



Software per Simulare Match Calcistici

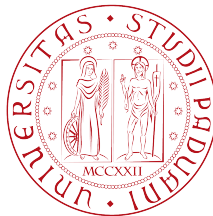
Autore: dott. Paolo Carletto

Matricola: 622276

Docente: dott. Tullio Vardanega

A.A. 2011/12

03 gennaio 2011



Indice

1	Introduzione	3
1.1	Tema	3
1.2	Requisiti	4
2	Analisi	5
2.1	La Struttura di Gioco	5
2.1.1	Campo	5
2.1.2	Arbitro	7
2.1.3	Manager	7
2.1.4	Giocatore	7
2.1.5	Palla	8
2.2	I Dettagli di Gioco	8
2.2.1	Movimento	8
2.2.2	Zone di Movimento	9
2.2.3	I Giocatori	10
3	Applicazioni delle Logiche	12
3.1	Giocatore con Palla	12
3.1.1	Logica di Tiro	13
3.1.2	Logica di Passaggio	14
3.1.3	Logica di Avanzamento	17
3.1.4	Logica di Posizione	17
3.2	Squadra con Palla	17
3.2.1	Logica di Smarcatura	17
3.2.2	Logica di Avanzamento	18
3.2.3	Logica di Posizione	18
3.3	Avversari con Palla	18
3.3.1	Logica di Contrasto	18
3.3.2	Logica di Indietreggiamento	19
3.3.3	Logica di Posizione	20
3.4	Parata del Portiere	20
3.4.1	Logica di Contrasto	20
3.4.2	Logica di Parata	21
3.4.3	Logica di Posizione	22
4	Modello	23
4.1	Campo	24
4.2	Arbitro	25
4.3	Manager	26
4.4	Squadra	28
4.5	Giocatore	29
4.6	Pallone	32
5	Concorrenza	33
5.1	Integrità dello Stato di Gioco	33
5.2	Unicità del Possesso di Palla	34
5.3	Priorità dei Processi	35
5.4	Prevenzione delle Situazioni di Stallo	36

6	Distribuzione	38
6.1	Estensione del Modello Distribuito	39
6.2	Creazione del Software Distribuito	40
6.3	Sincronizzazione Temporale	42
6.4	Stati di Gioco	42
7	Codifica	44
7.1	Field	44
7.2	Referee	45
7.3	Manager	46
7.4	Team	47
7.5	Player	48
7.6	Ball	49
8	Conclusioni	50
9	Bibliografia	51

1 Introduzione

1.1 Tema

Il tema del progetto è un simulatore software di una partita di calcio.

Il sistema software deve supportare almeno le seguenti caratteristiche:

- Gli utenti devono essere in grado di giocare un singolo match; il supporto per la giocabilità di una serie di partite con calendari e regole associate è facoltativo e può essere omesso;
- La variante scelta del gioco deve essere configurabile in tutti i parametri rilevanti, consentendo uno dei formati classici del calcetto a 5, 7 e il canonico 11 giocatori;
- I componenti delle squadre saranno caratterizzati da caratteristiche configurabili individualmente almeno per capacità tecniche e tattiche, velocità, parametri fisici tra cui anche la stanchezza; alcuni di questi parametri devono essere dinamici ed evolvere con la partita secondo una logica programmata;
- Ogni squadra deve avere un manager (software) in grado di configurare la formazione iniziale dei giocatori, la tattica iniziale e di impartire comandi per cambiamenti tattici e sostituzioni, tutti soggetti alle regole del gioco come definito nello standard corrispondente;
- Ogni squadra deve giocare secondo la **tattica comandato** dal manager; variazioni sono ammesse nella misura in cui risultano dalle caratteristiche programmabili dei giocatori;
- Ogni partita ha un arbitro (software) indipendente e 2 o 3 assistenti (software) subordinati che controllano il gioco e garantiscono che le norme vengano rispettate; il comportamento e le prestazioni dell'arbitro e degli assistenti non necessitano di mostrare i limiti fisici tipiche degli esseri umani;

Il sistema software deve includere almeno:

- Un software di base, centralizzato o distribuito, che implementi tutta la logica della simulazione;
- Un pannello grafico di sola lettura per la visualizzazione del campo di calcio, i giocatori, la palla, l'arbitro e gli assistenti; per le figure umane in campo è sufficiente a rappresentarle come puntini numerati che si muovono sul display senza ricorrere a sofisticati rendering grafici, come in una visione di un tavolo Subbuteo visto dall'alto;
- Due distinti pannelli grafici di lettura-scrittura per permettere all'utente di influenzare l'azione altrimenti indipendente del team manager; il pannello deve visualizzare i parametri attuali per ogni giocatore; la frequenza di aggiornamento dei display saranno configurabili dall'utente;
- Un pannello grafico di sola lettura per la visualizzazione di statistiche configurabili dall'utente; la frequenza di aggiornamento di questo display sarà configurabile dall'utente;
- Il nucleo del software deve essere sviluppato in Ada. La progettazione del software deve permettere la sostituzione a piacimento degli algoritmi principali: in altre parole, il sistema software deve essere il più modulare, configurabile e scalabile possibile. Queste qualità contribuiranno alla valutazione;
- I pannelli grafici possono essere programmati in qualsiasi linguaggio che i partecipanti considereranno idonei allo scopo. La bellezza grafica dei pannelli sarà tuttavia solo un fattore secondario nella valutazione. Ciò che importa invece è che l'interazione e il flusso di dati e di controllo tra il nucleo e il software

1.2 Requisiti

Possiamo riassumere brevemente i requisiti principali del sistema, ricapitolandoli in una tabella riassuntiva esplicativa. Non vengono proposti, nel presente elaborato, nessun tipo di diagramma Use-Case UML in quanto lo scopo del documento non è quello di effettuare un'analisi approfondita dei requisiti, ma di individuare le necessità principali che il software deve soddisfare. Si tratta principalmente di riepilogare quanto proposto nel capitolo precedente al fine di schematizzare gli obiettivi principali.

Requisito	Descrizione	Obbligatorio
R01	Possibilità da parte dell'utente di giocare un singolo match	Si
R02	Configurabilità dei parametri principali	Si
R03	Configurabilità dei parametri fisici dei giocatori	Si
R04	Dinamicità delle caratteristiche ed evoluzione durante il match (Es: stanchezza)	Si
R05	Presenza per ogni squadra di un manager	Si
R06	Il manager gestisce le configurazioni della squadra in campo	Si
R07	I giocatori giocano sulla base delle decisioni del manager	Si
R08	Presenza di un arbitro che controlla il rispetto delle regole	Si
***	***	***
***	***	***
***	***	***

Tabella 1: La lista dei requisiti obbligatori del sistema.

La colonna obbligatorio indica la necessità o meno del requisito al fine del funzionamento del sistema. Tutti i requisiti opzionali devono essere intesi come caratteristiche non necessarie all'implementazione di base del sistema, ma forniscono miglierie non trascurabili al sistema.

2 Analisi

2.1 La Struttura di Gioco

Il progetto prevede una serie di classi che compongono la struttura del gioco. Il riepilogo presenta gli elementi con una breve descrizione. Ogni singolo elemento verrà esaminato nelle sezioni successive entrando nel dettaglio delle varie componenti e dei vari metodi necessari per il corretto funzionamento del sistema.

Elemento	Descrizione
Campo	Dimensioni della superficie di gioco e gestione dei posizionamenti.
Arbitro	Controlla la regolarità del gioco.
Manager	Effettua le scelte per la squadra in base agli eventi.
Giocatore	Gioca la partita nella squadra.
Pallone	Oggetto condiviso dai giocatori

Tabella 2: La lista degli elementi che compongono il sistema.

2.1.1 Campo

Le dimensioni del campo dipendono dal numero dei giocatori. Indichiamo le dimensioni ufficiali LNP¹ sulla base del numero di componenti per ogni singola squadra:

Num. Giocatori	Lunghezza	Larghezza
5	42	22
7	55	32
8	65	38
11	105	68

Tabella 3: Le misure standard dei campi da calcio e calcetto.

Tutte le misure sopra indicate sono da considerarsi espresse in metri².

Le dimensioni del campo da gioco sono fondamentali rispetto al numero dei giocatori in quanto determinano la dimensione del movimento ovvero quanto il giocatore possa procedere in una direzione prima di finire fuori dal bordo.

¹Lega Nazionale Professionisti

²Un metro definito come la distanza percorsa dalla luce nel vuoto in un intervallo di tempo pari a $1/299.792.458$ di secondo.

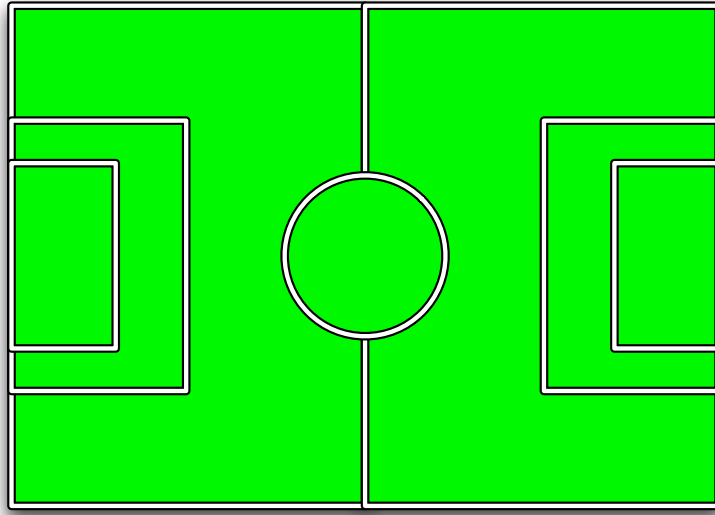


Figura 1: Il campo da gioco

Il campo è rappresentato attraverso un rettangolo in cui vengono definite le dimensioni dell'area complessiva di gioco e le aree in cui, esclusivamente per i portieri, è consentita presa con le mani. La rappresentazione del campo per l'elaboratore è una griglia composta da $2M \times 2N$ righe di separazione, dove M è la lunghezza del campo misurata in metri, mentre N è la larghezza. Potenzialmente, suddividendo il reticolato in maglie più strette, è possibile nascondere all'occhio umano il reticolato simulando un movimento fluido piuttosto che scattoso.

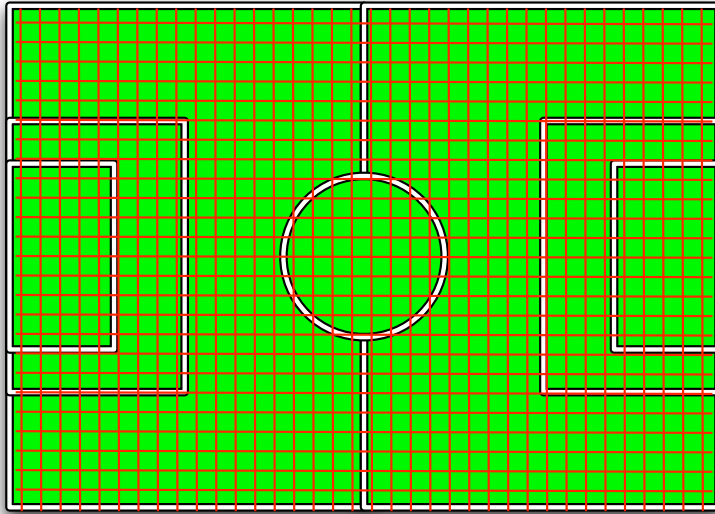


Figura 2: Il campo da gioco con il reticolato su cui avvengono i movimenti

2.1.2 Arbitro

Definiamo l'arbitro come un'entità che assolve il compito di giudicare alcuni eventi che sono segnalati come critici. Variabili aleatorie permettono di simulare il margine di errore decisionale rispetto a tali eventi.

L'arbitro viene assistito da due collaboratori, i guardalinee, che influenzano la decisione attraverso altre 2 variabili aleatorie, spostando il peso della decisione dell'arbitro a favore o contro il fischio.

Non vi sono elementi nel campo da gioco che rappresentano l'arbitro e i suoi collaboratori. Potenzialmente è possibile implementare solamente il fischio nel momento in cui viene segnalata una irregolarità di gioco.

2.1.3 Manager

Anche il manager della squadra, come per gli arbitri, non è rappresentato nel campo da gioco. Non interviene attivamente al gioco sul campo, ma si occupa di seguire la logica con la quale la squadra gioca la partita. Interviene nella scelta della strategia di gioco, nella disposizione iniziale della formazione, nelle eventuali variazioni necessarie a seguito dei risultati in campo e nelle sostituzioni dei vari giocatori all'interno del campo di gioco o tra giocatori in panchina.

Possiamo immaginare facilmente il manager come la mente della squadra che si occupa di interpretare la partita e di mettere la squadra in condizione di vincere.

2.1.4 Giocatore

Il giocatore rappresenta la forza in campo che gioca attivamente la partita, cerca di segnare il maggior numero di reti e di subirne il minimo al fine di vincere il match.

Ogni squadra dispone di n giocatori in campo di cui necessariamente:

- 1 portiere;
- 2 difensori;
- 2 centrocampisti;
- 1 attaccante;

Questa disposizione minima obbligatoria evita degli sbilanci eccessivi in favore di un reparto piuttosto che un altro, rendendo i match relativamente equilibrati e giocabili. In caso di calcetto a 5, la disposizione minima è leggermente differente per effetto del minor numero di giocatori in campo.

