderato che ha subito un'"aggressione" da parte di un ladro.

**SEND TOTAL AMOUNT:** attraverso questa chiamata lo strato superiore può avvertire gli utenti dell'avvenuta riduzione del premio totale.

## 4.6 API fornite allo strato superiore (di cooperazione) lato client

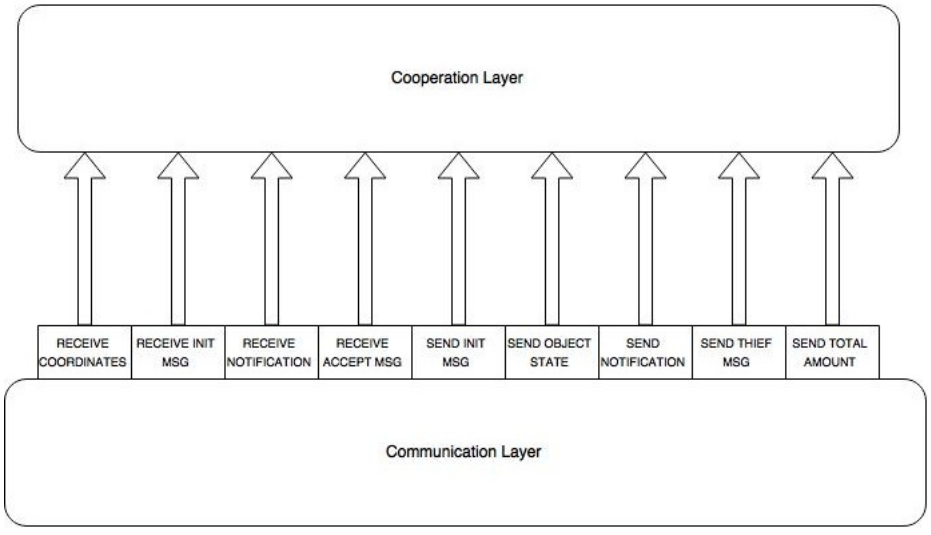


Figura 4.5: API fornite allo strato di cooperazione (CLIENT)

### 4.6.1 Receive API (API in ricezione):

**RECEIVE OBJECT STATE:** chiamata effettuata quando il client riceve un oggetto il cui stato è cambiato.

**RECEIVE INIT MSG:** chiamata effettuata quando il client riceve i dati di inizializzazione da parte del server.

**RECEIVE NOTIFICATION:** chiamata effettuata quando il client riceve una notifica da parte del server.

**RECEIVE THIEF MSG:** chiamata effettuata quando il client riceve il messaggio di avvenuto furto.

**RECEIVE TOTAL AMOUNT:** chiamata effettuata quando il client riceve il messaggio di riduzione dell'ammontare del premio.

### 4.6.2 Send API (API in invio):

**SEND INIT MSG:** attraverso questa chiamata lo strato superiore può richiedere al server le informazioni iniziali di gioco.

**SEND COORDINATES:** attraverso questa chiamata lo strato superiore può consegnare al server le coordinate attuali dell'utente.

**SEND NOTIFICATION:** attraverso questa chiamata lo strato superiore può consegnare al server una notifica.

**SEND ACCEPT MSG:** attraverso questa chiamata lo strato superiore può avvertire il server che l'utente accetta/rifiuta la richiesta di cooperazione.

## 4.7 Composizione dello strato di comunicazione lato Client

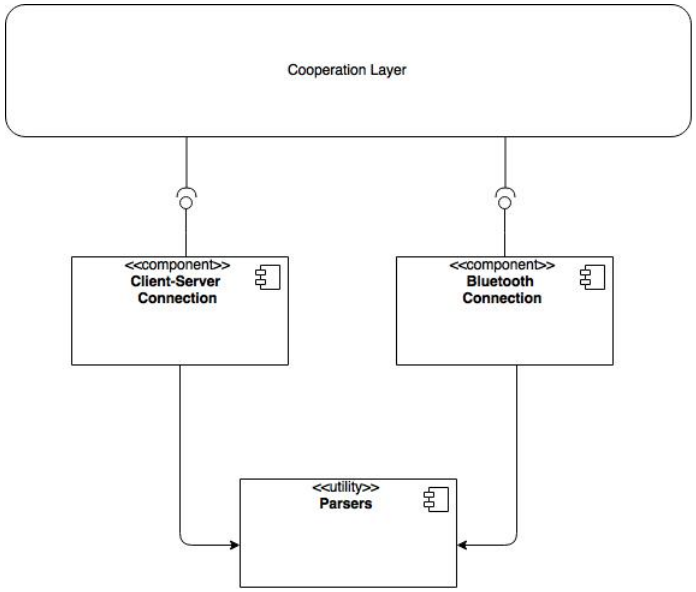


Figura 4.6: Diagramma dei componenti dello strato di comunicazione

Nello strato di comunicazione si possono identificare tre macro-componenti, di cui segue la descrizione.

### 4.7.1 Client-Server Connection

Si tratta del componente che si occupa della connessione tra client e server e di conseguenza dell'invio e ricezione di dati attraverso essa. Esso è composto da un'istanza che mantiene tutte le informazioni riguardanti le connessioni in corso e fornisce le API di invio allo strato superiore e da un'altra istanza bloccata in ricezione che fornirà le API di ricezione allo strato superiore. Sotto il diagramma dei sotto-componenti:

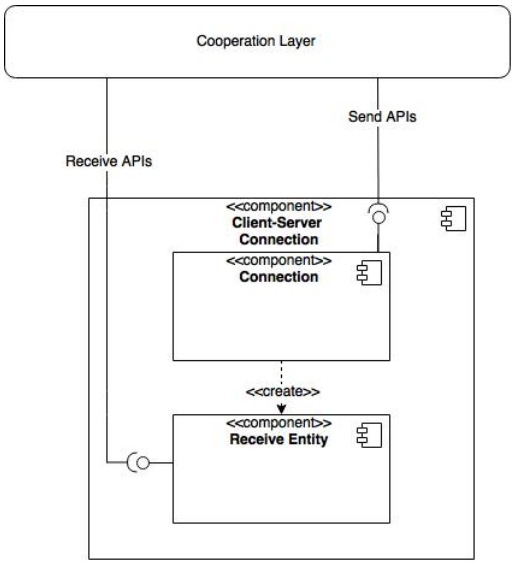


Figura 4.7: Diagramma dei sotto-componenti della connessione Client-Server

L'entità Connection viene invocata dallo strato superiore per poter inviare dati attraverso la connessione client-server. Viene perciò creata dallo strato cooperativo. L'entità in ricezione viene creata da Connection e fornisce le API in ricezione.

Il diagramma di sequenza che segue mostra lo scenario di inizializzazione del gioco, in cui viene invocata la creazione della connessione Client-Server una volta avviato il gioco.

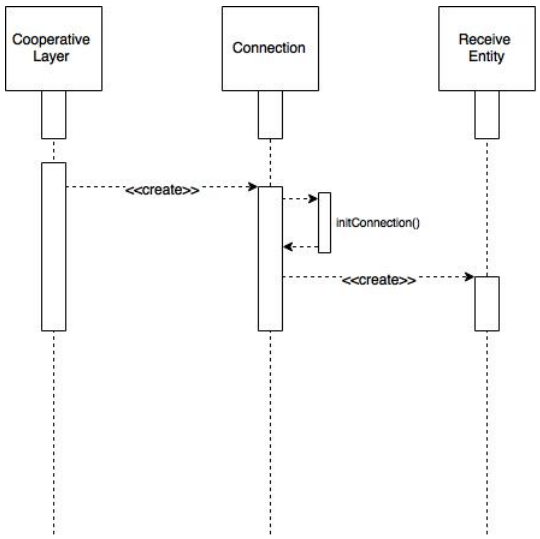


Figura 4.8: Diagramma di sequenza relativo all'inizializzazione della comunicazione

Come si può notare, una volta inizializzata la connessione col server, viene creata l'entità di ricezione utilizzando il canale aperto in seguito alla connessione. A questo punto, lo strato cooperativo è pronto ad utilizzare sia le API di invio sia quelle di ricezione.

