Ingegneria del software - Memory e Dama Multiplayer

Fabrizio Corriera

2019/2020

Indice

1 Introduzione

1.1 Obiettivo

L'obiettivo di questo documento consiste nel fornire una panoramica completa sul processo di pianificazione, analisi dei requisiti funzionali, studio delle risorse e sviluppo di un software utilizzando una strategia di tipo agile. Il software in questione è un tool che presenta all'interno sia un videogioco 3D che ripropone il classico gioco da tavolo della dama con la possibilità di giocare con altri giocatori in locale o online, sia un videogioco 3D che ripropone il classico gioco del memory in locale tra due giocatori.

1.2 Solo Extreme Programming

Ho deciso di utilizzare una metodologia agile simile all'Extreme Programming (XP) ma adattata al lavoro in solitario. L'XP è una metodologia di sviluppo software di tipo agile che pone enfasi sul lavoro di squadra e segue 12 regole fondamentali, divise in 4 categorie:

- **Feedback a scala fine**
  - Pair programming: lavorando in due sulla stessa macchina al medesimo codice, il prodotto sarà di qualità superiore.
  - Planning game: una riunione di pianificazione che avviene ad ogni iterazione (circa una volta a settimana).
  - Test driven development: test automatici scritti prima di scrivere il codice.
  - Whole team: il cliente deve essere presente e disponibile a verificare la qualità e la funzionalità del prodotto.
- **Processo continuo**
  - Integrazione continua: integrare continuamente i cambiamenti al codice eviterà ritardi più avanti nel ciclo del progetto.
  - Refactoring: riscrivere il codice senza alterarne le funzionalità esterne, in modo da renderlo più semplice e generico.
  - Small releases: frequenti rilasci di funzionalità.

- **Comprensione condivisa**
  - Coding standards: scrivere il codice seguendo uno standard di regole ben definite da tutto il team.
  - Collective code ownership: ognuno è egualmente responsabile della totalità del codice.
  - Simple design: sia nei confronti del codice che del cliente i programmatori dovrebbero mantenere un approccio orientato alla semplicità.
  - System metaphor: il sistema ed il suo funzionamento dovrebbero essere descritti con una metafora.
- **Benessere dei programmatori**
  - Sustainable pace: la cosiddetta 40-hour week; nessuno nel team dovrebbe lavorare più di 40 ore a settimana.

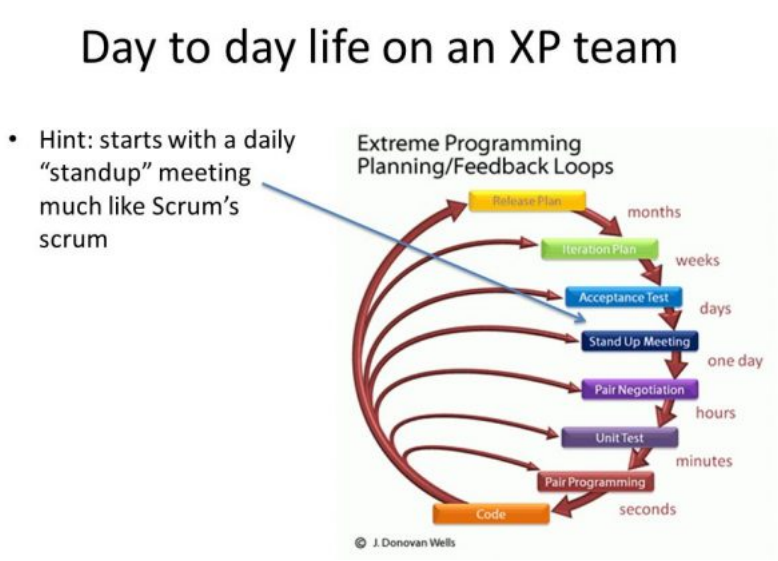

Figura 1: Rappresentazione del loop che definisce il lavoro di un team che usa XP

Ovviamente nel caso di un progetto sviluppato da un team composto da un'unica persona (come quello descritto in quella relazione), alcune regole

perdono di significato: sicuramente il pair programming non è possibile, seppure sia possibile ottenere comunque un altro punto di vista sul proprio codice attraverso il rubber duck debugging, e l'integrazione risulta costantemente immediata, a meno che non si lavori su più branch.

Generalmente si utilizzano le "user stories", delle descrizioni ad alto livello delle funzionalità del software che permettono di dividere il lavoro in blocchi logici, focalizzandosi sul risultato da dover ottenere. Le stories vanno scritte con un linguaggio molto semplice, senza la necessità di specificare dei requisiti che verranno descritti in seguito. Inoltre è fondamentale fare periodicamente dei test sull'accettabilità del codice, concordandone i criteri con il cliente.

Anche un unico programmatore può mantenere costanti le release del software: dividendo il progetto in diverse iterazioni e pianificandole di volta in volta. Resta comunque fondamentale la focalizzazione sulla comunicazione ed il feedback con il cliente (due elementi chiave nell'XP). Una trappola molto comune in cui è possibile cascare è cercare di lavorare più duramente o al lungo per mettersi in pari con la scaletta originale. Invece è sempre meglio, secondo la filosofia dell'XP, fare un nuovo "planning meeting", tirare le somme, accettare dove si è arrivati e studiare un nuovo piano di azione.

A tal proposito è infatti estremamente importante mantenere un andamento costante e sostenibile. Anziché condurre degli stand-up privati, più semplicemente è possibile dedicare un momento ogni giorno alla gestione degli imprevisti e allo scopo di riottenere il focus sul lavoro.

Utilizzare un design semplice ed una metafora per il sistema è quantomai fondamentale anche lavorando da soli. Assegnare la responsabilità alle classi e agli oggetti usando dei pattern GRASP (General Responsibility Assignment Software Patterns) piuttosto che delle carte CRC (Class Responsibility Collaboration) può essere d'aiuto in tal senso.

Lavorare iterativamente senza mai mostrare i risultati al cliente è un errore da evitare. In tal senso avrebbe addirittura senso pensare di lavorare faccia a faccia con il cliente per migliorare la comunicazione con quest'ultimo.

Anche se si lavora da soli è necessario stabilire e mantenere degli standard nella stesura del codice. Ciò renderà il codice molto più leggibile, comprensibile e facile da approcciare per il debug.

Se possibile potrebbe essere necessario testare il codice su più di una macchina. Non è detto che il funzionamento del software sia consistente se è stato testato su un'unica macchina.

Infine, ovviamente tutto il codice deve aver passato positivamente tutti i test prima della release. Se viene trovato un bug bisogna creare dei test a

riguardo.

