

Università degli studi di Bologna

Facoltà di Ingegneria e Scienze Informatiche

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Sistemi Autonomi

Boom Party

Simulazione Two Rooms and a Boom

Studenti

Luca Maestri 0900057223

Gioele Masini 0900056980

Andrea Pruccoli 0900057222

Alessandro Neri 0000741557

Indice

Indice	3
Introduzione	4
Requisiti	4
Analisi	6
Contesto	6
Preparazione del gioco	7
Comportamento in gioco di ogni giocatore	9
Dettaglio round	11
Progettazione	13
Entità	13
Stanze	13
Timer e Carta	14
Giocatore	14
Interfaccia utente	14
La coordinazione	15
Comunicazione tra i giocatori	15
Dettagli implementativi	16
BasicEnvironment	16
BoomPartyAgentArch	17
EnvironmentActions	17
BasicAgent	17
ITuplable	18
Movimento	18
Conclusioni	19
Miglioramenti	19
Interfaccia grafica	19
Ruoli e abilità	19
Comunicazione	20
Supporto al framework Jade	20

Introduzione

Il progetto è realizzato come artefatto finale per dimostrare la conoscenza acquisita durante il corso di Sistemi Autonomi.

Boom Party mira, partendo dalle logiche di un gioco di società, a modellare situazioni altamente dinamiche in cui le intenzioni del singolo individuo sono il risultato dell'unione tra gli obiettivi personali e quelli della collettività a cui appartengono.

La simulazione consisterà nella progettazione di un MAS con gli strumenti forniti durante il corso. In particolare nella relazione verrà descritto come è stata affrontata l'integrazione delle varie tecnologie (Jason, Tucson e Cartago) così come la logica adottata durante la fase di progettazione. Inoltre verranno presentati gli eventuali spunti per evoluzioni future, utilizzando il progetto realizzato come framework.

Requisiti

Prima di addentrarci nelle specifiche tecniche dell'applicazione, è bene avere una panoramica delle meccaniche e dinamiche del gioco ufficiale: Two Rooms and a Boom.

Le partite devono essere organizzate in un ambiente consono al gioco: deve essere composto da due aree collegate da un unico canale di passaggio. La comunicazione verbale e visiva tra le due stanze non è autorizzata. D'ora in poi ci si riferirà alle due aree come stanze e al passaggio come corridoio.

Prima dell'inizio della partita deve essere composto un mazzo di carte ruolo. Queste contengono tre informazioni: il nome della carta, la fazione e la descrizione. La fazione può essere Blu, Rossa o Grigia, la descrizione può contenere testo influente, abilità uniche di cui i giocatori potranno fare uso durante il gioco e obiettivi unici da perseguire. Se la carta è di fazione Rossa o Blu l'obiettivo, se presente, riguarda l'intera squadra del proprio colore (Es.: Presidente e Bombarolo descritti nel paragrafo successivo). Se la fazione è Grigia invece l'obiettivo è sempre presente e riguarda esclusivamente il giocatore in possesso della carta (Es.: **"Vinci se** al termine dell'ultimo round sei leader di una stanza.").

I giocatori vincono la partita se completano tutti i propri obiettivi entro la fine della partita. Gli unici due ruoli sicuramente presenti nel mazzo devono essere Presidente per la fazione Blu e Bombarolo per la Rossa. Questi ruoli contengono l'obiettivo basilare per la vittoria di entrambe le squadre:

- Carta Presidente: **"La Squadra Blu vince se** al termine dell'ultimo round non sei nella stessa stanza del Bombarolo."
- Carta Bombarolo: **"La Squadra Rossa vince se** al termine dell'ultimo round sei nella stessa stanza del Presidente."

Per le fazioni Rossa e Blu non sono stati rilevati altri requisiti fondamentali per il gioco. L'unico requisito è la disponibilità per entrambe le fazioni di un ruolo definito "base": carta la cui descrizione non contiene né abilità né obiettivi. I ruoli con fazione Grigia non sono fondamentali per il gioco ma

si desume buona prassi l'uso di almeno un ruolo di questa fazione per poter diversificare e completare l'esperienza di gioco.

All'inizio della partita ogni giocatore pesca casualmente una carta dal mazzo. Successivamente le persone vengono suddivise in modo equo fra due stanze. Per ogni stanza viene infine eletto un giocatore come leader in maniera pseudo-casuale.

Il gioco è normalmente composto da 5 turni, i quali si riducono a 3 se i giocatori sono 10 o meno. La durata di ogni turno è definita da uno specifico numero di minuti, normalmente misurati con un apposito timer. Come visibile dalla griglia qui di seguito, la durata dei turni è decrescente.

	5	4	3	2	1
6 - 10 PLAYERS	ONLY 3 ROUNDS & NO COLOR SHARING OR COLOR REVEALING				
11-13 PLAYERS	2	2	2	1	1
14-17 PLAYERS	3	2	2	1	1
18-21 PLAYERS	4	3	2	1	1
22+ PLAYERS	5	4	3	2	1
NUMBER OF HOSTAGES FOR EACH ROUND					
LEADER					

Ad inizio partita nessun giocatore conosce il ruolo e la fazione degli altri giocatori ed il gioco è basato sull'interazione con le altre persone presenti nella stanza, per capire chi è alleato e chi invece non lo è. Lo scambio di informazioni fra giocatori avviene previo accordo sulle modalità con cui effettuare tale scambio, ovvero mostrare solo parte della carta (solo la fazione a cui si appartiene), mostrare tutta la carta (rivelando anche il ruolo), fidarsi sulla 'parola' o offrendo in cambio informazioni su altri giocatori. Durante la partita i giocatori potranno fare azioni più complesse e coraggiose, come ad esempio bluffare per diffondere false informazioni, coordinarsi secondo una strategia di gruppo o tentare di formare una rete di comunicazione per nascondere le informazioni più importanti.