Un altro scenario interessante è quello dell'invio di informazioni da parte dello strato cooperativo una volta aperta la connessione. Si analizza tale scenario con un ulteriore diagramma di sequenza:

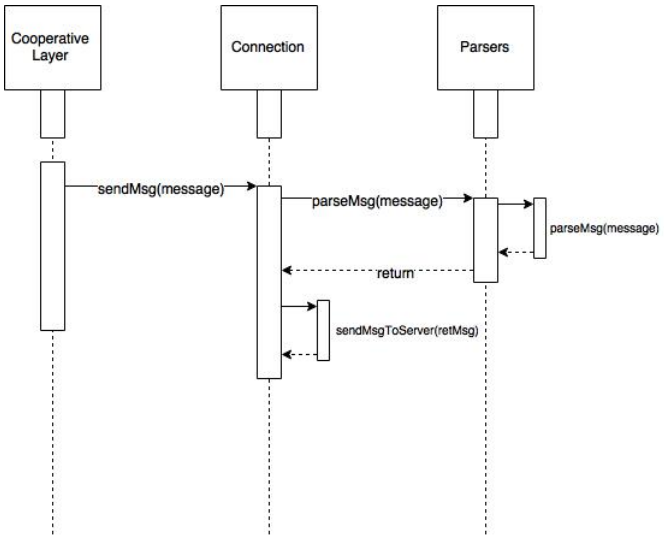


Figura 4.9: Diagramma di sequenza relativo all'invio di messaggi

Il messaggio viene ricevuto dallo strato superiore e ricevuto dal componente Connection. Il componente invoca il metodo di parsing sull'entità Parsers, per ottenere un oggetto sotto forma di stringa in modo tale da inviarlo agilmente al server attraverso la connessione. Una volta ricevuto il valore di ritorno, il messaggio viene spedito.

Per quanto riguarda la ricezione attraverso la connessione si veda il diagramma sottostante:

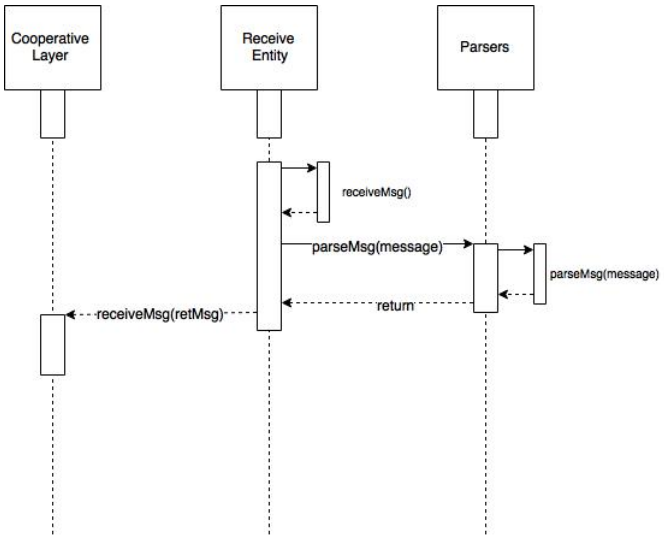


Figura 4.10: Diagramma di sequenza relativo alla ricezione di messaggi

L'entità in ricezione si blocca in attesa dell'arrivo di messaggi. Una volta ricevuto un messaggio, viene invocato un metodo di parsing per ottenere a partire dal messaggio testuale un oggetto da inviare allo strato cooperativo. Il valore di ritorno verrà perciò spedito attraverso l'apposita API.

### 4.7.2 Bluetooth Connection

Si tratta del componente che si occupa della connessione tra smartphone e glasses e di conseguenza dell'invio e ricezione di dati attraverso essa. Esso è composto da un'istanza che si occupa di stabilire la connessione Bluetooth tra i due dispositivi e da un'istanza che mantiene attiva la connessione e fornisce API in invio e ricezione. Sotto il diagramma dei sotto-componenti:

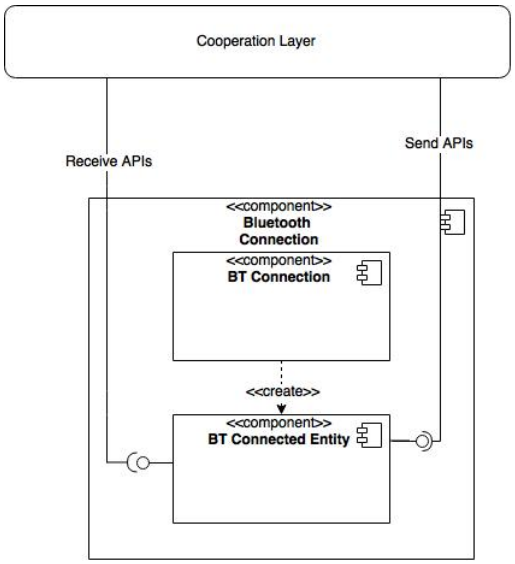


Figura 4.11: Diagramma dei sotto-componenti della connessione Bluetooth

L'entità BT Connection viene invocata dallo strato collaborativo quando è richiesta l'apertura della connessione Bluetooth. Tale entità, una volta aperta la connessione, crea una seconda entità BT Connected Entity, che si occuperà di mantenere attiva la connessione e sia di fornire direttamente le API di invio (Send APIs), sia di gestire le API di ricezione rimanendo in ascolto in modo bloccante sulla connessione in entrata. Per comprendere al meglio l'inizializzazione della connessione Bluetooth si veda il seguente diagramma di sequenza, in cui è compreso lo scenario della ricezione di un messaggio:

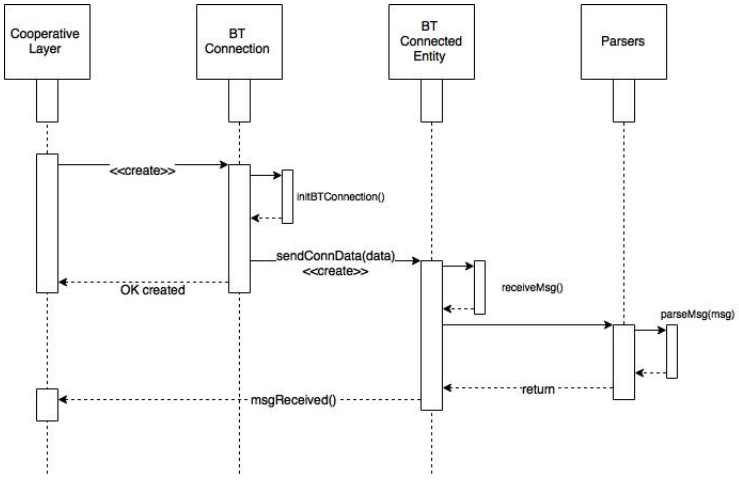


Figura 4.12: Diagramma di sequenza relativo all'inizializzazione e alla ricezione di messaggi

Lo strato cooperativo invoca la creazione di una connessione Bluetooth. Una volta stabilita la connessione, viene creata la BT Connected Entity, e una sua istanza viene restituita allo strato chiamante. Tale entità si pone in attesa di ricevere messaggi attraverso la connessione. Una volta ricevuto un messaggio si procede con il parsing per ottenere un oggetto da fornire allo strato cooperativo. A questo punto, una volta ottenuto l'oggetto, l'apposita API di ricezione permette allo strato superiore di ottenere i dati.