Ogni giocatore è contraddistinto da un pallino colorato. La squadra A è contraddistinta dal colore blu, mentre la squadra B dal colore rosso.

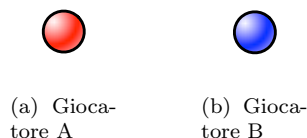


Figura 3: I giocatori delle 2 squadre.

Il colore della squadra viene assegnato automaticamente in modo casuale durante il processo di inizializzazione della partita.

2.1.5 Palla

La palla è uno degli elementi fondamentali del gioco. È l'oggetto condiviso dai giocatori e può essere controllato in ogni istante di tempo da al massimo un giocatore.

Viene mostrato nel campo da gioco come un pallino di colore nero, per contraddistinguerlo dai giocatori.



Figura 4: Il pallone da gioco

Nel momento in cui un giocatore entra in possesso del pallone, per distinguerlo dai compagni, il margine esterno viene rappresentato in grassetto. Questo permette di comprendere a colpo d'occhio quale giocatore detenga il pallone e come si muova con esso.



(a) Giocatore A con palla



(b) Giocatore B con palla

Figura 5: I giocatori delle 2 squadre in possesso del pallone.

2.2 I Dettagli di Gioco

2.2.1 Movimento

Ogni giocatore è in grado di muoversi all'interno del campo da gioco per interagire con gli altri giocatori della squadra, per contrastare gli avversari ed impossessarsi del pallone, per avanzare verso la porta con i compagni o per tirare il pallone nel momento in cui la distanza dalla rete è sufficiente per poter compiere un buon tiro e segnare un gol.

È stato deciso di semplificare il movimento nei 4 assi fondamentali:

- su;
- giù;
- destra;
- sinistra;

In questa maniera il giocatore è in grado di muoversi negli incroci del reticolato e dichiarare a compagni ed avversari la propria posizione al fine del corretto svolgimento del match. Inoltre, attraverso una serie di piccoli movimenti di questo tipo ogni giocatore è in grado di percorrere potenzialmente tutto il campo di gioco.

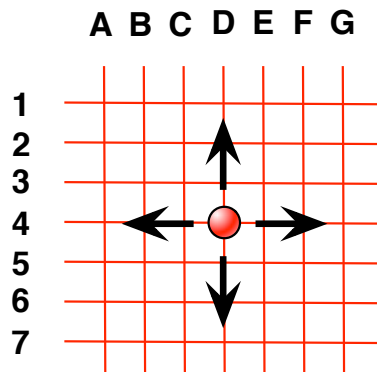


Figura 6: Gli assi del movimento dei giocatori.

2.2.2 Zone di Movimento

Per evitare che i giocatori sbilancino la squadra in modo esagerato è stato deciso di creare, per ogni area di gioco, delle “zone di movimento” che permettono di mantenere i difensori all’interno della zona di difesa, i centrocampisti entro dei margini di gioco centrali e gli attaccanti al limite del centrocampo.

In questo modo, in caso di contropiede avversario, non rimangono zone troppo scoperte e che gli avversari possono sfruttare a loro favore per effettuare azioni indesiderate.

La difesa muove sempre dalla porta fino al limite del centrocampo, evitando di entrare nella zona di campo avversaria.

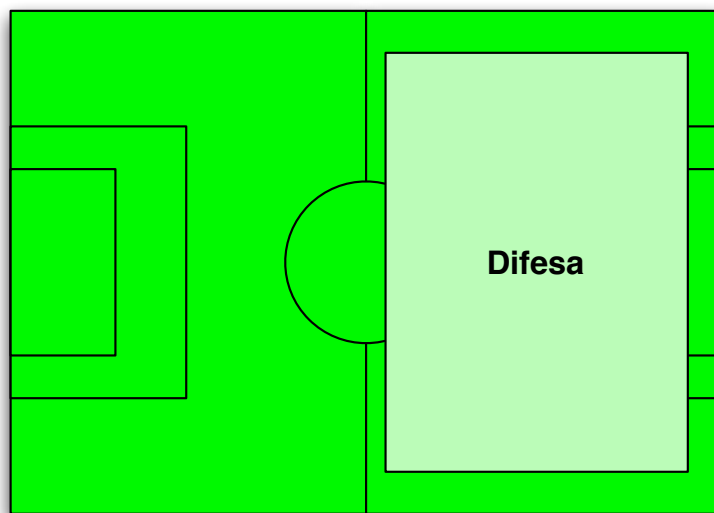


Figura 7: La zona di movimento dei difensori.

Il centrocampo può muoversi in tutta la parte centrale del campo e nelle fasce laterali per assistere gli attaccanti e i difensori.

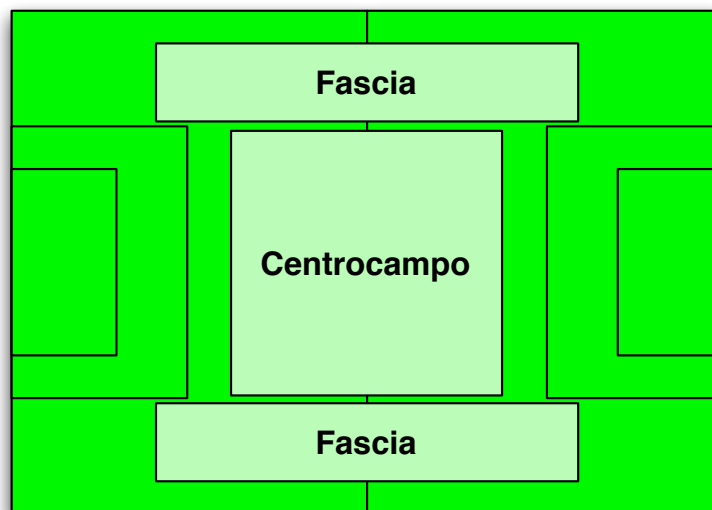


Figura 8: La zona di movimento dei centrocampisti.

Gli attaccanti invece muovono in tutta la zona del campo avversario senza mai retrocedere oltre il limite del centrocampo, per poter intervenire in caso di passaggi da parte dei compagni in caso di contropiede.

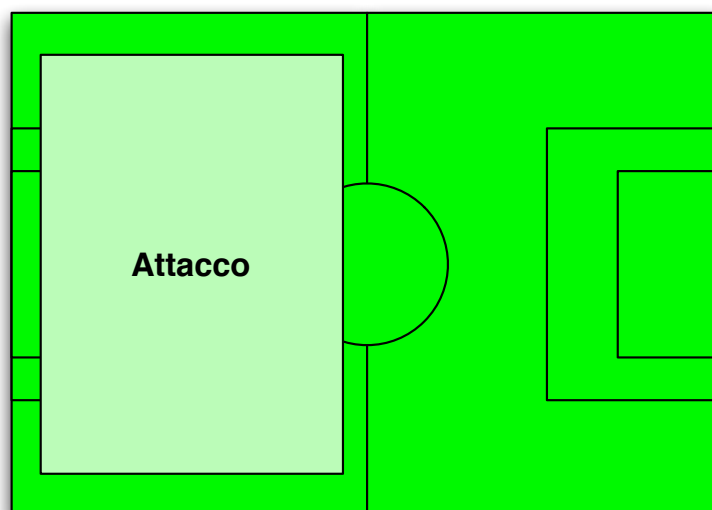


Figura 9: La zona di movimento degli attaccanti.

2.2.3 I Giocatori

Ogni giocatore, oltre al movimento all'interno del campo è in grado di effettuare alcune azioni di base ed in particolar modo legate ad alcune condizioni fondamentali:

- possesso del pallone da parte della propria squadra:
 - del giocatore stesso
 - di un compagno
- possesso del pallone da parte degli avversari

Definiamo con il termine *logica* una modalità di gioco relativa ad ogni singolo giocatore e legata alle condizioni di cui sopra.

Elenchiamo brevemente le possibilità che un giocatore può trovarsi a dover affrontare, analizzandole poi dettagliatamente nei paragrafi successivi:

1. il giocatore ha la palla:
 - logica di passaggio
 - logica di tiro
 - logica di avanzamento
 - logica di posizione
2. la squadra ha la palla:
 - logica di smarcatura
 - logica di avanzamento
 - logica di posizione
3. gli avversari hanno la palla:
 - logica di indietreggiamento
 - logica di contrasto
 - logica di posizione
4. il giocatore in porta:
 - logica di contrasto (sull'attaccante)
 - logica di parata (solo su tiro avversario)
 - logica di posizione

3 Applicazioni delle Logiche

Nella presente sezione analizziamo le formule ed i coefficienti che permettono di determinare, rispetto all'elenco visto precedentemente, le varie azioni che i giocatori possono compiere ed i risultati a cui possono portare. Come abbiamo detto, utilizzeremo il termine “logica” per definire una modalità con cui il giocatore determina le possibilità di gioco, sceglie l'azione migliore e la effettua.

Cominciamo definendo un particolare fondamentale: la “visibilità” del giocatore.

Con visibilità intendiamo la cognizione dello stato di gioco nell'istante t da parte del giocatore in questione. Per semplicità è stato deciso di definire con “visibilità” un quadrato di lato x (con x dispari) e il giocatore oggetto della funzione risiede nell'intersezione centrale. Questo permette al giocatore di conoscere la situazione di compagni ed avversari per tutte le celle contenute nel quadrato.

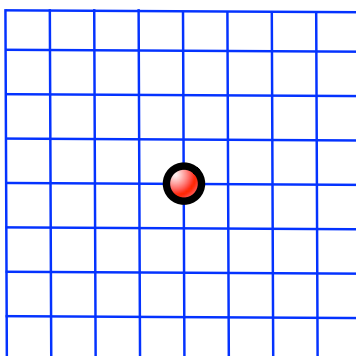


Figura 10: La “visibilità” dei giocatori in gioco.

Inoltre è possibile rendere il parametro x soggettivo rispetto ad ogni giocatore, il che permette a giocatori come il regista di avere una grande visione di gioco rispetto a difensori ed attaccanti.

Tutti i movimenti e i posizionamenti dei giocatori che verranno affrontati nei prossimi capitoli sono sempre da intendersi legati ai ruoli che i giocatori assumono nel campo e alle aree di movimento a cui sono vincolati. Si faccia riferimento alla sezione 2.2.2 per i dettagli delle zone in cui i giocatori sono liberi di muovere.

In tutto il capitolo trascureremo ogni aspetto legato alla concorrenza, tematica che verrà affrontata nel dettaglio al capitolo 5.

3.1 Giocatore con Palla

Il caso più complesso che possiamo affrontare è quello legato al possesso di palla da parte del giocatore. In questa casistica dobbiamo risolvere la maggior parte dei problemi legati alle scelte dei giocatori e determinare le regole con cui avanzano, tirano o passano il pallone ai compagni di squadra.

La priorità con cui un giocatore effettua un'azione è:

1. tiro in porta
2. passaggio ad un compagno
3. avanzamento con palla

4. mantenimento della posizione

Questo perchè se il giocatore vede la porta, effettua un tiro per tentare di segnare una rete. Altrimenti, il passaggio è il metodo più rapido per muovere velocemente la palla ed avanzare verso la porta. In caso non ci sia visione della porta e non ci siano compagni liberi a portata, si processa un avanzamento di posizione. Il mantenimento della posizione è una casistica limite che cercheremo sempre di evitare, ma che ci permetterà di risolvere situazioni complicate.

3.1.1 Logica di Tiro

La prima logica che andiamo ad analizzare è quella legata al tiro in porta.

Un giocatore che inizia la logica di tiro ha sicuramente visibilità sulla porta e pertanto non ne è relativamente distante.

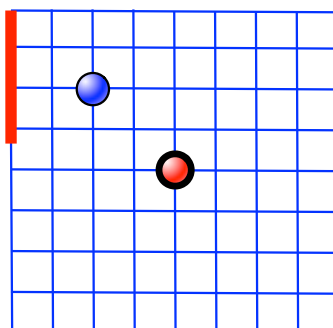


Figura 11: Giocatore con palla e visione della porta (linea rossa).

Il tiro può essere un buon tiro o un tiro mediocre. Inoltre può essere troppo potente o troppo debole e può venir parato dal portiere avversario. Per simulare un comportamento realistico del tiro, andiamo a considerare alcuni fattori fondamentali:

- potenza
- precisione
- distanza

Questi 3 coefficienti fortemente soggettivi rispetto al giocatore e alla posizione, vengono combinati assieme in un'equazione per produrre un coefficiente che verrà valutato assieme al coefficiente di parata del portiere (che vedremo in seguito nella sezione 3.4.2) per determinare il successo o il fallimento del tiro e la conseguente rete della squadra.

L'equazione è composta da

$$T_1 = \frac{potenza}{100} \cdot \frac{precisione}{100} \cdot \left(1 - \frac{distanza}{diagonale}\right)$$

dove potenza è il valore di potenza del giocatore espresso con un valore da 1 a 100, precisione è la precisione di tiro, anch'essa espressa con un valore da 1 a 100, mentre l'ultimo coefficiente è il risultato del rapporto inverso tra la distanza del giocatore dalla porta e la diagonale del campo.

Per dare un'idea di come varino i vari coefficienti ne esprimeremo alcuni campioni nella tabella sottostante ed andremo a definire che cosa rappresentano in termini di gioco.