Oltre allo scambio di informazioni l'unica attività che si può fare è quella di proporsi pubblicamente come nuovo leader della stanza. In quel momento ogni giocatore che vuole promuovere la nuova elezione alza la mano per votare. Il candidato diventerà leader se ottiene il voto da parte della maggioranza dei giocatori in stanza.

Alla fine di ogni turno il leader corrente di ogni stanza deve designare un certo numero di ostaggi, determinato in base all'attuale turno di gioco e al numero di giocatori, da spostare nell'altra stanza. Per regolamento questo è l'unico modo per spostarsi tra le due stanze e in questa fase i giocatori non possono parlarsi.

Obiettivo della squadra Blu è di tenere i giocatori con i ruoli Presidente e Bombarolo in stanze separate, quello della squadra Rossa è di tenerli nella stessa stanza. Alla fine dell'ultimo turno, dopo

l'ultimo scambio di ostaggi, tutti i giocatori rivelano la propria carta (mostrando ruolo e fazione) e si determinano i vincitori.

Le partite possono essere semplici se il mazzo è composto solo da Presidente, Bombarolo e tutti i restanti ruoli base (appartenenti ad una fazione e senza alcuna particolarità). Per partite più avanzate possono essere inseriti ulteriori ruoli di tutte e tre le fazioni.

Come requisito sarà quindi necessario rappresentare graficamente tutto l'ambiente con i giocatori al suo interno e gli spostamenti di questi ultimi sia all'interno della stanza, durante lo svolgimento dei turni, che tra le due stanze, nel caso in cui fossero designati come ostaggi dal leader della loro stanza.

Analisi

Partendo dai requisiti, la fase di analisi può suddividersi naturalmente nelle seguenti macroaree. È da tenere in considerazione che l'analisi è stata fatta su contesti reali con l'obiettivo di dettagliarli e semplificarli ai fini del progetto.

Contesto

Innanzitutto è importante definire tutti gli elementi necessari perché il gioco possa svolgersi correttamente, il cui ciclo di vita è direttamente collegato a quello dell'applicazione.

Lo spazio a disposizione per il movimento è composto da due aree camminabili collegate da un unico passaggio. Dall'interno di entrambe le aree non dev'essere possibile osservare gli ambienti esterni. Il passaggio dev'essere percorribile solo dai giocatori designati come ostaggi e in questo luogo non è permesso alcuno scambio di informazioni o raccolta di dati. L'unica comunicazione possibile tra le due stanze è solamente la minima indispensabile perché l'esecuzione della partita avvenga in modo coordinato, compiuta da parte dei due leader correnti.

Gli unici oggetti passivi presenti in gioco sono le carte e il timer. Le carte hanno ognuna un singolo proprietario il quale è l'unico giocatore con la possibilità di leggerne le informazioni liberamente. La carta di un giocatore può essere letta dagli altri giocatori soltanto se si trovano nella stessa stanza e se il proprietario sceglie di mostrarla, privatamente a specifici individui o pubblicamente a tutti i presenti nella stessa area del proprietario. Il timer deve essere presente in entrambe le stanze, condiviso o con due istanze dello stesso oggetto. Deve possedere un'interfaccia che permetta ai due leader di impostare il corretto timer all'inizio di ogni round di gioco e allo scadere del quale deve lanciare un evento verso tutti i giocatori.

Parte del contesto lo sono anche i giocatori. Prima che il gioco venga avviato deve essere conosciuto il numero di partecipanti. Il requisito fondamentale al riguardo è che il gioco venga portato a termine con successo e con realistica. Perché questo avvenga è importante che ogni giocatore sia completamente autonomo in termini di scelta, esecuzione e analisi del riscontro delle azioni principali del gioco quali movimento, comunicazione, votazione per cambio leader e definizione degli ostaggi.

Preparazione del gioco

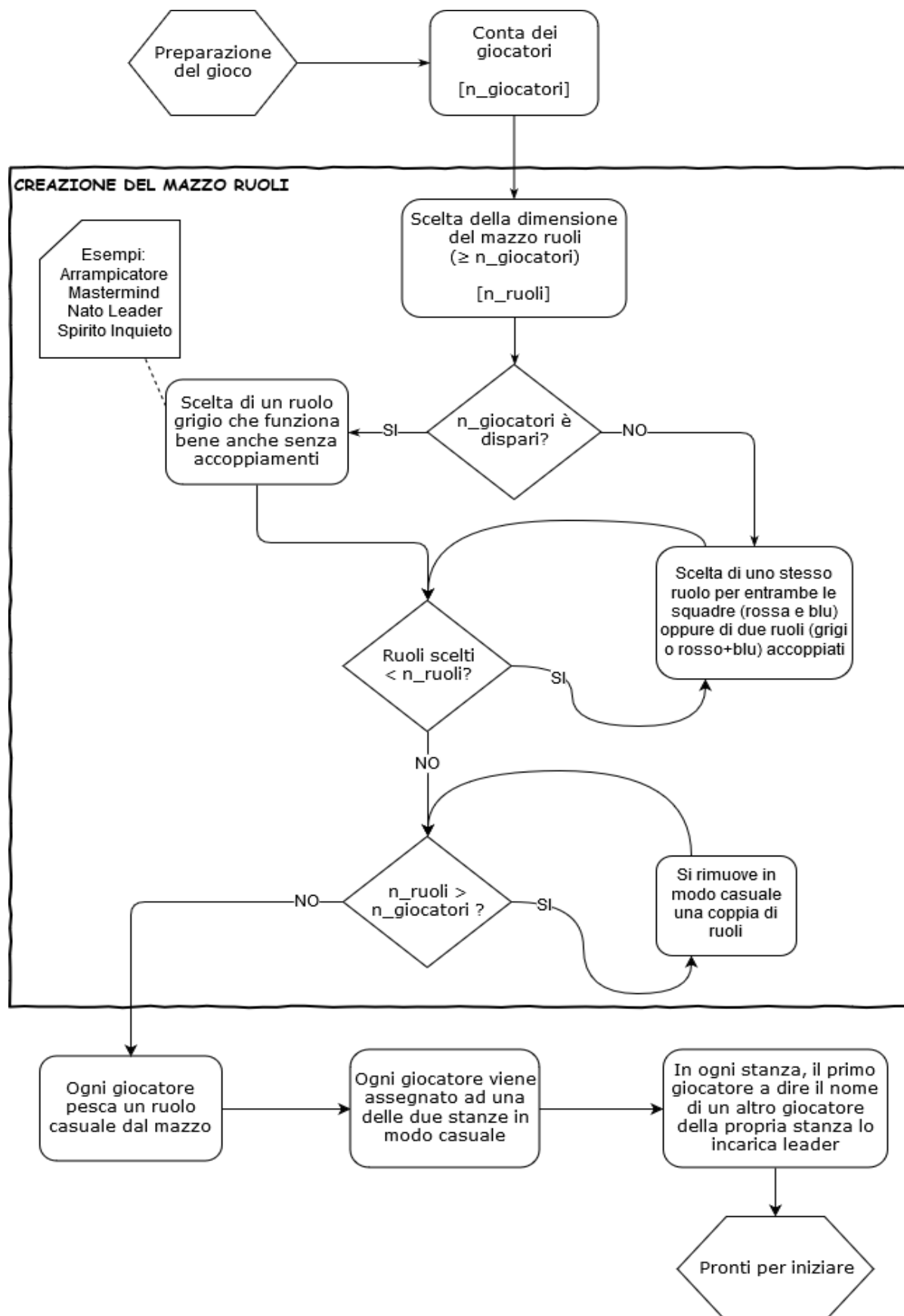
La preparazione del gioco è una fase importante della partita in quanto necessaria ma allo stesso tempo non fondamentale al funzionamento del software. In questa fase si effettuano una serie di azioni e di scelte che preparano il gioco all'avvio.