Lo scenario di invio di dati è descritto sotto:

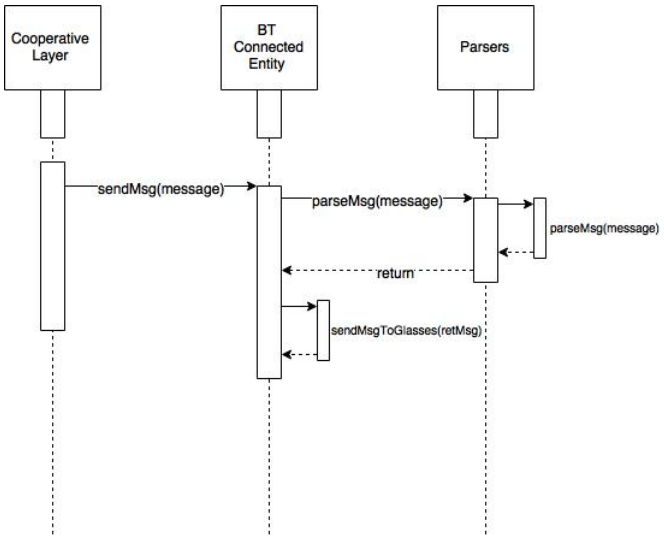


Figura 4.13: Diagramma di sequenza relativo all'invio di messaggi

Lo strato cooperativo invoca l'entità attiva della connessione Bluetooth passandole il messaggio da inviare. A questo punto avviene il parsing per poter ottenere i dati testuali da inviare attraverso la connessione. Una volta ottenuti i dati l'entità li spedisce al destinatario.

### 4.7.3 Parsers

Tale componente si occupa di effettuare le operazioni di parsing sugli oggetti che gli vengono inviati. Tali operazioni sono necessarie per poter inviare i dati attraverso le connessioni in maniera snella e veloce. Le operazioni di parsing avvengono in due direzioni:

**Da oggetto a forma testuale** Gli oggetti che devono essere inviati vengono passati dallo strato di cooperazione allo strato di comunicazione. Tali oggetti subiranno il parsing per poter essere effettivamente spediti sulla connessione.

**Da forma testuale a oggetto** I messaggi ricevuti devono essere "tradotti" in oggetti, per cui si procede all'operazione contraria rispetto alla precedente.

## 4.8 Composizione dello strato di comunicazione lato Server

Lato Server viene necessariamente a mancare il componente relativo alla connessione Bluetooth, mentre persistono i componenti di connessione tra client e server e di parsing. Il componente relativo alla gestione della connessione si differenzia però rispetto a quello lato client, in quanto deve essere in grado di gestire più connessioni simultanee. Analizziamo la sua struttura interna:

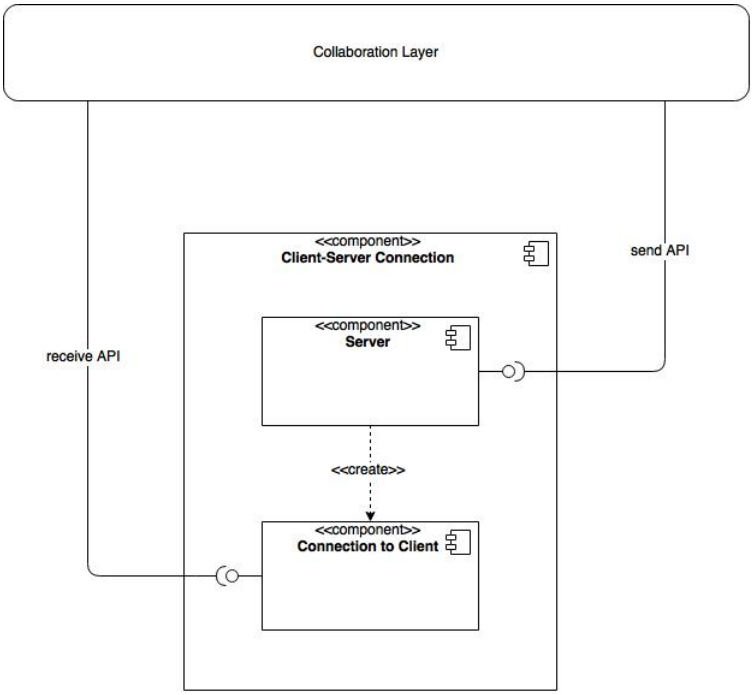


Figura 4.14: Diagramma dei sotto-componenti della connessione Client-Server

La connessione è gestita da due sotto-componenti : Server e Connection to Client.

Server si comporta come un vero e proprio server, ovvero si pone in ascolto di richieste di connessione per tutto il suo ciclo di vita. Ogni volta che riceve una richiesta, Server crea un'istanza di tipo Connection to Client. Connection to Client gestisce la connessione con uno specifico client. Fornisce perciò le API

in ricezione per quella particolare connessione, per cui lo strato di cooperazione saprà immediatamente la fonte dei messaggi una volta ricevuti.

Le API in invio sono offerte dall'entità Server, che smista i messaggi a una delle Connection to Client in base al destinatario. Si analizza a titolo di esempio lo scenario di una richiesta di connessione con conseguente ricezione di dati.

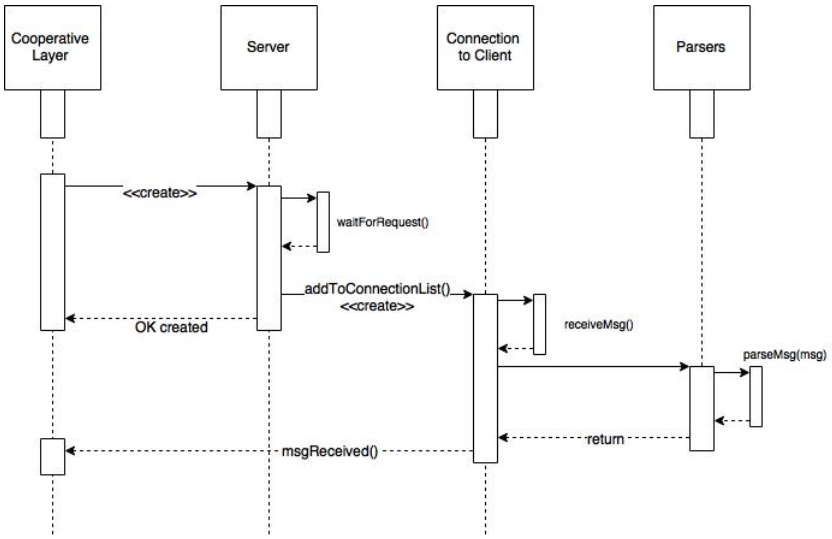


Figura 4.15: Diagramma di sequenza che mostra l'inizializzazione della connessione e la ricezione di messaggi

Lo strato di cooperazione invoca la creazione di un'entità Server. Tal entità si pone in attesa di richieste di connessione. Una volta ricevuta una richiesta, viene creata una nuova Connection to Client, la quale viene aggiunta alla lista delle connessioni attive. Tale nuova entità si pone in attesa di ricevere un messaggio, e una volta ricevuto ne effettua il parsing e lo fornisce allo strato di cooperazione attraverso le API di ricezione.