Potenza	Precisione	Distanza	Coefficiente
0.3	0.3	0.3	0.027
0.3	0.3	0.6	0.054
0.3	0.3	0.9	0.081
0.3	0.6	0.3	0.054
0.3	0.6	0.6	0.108
0.3	0.6	0.9	0.162
0.3	0.9	0.3	0.081
0.3	0.9	0.6	0.162
0.3	0.9	0.9	0.243
0.6	0.3	0.3	0.054
0.6	0.3	0.6	0.108
0.6	0.3	0.9	0.162
0.6	0.6	0.3	0.108
0.6	0.6	0.6	0.216
0.6	0.6	0.9	0.324
0.6	0.9	0.3	0.162
0.6	0.9	0.6	0.324
0.6	0.9	0.9	0.486
0.9	0.3	0.3	0.081
0.9	0.3	0.6	0.162
0.9	0.3	0.9	0.243
0.9	0.6	0.3	0.162
0.9	0.6	0.6	0.324
0.9	0.6	0.9	0.486
0.9	0.9	0.3	0.243
0.9	0.9	0.6	0.486
0.9	0.9	0.9	0.729

Tabella 4: Alcuni valori campione per il coefficiente di tiro (media 0.216).

E' facile comprendere osservando la tabella che al crescere delle caratteristiche ci siano differenti effetti sul tiro: un tiro può essere molto potente (ultimi risultati in basso) ma non efficace perchè poco preciso ed effettuato da distanze troppo elevate. Inoltre può essere molto preciso e puntare all'incrocio dei pali, ma non sufficientemente potente o magari effettuato da troppo distante e facilmente parabile dal portiere.

Chiameremo il valore ottenuto come T_1 e dovrà essere confrontato con il risultato del calcolo relativo alla parata per determinare il successo del tiro.

Il fattore aleatorio è stato escluso dal calcolo in quanto potrebbe complicare ulteriormente i conteggi e rendere difficile segnare una rete. Non sarebbe però complesso introdurre il classico fattore c generando un numero casuale e incrementando o moltiplicando il coefficiente ottenuto precedentemente per c ottenendo un coefficiente T_2 maggiormente influenzato da leggi pseudo-casuali.

3.1.2 Logica di Passaggio

Un giocatore che ha la palla può decidere di passare il pallone ad un compagno libero. La logica con cui avviene il passaggio è legato all'avanzamento verso la porta avversaria. Non è però sempre possibile avanzare

rispetto alla porta per effetto delle distanze dei giocatori avversari all'uomo a cui si vorrebbe passare il pallone.

Anche in questo caso entra in gioco la “visibilità” del giocatore e la sua capacità di vedere compagni liberi nelle vicinanze.

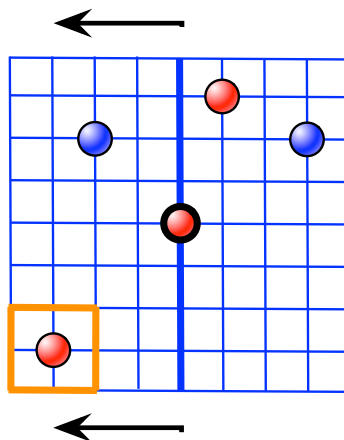


Figura 12: Giocatore con palla e visione di compagni ed avversari.

Il giocatore in possesso di palla, valuta la posizione del giocatore più vicino e più avanzato rispetto a se, controlla che sia libero entro un certo raggio ed eventualmente passa il pallone. L'equazione che regola il passaggio, in questo caso è più semplice della precedente e garantisce quasi sempre il buon fine del passaggio al giocatore destinatario entro un intervallo di tempo t . In particolare l'equazione è influenzata dalla distanza tra chi passa la palla e chi la deve ricevere e dalla precisione del passaggio:

$$P_1 = \left(1 - \frac{\text{distanza}}{\text{diagonale}}\right) \cdot \frac{\text{precisione}}{100}$$

Nel caso non si raggiunga la soglia, il pallone viene lanciato casualmente in una delle 4 direzioni per un numero variabile di intersezioni. Questo permette potenzialmente alla squadra avversaria di impossessarsi della palla in quanto ne prenderà il possesso il primo giocatore a venirne in contatto indipendentemente dallo schieramento.

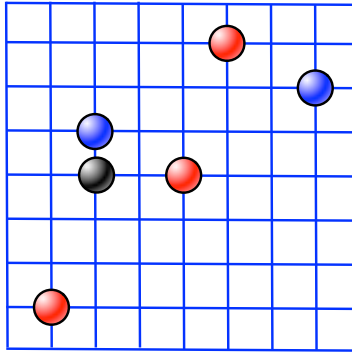


Figura 13: Giocatore avversario si avvicina al pallone prima degli altri.

Il buon fine del passaggio è soggetto al superamento della soglia minima per il passaggio pari a 0.3. Il valore di soglia è stato approssimato secondo il criterio per cui un passaggio non avviene mai a distanze elevate e la precisione dei giocatori non dovrebbe essere mai troppo bassa.

Precisione	Distanza	Coefficiente
0.3	0.6	0.18
0.3	0.8	0.24
0.6	0.6	0.36
0.6	0.8	0.48
0.9	0.6	0.54
0.9	0.8	0.72

Tabella 5: Alcuni valori campione per il coefficiente di passaggio (soglia 0.3).

Come si può facilmente vedere dalla tabella, per passaggi molto ravvicinati ($\frac{1}{4}$ della diagonale del campo), anche con una precisione molto bassa (intorno a 40 punti) il passaggio avviene con successo. Chiaramente, giocatori più precisi non sbaglieranno passaggi facili, ma potrebbero comunque avere qualche difficoltà in passaggi lunghi a giocatori distanti.

Anche qui, come nel tiro in porta, può essere introdotto un fattore aleatorio che può scombinare gli eventi: il fattore c può entrare in calcolo nell'equazione che assumerebbe la forma:

$$P_1 = \left(1 - \frac{\text{distanza}}{\text{diagonale}}\right) \cdot \frac{\text{precisione}}{100} \cdot c$$

dove c è un numero casuale tra 0,5 e 1,5. Questo potrebbe portare al fallimento di ottimi passaggi e al successo di passaggi inizialmente fallimentari.

Nella logica di passaggio non è stato affrontato il passaggio all'indietro ad un compagno di squadra. Questo per un semplice motivo. Se avessimo considerato anche i passaggi all'indietro, avremmo scatenato un meccanismo infinito di passaggi tra giocatori con una piccolissima percentuale d'errore. Potenzialmente gli interi 90 minuti di gioco potrebbero essere spesi in passaggi tra giocatori senza nemmeno un tiro in porta. Forzando il passaggio in avanti costringiamo i giocatori ad avanzare, passare o tirare in porta. In questo modo evitiamo diverse situazioni che possono causare un stallo del gioco inteso come assenza di azioni a causa di una politica non consistente.

3.1.3 Logica di Avanzamento

Un'altra azione a disposizione di un giocatore con la palla che non può effettuare un tiro e non può passare la palla è quella di avanzare con il pallone verso la porta. Un giocatore può anche muovere sulla linea verticale verso l'alto o verso il basso, specialmente verso un compagno di squadra che si sta smarcando per ricevere il passaggio del compagno.

Un giocatore potrebbe però continuare con un movimento ripetitivo dall'altro verso il basso e viceversa per tutta la durata della partita. Per evitare una situazione di questo genere, è stato scelto di limitare a y il numero di movimenti in orizzontale fino al successivo avanzamento verso la porta avversaria.

Per risolvere il problema è sufficiente inserire nella classe *Giocatore* una variabile y inizializzata al numero di movimenti verticali massimi. Un contatore decrementa il valore di y per ogni movimento consecutivo sull'asse verticale. Quando y raggiunge il valore 0, l'avanzamento è obbligatorio e y viene reimpostata al valore iniziale.

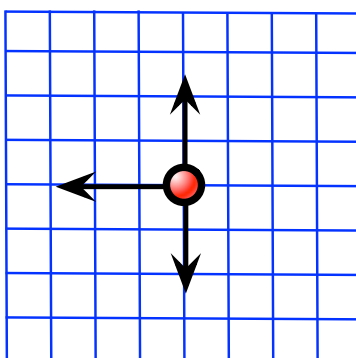


Figura 14: Giocatore in fase di movimento con palla.

3.1.4 Logica di Posizione

Il giocatore ha come azione di default in caso di emergenza un'ultima logica che viene utilizzata nel momento in cui nessuna delle logiche precedenti porti ad un risultato: il mantenimento della propria posizione in campo senza alcun tipo di movimento.

È una logica che non determina alcuna variazione del gioco, ma che permette di compiere in ogni caso un'azione e di restituire un parametro di "azione compiuta" alla classe "Timer" per poter procedere nell'invio del successivo istante temporale $t+1$.

3.2 Squadra con Palla

Un tipo di logica differente si applica quando il giocatore non è direttamente in possesso della palla, ma lo è un compagno di squadra. La strategia di movimento del giocatore deve essere finalizzata ad assistere il giocatore con la palla al fine di poter ricevere un passaggio e procedere verso la porta.

3.2.1 Logica di Smarcatura

Un giocatore senza palla deve garantire ai compagni di essere libero nel momento in cui sia necessario passare il pallone. È fondamentale infatti che assista la squadra nell'avanzamento verso la porta e si proponga per ricevere il pallone ed effettuare a sua volta un'azione.

In questo modo l'azione prioritaria è il movimento verso una intersezione distante almeno n intersezioni da uno o più avversari, dando prevalenza ad azioni di avanzamento rispetto a quelle di arretramento in modo da favorire l'attacco verso la porta avversaria.

L'azione di spostarsi verso un punto distante almeno n intersezioni consecutive da tutti gli avversari è detta "smarcatura".

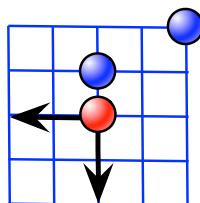


Figura 15: Giocatore in fase di smarcatura senza palla.

3.2.2 Logica di Avanzamento

Nel momento in cui il giocatore risulti già smarcato rispetto agli avversari si applica la stessa logica di avanzamento che si applica ad un giocatore in possesso di palla: il movimento deve essere direzionato in avanti verso la porta avversaria. Valgono le stesse regole viste nella sezione 3.1.3.

3.2.3 Logica di Posizione

Come già visto nella sezione 3.1.4, anche per il giocatore senza palla esiste un'azione di "emergenza" che permette di evitare situazioni di stallo determinate da impossibilità di compiere un movimento nel campo.

3.3 Avversari con Palla

Finora abbiamo visto tutte le logiche al possesso del pallone da parte della squadra o del giocatore, ma grossomodo per il 50% della durata della partita la palla sarà in mano ai giocatori avversari e la squadra dovrà assumere un atteggiamento difensivo al fine di evitare che gli avversari possano segnare una rete. Analizziamo le logiche di gioco che permettono di difendere la porta.

3.3.1 Logica di Contrasto

Nel momento in cui gli avversari prendono possesso del pallone, è necessario un meccanismo che permetta al giocatore di recuperare il pallone al fine di invertire le parti e portare la propria squadra in fase di attacco. Questa regola è definita come logica di contrasto ed è rappresentata nel campo di gioco da un contatto fisico tra giocatori. Nel contrasto, i due giocatori si scontrano per potenza e il più forte prende il controllo del pallone.

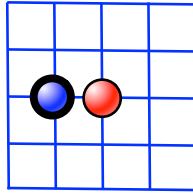


Figura 16: Giocatore in fase di contrasto contro un avversario con palla.

Il contatto fisico avviene tra due giocatori in posizione ravvicinata, a distanza di un'intersezione l'uno dall'altro. La logica con cui avviene il contrasto viene applicata ad entrambi i giocatori e rispetta la formula:

$$C_1 = \frac{potenza}{100} + c$$

dove potenza è la potenza del giocatore e c un fattore casuale con valore compreso tra -0.2 e 0.3 che rappresenta un rimpallo a favore o contro il giocatore.

Nel momento in cui avviene il contrasto l'oggetto condiviso viene liberato per l'istante t in cui viene valutato il contrasto e poi preso dal giocatore che ottiene il valore superiore rispetto al valore di contrasto.

In caso la formula produca due valori uguali, il giocatore in difesa vince il contrasto e prende il controllo della risorsa e il gioco ricomincia.

I dettagli legati alla liberazione della risorsa e alla ripresa della stessa a seguito del contrasto vengono discussi nel capitolo ??.

3.3.2 Logica di Indietreggiamento

Nel momento in cui un giocatore non ha visibilità sul giocatore con la palla può effettuare una sola azione: indietreggiare a supporto della fase difensiva.

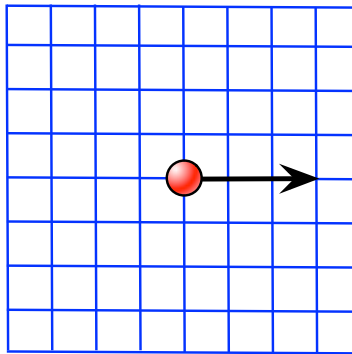


Figura 17: Giocatore in fase di indietreggiamento.

In questa maniera, il giocatore si avvicina ai giocatori in difesa, limita le aree di movimento degli avversari in fase di attacco e, in caso di recupero del pallone, si propone per il passaggio da parte dei compagni.