1.3 Descrizione delle funzionalità del software

Il software che desideriamo implementare consiste nella possibilità di scegliere tra giocare ad una rappresentazione 3D del gioco della dama, arricchita da una funzionalità multiplayer online e da una chat in-game o ad una rappresentazione 3D del gioco del memory. Ci aspettiamo che il software alla sua inizializzazione presenti un menù principale, con una grafica semplice ma d'impatto, e delle scelte da poter effettuare, ovvero:

1. Scegliere tra Memory e Dama Multiplayer
2. Iniziare una partita a Memory in locale contro un altro giocatore
3. Iniziare una partita a Dama in locale contro un altro giocatore.
4. Creare una stanza in cui ospitare una partita online di Dama.
5. Entrare in una stanza già esistente ed unirsi ad una partita online di Dama.

Dal menù della Dama si potrà anche inserire lo username con cui venire identificati durante le partite multiplayer (in caso di input vuoto si verrà identificati come host se si sta ospitando la partita o client se si è ospiti nella stanza di un altro giocatore).

Se si sceglie una partita in locale, due giocatori potranno confrontarsi giocando sulla stessa macchina e la partità inizierà subito dopo aver selezionato l'opzione dal menù.

Se si sceglie di creare una stanza per ospitare una partita si verrà messi in attesa di un giocatore che si unisca alla stanza e si potrà sempre avere la possibilità di annullare e tornare indietro al menù principale.

Se si sceglie di unirsi ad un'altra partita verrà chiesto di inserire l'indirizzo IP del giocatore che sta correntemente attendendo che qualcuno entri nella sua stanza. Di dafault sarà inserito l'indirizzo locale (127.0.0.1), ma sarà possibile modificarlo e poi si dovrà selezionare la conferma dell'indirizzo o di tornare indietro al menù principale.

Nel momento in cui inizia una partita qualunque vogliamo che il gioco della dama venga rappresentato fedelmente: su una scacchiera 8x8 composta

da un'alternanza di quadrati neri e bianchi ci sono 12 pedine nere e 12 bianche, situate ai due capi della scacchiera, posizionate solo sui quadrati neri. Le pedine possono essere mosse solo in avanti e in diagonale ed ovviamente non possono uscire dai limiti della scacchiera. Inizia il giocatore che muove le pedine bianche e ci aspettiamo che dopo che egli abbia mosso il turno passi al giocatore che muove le pedine nere e così via. Se una pedina si trova in una posizione adiacente ad una pedina avversaria, e la posizione seguente lungo la diagonale non è occupata, essa potrà e dovrà mangiare la pedina avversaria in questione: ciò vuol dire che si sposterà di due posizioni (fino ad occupare la poszione dietro la pedina "mangiata") e la pedina che è stata così scavalcata viene eliminata dal gioco. Se la pedina mossa ha mangiato e si trova in posizione di mangiare un'altra volta, il giocatore potrà e dovrà farlo prima di passare il turno; questo è l'unico caso in cui durante il proprio turno un giocatore muove più di una volta. Se una pedina raggiunge il limite avversario della scacchiera viene promossa e diventa una regina (re in inglese): ciò vuol dire che la suddetta pedina potrà muoversi in ambo le direzioni della scacchiera, mantenendo comunque la restrizione del movimento diagonale. Quando l'ultima pedina avversaria verrà mangiata, il giocatore che finisce il turno avrà vinto la partita.

Vogliamo che venga fatto presente di chi è il turno attuale, e che venga segnalato il vincitore. Inoltre intendiamo mettere un'evidenziazione dei pezzi che sono costretti a muovere in una determinata situazione. Ed infine, una volta conclusa la partita, il giocatore dovrà essere riportato al menù.

Durante la partita online sarà anche presente una chat in cui i due giocatori potranno scambiarsi messaggi in tempo reale mentre giocano. Nella chat sarà possibile scorrere i messaggi precedenti per leggerli o scriverne uno nuovo nell'apposita casella di input per poi inviarlo cliccando l'apposito tasto. I mittenti dei messaggi saranno evidenziati nella chat, in quanto i messaggi avranno un formato del tipo "Username: messaggio".

Quando inizia una partita di Memory vogliamo che vengano sistemate 20 coppie di carte coperte sul tavolo da gioco in ordine casuale, e che i giocatori, durante il loro turno, possano girare due carte. Se queste due carte sono uguali tra loro, quel giocatore ottiene 1 punto e potrà continuare a scoprire le carte. Se la coppia di carte girate non corrisponde, il giocatore dovrà rigirarle e passare il turno.

1.4 Analisi dei requisiti

Da questa breve descrizione possiamo individuare i requisiti di sistema del software, dividendoli in requisiti funzionali (descrivono ciò che il sistema dovrebbe fare) e requisiti non funzionali (non riguardano direttamente le funzioni fornite dal sistema ma ne specificano il comportamento, le politiche di organizzazione e sviluppo ecc). Partendo da questi requisiti scriveremo le User Stories che utilizzeremo durante lo sviluppo del progetto e che ci permetteranno di dividere il lavoro in maniera logica e incrementale.

- **Requisiti funzionali**
  - Menù principale dove è possibile scegliere tra Dama e Memory.
  - Menù del Memory da dove si può procedere verso una partita locale con un altro giocatore o tornare indietro.
  - Menù della Dama da dove si può procedere verso: partita locale, hosting di una partita, entrare in una stanza esistente o tornare indietro.
  - Possibilità di definizione di uno username personalizzato.
  - Sottomenù di hosting.
  - Sottomenù di accesso ad una partita.
  - Partita locale.
  - Partita online.
  - Chat online.
  - Ritorno al menù principale dopo la vittoria di un giocatore.
- **Requisiti non funzionali**
  - Il gioco deve essere 3D.
  - Il progetto deve essere sviluppato con il motore grafico Unity.
  - Il progetto deve usare il linguaggio di programmazione C#.
  - La funzionalità online deve usare il protocollo TCP senza ausilio di tool esterni.
  - La metodologia di sviluppo del software deve essere XP.
  - La consegna del software con annessa documentazione deve avvenire entro la data 02/09/2020.

1.5 Diagrammi dei casi d'uso

Presentiamo qui di seguito i diagrammi dei casi d'uso, o diagrammi use case, del nostro software. Un diagramma use case è un diagramma UML che viene usato per descrivere un set di azioni o funzionalità che un sistema può o dovrebbe svolgere con l'ausilio di uno o più utenti esterni.

Qui di seguito vediamo la traduzione in diagrammi use case di quanto è stato detto finora descrivendo le funzionalità del nostro software:

Figura 2: Diagramma Use Case del menù di gioco

Nella figura 2 è rappresentato il diagramma dei casi d'uso in riferimento al menù principale del software che ci accingiamo a sviluppare, e da qui possiamo vedere che l'utente al momento dell'avvio del programma può scegliere tra accedere al sottomenù della Dama e accedere al sottomenù del Memory.