Per quanto riguarda l'invio di messaggi dal server a un determinato client, si vada lo schema sottostante:

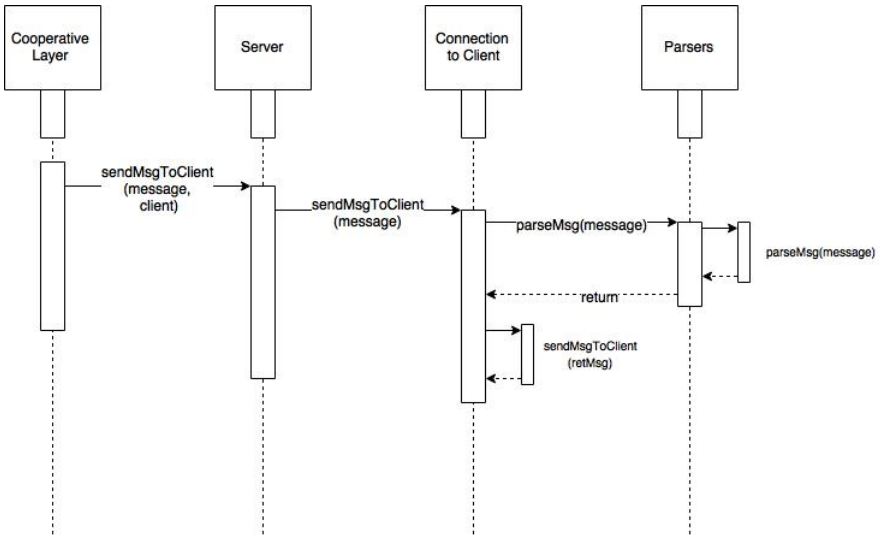


Figura 4.16: Diagramma di sequenza che mostra l'invio di messaggi

L'entità Server viene invocata dallo strato cooperativo specificando il client al quale inviare il messaggio. Server seleziona la Connection to Client corrispondente e demanda il compito di invio a tale entità. A quel punto l'entità di connessione effettua il parsing e spedisce il messaggio testuale al client.

## 4.9 Soluzioni architetturali al problema delle disconnessioni

Si è già introdotto precedentemente il problema dovuto alle possibili disconnessioni del client, e si è proposto un primo abbozzo di soluzione. Si va ora ad approfondire tale abbozzo illustrando le soluzioni architetturali necessarie.

L'idea è quella di attuare un meccanismo di recovery, ovvero un metodo per recuperare tutte le informazioni del gioco anche rimanendo disconnessi dalla partita per periodi più o meno lunghi. Con un meccanismo di questo tipo si può fare sì che ogni utente disconnesso non perda tutti gli eventi accaduti durante la sua assenza (per esempio, cambi di stato di oggetti, riduzione dell'importo). Si illustrano qui sotto le soluzioni studiate per implementare un meccanismo di questo tipo sia lato Server sia lato Client.

### 4.9.1 Soluzioni architetturali lato server

Lato server, sarà lo strato cooperativo a rendersi conto dell'avvenuta disconnessione di un utente, implementando meccanismi di timeout per la mancata ricezione di messaggi da parte di un giocatore. Una volta riconosciuta la

disconnessione di un utente, l'entità Server viene invocata con la richiesta di distruggere l'entità Connection to Client relativa all'utente disconnesso. Una volta effettuata tale operazione, l'entità Server continua ad attendere le connessioni di nuovi utenti, tra cui anche l'utente disconnesso. Una volta che il dispositivo del giocatore si sarà riconnesso, la richiesta sarà ricevuta da Server, che andrà ad aggiungere alla lista delle connessioni quella appena creata. A questo punto è ristabilita la connessione tra client e server e si può procedere prima col re-invio delle informazioni poi con il naturale svolgimento del gioco.

Con il seguente diagramma di sequenza si illustra lo scenario appena descritto:

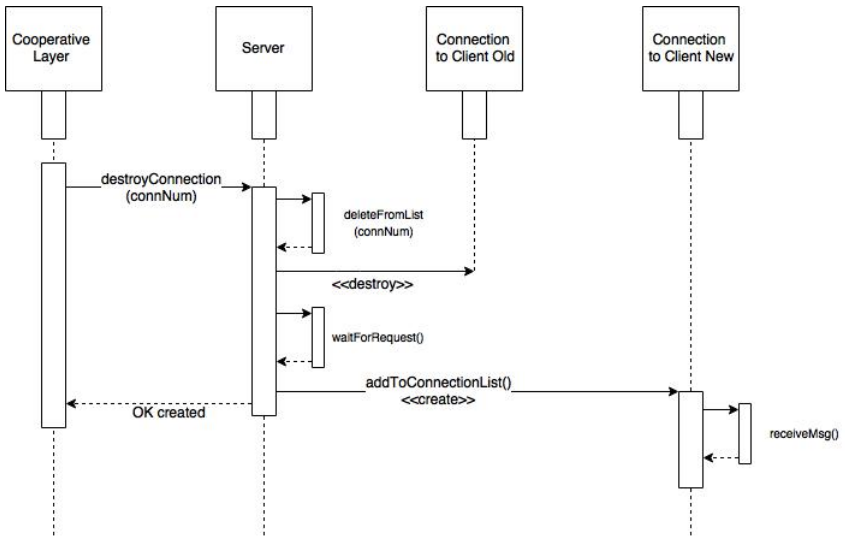


Figura 4.17: Diagramma di sequenza che mostra il recovery lato Server

Da notare che lo strato di cooperazione riceverà l'informazione relativa all'identità dell'utente solamente dopo l'avvenuta connessione, ovvero una volta ricevuto il messaggio di inizializzazione.

### 4.9.2 Soluzioni architetturali lato client

Lato Client, spetterà allo strato di comunicazione il compito di individuare la disconnessione dell'utente. Questo avviene controllando ad ogni invio di messaggio se c'è l'effettiva presenza di connettività di rete. Questo stratagemma assicura di evitare incongruenze nell'ottica di sistema condiviso. Infatti, le interazioni con il mondo condiviso si manifestano esclusivamente attraverso l'invio e ricezione di messaggi. L'essere offline significa non avere la possibilità di inviare o ricevere messaggi, per cui in un periodo di disconnessione un utente non può ricevere informazioni errate o non corrispondenti all'effettivo stato del

mondo condiviso. Oltretutto, è assicurata anche un'adeguata responsiveness, in quanto il periodo massimo di disconnessione senza conseguente consapevolezza da parte dello strato di cooperazione di essere effetttivamente offline è limitato inferiormente dal gap massimo di tempo che può intercorrere tra l'invio di due messaggi consecutivi. Ma tale periodo è limitato dalla frequenza di invio di messaggi relativi alle proprie coordinate GPS (dell'ordine di qualche secondo). Per cui un utente temporaneamente offline può non essere consapevole di esserlo nel peggiore dei casi per pochi secondi.

Nel caso in cui venga riscontrata l'effettiva assenza di connessione, si procede nel modo seguente:

**Si distrugge la connessione tra utente e server:** le entità Connection e Receive Entity vengono eliminate, in quanto è necessario ripristinare la connessione da zero.

**Si informa lo strato superiore dell'avvenuta disconnessione:** lo strato di cooperazione viene notificato dell'avvenuta disconnessione attraverso un messaggio spedito attraverso un'ulteriore API

**Si crea un'entità Recovery:** il suo compito sarà quello di monitorare il ripristino della connettività di rete. Una volta ripristinata che il dispositivo sarà uscito dal periodo di disconnessione, tale entità dovrà informare lo strato superiore riguardo la riconnessione dell'utente.

**Si ricrea l'entità Connection, e successivamente l'entità Receive Entity:** si procede dunque con la creazione delle due entità, esattamente come all'avvio del gioco.