In ogni partita reale, dopo aver contato i giocatori, la parte più importante è quella della scelta dei ruoli per la composizione del mazzo poiché potrà influire notevolmente sul grado di difficoltà della partita e sull'esperienza vissuta dai giocatori. Il numero di carte deve essere pari o superiore al numero di persone partecipanti. La fazione Grigia è formata da ruoli speciali che non è obbligatorio inserire nel mazzo. Normalmente almeno uno di questi ruoli viene inserito quando i giocatori sono dispari per mantenere l'equilibrio tra le altre due fazioni.

Successivamente si rimuovono casualmente ruoli dal mazzo (evitando di escludere bombarolo e presidente) riducendolo fino al numero di giocatori. Le carte rimanenti dovranno essere lo stesso numero per le fazioni Rossa e Blu, le altre apparterranno tutte alla fazione Grigia.

Terminata la scelta dei ruoli, ogni giocatore pesca una carta casuale senza svelarne le informazioni agli altri e viene assegnato ad una delle due stanze. Una volta divisi i giocatori tra le relative stanze si procede con l'ultima fase della preparazione: l'elezione di un leader per stanza in modo pseudo-casuale.

Di seguito si rappresenta graficamente il procedimento appena descritto.

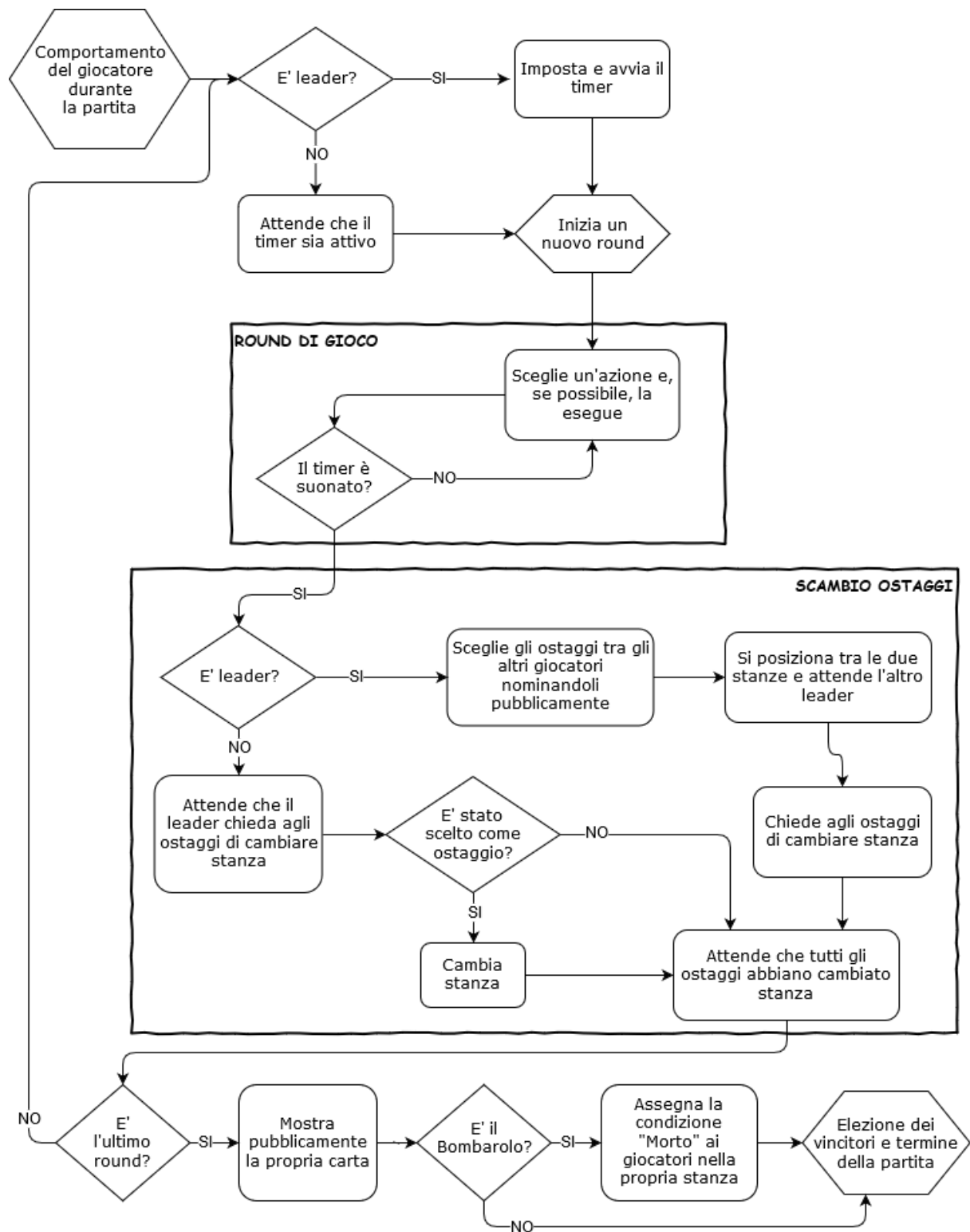