Nel primo caso avrà davanti a se la possibilità di impostare uno username, tornare al menù precedente o di scegliere la modalità di gioco tra quelle proposte: locale oppure online con il ruolo di host o di guest (in questi ultimi

due casi, come è possibile vedere, servirà l'ausilio di un ulteriore utente per poter avviare una partita di dama).

Nel secondo caso potrà scegliere tra avviare una partita a Memory o tornare al menù precedente.

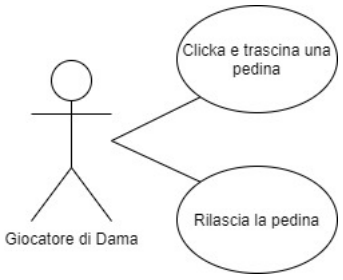

Figura 3: Diagramma Use Case di una partita in locale a Dama

L'utente, una volta che avvierà una partita di Dama, verrà riconosciuto come Giocatore di Dama, una generalizzazione del caso utente. Egli all'interno della partita di Dama potrà selezionare col click del mouse e la pressione continua del tasto sia il pezzo da muovere che la sua destinazione, trascinandolo verso di essa; oppure, rilasciando un pezzo della scacchiera già selezionato, potrà deselezionarlo o muoverlo.

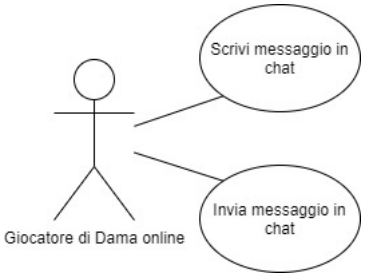

Figura 4: Diagramma Use Case di una partita online a Dama

Nel caso di una partita online a Dama, il Giocatore di Dama Online, generalizzazione del Giocatore di Dama, potrà scrivere dei messaggi in chat ed inviarli.

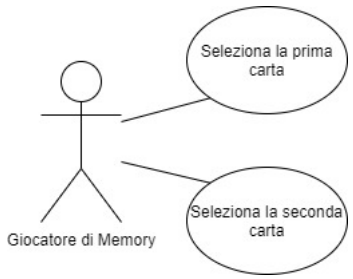

Figura 5: Diagramma Use Case di una partita a Memory

Nel caso invece di una partita a Memory, chiameremo la generalizzazione dell'Utente, Giocatore di Memory, ed egli, una volta avviata la partita di Memory, potrà selezionare la prima carta del suo turno e la seconda.

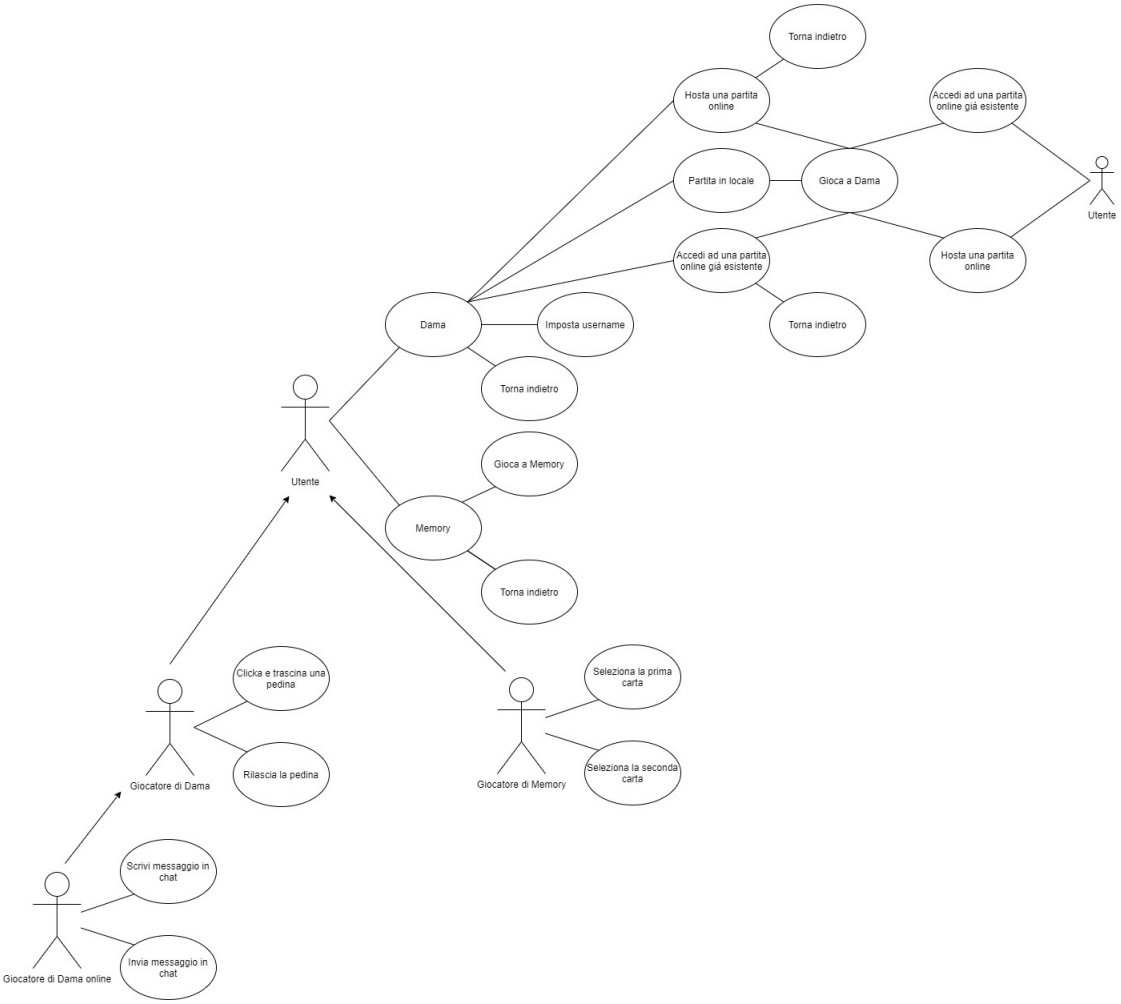

Figura 6: Diagramma Use Case generale del software

Nella figura 6 possiamo vedere il diagramma Use Case completo del software, dove gli attori e le loro generalizzazioni sono collegati tra loro ed è possibile vedere una panoramica completa dei casi d'uso del progetto.

2 Sviluppo del software

In questa parte del documento verrà trattato tutto il processo di sviluppo del software discutendone gli strumenti, l'organizzazione, le strategie, i metodi e valutandone infine i risultati.