**Il primo messaggio da inviare al server sarà un messaggio di inizializzazione:** in questo modo verranno ricevute le informazioni aggiornate relative allo svolgimento del gioco. Il diagramma che segue mostra lo scenario di disconnessione dell'utente:

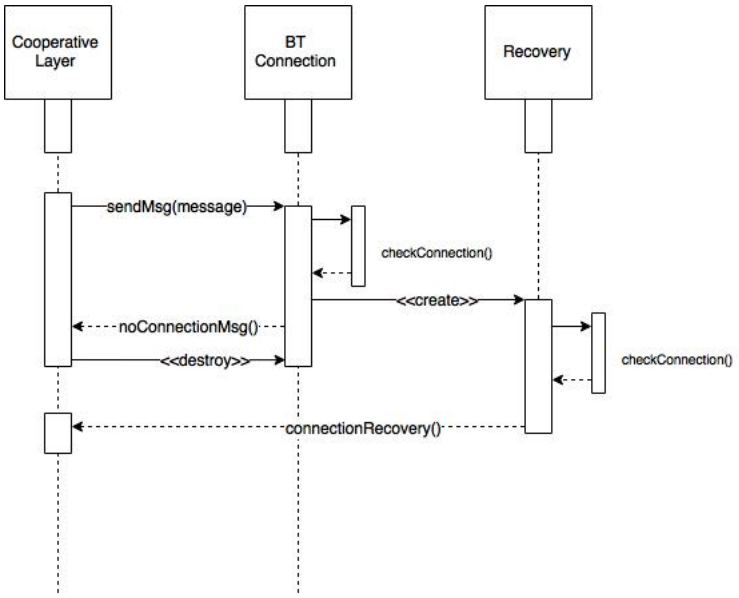


Figura 4.18: Diagramma di sequenza che mostra il recovery lato Client

Da notare che una volta ricevuto il messaggio di avvenuto ripristino della connettività, lo strato di cooperazione potrà ricreare la connessione esattamente come all'inizio del gioco.

## 4.10 Progettazione User Interface Layer

Lo strato User Interface si occuperà di fornire ai giocatori e ai gestori del gioco tutte le funzionalità necessarie al suo svolgimento.

Lato server, dovrà poter gestire la creazione del gioco con tutte le informazioni relative agli oggetti da posizionare nella mappa di gioco (con relative latitudine e longitudine, contenuto ed eventali informazioni aggiuntive). Oltretutto, dovrà permettere l'inserimento delle entità nemiche e le eventuali riduzioni dell'ammontare di denaro disponibile.

Lato client, lo strato User Interface si occuperà di fornire all'utente tutte le informazioni relative allo svolgimento del gioco, permettendone la visualizzazione sullo schermo dei glasses. Dovrà inoltre intercettare gli eventi causati dall'utente, come il rilascio di notifiche e le risposte alle richieste di cooperazione.

### 4.10.1 Lato Server

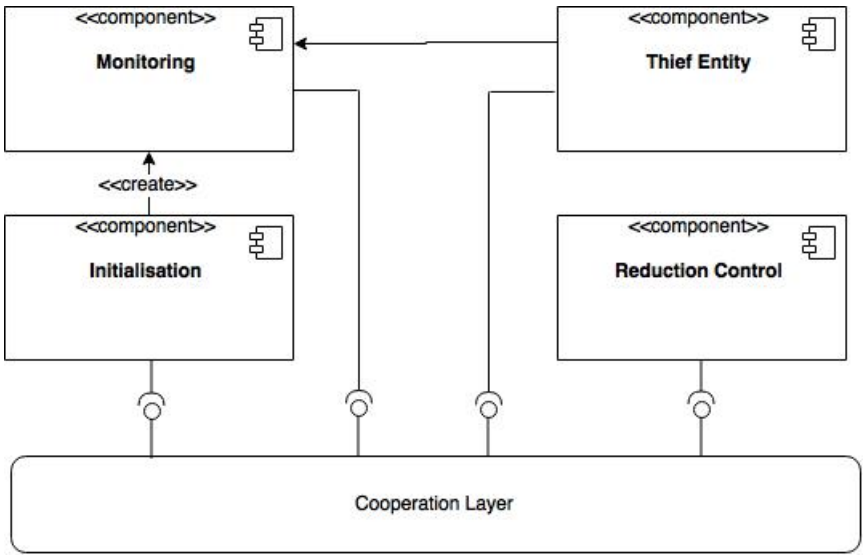


Figura 4.19: Diagramma componenti strato User Interface lato server

I componenti principali dello strato User Interface sono i seguenti:

- Initialisation
- Thief Entity
- Monitoring
- Reduction Control

**Initialisation:** è il componente che si occupa dell'inizializzazione del gioco. I gestori della partita potranno impostare tutte le informazioni attraverso un'apposita schermata.

**Thief Entity:** è il componente che rappresenta l'entità nemica del gioco. Tale entità cambia la sua posizione entro un raggio predefinito dai gestori del gioco in fase di inizializzazione.

**Monitoring:** è il componente che si occupa del monitoring del gioco. Permette ai gestori del gioco di visualizzare in tempo reale tutte le informazioni sulla posizione dei giocatori e delle entità nemiche e

**Reduction Control:** è l'entità che gestisce la riduzione del premio totale disponibile. Tale riduzione può essere temporizzata (dopo un certo periodo avviene automaticamente) oppure pilotata direttamente dai gestori del gioco.

### 4.10.2 Lato Client

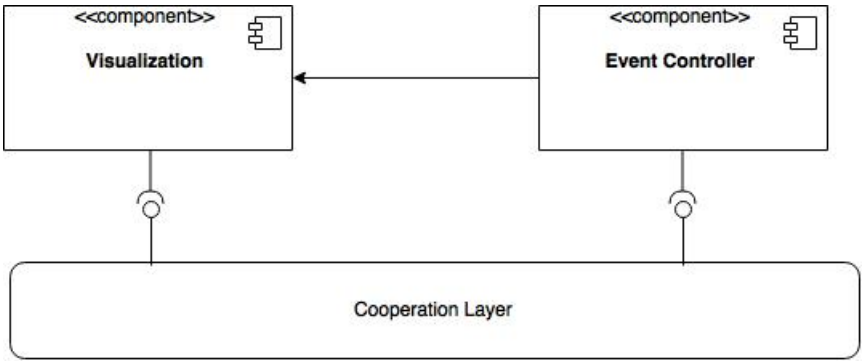


Figura 4.20: Diagramma componenti strato User Interface lato client

I componenti principali dello strato User Interface sono i seguenti:

- Visualization
- Event Controller

**Visualization:** è l'entità che gestisce la visualizzazione delle informazioni sullo schermo dei glasses dei giocatori. In base alle informazioni ricevute dallo strato di cooperazione le schermate saranno aggiornate.

**Event Controller:** intercetta gli eventi degli utenti, passandoli allo strato di cooperazione.

# Capitolo 5

# Sviluppo

## 5.1 Tecnologie utilizzate

Per lo sviluppo sono state utilizzate tecnologie dell'ambito mobile e gli smart-glasses Moverio BT-200 prodotti dalle Epson.

**Moverio BT-200** I Moverio BT-200 smart glasses sono una tecnologia molto recente ed offrono un supporto interessante alla realtà aumentata. Presentano un sistema operativo Android API 15, per cui sono programmabili come un qualsiasi dispositivo con tale sistema operativo.

I Moverio sono visori binoculari con lenti trasparenti. Questa loro caratteristica permette di sovrapporre i contenuti, visualizzati da entrambi gli occhi, alla realtà circostante così fornendo una nuova forma di realtà aumentata. In più questa sua caratteristica permette la "renderizzazione" di oggetti 3d con un maggiore coinvolgimento dell'utente. I glasses sono composti da visore e unità di controllo (un telecomando collegato al corpo dei glasses). Sul visore sono presenti diversi sensori tra cui accelerometri e bussola digitale e una videocamera. Una volta indossati, i glasses permettono di poter vedere uno schermo virtuale, come "proiettato" sull'orizzonte della grandezza di circa 50 pollici ad una distanza di 5 metri. Su tale schermo viene proiettata una view simile a quella di un comune tablet Android versione 4.0.4 in modalità landscape. Tramite l'unità di controllo touch-screen è possibile muovere un cursore sullo schermo per selezionare le azioni da intraprendere.