Comportamento in gioco di ogni giocatore

Un round parte quando il leader della stanza imposta e avvia il timer per il tempo definito per quel turno. Ogni giocatore decide quale azione compiere e tenta di eseguirla.

Quando il timer scade, il turno ha fine. Il leader sceglie gli ostaggi da scambiare ed una volta effettuato lo scambio imposta nuovamente il timer e lo avvia.

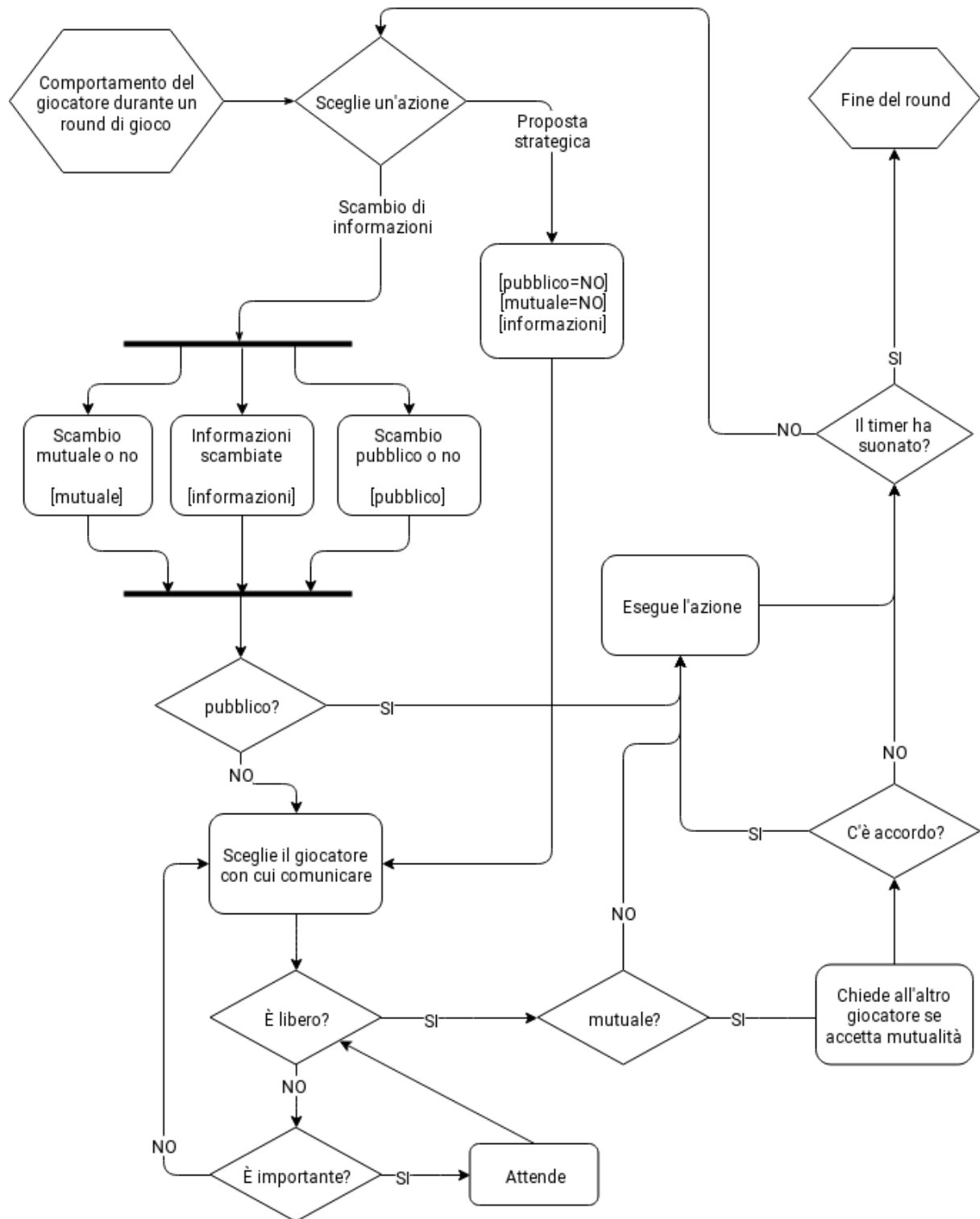
Se invece è l'ultimo turno, dopo lo scambio di ostaggi, ogni giocatore rivela pubblicamente la propria carta (con indicazione di ruolo e fazione) e si effettuano le considerazioni per decretare i vincitori.



Dettaglio round

All'interno del round di gioco ogni giocatore è spinto a relazionarsi con gli altri presenti nella stanza per acquisire e scambiare informazioni, operazione fondamentale per raggiungere l'obiettivo descritto sulla sua carta.

Uno scambio di informazioni avviene solitamente tra due giocatori, i quali si concordano sulle modalità con cui queste vengono scambiate (a volte si può decidere di cedere informazioni senza avere nulla in cambio). Una volta stipulato l'accordo, quest'ultimo deve essere mantenuto da entrambe le parti (anche se a volte non è possibile, per via di particolari regole di ruoli speciali).