3.3.3 Logica di Posizione

Anche nel caso in cui la propria squadra non sia in possesso del pallone, come già visto nella sezione 3.1.4, esiste la solita azione di “emergenza” ovvero il mantenimento della propria posizione in campo.

3.4 Parata del Portiere

L'unico giocatore eccezionale nel campo di gioco è il portiere. Questo particolare giocatore non segue le regole valide per tutti gli altri giocatori nel campo di gioco ma è soggetto a delle proprie logiche che lo differenziano nel modo di giocare, di muoversi e di reagire agli eventi come avanzamenti dell'attaccante avversario e al tiro in porta.

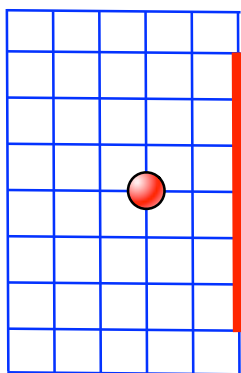


Figura 18: Portiere in posizione difensiva.

Vediamo nel dettaglio le logiche che ne regolano i movimenti e la posizione.

3.4.1 Logica di Contrasto

La logica primaria che permette il movimento del portiere è il contrasto con il giocatore avversario. Limitatamente all'area di gioco entro cui il portiere ha la possibilità di muoversi, il portiere cerca il contatto fisico con l'avversario che controlla il pallone. In questo modo può tentare uno scontro fisico e prendere il controllo del pallone evitando che l'attaccante possa segnare una rete.

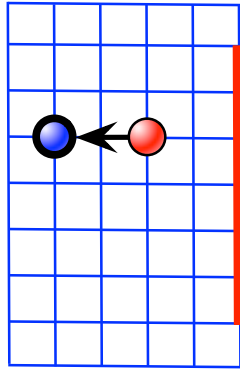


Figura 19: Portiere in fase di contrasto.

Le regole per la determinazione del contrasto sono uguali a quelle viste nella sezione 3.3.1. L'unica differenza è relativa al peso del fattore c . In questo caso, poichè il portiere è l'unico giocatore a cui è permesso l'intervento con le mani, il valore di c nella formula:

$$C_2 = \frac{potenza}{100} + c$$

varia tra 0 e 0.2, rendendo la potenza minima del portiere pari alla propria e permettendo solo incrementi su di essa.

3.4.2 Logica di Parata

Nel momento in cui un giocatore effettua un tiro in porta, il portiere effettua l'unica azione disponibile: tenta una parata del tiro.

L'azione di parare un tiro è indipendente dalla posizione attuale del portiere in quanto un tuffo gli permette di muoversi velocemente da una parte all'altra della porta.

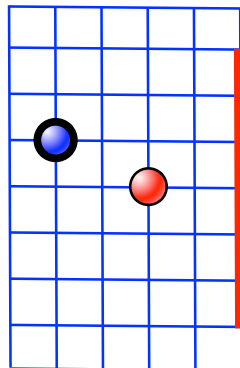


Figura 20: Portiere in fase di parata del tiro avversario.

L'equazione per determinare il valore della parata è uguale a:

$$P_1 = \frac{agilità}{100} \cdot \frac{reattività}{100} \cdot \left(1 - \frac{distanza}{diagonale}\right)$$

dove agilità è il valore di agilità del portiere espresso con un valore da 1 a 100, reattività è la velocità di reazione, anch'essa espressa con un valore da 1 a 100, mentre l'ultimo coefficiente è il risultato del rapporto inverso tra la distanza del portiere dal giocatore che sta effettuando il tiro e la diagonale del campo.

La parata è in contrasto con il tiro visto nella sezione 3.1.1. Il tiro genera un valore che viene confrontato con quello generato dalla parata del portiere. In caso il valore P_1 sia inferiore a T_1 , il portiere non è riuscito a parare il tiro (per merito dell'attaccante o demerito del portiere) e la palla è finita in rete. Nel caso in cui il valore P_1 sia uguale o maggiore di T_1 il portiere ha compiuto una parata e ha salvato la porta.

L'azione ricomincia dal portiere con un passaggio verso un compagno per tentare un'azione d'attacco.

3.4.3 Logica di Posizione

Anche il portiere, in qualità di giocatore come tutti gli altri, ha la possibilità di gestire una situazione di "emergenza" rimanendo in posizione e non generando movimenti. Si faccia riferimento alla sezione 3.1.4 per i dettagli della logica.

4 Modello

Analizziamo ora il modello del sistema e la composizione delle classi e dei metodi principali necessari al corretto funzionamento del software come descritto nella tabella sottostante.

Classe	Identifica	Descrizione
Field	Campo	Gestisce le dimensioni della superficie di gioco e il posizionamento di giocatori e palla su di esso.
Referee	Arbitro	Controlla la regolarità del gioco e fischia eventuali scorrettezze.
Manager	Manager	Gestisce il modulo di gioco, effettua i cambi tra giocatori in campo ed in panchina e schiera la formazione d'attacco o di difesa.
Team	Squadra	Associa i vari giocatori alla squadra di cui fanno parte.
Player	Giocatore	Gioca la partita assieme ai compagni di squadra.
Ball	Pallone	Oggetto condiviso dai giocatori

Tabella 6: La lista degli elementi principali che compongono il sistema.

Le classi principali sono le 6 classi fondamentali appena presentate. Se mancasse anche solo una di esse, non si potrebbe realizzare il gioco nella nostra realtà virtuale. Ognuna di esse rappresenta un elemento presente e visibile nel campo di gioco ed attivamente partecipa allo svolgimento della partita. La sola eccezione vale per la classe *Team* che si occupa di raggruppare i giocatori in un'unica squadra.

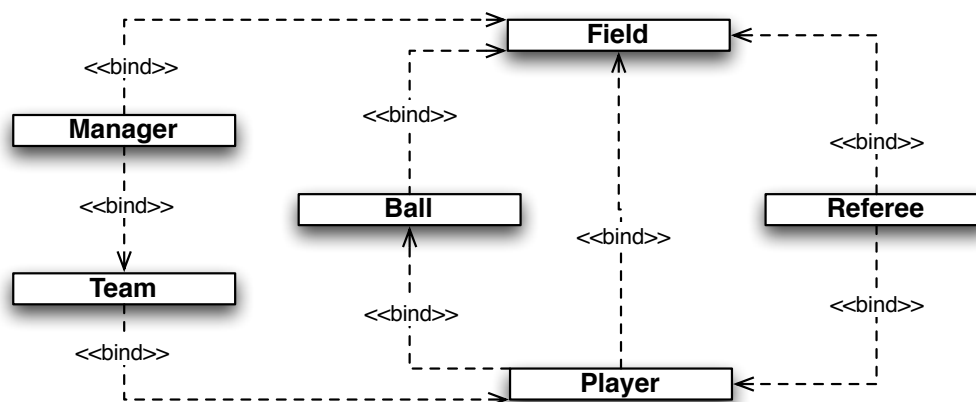


Figura 21: Diagramma generale delle classi del sistema.

Lo schema esposto presenta le relazioni che intercorrono tra le classi principali del nostro gioco oltre all'indicazione del verso in cui la comunicazione è direzionata.

Approfondiamo nelle prossime sezioni le singole classi entrando nel dettaglio per comprendere la loro tipologia e le funzioni principali che permettono il corretto svolgimento del gioco.

4.1 Campo

Il campo deve essere la classe di riferimento necessariamente accessibile a tutti i giocatori in campo. Si occupa di salvare i posizionamenti dei giocatori nell'area di gioco a seguito dei movimenti effettuati e di fornire alcune informazioni legate al campo di gioco come ad esempio la posizione del pallone e la dislocazione delle porte.

Ad ogni giocatore che effettua l'interrogazione di stato restituisce:

- la posizione nel campo di gioco;
- le direzioni percorribili;
- la posizione dei compagni;
- la posizione degli avversari;
- la posizione del pallone;

Ogni giocatore deve poter accedere alla classe *Field*, effettuare la richiesta del vettore delle posizioni ed attendere una risposta dalla classe. Essa restituirà una lista contenente le informazioni relative alla posizione dei compagni, degli avversari e del pallone necessarie al giocatore per poter compiere il proprio movimento in gioco coerentemente con gli spazi disponibili e con la logica applicata.

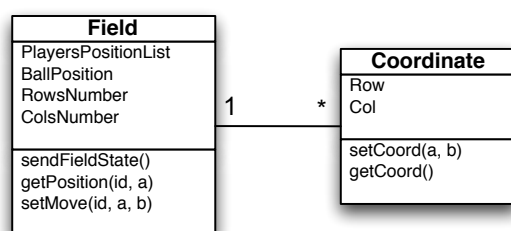


Figura 22: Diagramma delle classi per il campo di gioco.

I valori delle dimensioni di gioco e delle aree di movimento sono inizializzate non appena vengono definite la tipologia di partita (calcio/calciotto) e il numero di giocatori in campo.

Un vettore di posizioni contiene i vari posizionamenti dei giocatori e lo schieramento a cui appartengono. Una variabile ad-hoc contiene invece il posizionamento del pallone e viene trattato in modo simile al vettore precedente.

Le funzioni che permettono di accedere ai vettori di stato sono:

function GetPositions return PositionList che ritorna un listato di posizioni dei vari giocatori in campo;

function GetBallPosition return Position che restituisce la posizione della palla in campo;

Nel caso si trovi al limite della propria area di movimento, le uniche azioni possibili sono: rimanere nella propria posizione e non compiere alcun movimento o, in alternativa, muovere lateralmente per avvicinarsi al giocatore con la palla (se visibile).

Il campo gestisce anche alcuni eventi particolari legati all'uscita del pallone dall'area di gioco. Mentre possiamo ammettere che un giocatore rimanga tutta la partita in campo e quando raggiunga il bordo del campo non possa attraversarlo, è lecito pensare che il pallone fuoriesca dall'area di gioco. In questo caso viene lanciata una eccezione e la palla passa alla squadra avversaria. Si possono verificare 3 casi principali:

Fallo laterale avviene quando un giocatore lancia la palla oltre la linea laterale del campo. La palla passa ad un giocatore avversario e il gioco riprende dal bordo in cui è avvenuta l'uscita, punto in cui vengono riposizionati giocatore e pallone;

Rimessa dal fondo è causata da un giocatore avversario che scaglia la palla oltre la linea della porta avversaria. Il pallone passa al portiere e il gioco ricomincia dal fondo del campo;

Calcio d'angolo viene fischiato quando un giocatore lancia la palla oltre la linea della propria porta. Il gioco riprende dall'angolo con il riposizionamento di palla e giocatore;

La funzione che permette all'arbitro di resettare le posizioni in base all'evento è:

procedure ResetPosition(ID : in Integer; P : Position) nella quale viene passata l'ID del giocatore che andrà a battere la rimessa e la posizione in cui verranno posizionati giocatore e pallone;

4.2 Arbitro

L'arbitro è di per se un'entità molto semplice: non è presente direttamente nel campo di gioco e viene richiamato in casi chiaramente definiti per determinare il corretto svolgimento del gioco.

Le casistiche in cui l'arbitro viene chiamato in causa sono:

- contrasto tra due giocatori (possibile fallo);
- palla oltre il limite del campo:
 - fallo laterale;
 - rimessa dal fondo;
 - calcio d'angolo;
- fuorigioco (non trattato);

In tutti i casi, il fischio dell'arbitro comporta il fermo del gioco, la riassegnazione della palla e l'inizio di una nuova azione.

Definiamo cosa fa l'arbitro nei casi sopra esposti:

- fallo tra giocatori:
 1. mantiene la posizione della palla sul punto del fallo;
 2. mantiene la posizione del giocatore che ha subito il fallo vicino alla palla;
 3. riposiziona posiziona l'altro giocatore a distanza definita;
 4. fa riprendere il gioco;
- palla fuori dal campo:
 1. posiziona la palla sul bordo del campo nel punto di uscita;
 2. posiziona il giocatore sulla palla;
 3. ne assegna il possesso;

4. fa riprendere il gioco;

Nello schema sottostante si comprende facilmente come l'arbitro intervenga nelle varie fasi di gioco ed interagisca con le altre classi.

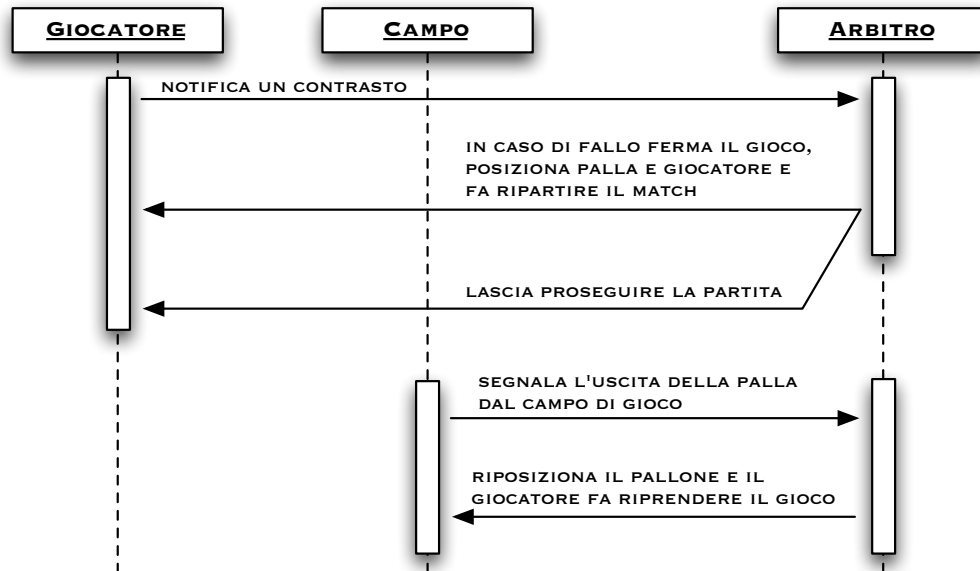


Figura 23: Diagramma di sequenza dell'arbitro di gioco.