## 5.2 Linguaggi utilizzati

Lo sviluppo del progetto ha avuto luogo utilizzando come linguaggi di programmazione Java e Android.

Lato server è stata sviluppata un'applicazione Java-based utilizzando come IDE Eclipse.

Lato client sono state sviluppate due applicazioni Android, una su Smartphone e una su smart-glasses Moverio BT-200.

## 5.3 Gestione delle connessioni

Per lo sviluppo della connessione client-server sono state utilizzate delle socket TCP. La scelta di TCP è dovuta al fatto che data la natura del sistema è stato necessario utilizzare un protocollo affidabile e connection-oriented.

La connessione tra Smartphone e glasses è stata sviluppata utilizzando il protocollo Bluetooth.

## 5.4 Scelte implementative

I principali componenti legati alle connessioni sono stati implementati come thread.

**Lato Server** un thread (**ServerThread**) rimane in attesa di nuove connessioni su una **ServerSocket** e una volta ricevuta una richiesta di connessione fa partire un secondo thread di tipo **ConnectionThread** che gestirà la connessione col singolo client, attraverso l'utilizzo di un oggetto di tipo **Socket**.

```
package model;

import java.io.IOException;
import java.net.ServerSocket;
import java.util.ArrayList;
import java.util.List;

public class ServerThread implements Runnable {
   List<ConnectionThread> clients;
   ServerSocket listenSock;

   public ServerThread() throws IOException{
      final int port = 6789;
      this.clients = new ArrayList<>();
      this.listenSock = new ServerSocket(port);
   }

   @Override
```

```
   public void run() {
      while (true) {
         ConnectionThread server;
         try {
            server = new ConnectionThread(listenSock.accept());
            this.clients.add(server);
            Thread thread = new Thread(server);
            thread.start();
         } catch (IOException e) {
            e.printStackTrace();
         }

      }
   }

}
```

Listato 5.1: ServerThread.java

Il thread appena creato viene aggiunto alla lista clients, che tiene conto di tutte le connessioni aperte con i vari utenti.

La classe **ConnectionThread** rappresenta un thread che attende la ricezione di messaggi da parte dell'utente con cui è aperta la socket. Una volta ricevuto il messaggio, questo viene passato al metodo process(String s), che ne effettua il parsing e lo passa allo strato di cooperazione.

```
package model;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.*;

import org.json.JSONException;
import org.json.JSONObject;

import parsers.ParsersUtils;

public class ConnectionThread implements Runnable {
   Socket sock;
   InputStream recvStream;
   OutputStream sendStream;
   String request;
   String response;
```

```
ConnectionThread(Socket s) throws IOException {
   this.sock = s;
   this.recvStream = sock.getInputStream();
   this.sendStream = sock.getOutputStream();
   System.out.println("Connected on local port: " +
       s.getLocalPort());
   System.out.println("Connected from client with IP "
        + (s.getInetAddress()).getHostAddress() + " : " +
           s.getPort());
}

public void run() {
   while (true) {
      getRequest();
      process();
   }
}

void getRequest() {
   try {
      int dataSize;
      byte[] recvBuff = new byte[1024];
      dataSize = recvStream.read(recvBuff, 0, 1024);
      request = new String(recvBuff, 0, dataSize);
      System.out.println("Received: " + request + dataSize);
   } catch (IOException ex) {
      System.err.println("IOException in getRequest");
   }
}
```

Listato 5.2: ConnectionThread.java

La classe **ConnectionThread** fornisce le API in invio per poter spedire messaggi al client attraverso la socket. Eccone alcuni esempi:

```
public void sendMatchToClient(Match match, Player player) throws
    JSONException, IOException{
   JSONObject jsonToSend = ParsersUtils
        .getMatchJSONObject(match, player);
   String stringToSend = jsonToSend.toString() + '\n';
   byte[] sendBuff = new byte[stringToSend.length()];
   sendBuff = stringToSend.getBytes();
   sendStream.write(sendBuff, 0, sendBuff.length);
```

```
}

public void sendTreasureChestToClient(TreasureChest
    treasureChest) throws JSONException, IOException{
   JSONObject jsonToSend = ParsersUtils
         .getTreasureChestJSONObject(treasureChest);
   String stringToSend = jsonToSend.toString() + '\n';
   byte[] sendBuff = new byte[stringToSend.length()];
   sendBuff = stringToSend.getBytes();
   sendStream.write(sendBuff, 0, sendBuff.length);
}

public void sendConfirmRefuseCooperation(int idPlayer,String
    toSend) throws JSONException, IOException{
   JSONObject jsonToSend = ParsersUtils
         .getResponseJSONObject(idPlayer, toSend);
   String stringToSend = jsonToSend.toString() + '\n';
   byte[] sendBuff = new byte[stringToSend.length()];
   sendBuff = stringToSend.getBytes();
   sendStream.write(sendBuff, 0, sendBuff.length);
}
```

Listato 5.3: ConnectionThread.java send API

Si può notare che il parsing dei messaggi è stato impostato sulla libreria JSON, che offre la possibilità di convertire oggetti in stringhe in maniera veloce e standardizzata.

**Lato Client** viene utilizzata una classe **TCPConnection**, il cui compito è quello di instaurare la connessione col server e di far partire un thread di tipo **TCPClientThread**. **TCPConnection** fornisce le API in invio, mentre **TCPClientThread** notifica lo strato collaborativo attraverso **Handler** ad ogni ricezione di messaggio da parte del server. Nel codice sottostante compaiono anche le API in invio relative all'invio della posizione e della conferma o rifiuto di cooperazione al server.

```
package it.unibo.bbc_smartphone.tcp_connection;

import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.net.Socket;

import it.unibo.bbc_smartphone.ParserUtils;
import it.unibo.bbc_smartphone.activity.MainActivity;
import it.unibo.bbc_smartphone.model.Alert;

public class TCPConnection extends Thread {
    private DataOutputStream outToServer;
    private MainActivity.TCPConnectionHandler tcpConnectionHandler;
    private static final String serverIP = "192.168.2.104";
    private static final int port = 6789;

    public TCPConnection(MainActivity.TCPConnectionHandler
        tcpConnectionHandler) {
        this.tcpConnectionHandler = tcpConnectionHandler;
    }

    @Override
    public void run(){
        Socket clientSocket = null;
        try {
            clientSocket = new Socket(serverIP, port);
            outToServer = new
                DataOutputStream(clientSocket.getOutputStream());
            BufferedReader inFromServer = new BufferedReader(new
                InputStreamReader(clientSocket.getInputStream()));
            TCPClientThread tcpClientThread = new
                TCPClientThread(inFromServer, tcpConnectionHandler);
            tcpClientThread.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void sendPositionToServer(long latitude, long longitude,
        int idPlayer) throws JSONException, IOException {
        JSONObject jsonObject =
            ParserUtils.getPositionToSend(latitude,
            longitude,idPlayer);
        String stringToSend = jsonObject.toString() + '\n';
        outToServer.writeBytes(stringToSend);
    }
    public void sendConfirmRefuseCooperationToServer(String toSend,
```

```
        int idPlayer) throws JSONException, IOException {
        JSONObject jsonObject =
            ParserUtils.getConfirmOrRefuseCooperationMsg(toSend,
            idPlayer);
        String stringToSend = jsonObject.toString() + '\n';
        outToServer.writeBytes(stringToSend);
    }
```