Oltre allo studio comportamentale, l'analisi dei requisiti ci ha permesso di individuare i seguenti requisiti non funzionali da soddisfare:

- **Autonomia:** ogni giocatore avrà un obiettivo personale da perseguire. Dovrà adattarsi ai cambiamenti che avvengono nell'ambiente in cui è immerso.

- **Interazione e cooperazione:** scopo del gioco è quello di raccogliere informazioni tramite la comunicazione con i giocatori della propria stanza (che rappresenta il perimetro di comunicazione). In base al ruolo e allo stato della partita un giocatore potrà decidere di collaborare o dare informazioni parziali/inesatte a coloro che lo interrogano. Scopo della simulazione sarà quello di studiare come evolve l'interazione tra gruppi di giocatori.
- **Scalabilità e extensibility:** la simulazione deve poter gestire un numero elevato di giocatori, eventualmente su nodi eterogenei e distribuiti.

Dai requisiti emersi (e rimanendo coerenti con gli obiettivi del corso), scegliamo di realizzare una modellazione di tipo **ABM** (Agent-Based Model). Nei prossimi capitoli verranno esposte le componenti che caratterizzano questa tipologia di modellazione.

Progettazione

Si è scelto di realizzare un framework Java che funga da base per l'implementazione di agenti autonomi che siano in grado di approcciarsi con gli altri agenti (giocatori) della stanza, scambiarsi informazioni ed elaborare strategie per raggiungere il proprio obiettivo (che può essere comune ad altri giocatori o personale).

Per quanto riguarda le tecnologie, utilizzeremo Jason come framework **BDI** per la definizione dei comportamenti degli agenti, Tucson per quanto riguarda lo stato globale della simulazione, CArtaGo per la creazione e gestione di artefatti semplici (le carte che definiscono il team e il ruolo, i timer presenti nelle due stanze), ReSpecT come supporto nelle fasi iniziali e finali del gioco.

L'interfaccia verrà sviluppata anch'essa in Java tramite la libreria Swing (estendendo dei componenti grafici messi a disposizione dal framework Jason) per poter essere implementata in maniera indipendente dalla piattaforma di esecuzione, in modo che funzioni su ogni sistema operativo. Anch'essa farà uso di Tucson per una migliore integrazione con il sistema.

Entità

Stanze

La modellazione di questa entità permette di rappresentare in maniera virtuale lo spazio fisico e di raggruppare secondo criteri logici i giocatori. È un elemento di coordinazione fondamentale ma non possiede logica in quanto sono i singoli giocatori a fare uso di questa informazione quando necessario. Viene rappresentato da una tupla apposita sul tuple centre che contiene le seguenti informazioni: numero identificativo univoco generato manualmente in scrittura del codice e giocatore che è attualmente leader di quella stanza. Entrambe le stanze vengono caricate all'avvio dell'applicazione.

Timer e Carta

La carta posseduta da ogni giocatore, come già indicato in precedenza, lo identifica nei panni di un particolare ruolo e all'interno di una fazione. Il timer invece scandisce il tempo che rimane all'interno di ogni round. Poiché sono due oggetti che vengono sfruttati in maniera attiva dai giocatori (agenti), si è deciso di modellarli come artefatti.

Il timer viene rappresentato sul tuple centre da una tupla che ne specifica il nome (riferimento all'artefatto) e la stanza di riferimento. La carta viene rappresentata con il solo nome, poiché utile solo in fase iniziale della partita.

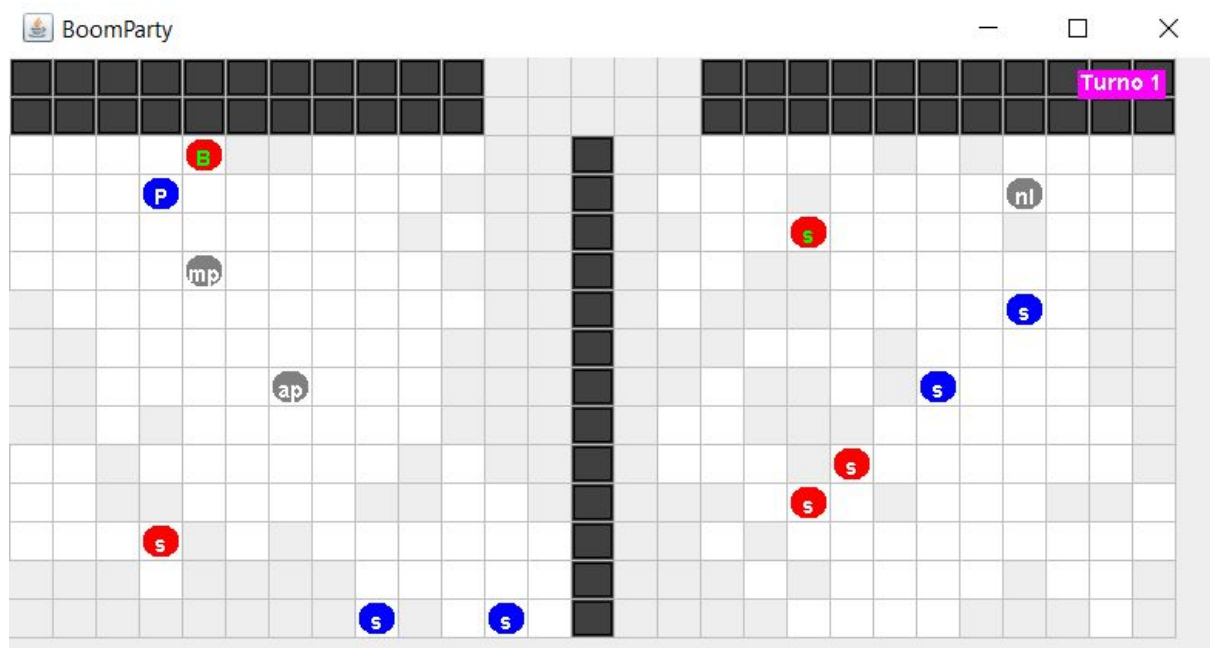
Giocatore

Il giocatore è l'elemento cardine e anche quello più complesso identificato. Ogni giocatore ha un obiettivo che cercherà di raggiungere durante la partita tramite le azioni che può compiere all'interno dell'ambiente in cui è posto, ovvero muoversi all'interno delle stanze e comunicare con gli altri giocatori (altri agenti) per scambiare/estorcere le informazioni che potrebbero condurlo alla vittoria.