2.1 Strumenti utilizzati

Nello sviluppo di questo software sono stati impiegati i seguenti strumenti:

- **Unity:** Unity è un motore grafico multipiattaforma sviluppato da Unity Technologies che consente lo sviluppo di videogiochi e altri contenuti interattivi, quali visualizzazioni architettoniche o animazioni 3D in tempo reale. La scelta è ricaduta su Unity piuttosto che sugli altri concorrenti sul mercato in quanto esso offre un piano gratuito completo di ogni funzionalità, avevo già delle esperienze di programmazione in C# e soprattutto, al momento della scrittura di questo documento (Agosto 2020), Unity è uno dei motori grafici più utilizzati nel panorama professionale dello sviluppo di videogiochi. Inoltre Unity presenta un framework built-in per l'automated testing.
- **Microsoft Visual Studio:** Microsoft Visual Studio (o più comunemente Visual Studio) è un ambiente di sviluppo integrato (Integrated development environment o IDE) sviluppato da Microsoft. In questo caso la scelta è stata quasi ovvia, in quanto l'integrazione tra Unity e Visual Studio è totalmente supportata dai loro sviluppatori.
- **LaTeX:** LaTeXè un linguaggio di markup per la preparazione di testi, basato sul programma di composizione tipografica TeXed è stato utilizzato per la scrittura di questo documento.
- **Asset grafici gratuiti:** Alcuni asset grafici utilizzati in questo progetto sono asset gratuiti o con licenza freeware acquistati sull'asset store di Unity. Lo skybox e l'acqua utilizzati nella scenografia del gioco sono asset gratuiti utilizzati in maniera lecita secondo le norme delle loro licenze.
- **Draw.io:** Draw.io è un'applicazione di diagrammi gratuita che è stata impiegata per disegnare tutti i diagrammi presenti in questo documento.

2.2 Progettazione

Suddividiamo il lavoro in maniera logica ed iterativa, individuando le caratteristiche del progetto che possono essere implementate in maniera "standalone", permettendoci quindi di poter sostenere delle release continue augurandoci un ritmo di almeno una release al giorno. Scriviamo quindi le user

stories.

User stories 1

1. Creazione scacchiera e pezzi
2. Codifica del movimento dei pezzi
3. Regole per il movimento dei pezzi
4. Codifica del codice per il check della validità delle mosse
5. Promozione a regina
6. Codifica lato server
7. Codifica lato client
8. Creazione Menù
9. Connessione utenti
10. Invio delle mosse online
11. Miglioramenti estetici e inserimento degli alert
12. Codifica della partita in locale
13. Effetti grafici per i pezzi costretti a muovere
14. Codifica della chat in-game
15. Abbellimenti grafici

Questa lista è stata stilata nell'ordine in cui le stories sono state prese in carico e le funzionalità da loro descritte implementate. Non tutte esistono dai primi istanti di vita del progetto, infatti la 12, la 13 e la 14 sono state inserite in un secondo momento per far fronte a delle necessità, per favorire lo sviluppo di altre funzionalità o in generale per migliorare il risultato finale del progetto.

A seguito della richiesta da parte del cliente dell'aggiunta di un secondo gioco (Memory) nel softwre, sono state aggiunte le seguenti storie:

User Stories 2

1. Creazione tavolo e carte
2. Codifica delle regole di gioco
3. Codifica notifiche a schermo
4. Creazione menù Memory e menù principale
5. Perfezionamenti

In questo caso le storie sono di numero molto inferiore perché dovranno essere esaurite in un tempo molto più ristretto, e ci si è basati maggiormente sulle macrofunzionalità di gioco, suddividendo così il progetto in poche ma grandi aree tematiche (ciò è stato reso possibile anche dalla maggiore semplicità del gioco da implementare).

2.3 Sviluppo

- **Inizio - 11/08/2020**
  - **Inizio Storia 1**
  - Creazione dei modelli 3D della scacchiera e dei pezzi
  - Creazione e applicazione delle texture ai modelli 3D
  - Creazione dello script CheckersBoard.cs
  - Creazione dello script Piece.cs
  - Generazione dei pedoni sulla scacchiera (metodi GenerateBoard, GeneratePiece e MovePiece dello script CheckersBoard.cs)
  - **Fine Storia 1**
  - **Release 0.1:** Questa release all'avvio genera i modelli 3D della scacchiera e dei pedoni, applica loro le texture e li posiziona nelle loro posizioni corrette
  - **Inizio Storia 2**
  - Creazione del metodo UpdateMouseOver di CheckersBoard.cs per ottenere la posizione del mouse come 2 int (x e y)
  - Generazione e fix delle collisioni con la scacchiera

- Creazione del metodo SelectPiece di CheckersBoard.cs per la selezione del pedone nella posizione [x,y] con cui iniziare il drag per lo spostamento
- Creazione del metodo TryMove di CheckersBoard.cs e inizio della codifica delle regole di movimento, ma per permettere la release ci limitiamo a permettere qualunque movimento finché rispetti i limiti della scacchiera
- **Fine Storia 2**
- **Release 0.2:** Questa release aggiunge alla precedente il movimento delle pedine tramite il loro drag, che presenta dei bug che dovranno essere risolti nella prossima iterazione.

• **12/08/2020**

- **Inizio Storia 3**
- Creazione del metodo UpdatePieceDrag di CheckersBoard.cs per ottenere un effetto dove la pedina selezionata viene spostata verso l'alto e, finché viene tenuto il click su di essa, viene spostata in maniera concorde ai movimenti del mouse
- Update sul metodo TryMove di CheckersBoard.cs: viene specificato che se le coordinate della destinazione del pezzo sono uguali a quelle di partenza ai fini del codice il pezzo non sarà stato mosso
- Creazione del metodo ValidMove di Piece.cs per specificare le regole che definiscono una mossa come valida o no, considerando la direzione di movimento rispetto il colore della pedina, l'obbligo di movimento diagonale e se la pedina in questione è una regina o no. Inoltre iniziamo a codificare le condizioni per mangiare una pedina avversaria
- Concludiamo la codifica delle regole per mangiare una pedina avversaria sul metodo TryMove di CheckersBoard.cs distruggendo la pedina "mangiata"
- Creazione del metodo EndTurn di CheckersBoard.cs per liberare le variabili riguardanti la pedina appena mossa, passare il turno all'altro giocatore e controllare se è stata vinta la partita
- **Fine Storia 3**