Listato 5.4: TCPConnection.java

La classe **TCPClientThread** è implementata come segue:

```
package it.unibo.bbc_smartphone.tcp_connection;

import android.util.Log;

import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.IOException;

import it.unibo.bbc_smartphone.ParserUtils;
import it.unibo.bbc_smartphone.activity.MainActivity;

public class TCPClientThread extends Thread {
    private BufferedReader inFromServer;
    private MainActivity.TCPConnectionHandler tcpConnectionHandler;
    public TCPClientThread(BufferedReader inFromServer,
        MainActivity.TCPConnectionHandler tcpConnectionHandler){
        this.inFromServer = inFromServer;
        this.tcpConnectionHandler = tcpConnectionHandler;
    }

    @Override
    public void run() {
        while (true){
            try {
                String s = inFromServer.readLine();
                this.receiveMessage(s);
            } catch (IOException e) {
                e.printStackTrace();
            } catch (JSONException e) {
                e.printStackTrace();
            }
```

```
    }
}
```

Listato 5.5: TCPClientThread.java

Il thread rimane in attesa della ricezione di messaggi da parte del server e una volta ricevuti dati sullo stream in input li elabora attraverso il metodo *receiveMessage(String s)*. Il metodo ha la seguente struttura:

```
private Object receiveMessage(String s) throws JSONException {
    JSONObject jsonObject = new JSONObject(s);
    int messageType = -1;
    Object object = null;
    switch (jsonObject.getInt("messageType")) {
        case 0:
            messageType=0;
            object =
                ParserUtils.getMatchFromJSONObject(jsonObject);
            break;
        case 1:
            messageType=1;
            object =
                ParserUtils.getTreasureChestFromJSONObject(jsonObject);
            break;
        case 2:
            messageType=2;
            object =
                ParserUtils.getConfirmedOrRefused(jsonObject);
            break;
        case 3:
            messageType=3;
            object =
                ParserUtils.getAlertFromJSONObject(jsonObject);
            break;
        case 4:
            messageType=4;
            object = ParserUtils.getMoneyTheft(jsonObject);
            break;
        case 5:
            messageType=5;
            object = ParserUtils.getNewAmount(jsonObject);
            break;
    }
```

```
        this.tcpConnectionHandler.obtainMessage(messageType,
            object).sendToTarget();
        return object;
    }
```

Listato 5.6: TCPClientThread.java receive API

Il tipo del messaggio è individuato tramite il primo campo dell'oggetto JSON ricevuto. In base al tipo, si effettua il relativo parsing dell'oggetto per poi inviarlo all'**Handler** che permetterà allo strato di cooperazione di ricevere il messaggio.

# Capitolo 6

# Valutazioni finali

L'applicazione è stata sviluppata in base alle fasi di analisi, progettazione e prototipazione descritte nei capitoli precedenti. Le funzionalità principali del sistema sono state implementate con successo ed è stato possibile verificarne l'effettivo funzionamento sul campo. L'applicazione ha risposto bene ai requisiti e agli scenari descritti attraverso i casi d'uso.

In particolare, l'obiettivo di creare un sistema condiviso di informazioni è stato pienamente raggiunto, in quanto l'esperienza di ogni utente è risultata essere coerente con quella dei compagni, aspetto fondamentale e basilare dell'intero gioco.

Per quanto riguarda le problematiche introdotte nel capitolo relativo alla progettazione,esse sono state risolte in maniera considerevolmente soddisfacente. La latenza di gioco non ha rappresentato un problema, in quanto la scelta di ridurre al minimo le informazioni da inviare ad ogni scambio di messaggi si è rivelata vincente per snellire la comunicazione e fornire un'esperienza reattiva. Chiaramente, questo aspetto è limitato anche dalle tecnologie disponibili. In uno scenario di gioco abbastanza ampio che richiede molto spesso l'utilizzo di reti 3G la qualità del segnale risulta cruciale e in alcuni casi limitante per un sistema condiviso. Nel nostro caso comunque non ha quasi mai rappresentato un problema impedente.

L'affidabilità della comunicazione è stata ottenuta affidandosi ad un protocollo connection-oriented e affidabile come TCP. La scelta di questo protocollo nasce proprio dall'esigenza di avere la certezza che tutti i messaggi inviati giungano a destinazione. Una possibile alternativa avrebbe potuto essere quella di utilizzare un protocollo snello come UDP, per ridurre ulteriormente la latenza, ma sarebbe stato necessario introdurre numerosi controlli a livello di applicazione. Infatti nell'economia del sistema la perdita anche di un solo pacchetto può risultare cruciale e causare la perdita di coerenza dell'intero gioco. Basti pensare ai pacchetti che registrano il cambiamento di stato di un oggetto o la

conferma/rifiuto di cooperazione: la perdita di una di queste informazioni può compromettere la cooperazione e la coerenza del mondo condiviso. Oltretutto tali controlli sarebbero stati necessari per ogni singolo pacchetto inviato, in quanto anche i pacchetti inviati più frequentemente (quelli contenenti la posizione attuale dell'utente) devono necessariamente essere ricevuti dal server per evitare che un giocatore giunga nei pressi di un oggetto senza esserne notificato. Inoltre, la creazione di un canale di comunicazione semplifica l'invio di informazioni dal server al client. Anche l'utilizzo di un protocollo di più alto livello come HTTP avrebbe rappresentato una valida alternativa, ma il rischio sarebbe stato quello di aggiungere ai messaggi informazioni non necessarie al nostro caso di studio.

Per quanto riguarda il problema dovuto alle disconnessioni, la soluzione proposta si è rivelata solida per quanto riguarda la coerenza del mondo condiviso, in quanto la possibilità da parte degli utenti di riottenere le informazioni aggiornate una volta ripristinata la connessione elimina possibili problemi di non correttezza dei dati. Un problema emerge dallo scenario di richiesta di cooperazione. Difatti, se viene inviata una richiesta di cooperazione da parte di un utente ad un suo compagno proprio nel momento in cui il compagno è offline, si rischia che l'utente rimanga in attesa a lungo tempo di una risposta che potrebbe non arrivare. Questo problema è stato risolto (come descritto nel capitolo di progettazione) notificando tutti i giocatori dell'avvenuta disconnessione di uno dei compagni qualora avvenga tale circostanza. Con questa soluzione l'utente che invia una richiesta, una volta resosi conto che effettivamente il destinatario della richiesta è offline, può evitare di attendere una risposta.

Il meccanismo proposto però fa sorgere un ulteriore problema, ovvero il fatto che un utente disconnesso dal gioco ne rimane totalmente escluso per tutta la durata della disconnessione. Questo problema potrebbe essere risolto introducendo nuovi meccanismi di gioco. Per esempio, durante una disconnessione, l'utente potrebbe avere la possibilità di tornare in un punto dove è presente un'entità nemica e grazie alla sua invisibilità riappropriarsi del tesoro precedentemente rubato. Una volta tornato online, questa riappropriazione verrebbe notificata al server.

E' però chiaro che qualsiasi possibile soluzione al problema delle disconnessioni in un sistema distribuito di questo tipo presenterà in ogni caso dei limiti.

Nell'architettura logica del sistema descritta alla fine del capitolo sull'analisi e la modellazione si sono identificata tre livelli: communication layer, cooperation layer e user interface layer. Nel nostro caso di studio buona parte dei meccanismi di gioco vengono già modellati nei primi due livelli, difatti il livello di user interface ha lo come principali funzioni quelle di visualizzazione

dei dati lato utente e di monitoring lato server. Si può però pensare di generalizzare i due livelli, astraendoli dal particolare caso di studio per far sì che vengano a far parte di un middleware che possa fornire API e strumenti per applicazioni con requisiti simili a quelli della nostra.