La tupla che lo rappresenta contiene le informazioni sul nome del giocatore (riferimento all'agente) e sulla stanza in cui si trova attualmente.

Interfaccia utente

L'interfaccia di gioco è stata realizzata estendendo le classi GridWorldView e GridWorldModel, che ci hanno permesso di poter rappresentare la parte grafica del software come una griglia, scelta che ci ha anche aiutato poi durante lo sviluppo per la rappresentazione grafica delle due stanze, dei giocatori sulla mappa, i loro spostamenti fra le stanze (durante lo scambio degli ostaggi) o per raggiungere un altro giocatore con cui interagire.



Contestualmente alla finestra principale viene avviata anche una console di MAS e, se la modalità della simulazione è settata su 'debug', viene istanziata anche una finestra per i tool di Tucson.

Per una maggiore comprensione della rappresentazione grafica sopra proposta, di seguito una legenda dei colori e delle lettere:

- Sfondo rosso/blu/grigio: fazione a cui appartiene il giocatore;
- Lettere:
 - s: ruolo 'semplice';
 - P: **Presidente**, sempre nella squadra blu;
 - B: **Bombarolo**, sempre nella squadra rossa;
 - mp/ap: **Moglie del Presidente / Amante del Presidente**;
 - nl: **Nato Leader**.
- Colore lettera: verde per il leader della stanza, bianco tutti gli altri.

La coordinazione

La coordinazione fra gli agenti Jason, oltre all'uso dei goal e belief propri del linguaggio, è stata raggiunta sfruttando anche i mezzi messi a disposizione da Tucson: il tuple centre.

Esso è utilizzato ad inizio partita quando dal sistema vengono create le tuple dei giocatori e dei ruoli disponibili, al termine della partita quando devono essere fatte le valutazioni su chi abbia vinto la partita e durante la comunicazione fra due giocatori.

Per impersonare il ruolo del mazziere, ovvero quello che si occupa ad inizio partita di creare gli artefatti carta per ogni ruolo/fazione e timer per ogni stanza, il sistema crea una tupla **token(mazziere)** che viene letta in modo randomico da uno degli agenti/giocatori prima che il primo turno abbia inizio. Quando è stato giocato l'ultimo round, ogni giocatore scrive sul tuple centre le informazioni su chi è, che ruolo ha e di quale fazione fa parte. Sempre il mazziere è incaricato poi di valutare, in base alle informazioni così fornite, quale fazione e giocatore ha vinto la partita.

Nel corso della simulazione, la comunicazione fra due giocatori viene gestita facendo uso di tuple e di reaction, permettendo una scrittura del codice Jason più fluida e astratta.

Per automatizzare la conversione di oggetti java in tuple e viceversa, sono state implementate delle classi apposite per ogni tupla scritta sul tuple centre dal sistema (**initialRole** e **player**), che contengono i metodi per generare le tuple o per popolare l'istanza della classe a partire da una LogicTuple letta dal tuple centre, che le rappresentano interamente o parzialmente (tramite l'uso delle variabili Prolog) ed è anche possibile popolare i dati a partire da una LogicTuple.

Comunicazione tra i giocatori

La gestione del processo di comunicazione tra i giocatori e la relativa gestione delle informazioni ottenute è stata suddivisa, per praticità ed estendibilità, in più moduli:

communication: modulo adibito allo scambio di messaggi, al quale viene affidata la gestione di tutte le operazioni riguardanti questa fase.

conversationTracker: modulo sviluppato per tenere traccia delle conversazioni tenute fino a quel momento e dell'esito ottenuto.

knowlegde: modulo che offre le funzionalità necessarie alla gestione della conoscenza che ogni giocatore possiede relativamente agli altri giocatori, dotato di un valore di affidabilità specifico per ogni dato, conferito dal metodo di comunicazione usato nella determinata operazione di comunicazione.

I giocatori interessati a scambiarsi informazioni relative ai ruoli di altri giocatori si appoggiano alle funzioni messe a disposizione dal modulo di comunicazione specificando gli appositi parametri, quali: il giocatore con cui si vuole interloquire, il nome del giocatore di cui si vuole avere informazioni, il tipo di modalità di scambio e quali informazioni si desidera ricevere (solo team di appartenenza o anche ruolo). I tipi di modalità di scambio informazioni messe a disposizione degli agenti sono i seguenti:

- **Carta:** rappresenta uno scambio di informazioni tra giocatori, simulando il caso reale di due persone che rivelano le informazioni richieste mostrandosi a vicenda la carta in proprio possesso simultaneamente. Questo tipo di scambio si affida a tucson in modo da garantire una maggiore affidabilità, grazie alla sua introduzione come elemento esterno di controllo e supervisione dell'operazione, oltre a fornire un minore ritardo tra i tempi di ricezione delle informazioni tra i due soggetti principali, infatti tramite Respect vengono inseriti i dati nel tuple space atomicamente solo al conseguimento di tutte le informazioni necessarie.
- **Parlato:** questa tipologia di comunicazione sta ad indicare un normale scambio di informazioni basato sulla fiducia reciproca, equivalente ad uno scambio di informazioni verbale tra due concorrenti senza alcun riscontro sulla veridicità dei dati ottenuti, è un metodo adatto per comunicare con giocatori del quale si conoscono poche o nessuna informazione o nel caso si sia interessati ad effettuare un bluff.

Una volta ricevute le informazioni necessarie il modulo di comunicazione si appoggia alle altre due componenti citate per aggiornare i dati a disposizione dell'agente, andando quindi ad arricchire le informazioni in proprio possesso riguardo gli altri agenti, nel caso la richiesta di scambio informazioni sia stata accettata e si sia correttamente conclusa, infine procedendo ad aggiornare lo storico delle conversazioni effettuate sino a quel momento.