L'unico aspetto determinante rispetto a quanto visto sotto è legato esclusivamente alla priorità con cui l'arbitro interviene nel gioco: per un'analisi dettagliata delle priorità in campo, rimandiamo alla sezione [5.3](#) in cui vengono trattate nello specifico.

4.3 Manager

Il Manager deve essere considerato la mente del gioco in quanto non partecipa attivamente alla partita ma si occupa di definire le strategie di gioco e i cambi tra giocatori in campo o in panchina. Le scelte legate alla posizione dei giocatori e alle strategie sono legate ad alcuni fattori oggettivi che permettono al manager di prendere le decisioni opportune per modificare la squadra in campo:

- risultato parziale della partita;
- numero di giocatori in campo (a seguito di potenziali espulsioni o infortuni);
- sbilanciamento attuale della formazione (attacco o difesa);
- formazione in campo;
- modulo di gioco;

Manager è un thread che viene risvegliato ad intervalli regolari per visionare e comprendere l'andamento del match ed apportare correzioni alla squadra oppure in casi particolari in cui viene risvegliato. Gli eventi che scatenano il Manager possono essere identificati da:

- gol della propria squadra;

- gol della squadra avversaria;
- energia di un giocatore che scende sotto una soglia predefinita (e necessita un cambio);
- giocatore che subisce un infortunio durante la partita;
- fischio dell'arbitro per contrasto tra giocatori;
- espulsione di un giocatore in campo;

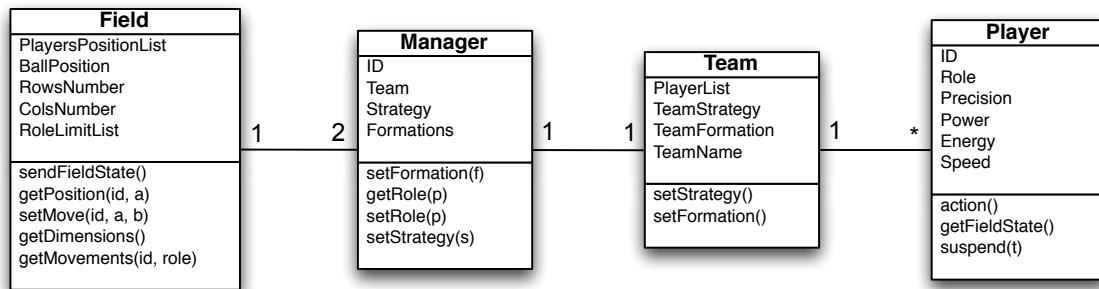


Figura 24: Diagramma delle classi per il manager della squadra.

Come descritto nel diagramma sopra esposto, il manager ha una relazione diretta con i giocatori attraverso la classe *Team* con la quale può visionare lo stato dell'affaticamento ed eventuali ulteriori caratteristiche proprie di ogni singolo elemento. Inoltre è direttamente collegato con la classe squadra che raggruppa tutti i giocatori per apportare eventuali variazioni a cui è soggetta tutta la squadra piuttosto che un singolo giocatore specifico.

Può inoltre accedere alla classe *Field* che gli permette di conoscere il posizionamento dei giocatori in campo nonché quello della palla attraverso la funzione *getFieldState()*, uguale a quella utilizzata dai giocatori. Anche questi fattori sono determinanti al fine di prendere decisioni per la propria squadra.

A seguito dei risultati della propria elaborazione, possono essere effettuati interventi come:

- cambio di formazione;
- sostituzione di un giocatore;
- cambio di posizione tra giocatori in campo;
- cambio della propensione di gioco (attacco/difesa);

Nel diagramma sottostante invece viene analizzato lo schema di comunicazione degli stati da parte del Manager che, come già visto, interroga il campo e i giocatori per ottenere il quadro della situazione per effettuare le variazioni necessarie.

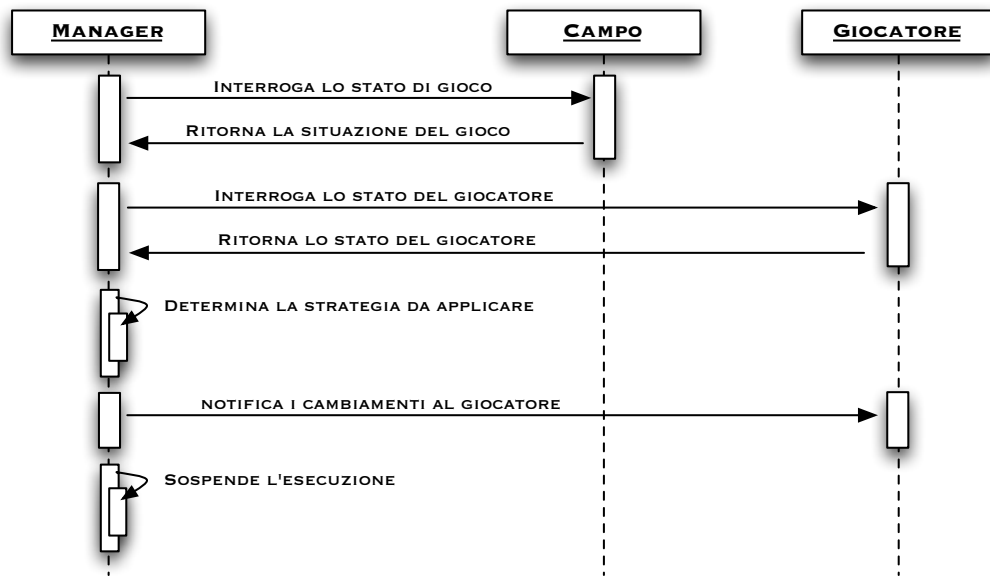


Figura 25: Diagramma di sequenza del manager.

Anche il Manager deve avere una priorità differente rispetto a quella dei giocatori in campo. Rimandiamo alla sezione 5.3 per ulteriori chiarimenti.

4.4 Squadra

La classe squadra è una classe molto semplice nella nostra realtà di gioco. È l'elemento che funge da contenitore per i giocatori e che permette di raggrupparli in due insiemi differenti.

La classe squadra permette di facilitare il coordinamento dei giocatori, la diffusione di alcune specifiche impartite dal Manager a tutti i giocatori della squadra in campo e non soltanto a qualche istanza di giocatore particolare. Il modulo in campo e lo sbilanciamento della squadra in attacco o in difesa sono caratteristiche generali e l'oggetto di riferimento è la classe squadra, piuttosto che tutti i singoli giocatori che le appartengono.

I metodi principali appartenenti alla classe *Team* sono:

procedure SetStrategy(S : in Strategy) che imposta la strategia di gioco;

procedure SetModule(M : in Module) che setta il modulo di gioco;

function GetStrategy return Strategy restituisce ai giocatori ed al manager la strategia applicata;

function GetModule return Module restituisce ai giocatori ed al manager il modulo di gioco in uso;

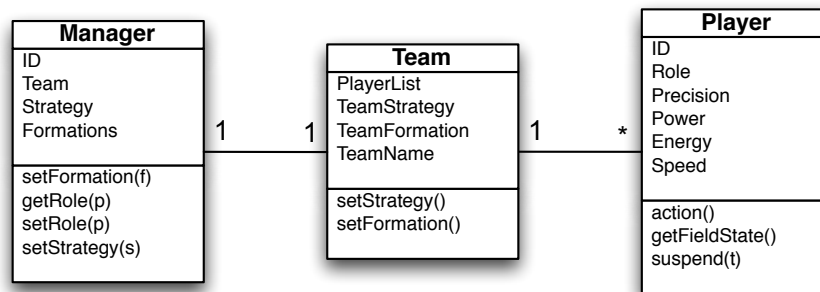


Figura 26: Diagramma delle classi per la classe squadra.

Manager è l'unica entità in grado di accedere direttamente alla squadra ed effettuare variazione. I giocatori subiscono tali modifiche in riferimento alla classe squadra di cui fanno parte e leggono le modifiche apportate attraverso i comandi *Get*.

4.5 Giocatore

I giocatori sono i task di base necessari per lo svolgimento del gioco e vengono intesi come multiple istanze dello stesso task.

Ognuno di essi è contraddistinto da una serie di caratteristiche proprie e legate al ruolo, alla posizione e alle abilità:

- **ID:** ogni giocatore possiede un **ID univoco** che permette di essere identificato rispetto a tutti gli altri giocatori in campo per poter essere associato come possessore della risorsa condivisa o identificato nel vettore delle posizioni in campo;
- **Ruolo:** determina il posizionamento in campo e le possibilità di movimento all'interno dell'area specifica di gioco.
- **Visibilità:** è il parametro che determina l'abilità di visione del gioco di ogni singolo giocatore.
- **Potenza:** è il parametro che determina la potenza di tiro e di contrasto del giocatore. È un valore tra 0 e 100.
- **Precisione:** contraddistingue l'abilità del giocatore in questione nel giostrare il pallone.
- **Velocità:** identifica la velocità del giocatore ovvero la quantità di movimenti che può compiere rispetto ad altri.
- **Energia:** rappresenta la resistenza del giocatore durante la partita. È un parametro che decrementa (in modo lineare o esponenziale) con il proseguire delle azioni di gioco fino alla fine della partita.

Ogni giocatore è definito da un processo che compie in maniera ciclica tre azioni base:

- **getFieldState():** il giocatore apprende lo stato del gioco ed in particolare il posizionamento dei compagni, degli avversari e della palla;
- **action():** sulla base dello stato del gioco e limitatamente alla propria visibilità, ogni giocatore effettua una mossa tra quelle definite nelle logiche di gioco a disposizione;

- **suspend()**: al termine dell'azione, ogni giocatore sospende la propria esecuzione per un tempo definito. Al termine dell'attesa procede con una nuova azione;

Il diagramma esposto identifica quali sia la sequenza delle azioni che ogni giocatore compie durante la propria fare di gioco. Inizialmente procede con una interrogazione del campo al fine di ottenere lo stato del gioco. Il campo risponde con un vettore contenente i posizionamenti dei vari giocatori in campo nonché quella del pallone. Segue immediatamente una fase di analisi dell'area "visibile" al fine di determinare quale sia l'azione ottimale da compiere. L'azione viene portata a termine e viene inviato al campo il posizionamento del giocatore a seguito della manovra effettuata. Dopo la notifica, il processo si sospende per un tempo t noto al fine di lasciare agli altri processi giocatore la possibilità di operare a loro volta.

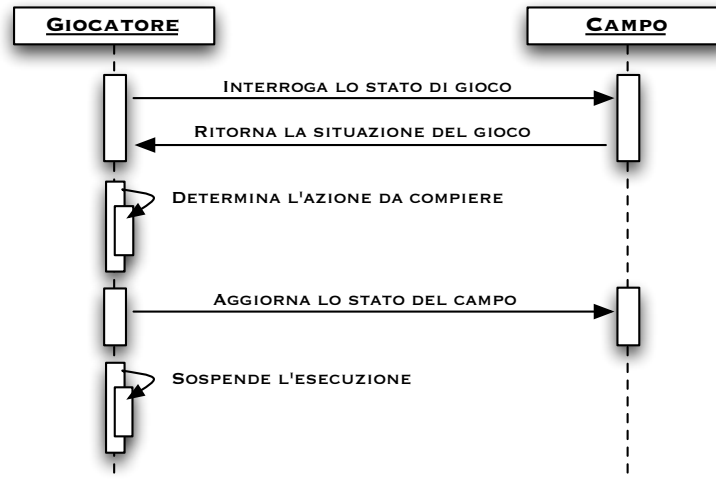


Figura 27: Diagramma di sequenza del giocatore.

Il diagramma degli stati presenta in forma semplificata i vari stadi in cui il giocatore si può trovare in ogni istante di tempo.

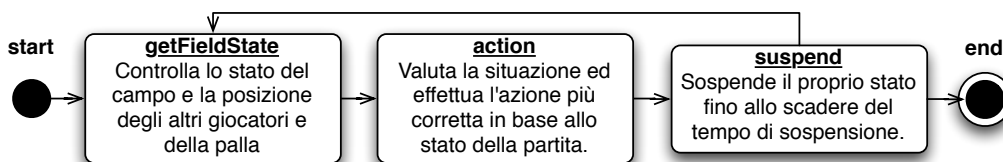


Figura 28: Diagramma di stato dei giocatori in campo.