- **Release 0.3:** Questa release risolve alcuni bug della precedente, aggiunge le regole di movimento delle pedine, rendendo impossibile il movimento sulle tessere bianche e permette di muovere le pedine di due posizioni se sulla seguente tessera di una delle loro due diagonali vi sono una pedina avversaria e subito dopo una tessera vuota. La release presenta comunque ancora dei bug riguardanti il movimento delle pedine e notiamo che in particolare l'istanza della pedina che dovrebbe venire "mangiata" non viene distrutta.
- **Inizio Storia 4**
- Risoluzione del bug della release precedente: dopo una mossa non valida le pedine tornano correttamente alla loro posizione originale
- Creazione del metodo IsForcedToMove di Piece.cs per controllare se una pedina si trova nella posizione di poter mangiare e quindi di conseguenza secondo le regole della dama dovrebbe essere costretta a farlo e implementiamo questo metodo nello script CheckersBoard.cs in un nuovo metodo chiamato ScanForcedMoves che ci restituirà una lista dei pezzi che saranno costretti a muovere
- Risolto il bug per cui una pedina non veniva distrutta dopo essere stata mangiata
- **Fine Storia 4**
- **Release 0.4:** Con questa release abbiamo un gioco quasi funzionante, possiamo muovere le pedine solo nelle posizioni valide, se una o più di una delle nostre pedine sono costrette a mangiare saranno le uniche pedine che saremo in grado di muovere e una volta che le pedine vengono mangiate, la loro istanza viene distrutta

- **13/08/2020**
  - **Inizio Storia 5**
  - Apportati piccoli cambiamenti nel codice per permettere il playtesting in locale
  - Update sul metodo EndTurn per controllare la posizione di una pedina al termine del suo movimento. Se essa avrà raggiunto il limite opposto della scacchiera e non è già una regina vuol dire che verrà promossa

- Creiamo un override del metodo ScanForcedMoves che prende in ingresso una pedina e la sua posizione. Ciò ci servirà per implementare il doppio salto nel nostro codice, ovvero quella situazione in cui una pedina che ha appena mangiato può mangiare nuovamente nello stesso turno se ne ha la possibilità
- Creazione del metodo Victory di CheckersBoard.cs a scopo di testing utilizzando dei Log
- Debug e Update del metodo CheckVictory di CheckersBoard.cs, dove attuiamo una scansione delle pedine ancora in gioco per vedere se entrambe le squadre hanno ancora almeno una pedina
- **Fine Storia 5**
- **Release 0.5:** Questa release presenta il gioco nella sua versione locale perfettamente funzionante e privo di bug.

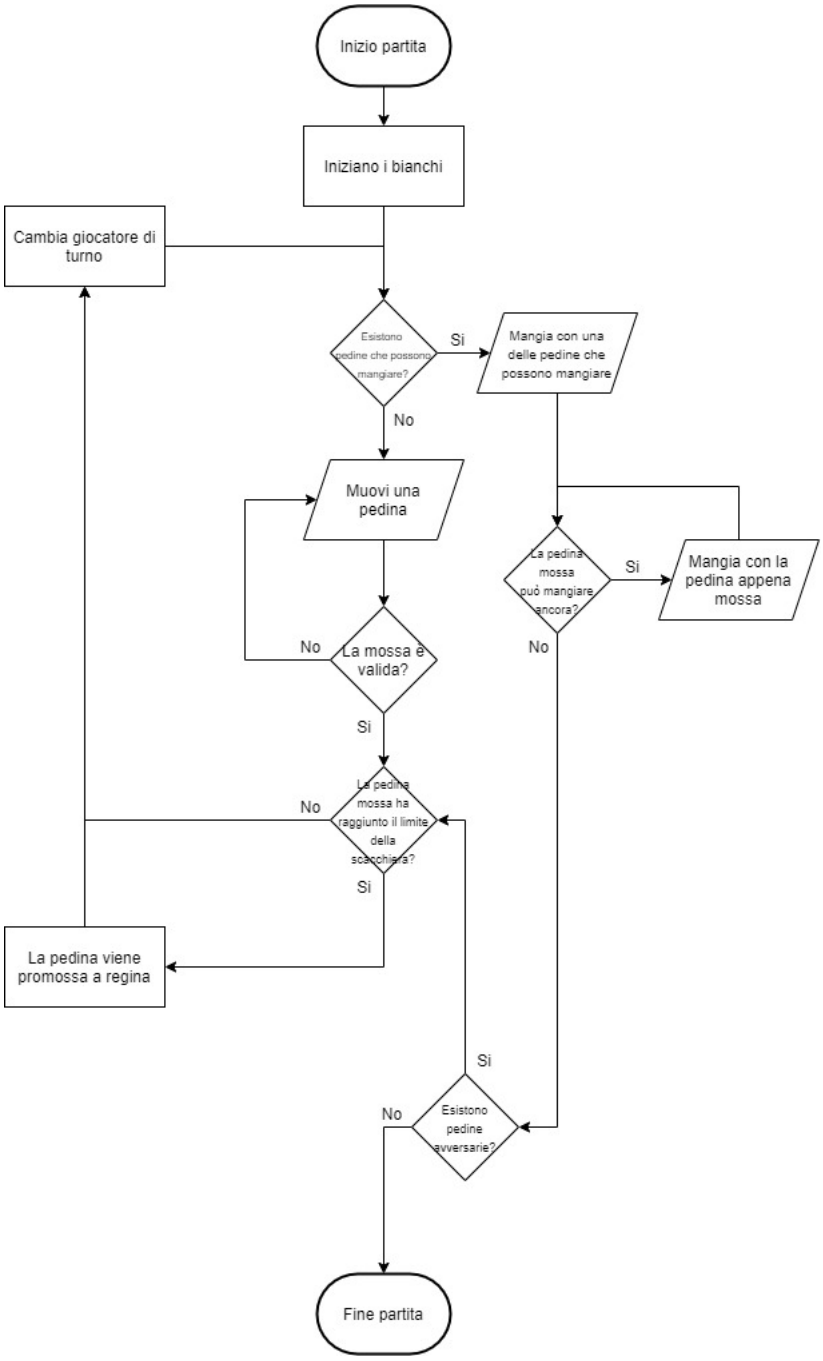

Figura 7: Diagramma di flusso di una partita di dama usato come riferimento durante lo sviluppo