Dettagli implementativi

BasicEnvironment

L'environment di default in cui agenti e artefatti operano. Costruito estendendo **CartagoEnvironment** per poter utilizzare le funzionalità fornite dalla libreria di **CARTAGO**, cioè gli artefatti.

A questa classe è affidato il compito di inizializzare il modello del "mondo" e la relativa view con le informazioni contenute all'interno del file mas2j (principalmente la lista di giocatori), non che anche

l'implementazione delle operazioni da effettuare quando gli agenti richiamano le azioni personalizzate del sistema (quelle definite all'interno di **EnvironmentActionsEnum** e **EnvironmentActions**) e l'aggiornamento dei percepts di ogni agente.

BoomPartyAgentArch

Architettura utilizzata all'interno del sistema, implementata estendendo l'architettura di **CARTAgO** ed implementando l'interfaccia di **Tucson4Jason** (rispettivamente **CAgentArch** e **T4JnArch**), scelta obbligatoria in quanto, come già detto più volte, gli agenti all'interno del software devono essere in grado di interagire sia con gli artefatti che con il tuple space.

EnvironmentActions

All'interno del file **EnvironmentActionsEnum** sono elencate quali sono le possibili azioni eseguibili all'interno della simulazione. Mentre le più semplici sono implementate direttamente all'interno del **BasicEnvironment**, quelle che richiedono un maggior livello di approfondimento sono state implementate all'interno del file **EnvironmentActions**.

BasicAgent

Elemento fulcro del software, questo file Jason definisce i goal, i belief e i plan fondamentali per un agente per poter garantire il corretto funzionamento del gioco. All'interno del file vengono definiti quelli che sono gli eventi e le azioni di base che un agente può intraprendere all'interno di un turno di gioco, dalla preparazione della partita, l'interazione con i vari artefatti (carta/timer), le operazioni da eseguire alla fine del round (gestione degli ostaggi) e quelle da eseguire a fine partita (ogni giocatore rivela la propria identità ed appartenenza). Poiché il codice per la gestione di un agente è risultato essere abbastanza complesso, sono stati creati vari file di supporto, di utilità, per meglio separare le varie aree di interesse. Il file **basicAgent** infatti importa diversi altri file:

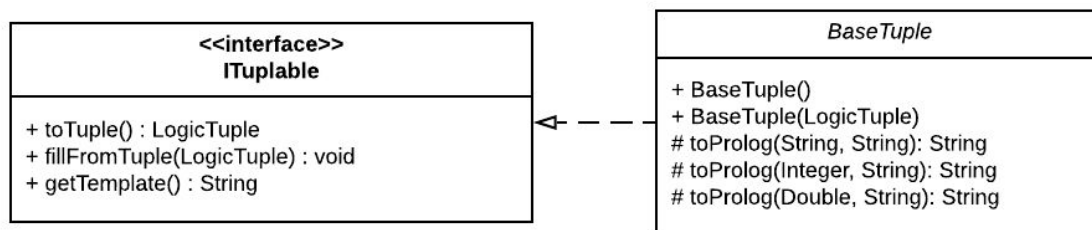
- **tucsonBaseWrapper**: file che fa da wrap per l'interazione degli agenti con il tuple space, permettendo il recupero e la scrittura delle informazioni sullo stesso;
- **basicRules**: elenco di regole base valide per ogni agente durante la simulazione (come ad esempio quanti turni compongono una partita e quanti minuti dura ogni turno);
- **communication**: contiene la gestione dell'interazione fra due giocatori, cioè lo scambio di informazioni che avviene mostrando la propria carta o solo tramite il parlato;
- **timerHelper**: file di utility per il focus e defocus su l'artefatto **Timer** da parte dell'agente;
- **movement**: contiene i plan che richiamano le azioni eseguite dall'agente nei confronti dell'ambiente (in questo caso il movimento all'interno e fra le stanze).

Legato in maniera indissolubile a questo file è **intelligentAgent**, estensione di **basicAgent** al cui interno è implementata la logica più pura della sequenza di azioni che un agente deve intraprendere durante un turno di gioco e altre azioni, altre intenzioni, che ne definiscono il livello di esperienza con cui il giocatore sa giocare. Infatti il file rappresenta quello che può essere definito il giocatore medio, quello che segue strettamente le regole e che non elabora particolari strategie o non esegue ragionamenti avanzati sull'attuale situazione della partita.

ITuplable

Interfaccia che ogni Tuple deve implementare se deve essere gestita da Java.

Per guidare e semplificare la creazione di una Tuple è stata implementata la classe astratta BaseTuple che offre delle funzioni per gestire i singoli attributi dell'oggetto, ed eventualmente sostituirne il valore con una variabile Prolog. BaseTuple implementa due costruttori: il primo senza parametri, per le interrogazioni Tucson, il secondo per popolare l'oggetto con i dati di una tuple compatibile.



Implementazioni di questa interfaccia sono molto utili per la gestione dei dati su tuple centre in quanto, oltre a contenere i valori che compongono il dato, possiedono le seguenti funzionalità:

- Generazione della tuple contenente i singoli dati, con la possibilità di sostituirne uno o più con variabili Prolog (metodi `toTuple` e `asTupleString`);
- Valorizzazione di un'istanza **ITuplable** con i valori di una tuple corrispondente (metodo `fillFromTuple`).

Movimento

Per permettere agli agenti di muoversi liberamente nell'ambiente di gioco si è scelto di utilizzare un algoritmo di ricerca su grafi per individuare il miglior percorso dalla posizione corrente del giocatore fino alla posizione "bersaglio". Questa scelta permette di garantire efficienza computazionale all'applicazione e simulazione di un movimento realistico.