La parte più consistente del processo è tutta contenuta nello stato *action*, il quale si occupa di valutare le posizioni ed applicare la logica opportuna. Lo schema che segue indica dettagliatamente, passo per passo, tutte le funzioni che vengono applicate per creare il quadrato visibile al giocatore al fine di consentirgli il passaggio ad altri compagni o movimenti in una delle quattro direzioni possibili. All'interno dello schema esistono due diramazioni, legate alla possibilità che il giocatore detenga o meno il possesso di palla oppure che la squadra sia in possesso di palla attraverso un compagno. Nel caso entrambi i test siano negativi, prendiamo come regola che gli avversari siano in possesso del pallone, senza applicare ulteriori test.



Figura 29: Diagramma dell'azione di gioco del giocatore.

Andiamo ad analizzare anche la situazione della memoria e della CPU durante il gioco nel diagramma di Garrett esposto. Ogni giocatore comincia la partita in una coda di processi pronti. Nel primo istante utile, un processo prende possesso della CPU e compie le azioni viste precedentemente fino allo scadere del tempo di utilizzo del processore o fino alla sospensione. Una volta sospeso, entra in un'altra coda, quella dei processi sospesi, e attende il termine del tempo di sospensione t . Una volta pronto, passa nuovamente nella coda dei processi pronti per procedere con una nuova azione non appena avrà a disposizione la CPU della macchina su cui risiede.

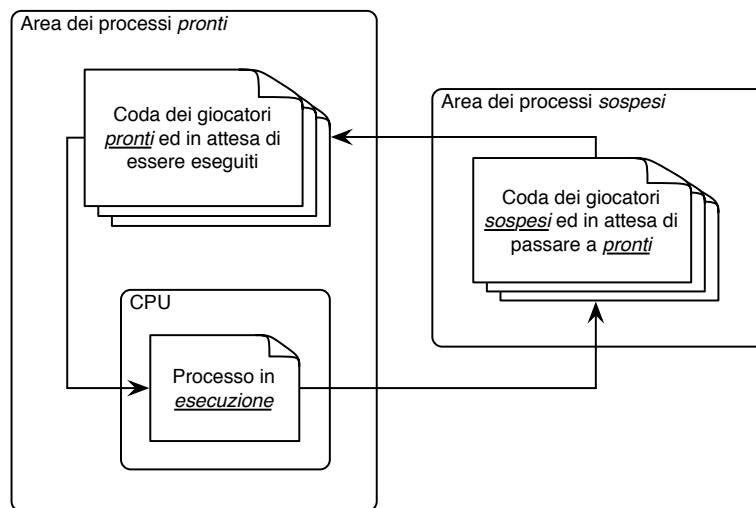


Figura 30: Diagramma di stato dei giocatori in campo.

4.6 Pallone

Il pallone rappresenta la risorsa condivisa nel campo di gioco. Ogni giocatore concorre per ottenerne il possesso e poter condurre la palla nella rete avversaria e segnare un gol in favore della propria squadra.

Il pallone è rappresentato nel campo come una risorsa condivisa il cui accesso da parte dei giocatori è limitato e regolato da un monitor. Ciò preclude la possibilità che la palla possa essere in possesso di più di un giocatore contemporaneamente. Ada 95 mette a disposizione il costrutto *protected type* che si occupa di costruire e gestire i monitor per l'accesso in mutua esclusione alla risorsa.

Dal punto di vista logico possiamo immaginare il pallone come un oggetto sul campo di gioco il quale viene spostato dai giocatori attraverso avanzamenti con palla, passaggi, tiri in porta o contrasti. Dal punto di vista tecnico, si tratta di un'entità passiva che “subisce” le azioni che vengono compiute su di esso da parte dei giocatori.

In questo caso la classe si riduce ad un oggetto protetto con pochi metodi accessibili per ottenere la sua posizione, l'eventuale possessore, effettuare un movimento della sfera in una delle 4 direzioni o eventualmente settarne il possessore.

La risorsa protetta contiene una variabile fondamentale che permette di identificare il thread che ne detiene il possesso. In questa maniera, anche nel caso in cui il giocatore possessore della palla entri in stato di *sleep*, una guardia garantirà che nessun altro dei 21 giocatori rimanenti possa accedere alla risorsa.

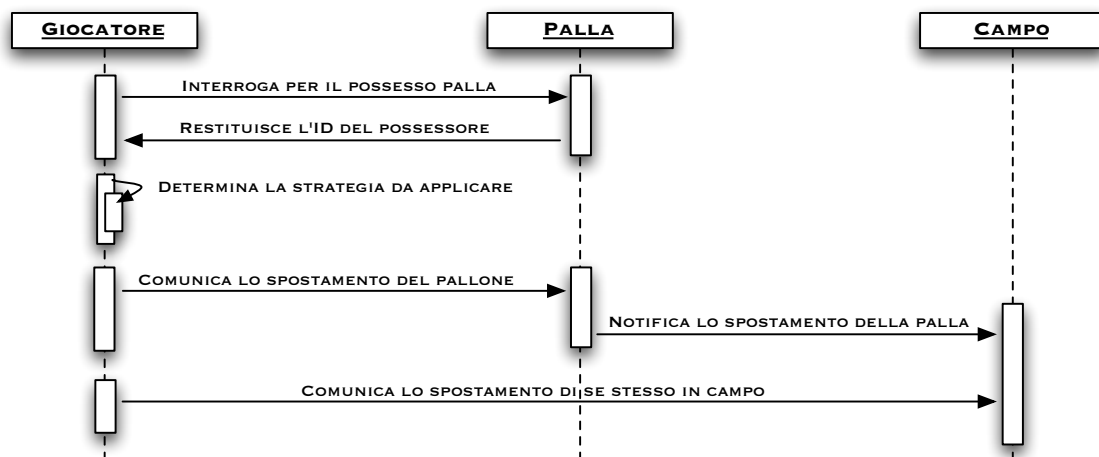


Figura 31: Diagramma di sequenza dello spostamento della palla in campo.

Per maggiori dettagli sui metodi legati alla risorsa protetta rimandiamo alla sezione 5.2 in cui vengono analizzati gli aspetti relativi alla concorrenza per un oggetto condiviso.

5 Concorrenza

Prima di entrare nel dettaglio dei problemi legati alla concorrenza, cerchiamo di identificare chiaramente quali sono le entità che fanno parte del nostro gioco e come interagiscano al fine di un corretto funzionamento:

Entità Attive: rappresentate dai **giocatori in campo**. Sono i thread attivi che compiono movimenti sul campo secondo scelte determinate da loro stessi in base alla situazione che rilevano nel momento in cui vengono attivati.

Entità Reattiva: identificata dal pallone e dal campo di gioco. Rappresentano entità che non hanno potere decisionale, ma subiscono variazioni sulla base dell'intervento dei giocatori che ne hanno via via il possesso. Il pallone può muovere assieme al giocatore in caso di avanzamento con palla o in maniera autonoma nei casi di passaggio o tiro in porta. Il movimento della palla è comunque sempre subordinato alla potenza impressagli dal giocatore che effettua l'azione. Il campo subisce tutte queste variazioni e si occupa di mantenere uno stato di gioco consistente.

Le situazioni di concorrenza che dobbiamo necessariamente analizzare e risolvere al fine di perfezionare il gioco ed eliminare situazioni impreviste sono fondamentalmente:

- integrità dello stato di gioco;
- unicità del possesso di palla;
- prevenzione di situazioni di stallo;

Esaminiamo perciò ognuna di esse e forniamo gli elementi necessari per la loro risoluzione.

5.1 Integrità dello Stato di Gioco

Avere 22 processi che modificano costantemente lo stato del campo di gioco in un ciclo di durata pari al tempo totale di gioco può causare facilmente seri problemi di inconsistenza a livello di campo di gioco. In questo senso è necessario prevedere tale situazione ed utilizzare tecniche di programmazione che permettano di evitarla per poter procedere sempre correttamente con l'aggiornamento dello stato e dare la possibilità ai thread che lo necessitano, di conoscere la condizione di gioco.

Le strade percorribili sono **fondamentalmente due**:

1. istanziare la classe *Field* come processo;
2. creare la classe *Field* come tipo protetto;

La prima soluzione, più simile ad una struttura di tipo client/server, garantisce attraverso l'utilizzo di una struttura di tipo *select* a cui fanno seguito una serie di *accept* legate alla scrittura e alla lettura, accesso esclusivo al thread che gestisce i posizionamenti nel campo. La soluzione è percorribile se immaginiamo di istanziare su macchine differenti i giocatori e il thread del campo di gioco. Nel caso l'istanza fosse residente sulla stessa macchina dei giocatori, la soluzione diventerebbe costosa elevando il livello di utilizzo di CPU a causa del numero di processi attivi che necessiterebbero di essere accodati nella lista dei thread pronti. **Si renderebbe pertanto necessario l'utilizzo di politiche di priorità per garantire assenza di attesa una volta richiamati i metodi di lettura e scrittura.** Nonostante Ada 95 permetta di risolvere il problema in questo senso, preferiamo scegliere una soluzione differente.

Diventa perciò più semplice gestire la classe *Field* come un tipo protetto che garantisce la mutua esclusione delle operazioni che intervengono sulla classe, mantenendo pertanto la correttezza dello stato di gioco attraverso la gestione da parte del linguaggio degli accessi alle variabili protette. La soluzione non impatta direttamente sul processore in quanto la CPU non viene occupata direttamente dalla classe *Field* ma dai thread dei giocatori che vi devono accedere.

La classe *Field* è pertanto dotata dei metodi:

function GetPositions return PositionList che permette di interrogare il campo ed ottenere il vettore contenente le posizioni di tutti i giocatori nell'area di gioco, compresa quella del giocatore interrogante. La funzione, come specifica del linguaggio Ada 95, ha come prerogativa l'accesso ai dati in sola lettura, evitando la modifica del campo di gioco. Questo permette l'accesso concorrente in *read only* al vettore delle posizioni;

procedure SetPosition(ID : in Integer; M : in Move) da la possibilità al giocatore che lancia la procedura di modificare la propria posizione nel campo di gioco, indicando il proprio ID univoco e la tipologia di movimento come da specifiche già trattate nella sezione 2.2.1. In questo modo viene aggiornata la posizione nel vettore senza incorrere in problemi di inconsistenza;

function GetBallPosition return Position permette al giocatore di conoscere la posizione della palla all'interno del campo di gioco. È necessaria al fine di permettere al processo di decidere quale azione compiere, limitatamente alla propria visibilità di gioco;

Queste tre funzioni da sole permettono di ottenere tutte le informazioni necessarie al corretto svolgimento del gioco non solo per i thread giocatori, ma anche eventualmente a manager ed arbitri per effettuare decisioni o correzioni durante lo svolgimento della partita. La garanzia di mutua esclusione è determinata dalla stessa classe *Field* in quanto dichiarata come risorsa protetta ad accesso limitato.

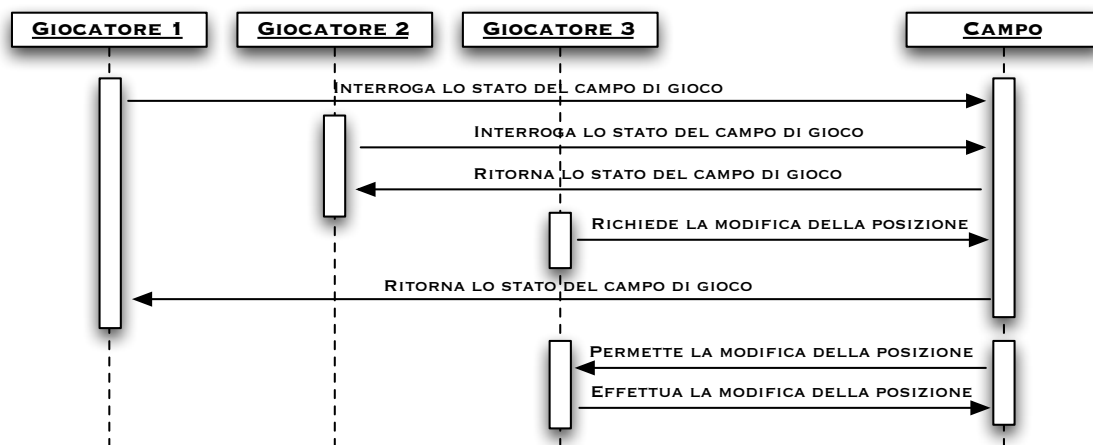


Figura 32: Diagramma di sequenza delle richieste dei giocatori per la notifica e la modifica delle posizioni.

Nello schema sopra esposto è possibile notare come la lettura delle posizioni in campo sia un processo di **lettura concorrente**, mentre l'aggiornamento della posizione del giocatore sia un'operazione esclusiva. In questo senso, una volta terminate le eventuali operazioni di lettura, il processo prende il controllo della risorsa, effettua il cambiamento e la rilascia.

5.2 Unicità del Possesso di Palla

Un altro problema evidente legato alla concorrenza tra thread è quello del possesso palla da parte dei giocatori in campo. Il pallone risulta l'unica istanza esistente e non è realistico, nella nostra realtà virtuale così come nella realtà di gioco vero e proprio, la condizione in cui due giocatori distinti detengano contemporaneamente il possesso del pallone.

Si rende pertanto necessario l'utilizzo di un meccanismo che garantisca il mutuo accesso alla risorsa. Anche in questo senso Ada 95 ci viene incontro fornendo uno strumento per la gestione delle risorse protette. Dichiarando la palla come risorsa protetta, garantiamo l'accesso esclusivo ai dati in essa contenuti.