In questo modo si offrirebbero due livelli a supporto di un terzo livello che potrebbe essere definito application layer, in cui verrebbero implementate tutte le funzionalità della specifica applicazione. Le API fornite a questo livello potrebbero essere API di gestione di oggetti aumentati le cui informazioni andrebbero condivise tra vari utenti. Tali oggetti dovrebbero però essere personalizzabili a livello di applicazione, per permettere di sviluppare sistemi molto eterogenei.

In ottica di cooperazione, sarebbe importante fornire a livello di middleware anche la possibilità di inviare automaticamente la posizione dei vari utenti ad un server centralizzato, in quanto questo rappresenterebbe un requisito necessario e vitale per qualsiasi applicazione di questo tipo.

Entrando nello specifico del nostro sistema, sarebbe necessario a livello di comunicazione evitare di tenere conto del tipo di messaggi ricevuti e inviati, ma fornire solamente API che garantiscano una comunicazione sicura, protetta e che gestisca tutte le problematiche di cui si è già discusso. A livello di cooperazione, sarebbe necessario astrarre dai meccanismi di gioco, per fornire API generalizzate che permettano di agire sugli aspetti legati alla realtà aumentata degli oggetti di cui si mantengono le informazioni. Per esempio, il middleware che si potrebbe proporre a partire dagli strati individuati dovrebbe poter gestire il cambio di stato di oggetti situati e aumentati senza però conoscere i dettagli di questo passaggio di stato. Le interazioni tra oggetti e utenti (o anche tra due utenti, o due oggetti) dovrebbero essere individuate a livello di middleware, ma successivamente gestite per il caso concreto a livello di applicazione, in modo da rendere il sistema altamente scalabile.

Un ulteriore aspetto che un middleware di questo tipo dovrebbe gestire è quello relativo alle disconnessioni. Infatti, mentre le eventuali reazioni relative alla disconnessioni dovrebbero essere gestite a livello di applicazione, la loro rilevazione dovrebbe essere effettuata a livello di middleware, con tecniche analoghe a quelle definite in questo scritto.

In definitiva, il sistema da noi progettato ha l'ambizione di non rimanere entro i confini del caso di studio, ma di poter rappresentare, con le dovute modifiche e accorgimenti, un vero e proprio middleware per sistemi distribuiti a supporto della cooperazione situati e basati su realtà aumentata.

# Conclusioni

Il lavoro svolto è stato soddisfacente sotto tutti i punti di vista. Innanzitutto, l'analisi di un sistema di tale complessità ha offerto svariati spunti interessanti su cui riflettere e continue sfide da affrontare. I problemi che si sono dovuti affrontare fin dall'ideazione dell'applicazione mi hanno permesso di ampliare le mie conoscenze per trovare soluzioni adeguate e ponderate. Progettare da zero un sistema distribuito risulta sicuramente un compito arduo e pieno di insidie, ma anche stimolante e formante. Tutte le nozioni apprese su augmented reality, distributed systems, location-based games sono entrate a far parte del mio bagaglio culturale, ampliando i miei orizzonti verso ambienti prima di questa esperienza sicuramente o ignorati o conosciuti solo marginalmente.

La possibilità di lavorare in un team è stata determinante e motivo di grande soddisfazione personale, in quanto mi ha permesso di confrontarmi con altri colleghi e di risolvere problematiche più o meno complesse in maniera cooperativa. Oltretutto trovo che l'aver suddiviso il compito di progettazione e sviluppo su più livelli gestiti individualmente abbia contribuito a rendere il sistema più scalabile e modulare.

Il sistema sviluppato ha soddisfatto le aspettative e ha anche posto le basi per eventuali sviluppi futuri o applicazioni pratiche, descritti nella sezione seguente.

## Sviluppi Futuri

Per l'immediato futuro, è necessario distinguere due prospettive, ovvero l'ampliamento dell'applicazione basandosi sullo specifico caso di studio oppure lo sviluppo del middleware descritto alla fine del capitolo sulle valutazioni finali.

Per quanto riguarda il primo caso, l'applicazione potrebbe essere ampliata aggiungendo una parte dedicata al rendering degli oggetti aumentati finora rappresentati attraverso la visualizzazione di messaggi. Questo porterebbe ad avere una forma di realtà aumentata ancora più avanzata e conciliante con

la concezione più classica di augmented reality. Ovviamente tale ampliamento presenta numerose difficoltà soprattutto legate alle tecnologie attualmente disponibili. Per non perdere la scalabilità che caratterizza il sistema sarebbe infatti da evitare l'utilizzo di markers e tecniche di riconoscimento di immagini che riducano il raggio d'azione del gioco e la sua dinamicità. Un'alternativa potrebbe essere quella di utilizzare tecniche avanzate di triangolazione con dispositivi quali beacon o NFC sparsi sullo scenario di gioco, ma anche questo ne diminuirebbe la scalabilità. L'alternativa più valida sembra quindi quella di affidarsi alla geolocalizzazione, tentando di migliorare al massimo il segnale che viene ricevuto. I progressi tecnologici in questo senso determineranno quando questa soluzione sarà effettivamente applicabile.

Per quanto riguarda lo sviluppo di un middleware per applicazioni AR location-based, questa prospettiva potrebbe risultare estremamente interessante. Infatti aprirebbe la strada allo sviluppo di applicazioni innovative e molto funzionali in ottica di cooperazione. Per esempio, si potrebbe pensare di fornire dispositivi collegati ad un sistema cooperativo a soccorritori che devono agire in un luogo in cui è avvenuto un disastro di qualche tipo. Un middleware come quello teorizzato potrebbe fornire degli strumenti abbastanza scalabili per costruire un sistema che permetta di gestire dinamicamente dei punti d'interesse geolocalizzati, in cui visualizzare informazioni relative al tipo di emergenza, al numero di persone presenti, alla priorità ecc...

Insomma, le possibili applicazioni pratiche e i possibili sviluppi futuri del sistema sono molteplici, e possono aprire la strada a scenari molto interessanti ed estremamente innovativi.

# Ringraziamenti

Ringrazio familiari, amici e tutte le persone che mi sono state vicine in questi tre anni. Ringrazio i co-relatori Angelo Croatti e Pietro Brunetti e il prof. Alessandro Ricci per i preziosissimi consigli che hanno dispensato. Infine ringrazio i miei inseparabili compagni Berlo e Brando per questo meraviglioso percorso insieme.

# Bibliografia


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

[2] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, Misa Ivkovic, *Augmented reality technologies, systems and applications*

[3] It's Alive Mobile Games AB, *What is Pervasive Gaming?*, 2005.

[4] Montola, Markus, Stenros, Jaakko, Waern, Annika , *Pervasive Games. Theory and Design. Experiences on the Boundary Between Life and Play.*, Morgan Kaufmann Publishers, 2009.

[5] Wolfgang Broll, Jan Ohlenburg, Irma Lindt, Iris Herbst, Anne-Kathrin Braun, *Meeting Technology Challenges of Pervasive Augmented Reality Games*

[6] Juan Rodriguez-Covili, Sergio F. Ochoa, Jose A. Pino, Roc Messeguer, Esunly Medina , Dolors Royo, *A communication infrastructure to ease the development of mobile collaborative applications*

[7] Lindt, Broll, *NetAttack – First Steps Towards Pervasive Gaming*, ERCIM NEWS Special Issue on Games Technology, 2004.

[8] Ohlenburg, J. Herbst, I. Lindt, I. Fröhlich, T. Broll, *The MORGAN Framework: Enabling Dynamic Multi-User AR and VR Projects*, Proceedings of the ACM Symposium on Virtual Reality Software and Technology , 2004.

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

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

# Elenco delle figure