- **14/08/2020**
  - **Inizio Storia 6**
  - Creazione dello script Server.cs
  - All'interno dello script Server.cs definiamo la classe ServerClient per dare una definizione dei client che stabiliranno una connessione con il server
  - Creazione del metodo Init di Server.cs dove inizializziamo le variabili del server come la lista dei client, settiamo un nuovo TcpListner che usa la socket 6321, avviamo il server e mettiamo il server in ascolto
  - Creazione del metodo StartListening di Server.cs che chiama la funzione BeginAcceptTcpClient che svolge l'handshake per la connessione Tcp
  - Creazione del metodo AcceptTcpClient di Server.cs per aggiungere alla lista clients l'oggetto ServerClient che verrà creato nel momento in cui un nuovo client si connetterà e rimettere subito dopo il server in ascolto
  - Creazione del metodo IsConnected di Server.cs che controlla se un client è connesso al server
  - Creazione del metodo Update di Server.cs dove richiamando il metodo IsConnected è possibile scoprire i client disconnessi, chiudere la connessione Tcp ed aggiungerli alla lista degli utenti disconnessi, oltre a creare un oggetto StreamReader che legga i messaggi inviati dai client ancora connessi. I client disconnessi verranno poi eliminati sia dalla lista disconnectList che dalla lista clients
  - Creazione del metodo OnIncomingData di Server.cs per stabilire il protocollo che il server seguirà quando riceverà messaggi dai client (al momento lo definiremo solo con un log che stamperà a video il nome del client che ha inviato il messaggio ed il testo del messaggio ricevuto)
  - Creazione del metodo Broadcast di Server.cs per inviare messaggi da parte del server alla lista di client connessi
  - **Fine Storia 6**
- **17/08/2020**

  - **Inizio Storia 7**
  - Creazione dello script Client.cs
  - All'interno dello script Client.cs definiamo la classe GameClient che sarà il tipo con cui identifichermo i client veri e propri
  - Creazione del metodo ConnectToServer di Client.cs dove viene fatta richiesta di connessione al server alla socket 6321 e vengono inizializzati i reader e i writer
  - Creazione del metodo OnIncomingData di Client.cs che definisce il protocollo che il client seguirà quando riceverà messaggi dal server (al momento lo definiremo solo con un log che stamperà a video il testo del messaggio ricevuto)
  - Creazione del metodo Send di Client.cs per inviare messaggi al server
  - Creazione del metodo Update di Client.cs che terrà il client in ascolto per eventuali messaggi da parte del server
  - Creazione del metodo CloseSocket di Client.cs che terminerà la connessione chiudendo reader, writer e socket
  - Creazione dei metodo OnApplicationQuit e OnDisable di Client.cs che chiama il metodo CloseSocket per terminare la connessione rispettivamente alla chiusura dell'applicazione e alla disattivazione dell'oggetto client
  - **Fine Storia 7**

- **18/08/2020**
  - **Inizio Storia 8**
  - Creazione di una nuova scena in Unity che chiameremo Menu e utilizzeremo per l'appunto per il menù principale del gioco
  - Creazione dello script GameManager.cs in cui viene codificato il funzionamento dei pulsanti sul menù.
  - Creazione dei metodi Connect Button e HostButton di GameManager.cs che saranno i metodi che verranno chiamati nei metodi OnClick dei pulsanti Connettiti e Ospita

- Utilizzando l'editor di Unity, nella scena Menu creiamo i pulsanti e le interfacce grafiche del menù (pulsanti Connettiti e Ospita, schermata di inserimento dell'IP con pulsanti di conferma e per tornare indietro e schermata di attesa di connessione con pulsante per tornare indietro)
- Creazione del metodo ConnectToServerButton di GameManager.cs, che sarà il metodo che verrà chiamato nel metodo OnClick del pulsante Connettiti del sottomenù Connect, che per il momento resterà vuoto
- Creazione del metodo BackButton di GameManager.cs che sarà il metodo che verrà chiamato nel metodo OnClick dei pulsanti Indietro e Annulla rispettivamente del sottomenù Connect e Host
- **Fine Storia 8**
- **Release 0.6:** In questa release tardiva è stata aggiunta l'interfaccia del menù e la sua struttura base, oltre alle basilari funzionalità di navigazione all'interno del menù stesso
- **Inizio Storia 9**
- Nel metodo ConnectToServerButton di GameManager.cs inseriamo l'indirizzo Ip per la connessione di default come localhost, generiamo l'istanza di un oggetto Client e ne chiamiamo il metodo ConnectToServer
- Nel metodo HostButton di GameManager.cs codifichiamo la generazione di un'istanza di un oggetto Server e ne chiamiamo il metodo Init. Generiamo anche qui l'istanza di un oggetto Client e ne chiamiamo il metodo ConnectToServer specificando come indirizzo Ip il localhost
- Creiamo un inputField nel menù per l'inserimento del nome del client
- Modifichiamo lo script Client.cs per specificare un nome dell'oggetto Client, e modifichiamo anche i metodi ConnectToServerButton e HostButton di GameManager.cs per acquisire dall'input del menù il nome del client o per dare di default rispettivamente il nome Client e Host
- Bugfix per risolvere il problema che avviene nel momento in cui si crea un server ed un client, e si esce dalla schermata di attesa: in

questo caso il server precedentemente creato non viene distrutto e rientrando nella schermata di attesa connessione ci ritroveremo con due istanze di un oggetto Server e di un oggetto Client. Aggiungiamo quindi il codice necessario per distruggere il server (se esiste) ed il client nel metodo BackButton di GameManager.cs

  - **Fine Storia 9**
  - **Release 0.7:** In questa release, dai log di debug è possibile notare che la connessione tra due giocatori funziona perfettamente seppure non sono ancora in grado di comunicare tra loro

- **19/08/2020**
  - **Inizio Storia 10**
  - Definiamo il nostro standard per i messaggi che verranno inviati tra client e server: la prima lettera indicherà il mittente, "S" (server) o "C" (client); Il resto dei caratteri indicherà il tipo di messaggio, "WHO" (identificazione), "CNN" (connessione), "MOV" (mossa), "MSG" (messaggio chat); Poi un carattere pipe (|) ricoprirà il ruolo di divisore tra i parametri passati nel messaggio Tcp in base alla sua natura (nome client + bool per identificare se il client è anche l'host nei messaggi di identificazione, nome client nei messaggi di connessione, posizione iniziale della pedina + posizione di arrivo della pedina nei messaggi di mossa, stringa contenente il messaggio di chat)
  - Nello script Server.cs creiamo un override del metodo Broadcast che anziché prendere una stringa ed una lista di ServerClient come parametri in ingresso, prenderà un solo oggetto ServerClient anziché la lista. In questo metodo ci limiteremo a creare una nuova lista di ServerClient con un unico elemento che sarà il ServerClient che è stato passato in ingresso
  - Modifichiamo il metodo AcceptTcpClient in Server.cs per inserire un Broadcast diretto agli altri client connessi al server per notificarli della connessione di un nuovo utente. Modifichiamo il metodo OnIncomingData in Client.cs per gestire il messaggio in questione inviato dal server
  - Il metodo OnIncomingData sia in Server.cs che in Client.cs prenderà la stringa ricevuta attraverso la connessione Tcp e eseguirà

uno Split in corrispondenza del carattere "|", per poi eseguire uno switch dove i vari casi dipenderanno dalla prima stringa. Quindi in questo caso ci curiamo del case "SWHO": per ogni client nella lista dei client connessi verrà chiamato il metodo UserConnected che istanzierà un oggetto GameClient e assegnerà il loro nome ad ognuno di essi. Infine il GameClient verrà aggiunto alla lista "players". Dopodiché verrà inviato dal client al server il messaggio "CWHO" che notificherà la volontà di connettersi