I metodi utilizzabili con la classe palla sono:

function GetOwner return Integer permette di ottenere l'ID dell'attuale possessore del pallone o il valore *null* nel momento in cui non esistesse in quell'istante un possessore della risorsa;

function GetBallPosition return Position ritorna la coordinata relativa alla posizione del pallone con lo scopo di prenderne possesso nel caso si trovi in intersezioni adiacenti a quella del giocatore che sta effettuando la richiesta;

procedure SetOwner(ID : in Integer) è forse il metodo principale della classe. Esso permette di settare il valore della variabile relativa al proprietario della risorsa. Essendo una procedura, garantisce la mutua esclusione nell'attività attraverso il meccanismo implicito del monitor, impedendo che un giocatore, una volta flaggato il *lock* sulla risorsa, non possa settare il proprio valore ID nella variabile a causa di una sospensione. Una situazione di questo genere porterebbe ad un'evidente inconsistenza di sistema, causando l'impossibilità di accedere alla risorsa da parte di tutti gli altri giocatori per tutta la durata del match;

procedure KickBall(M : in Move; P : in Integer) rappresenta la liberazione della risorsa attraverso un ideale "calcio". In questa procedura, anch'essa atomica ed indivisibile, il giocatore rimuove il blocco sulla risorsa settando la variabile a *null* e applica una potenza al colpo in una direzione determinata;

Come abbiamo avuto modo di esaminare per la classe *Field*, i metodi appena visti garantiscono accesso esclusivo alla risorsa per eventuali modifiche al suo stato interno, mentre garantiscono accesso concorrente alle informazioni, permettendo a tutti i giocatori di identificarne proprietario e posizione, ma assicurando l'integrità della risorsa.

5.3 Priorità dei Processi

Al fine di gestire il corretto andamento del gioco, è necessario gestire le priorità dei thread. Se così non fosse, ognuno di essi entrerebbe nella coda dei pronti e **dovrebbe attendere diversi cicli di CPU** prima di poter eseguire. Per questo motivo è necessario raggruppare le tipologie di processi e ordinarle seguendo una logica prioritaria d'esecuzione.

I processi più comuni sono quelli dei giocatori in campo. **Tutti i giocatori eseguono con la stessa priorità**, demandando eventuali gestioni ulteriori allo scheduler di sistema che li tratterà come processi ordinari.

Con priorità maggiore intervengono i manager delle squadre che, una volta pronti, devono entrare in esecuzione prima dei giocatori al fine di modificare le condizioni di gioco preventivamente rispetto al successivo intervento dei giocatori.

Infine l'arbitro, in **qualità** di giudice di gioco deve poter intervenire immediatamente interrompendo il gioco, effettuando eventuali riposizionamenti di giocatori e di palla, prima di ogni giocatore e ancor prima delle decisioni dei manager.

Classe	Priorità	Descrizione
Giocatore	1	Priorità minima di accesso alla CPU.
Manager	2	Priorità superiore a quella dei giocatori.
Arbitro	3	Priorità massima.

Tabella 7: La lista delle priorità dei processi.

In questo modo viene garantito il corretto accesso al processore ed eventuali modifiche che devono essere applicate al gioco vengono recepite immediatamente da tutti i processi con priorità inferiore per effetto dell'ordine di esecuzione.

5.4 Prevenzione delle Situazioni di Stallo

Avere un numero elevato di processi che concorrono ad un'unica risorsa può portare facilmente a situazioni di stallo. Elenchiamo per completezza le tre situazioni in cui il sistema non è in grado di procedere oltre determinando una partita infinita o immodificabile fino alla fine del tempo di gioco:

- deadlock;
- starvation;
- lockout;

Per non incappare nelle situazioni di cui sopra è necessario risolvere una o più tra le precondizioni elencate:

Mutua esclusione: non più di un processo alla volta ha accesso alla risorsa condivisa;

Cumulazione di risorse: i processi possono accumulare risorse e trattenerle mentre attendono di acquisirne altre;

Assenza di prerilascio: le risorse vengono rilasciate solo volontariamente;

Attesa circolare: un processo attende una risorsa in possesso del successivo processo in coda;

Il primo punto è già stato esaurientemente coperto e risolto nelle sezioni 5.2 e 5.1. Non ci dilunghiamo oltre nell'analisi e nella risoluzione ma rimandiamo ad essi per ulteriori approfondimenti.

Il secondo punto, in parte, è risolto per definizione: il prerequisito di unicità della palla nel campo di gioco è sufficiente per poter rendere inattuabile la precondizione ed evitare situazioni di stallo legate ad esso. Il giocatore che detiene il possesso della sfera non limita le possibilità di gioco degli altri giocatori che possono procedere nella loro strategia fino ad ottenerne il possesso per loro o per la propria squadra. Per quanto riguarda l'altra risorsa protetta, ovvero il campo di gioco, le azioni di spostamento sono atomiche e non comportano il blocco della risorsa né il possesso di risorse multiple che possono determinare il blocco della procedura. Una volta settata la nuova posizione, la risorsa viene rilasciata immediatamente per eventuali consultazioni o ulteriori modifiche.

La terza precondizione coinvolge per lo più il possesso di palla e non è risolvibile “realisticamente” se non utilizzando un sistema di rilascio forzato della risorsa attraverso, per esempio, un meccanismo di conteggio dei movimenti o dei tempi di possesso. Possiamo assumere che dopo n movimenti consecutivi con palla, il giocatore debba necessariamente effettuare il passaggio ad un compagno o tentare un tiro in porta. Questi due eventi però sono legati alla visibilità del giocatore in questione e possono non poter essere soddisfatti nel momento in cui viene raggiunto il valore limite. Non è pertanto una soluzione ottimale. È possibile eventualmente conteggiare gli istanti temporali in cui il giocatore trattiene il pallone. A seguito di t istanti, il processo deve liberare la risorsa, ma come nel caso precedente, si possono verificare condizioni in cui, per effetto della visibilità di gioco non ci siano compagni vicini e la porta sia troppo distante per poter tirare.

Altro tipo di approccio dalle linee decisamente più semplicistiche, ma molto più realistico, è quello di assumere che nel corso della partita il giocatore con palla si trovi in condizione di tirare, passare o subire da 1 a x contrasti e questo comporti inevitabilmente, nel corso del match, la perdita del pallone. Analizziamo il nostro modello per comprendere cosa può succedere nel caso in cui un giocatore non liberi la palla:

1. **Avanzamento:** il giocatore continua ad avanzare nel campo di gioco con la palla. Nel caso non subisca contrasti da parte di avversari o vinca tutti quelli sostenuti, arriverà in prossimità della porta. Per effetto delle logiche di gioco, effettuerà il tiro per tentare di segnare una rete rilasciando la risorsa. Nel caso non tenti il tiro e continui a trattenere il pallone, possiamo assumere che a seguito di n contrasti durante la fase di avanzamento, in almeno uno risulti sconfitto e perda il possesso del pallone, rilasciandolo.
2. **Assenza di compagni per il passaggio:** assumiamo che concorrentemente al giocatore con palla anche tutti i compagni in gioco continuino a muoversi per smarcarsi da avversari e ricevere il pallone. È lecito ritenere che almeno un giocatori visibile dal portatore di palla sia smarcato per dare modo al nostro thread di passare il pallone.
3. **Assenza di porta avversaria:** come già visto nel punto 1 del presente elenco, l'avanzamento porta inevitabilmente il giocatore a trovarsi di fronte alla porta avversaria e questo gli permetterà di effettuare il tiro.
4. **Mantenimento della posizione:** nel peggiore dei casi, il giocatore non effettuerà alcuno spostamento. In questo caso possiamo assumere che sia il gioco svolto intorno ad esso ad essere dinamico e questo porta inevitabilmente all'avvicinamento da parte di compagni a cui passare la risorsa o di avversari che tenteranno di effettuare un contrasto per sottrarre il pallone. Anche in questo caso è perfettamente lecito pensare che un giocatore, anche se fermo, non si trovi mai nella condizione di passare o di perdere la palla.

Ai fini della nostra realtà, quanto asserito nei punti precedenti è sufficiente per poter considerare risolta l'assenza di prerilascio per effetto della costruzione del nostro sistema di gioco. In questo caso possiamo affermare la solidità del modello di gioco rispetto alla mancanza di prerilascio.

La quarta preconditione non si verifica per ipotesi. Per quanto riguarda l'accesso alla classe *Field*, la modifica dello stato di gioco avviene istantaneamente a seguito dell'applicazione di una delle logiche di gioco. Questo non comporta attesa circolare in quanto un processo che ottiene accesso alla risorsa *Field* la libera prima di sospendere la propria esecuzione. Questo comporta che il processo successivo in coda si troverà in condizione di accedere anch'esso alla risorsa, liberandola prima di sospendersi. Per quanto riguarda invece la risorsa *Ball*, nessun giocatore rimane immobile in caso di impossibilità di accedere alla risorsa, ma compie azioni legate all'avanzamento verso la porta avversaria o al mantenimento della propria posizione in base alle condizioni di gioco con cui si trova ad interagire.

Alla luce di quanto visto finora possiamo permetterci di escludere il verificarsi di situazioni di stallo che comportino l'impossibilità di portare a termine la partita.

6 Distribuzione

Il concetto di distribuzione prevede che un sistema residente su di un unico calcolatore venga scomposto in parti al fine di poter essere suddiviso in una moltitudine di calcolatori differenti. La distribuzione di un sistema permette di ripartire il carico di lavoro imputato ad un solo calcolatore su una serie di macchine, ognuna di esse con una funzione determinata e dei compiti precisi.

Ada 95 permette di generare in modo estremamente semplice un sistema completamente distribuito partendo dallo sviluppo di un sistema monolitico e definendo in seconda battuta le componenti distribuite attraverso l'uso di alcune direttive per determinare la tipologia di comunicazione e di oggetti nonché attraverso un file di configurazione che aiuta il compilatore nella generazione dei vari eseguibili. Ognuno di essi conterrà al suo interno gli "stub" generati dal compilatore. Attraverso questi riceverà ed invierà messaggi alle varie componenti con cui è direttamente collegato.

Il linguaggio permette di distribuire il nostro programma secondo differenti tipologie paradigmi di distribuzione:

- **Sistema Monolitico:** tutto il programma risiede su di un'unica macchina sulla quale vengono eseguiti tutti i processi. Questo comporta chiaramente un elevato carico per la CPU a seguito dell'elevato numero di processi che ne necessitano l'utilizzo;
- **Sistema Peer-to-Peer:** il sistema è completamente decentralizzato. Ogni entità che viene creata può potenzialmente risiedere su una macchina differente. Tradotto in termini di gioco, ogni giocatore può risiedere su una macchina diversa, i manager delle squadre su altre macchine ancora e così via fino all'esaurimento dei processi, attivi e passivi, coinvolti;
- **Sistema Client/Server:** è la tipologia tradizionale con cui vengono gestite le applicazioni che necessitano di una centralizzazione, ovvero una macchina funge da servente e gestisce il gioco. Tutte le altre macchine sono i clienti che si connettono direttamente al server al fine di conoscere lo stato di gioco, ma procedono in maniera autonoma nelle valutazioni e nella realizzazione delle azioni. Ad ogni termine di azione, viene aggiornato lo stato del server;

Ai fini della nostra realtà virtuale, la soluzione più pratica è la terza, legata all'architettura client/server in quanto possiamo immaginare di distribuire le due squadre in campo in due macchine differenti pilotate da due intelligenze artificiali o da due giocatori reali, mantenendo però la gestione delle risorse condivise in un server accessibile ad entrambe. Gli "stub" si occupano di gestire la comunicazione tra macchine differenti, convertendo messaggio in ingresso ed uscita e facendo credere al software che tutto il sistema risieda su un'unica macchina.

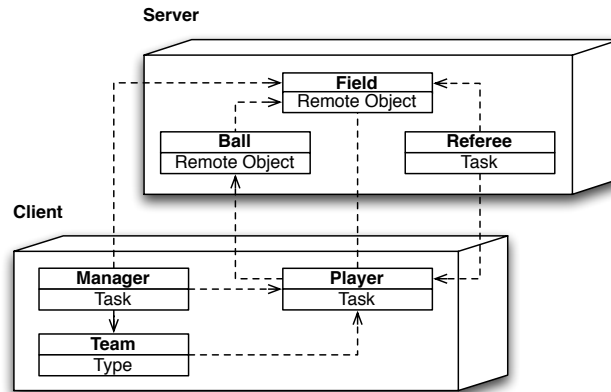


Figura 33: Diagramma del sistema distribuito.

Presentiamo qui sopra lo schema con cui è composto il nostro sistema

Le entità principali sono fondamentalmente 6 di cui 3 risiedono sulla macchina server:

- **Campo:** shared protected object;
- **Arbitro:** task;
- **Pallone:** shared protected object;

Le rimanenti sono proprie della squadra in gioco e risiedono sul client:

- **Manager:** task;
- **Squadra:** shared protected object;
- **Giocatore:** task;

La comunicazione tra entità avviene come indicato dai segmenti che uniscono gli elementi presentati nella figura precedente. Viene inoltre indicato il verso in cui la comunicazione viene inizializzata, ovvero quale elemento si occupa di effettuare la chiamata di funzione/procedura che permette di interrogare lo stato degli elementi o modificare alcuni di essi.