Per ottenere il massimo risultato, si è scelto di far calcolare il percorso all'environment il quale è sempre a conoscenza di tutte le informazioni necessarie. Questi fornisce una semplice interfaccia agli agenti per effettuare i movimenti rispettando le necessità di gioco:

- `move_towards(PLAYER)`, muove l'agente di una casella in direzione del giocatore in input;
- `move_in(Area)`, muove l'agente di una casella in direzione dell'area in input;
- `move_randomly`, muove l'agente in una casella adiacente casuale.

Le prime due azioni utilizzano un'implementazione dell'algoritmo di pathfinding A* per calcolare il percorso alla prima richiesta. Il risultato viene salvato in memoria in modo da essere applicato fino al raggiungimento dell'obiettivo. Il ricalcolo avviene solo nel caso in cui un nuovo ostacolo si presenta

sul percorso o, ad esempio in caso di spostamento verso un agente in movimento, se la posizione “bersaglio” è cambiata.

In quanto l’ambiente è composto da numerosi agenti in movimento e considerando che i giocatori in partita potrebbero essere un numero considerevole, si presenta un’importante problematica non coperta dal solo algoritmo di pathfinding. Durante lo scambio di ostaggi si possono avere fino a 10 agenti che contemporaneamente si muovono attraverso uno stretto passaggio: il corridoio. Per evitare uno stallo è stato incluso l’utilizzo della funzionalità `move_randomly`, che viene utilizzata ogni qual volta l’algoritmo non riesce a generare nessun tragitto percorribile. Questo spostamento mantiene gli agenti in continuo movimento, condizione che permette di liberare lo spazio di passaggio in caso di stallo.

Un ulteriore problema è dovuto all’esecuzione in parallelo dei diversi agenti. Per avere le migliori performance è importante eseguire i calcoli in parallelo ma è altresì importante non far sovrapporre gli agenti. Per risolvere questo problema è stato implementato un metodo `synchronized, setAgentPosition`, come unica via per cambiare la posizione degli agenti. Mentre tutto il resto del codice viene eseguito in parallelo, questo metodo garantisce che i movimenti vengano attuati solo se risulta realmente possibile, e restituisce un booleano indicante il successo o il fallimento dell’operazione.

Conclusioni

Miglioramenti

Grazie alla forte ingegnerizzazione del software e all’approccio utilizzato durante lo sviluppo del progetto, il sistema, già completo, può essere esteso in numerosi modi, tra i possibili miglioramenti elenchiamo alcune possibili modifiche che permetterebbero l’introduzione di nuove interessanti funzionalità e una serie di applicazioni concrete valide per ogni contesto.

Interfaccia grafica

Ora come ora, la decisione della quantità di giocatori e la scelta della composizione del mazzo sono tutte a carico del codice, ma la possibilità da parte dell’utente di poter modificare questi valori è uno dei primi miglioramenti che viene in mente una volta avviato il software. Un’altra possibilità che potrebbe essere lasciata all’utente potrebbe essere quella di decidere il numero di turni di gioco, la durata in minuti di questi e anche la velocità di gioco, ad esempio raddoppiata o dimezzata.

Ruoli e abilità

Una parte interessante e utile al fine di variare le strategie di gioco e quelli che potrebbero essere alcuni momenti chiave della partita, potrebbe essere sicuramente l’introduzione di **ruoli** specifici all’interno delle due fazioni principali e delle relative **abilità** che questi possiedono. In realtà con ‘ruolo’ ci si riferisce solamente al nome della carta che porta con sé una particolare abilità, come ad esempio:

- **Buttafuori:** una sola volta durante tutta la partita può decidere 1 ostaggio da mandare nell'altra stanza, azione che deve essere compiuta prima che il leader della stanza nomini tutti e quanti gli ostaggi da mandare;
- **Spia:** fa parte della fazione opposta a quella scritta sulla sua carta (un ruolo molto particolare da giocare);
- **Agente:** una volta per round può costringere un giocatore a mostrare la sua carta;
- **Angelo:** inizia la partita con la condizione 'Onesto', per via di tale condizione non può mentire verbalmente (può comunque farlo in altro modo, ad esempio gesticolando).

Oltre alle abilità possono essere introdotte anche le **condizioni** come si è appena visto con il ruolo **Angelo**. Le condizioni sono poste sulle carte dei giocatori (e quindi sul ruolo) ed impongono al giocatore che possiede quella carta di rispettare le regole descritte su di essa, come appunto il non poter mentire verbalmente, o un altro esempio può essere quello di non poter comunicare affatto verbalmente. Sono un buon mezzo per variare ulteriormente l'esperienza di gioco.

Comunicazione

Le informazioni che i giocatori attualmente si scambiano sono solo quelle di cui sono a conoscenza, ma anche il bluff, quindi il dare (e ricevere) informazioni false è un altro aspetto che contraddistingue questo gioco, ma che non è stato trattato poiché considerato come una parte abbastanza complessa da implementare e gestire, nonostante il terreno sia comunque stato preparato in previsione di questa possibilità (vedasi il valore di affidamento all'interno del file **knowledge**). Altro punto che manca alla comunicazione, questa volta non più fra 2 giocatori, è la comunicazione pubblica di informazioni, ovvero il rivelare le proprie informazioni (che siano date dalle proprie carte o dalla propria conoscenza) ad un diverso numero di giocatori contemporaneamente, senza però poter pretendere che la parte ricevente delle informazioni faccia lo stesso, poiché nella rivelazione pubblica non è richiesto il consenso delle due parti per effettuare lo scambio, quindi non viene preso nessun accordo vincolante (si ricorda che ogni 'contratto' pattuito fra due parti deve essere rigorosamente rispettato).

Supporto al framework Jade

Abilitare la simulazione distribuita tra più nodi, sfruttando il framework ad agenti Jade.

A tale scopo, per quanto Jason abbia già la nozione di JadeMAS, occorre comunque sviluppare un Environment ed un'architettura per gli agenti che consentano di utilizzare contemporaneamente le funzionalità di tutte le tecnologie in gioco (Cartago, Jade, Tucson).

Per maggiori informazioni vedere le integrazioni ad oggi [disponibili](#), in particolare **C4Jade**.