- Nello script Server.cs, nel metodo OnIncomingData, nel case "CWHO", il server prenderà i parametri del client che sta cercando di connettersi e dopodiché notificherà a tutti i client con un broadcast con un messaggio "SCNN" l'avvenuta connessione del nuovo client. Nel metodo OnIncomingData di Client.cs ogni client aggiungerà alla propria lista players il nuovo client (se la lista players raggiung ela dimensione 2 si inizia la partita)
- Bugfix: entrambi i giocatori una volta entrati in partita risultano come giocatore nero; risolviamo assegnando di default le pedine bianche all'host
- Modifichiamo il metodo EndTurn in CheckersBoard.cs per creare il messaggio "CMOV" che verrà inviato al server per notificare la mossa appena conclusa. Il messaggio conterrà le coordinate x ed y della posizione iniziale e della posizione finale della pedina. Nello script Server.cs, nel metodo OnIncomingData, nel case "CMOV", semplicemente, invieremo il medesimo messaggio in broadcast ai client connessi con la sigla "SMOV". Nello script Client.cs, nel metodo OnIncomingData, nel case "SMOV", chiamiamo il metodo TryMove dell'istanza CheckersBoard, passando le due coppie di coordinate x e y contenute nel messaggio
- Bugfix nel metodo CheckVictory di CheckersBoard.cs: il conteggio delle pedine rimanenti avveniva in un timing errato rispetto all'ultima pedina mangiata
- **Fine Storia 10**
- **Release 0.8:** Il gioco nella sua versione multiplayer è funzionante e senza bug. Sono ancora richieste delle migliorie, ma questa potremmo considerarla la prima release totalmente funzionale

- **20/08/2020**

- **Inizio Storia 11**
- Dopo aver acquistato il pacchetto base degli asset gratuiti di Unity, nella scena Game inseriamo l'elemento acqua sotto la scacchiera ed uno skybox notturno
- Copiamo il modello 3D della scacchiera insieme al resto della scena Game e li incolliamo nella scena Menu, eliminando lo script CheckersBoard.cs da questa versione dell'oggetto
- Inseriamo anche qualche pedina sull'oggetto 3D, dando loro un aspetto disordinato e piacevole da vedere. Regoliamo inoltre la posizione della camera sulla scena, cercando di dare una maggiore idea di dinamismo
- Creiamo le strutture grafiche che ci serviranno per implementare le notifiche durante la partita
- Nello script CheckersBoard.cs codifichiamo i metodi che ci permetteranno di gestire queste notifiche: Alert e UpdateAlert. Nel primo semplicemente verrà settato il testo della notifica, impostato il tempo in cui è stato avviato l'ultimo alert e settato un bool che indica la notifica come attiva. Nel secondo faremo in modo che dopo 1.5 secondi l'opacità della notifica inizi a calare e che dopo 2.5 secondi sia sparita del tutto
- Nel metodo Start di CheckersBoard.cs chiamiamo il metodo Alert in modo tale che ci notifichi il nome dei due giocatori della partita. Aggiungiamo le chiamate ad Alert anche nel metodo EndTurn, così che ogni volta che un giocatore passa il turno, ci sarà una notifica che ci avvertirà di chi è il turno attuale
- **Fine Storia 11**
- **Release 0.9:** Non ci sono stati molti cambiamenti funzionali dalla release precedente a questa, ma da un punto di vista grafico possiamo decisamente considerare questa nuova release molto più piacevole (anche se avevo iniziato a lavorarci non sono state implementate la telecamera mobile nella scena del menù e l'animazione "flip" della promozione a regina di una pedina in quanto non strettamente necessarie e per questo sono state depennate)
- **Inizio Storia 12**

- Creazione nuovo pulsante "Gioca in locale" nel menù di gioco per l'implementazione della partita in locale. Nello script GameManager.cs implementiamo il metodo LocalButton che verrà chiamato dal metodo OnClick del pulsante. In questo metodo verrà semplicemente cambiata la scena passando da Menu a Game
- Nello script CheckersBoard.cs modifichiamo vari metodi per permettere la partita in locale, utilizzando come discriminatore (o come valore booleano negli if) l'esistenza o meno di un client
- Diverse iterazioni fa avevo commentato una parte di codice che serviva per testare il funzionamento del gioco cambiando il turno del giocatore che muoveva. Ora questo frammento di codice verrà utilizzato per far funzionare la partita locale
- Aggiungiamo il caso dell'alert in assenza di client: in questo caso l'alert ci avvertirà solo del colore di pedine che deve muovere.
- **Fine Storia 12**
- **Release 0.10:** L'unico cambiamento di questa release rispetto alla precedente è la possibiltà di giocare anche in locale. Originariamente non era considerato un requisito funzionale, ma essendo di facile implementazione è stato aggiunto

- **21/08/2020**
  - **Inizio Storia 13**
  - Creiamo un generico oggetto 3D quadrato 1mx1m (le dimensioni di una tessera della scacchiera) e nel metodo Start di CheckersBoard.cs lo posizioniamo 100m sotto la scacchiera
  - Creiamo il metodo Highlight in CheckersBoard.cs che servirà per posizionare l'oggetto precedentemente creato sotto una pedina e chiamiamo questo metodo alla fine di ScanForcedMoves
  - Bugfix dovuto al mancato spawn dell'oggetto 3D. Inseriamo una chiamata a ScanForcedMoves alla fine di EndTurn
  - Bugfix dovuto ad uno strano posizionamento dell'oggetto 3D dopo una mossa non valida. Inseriamo una chiamata ad Highlight prima di ogni return in TryMove
  - Animiamo gli oggetti 3D dando loro una rotazione all'interno del metodo Update in CheckersBoard.cs

- All'oggetto 3D colleghiamo un effetto particellare per renderlo maggiormente visibile: dopo un lungo processo di trial and error è stato raggiunto un risultato soddisfacente con dei raggi di luce che si diffondono in maniera radiale sopra l'oggetto 3D, che alla fine decido di rendere invisibile, lasciando visibile solo la luce per gusti estetici
- Copio l'oggetto 3D 1 volta poiché nella dama è impossibile avere più di due pedoni che possono mangiare contemporaneamente e modifico il codice per agire su entrambi gli oggetti
- **Fine Storia 13**
- **Release 0.11:** Questa aggiunta estetica rende il gioco sicuramente migliore, mettendo così in risalto una regola che potrebbe sfuggire alla vista di giocatori meno esperti, risultando così una feature sia estetica che funzionale