6.1 Estensione del Modello Distribuito

Potenzialmente risulta estremamente facile estendere il modello a partita singola in un campionato composto da una serie di squadre e una serie di singola partite. Il principio è quello di inserire un'ulteriore strato che si occupa di gestire il calendario, stabilendo gli incontri e gestendo i risultati al termine dei match al fine di gestire la classifica per decretare la formazione vincitrice. Il modello viene presentato nello schema sottostante:

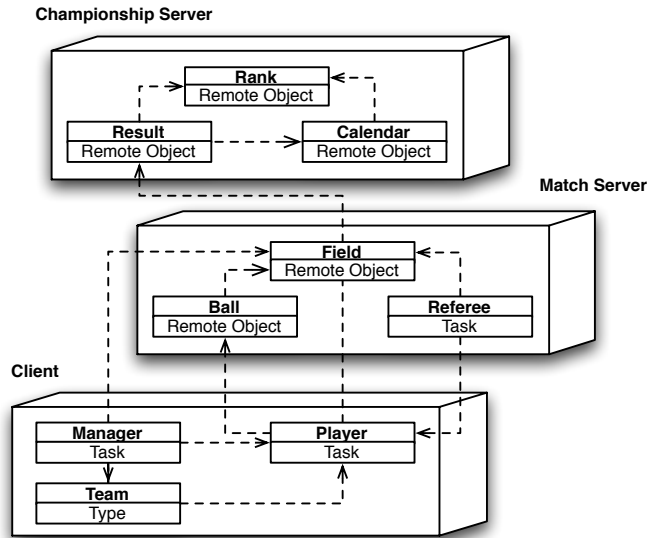


Figura 34: Diagramma del campionato distribuito.

6.2 Creazione del Software Distribuito

Attraverso l'utilizzo di Ada 95 DSA³, la costruzione di un sistema distribuito è notevolmente semplificata per effetto dell'integrazione nello stesso linguaggio di programmazione di strutture di definizione di interfacce che permettono di creare un sistema completamente distribuito senza dover ricorrere a linguaggi esterni (come ad esempio CORBA).

I passaggi che permettono di realizzare in Ada 95 un sistema distribuito sono:

1. Progettare un'applicazione come se non fosse distribuita;
2. Identificare procedure remote (*RPC*), variabili condivise (*SP*) e oggetti distribuiti (*RT*) e categorizzare i pacchetti;
3. Realizzare e testare il sistema non distribuito;
4. Redigere un file di configurazione (*Configuration File*) per partizionare l'applicazione;
5. Costruire le partizioni e testare l'applicazione distribuita.

Avendo già sviluppato tutta l'applicazione come non distribuita, realizzarne la distribuzione si riduce all'identificazione delle entità e alla ripartizione attraverso una configurazione. Nella box sottostante possiamo vedere il codice relativo al file di specificazione **Configuration.ads**:

```

1 with Field_Package; with Ball_Package; with Referee_Package; with Player_Package;
  with Team_Package; with Manager_Package;

3 configuration Config is

5   -- definizione di Manager e uso dei package

```

³Distributed System Annex

```

7      fragment Manager1 is
          use Manager_Package;
      end Manager1;

9

11     fragment Manager2 is
          use Manager_Package;
      end Manager2;

13

15     — definizione di Player e uso dei package
      fragment Player1 is
          use Player_Package;
17     end Player1;

19     fragment Player2 is
          use Player_Package;
21     end Player2;

23     — definizione di Team e uso dei package
      fragment Team1 is
          use Team_Package;
25     end Team1;

27     fragment Team2 is
          use Team_Package;
29     end Team2;

31

33     — definizione del campo
      fragment Field0 is
          use Field_Package;
35     end Field0;

37     — definizione del pallone
      fragment Ball0 is
          use Ball_Package;
39     end Ball0;

41

43     — definizione dell'arbitro
      fragment Referee0 is
          use Referee_Package;
45     end Referee0;

47 end Config;

```

Nella box successiva, invece, il codice relativo al file di implementazione `Configuration.adb`:

```

1 configuration body Config is

3     — definizione del server
      map Field0, Ball0, Referee0 onto Server;

5

7     — definizione del primo client
      map Manager1, Player1, Team1 onto Client1;

9

11    — definizione del second client
      map Manager2, Player2, Team2 onto Client2;

```

```
end Config;
```

In questo modo, come già visto, vengono creati gli stub di comunicazione tra client e server, il codice viene ripartito correttamente tra le entità e vengono generati gli eseguibili da utilizzare per il nostro gioco.

6.3 Sincronizzazione Temporale

Per poter gestire la partita su di un sistema completamente distribuito è fondamentale gestire la componente temporale. Ogni istanza, creata su macchine differenti, deve essere in grado di gestire in maniera autonoma il trascorrere del tempo.

Anche in questo caso il linguaggio di programmazione ci viene incontro in quanto Ada 95 ci permette di accedere all'orologio di sistema e sfruttare sospensioni assolute per determinati intervalli temporali attraverso l'uso del package `Ada.Real_Time`. Questa classe è in grado di fornire il controllo del tempo sulla base degli intervalli tra istanti temporali piuttosto che sul tempo orario. Come possiamo vedere nella figura sottostante, usare il tempo in maniera assoluta permette di gestire le sospensioni dei giocatori e garantire una durata unica di gioco per tutte le istanze clienti che partecipano alla partita.

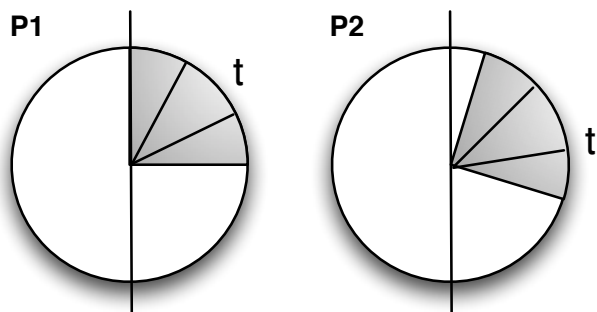


Figura 35: Schema di avanzamento temporale nel gioco su macchine differenti.

Consideriamo una partita tipo: tutti i task vengono inizializzati e cominciano a lavorare contemporaneamente. A mano a mano che i singoli processi eseguono, vengono sospesi per n istanti temporali, dove n è un valore univoco e definito per tutti i calcolatori su cui viene distribuito il nostro sistema. Ogni processo, dopo aver atteso il proprio turno nella coda dei processi pronti, prende il controllo della CPU, esegue il proprio lavoro e si autosospende usando la direttiva `delay until` per $x \cdot n$ istanti temporali, dove x dipende dalla velocità di gioco del giocatore ed n è appena stato definito.

In Ada 95 il comando `delay until` limita inferiormente la sospensione del processo garantendo attesa passiva fino allo scadere del tempo. Una volta risvegliato, il processo ritorna nella coda dei processi pronti, in attesa di eseguire nuovamente. Tali operazioni avvengono in maniera ciclica, fino allo scadere dei 90 minuti di gioco i quali riproporzionati su un equo valore di istanti temporali calcolati sulla base del rapporto tra tempo effettivo e tempo accelerato di gioco, nonché in rapporto alla media delle sospensioni dei thread giocatori.

6.4 Stati di Gioco

Con stato di gioco si intende una procedura attraverso cui il server deve o meno ricordare lo stato del client al fine di poterlo riconoscere e procedere con eventuali azioni nei suoi confronti. Viene definito “stateless” un

server che non ha necessità di riconoscere la sessione con il client. Nel caso opposto il server viene definito come “stateful”.

Nel nostro caso possiamo definire il server come “stateless”. Non c’è alcun motivo per cui debba memorizzare il client con il quale è connesso e neppure l’eventuale sessione.

Tutti i client, indistintamente, si connettono con il server e in modo ciclico richiedono lo stato del campo e della palla. Una volta restituiti i dati i processi, in forma del tutto autonoma, processano le informazioni al fine di procedere con l’azione più corretta legata alle informazioni che hanno a disposizione e alle caratteristiche di ogni singolo giocatore.

Una volta determinata la migliore azione, si connettono con il server, aprono una nuova sessione con la classe *Field* per compiere il proprio movimento, eventualmente aprono una sessione con *Ball* per effettuare modifiche allo stato del pallone e chiudono il collegamento. La trasmissione del proprio ID alla classe *Field* permette di chiudere la sessione ed aprirne una nuova, minimizzando i tempi di collegamento. Nel caso in cui non venisse trasmesso l’ID del processo che deve effettuare la modifica, sarebbe necessari mantenere il collegamento dalla richiesta dello stato del campo fino alla trasmissione dell’aggiornamento.

Questo però porterebbe a tempi di trattenimento della risorsa potenzialmente elevati e quindi ad un blocco delle risorse in gioco e ad un conseguente ritardo nel ritmo della partita.

6.5 Sincronizzazione Stati

– SNAPSHOT: stato di *Field*

7 Codifica

7.1 Field

```
— rappresenta la classe che contiene i dettagli del campo di gioco
2
package Field_Package is
4
  — dichiarazione di oggetto remoto
  pragma Remote_Types;
6
  type Move is (Up, Down, Left, Right);
8
  type Position is
  record
12      ID : Integer;
      Row : Integer;
14      Col : Integer;
  end record;
16
  — da ridefinire
  type PositionList is array (1 .. 23) of Position;
18
  protected type Field is
20
    — dimensioni del campo standard (variabili in base all'inizializzatore)
    NumRows : Integer := 315;
24    NumCols : Integer := 204;
26
    — funzione per leggere lo stato del campo
    function GetPositions return PositionList;
28
    — procedura per la modifica dello stato del campo su un oggetto protetto
    procedure SetPosition(ID : in Integer; M : in Move);
30
    — funzione per ottenere la posizione della palla
    function GetBallPosition return Position;
32
    — funzione per ottenere la posizione della porta avversaria
    function GetGoalPosition return Position;
34
    — funzione esclusiva dell'arbitro
    procedure ResetPosition(ID : in Integer; P : Position);
36
    — funzione esclusiva dell'arbitro
    procedure ResetPosition(ID : in Integer; P : Position);
38
  end Field;
40
end Field_Package;
```

7.2 Referee

```
1 with Field_Package; use Field_Package;  
  with Player_Package; use Player_Package;  
3  
  package Referee_Package is  
5  
    end Referee_Package;
```

7.3 Manager

```
— rappresenta la classe che contiene i dettagli del manager di gioco
2 with Field; use Field;
  with Player; use Player;
4
6 task type Manager is
8     function GetFieldState() return PlayerList;
10     procedure ChangeModule(M : in Module);
12     procedure ChangePlayers(A : in Player; B : in Player);
14     procedure ChangeStrategy(A : in Strategy)
16         type Strategy is (Attack, Middle, Defense);
18 end Manager;
```

7.4 Team

```
— rappresenta la classe che contiene i giocatori in campo
2 with Player_Package; use Player_Package;

4 package Team_Package is

6   — dichiarazione di oggetto remoto
   pragma Remote_Types;

8   — dichiarazione della variabile strategia:
10  type Strategy is (Defense, Middle, Attack);

12  — enumerazione dei moduli di gioco
   type Module is ('352', '442', '343', '532');

14

16  — da ridefinire
   type PlayerList is array (1 .. 11) of Player;

18  protected type Team is

20     S : Strategy := Middle;
     M : Module := '442';

22

24     — vettore dei giocatori
     List : PlayerList := puntatore****

26     — funzione per cambiare la strategia
     procedure SetStrategy(S : in Strategy);

28

30     — funzione per ottenere la strategia di gioco
     function GetStrategy return Strategy;

32

34     — funzione per impostare il modulo di gioco
     procedure SetModule(M : in Module);

36     — funzione per verificare il modulo di gioco
     function GetModule return Module;

38  end Team;

40 end Team_Package;
```


7.5 Player

```

2  -- processo giocatore
  with Field_Package; use Field_Package;
  with Ball_Package; use Ball_Package;
4
  package Player_Package is
6
      task type Player is
8
          -- ID : Integer
          -- Direction : Integer := 0; -- da inizializzare con 1: sinistra a destra; 2:
10         -- destra a sinistra;
          -- R : Role ;
12
14         -- entry Action(T : in Integer);
16         -- function Throw()
18         -- function Pass()
20         -- function Attack()
22         -- function Defense()
24         -- function Contrast()
26         -- function Rest()
28         -- function Unmark()
30         -- function Save()
32     end Player;
34 end Player_Package;
```

7.6 Ball

```
2 package body Ball_Package is
4     protected type Ball is
        pragma SHARED(Ball);
6         entry Ownership;
        private
8             Possess : Boolean := False;
            Player : String := null;
10    end Ball;

12    protected body Ball is
        entry Ownership when Ownership Count = 0 or Possess is
14        begin
            if Ownership Count = 0 then
16                Possess := False;
                — indicazione del giocatore che possiede la palla
18            else
                Possess := True;
20            end if;
            end Ownership;
22    end Ball;

24 end Ball_Package;
```

8 Conclusioni

9 Bibliografia

Riferimenti bibliografici

- [1] A. Burns, A. Wellings, *Concurrent and Real-time Programming in Ada* Cambridge University Press, 3rd Edition, 2007.
- [2] A. S. Tanenbaum, M. van Steen, *Distributed Systems: Principles and Paradigms* Prentice Hall, 2nd Edition, 2006.