- **22/08/2020**
  - **Inizio Storia 14**
  - Creiamo il case "CMSG" nello switch nel metodo OnIncomingData di Server.cs per poter gestire i messaggi della chat che andremo ad implementare: molto semplicemente il messaggio ricevuto verrà mandato in broadcast ai client connessi con la sigla iniziale "SMSG" e il nome del client che l'ha inviato
  - Creiamo il case "SMSG" nello switch nel metodo OnIncomingData di Client.cs per poter gestire i messaggi della chat ricevuti dal server. Chiamiamo il metodo ChatMessage dall'istanza di CheckersBoard, passando il testo del messaggio.
  - Creiamo gli elementi grafici che comporranno la chat nella schermata in-game: un pannello dove verranno aggiunti i messaggi inviati e ricevuti, uno spazio di input dove inserire il messaggio da inviare e il punlsante per confermare l'invio del messaggio
  - Creazione metodo ChatMessage e SendMessage di CheckersBoard.cs rispettivamente per stampare a video sul pannello della chat ed inviare i messaggi scritti sullo spazio di input
  - Modifichiamo il metodo Start di CheckersBoard.cs per creare un'istanza della chat nella partita online ma non nella partita in locale

  - **Fine Storia 14**
  - **Release 0.12:** Questa release aggiunge una chat in-game perfettamente funzionante che permette di comunicare tra i due giocatori. A livello funzionale il progetto è completo. Mancano solo alcune migliorie grafiche

- **Fine - 24/08/2020**
  - **Inizio Storia 15**
  - Nella scelta di migliorare graficamente il progetto decidiamo uno stile semplice ma di impatto, usando come font predominante un font pixelato gratuitamente acquisito sullo Unity asset store e come colori le scritte bianche su sfondo nero
  - Creiamo una scritta che graficamente faccia da titolo per l'applicazione. Usiamo i tool grafici direttamente messi a disposizione da Unity, senza servirci di programmi di grafica esterni
  - Modifichiamo tutte le interfacce grafiche del software secondo lo standard scelto
  - Creiamo un'animazione per il mouseover dei pulsanti del menù principale
  - Modifichiamo il metodo Victory in CheckersBoard.cs , inserendo una chiamata al metodo Alert che ci notifichi quale giocatore ha vinto. Inoltre modifichiamo il metodo Update in CheckersBoard.cs per chiudere la partita, distruggendo le istanze di Server e Client e tornare al menù principale dopo 3 sec
  - **Fine Storia 15**
  - **Release 1.0:** Questa ultima release la possiamo considerare completa e priva di bug e pronta per essere consegnata al cliente entro il tempo limite che ci siamo fissati. i requisiti funzionali sono stati implementati tutti correttamente ed il prodotto è visivamente piacevole

Dopo aver discusso col cliente e aver preso in carico le modifiche concordate, ricomincia il processo di sviluppo:

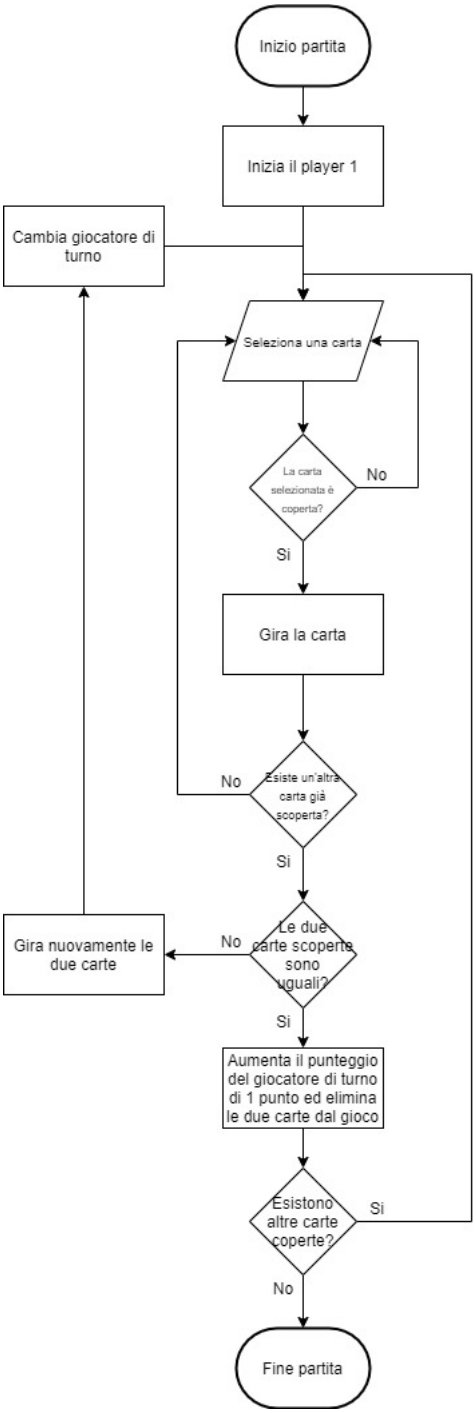

Figura 8: Diagramma di flusso di una partita di memory usato come riferimento durante lo sviluppo

- **Inizio - 31/08/2020**
  - **Inizio Storia 1**
  - Creazione del modello 3D del tavolo da gioco e delle carte. Si è scelto uno stile grafico affine a quello del gioco della Dama già implementato
  - Creazione dello script MemoryTable.cs: conterrà la maggior parte del codice riguardante il gioco. Qui verranno implementati i metodi che regoleranno il funzionamento della partita
  - Creazione dello script Card.cs usato per definire la classe Card che rappresenta le carte del gioco e ne contiene gli attributi ed i metodi
  - Creazione dei metodi GenerateDeck, GenerateCard, GenerateCopy, ShuffleDeck e PositionCards di MemoryTable.cs: il primo chiamerà gli altri 4 rispettivamente per creare le prime 20 carte di cui è composto il mazzo, creare le restanti 40 carte copie delle prime 20, mescolare il mazzo (usando dei numeri randomici per ogni carta, che detteranno di quante posizioni verrà spostata la suddetta carta) e posizionare le carte del mazzo coperte sul tavolo da gioco
  - **Fine Storia 1**
  - **Inizio Storia 2**
  - Creazione del metodo CardClicker di MemoryTable.cs che viene chiamato nel metodo Update per controllare se l'utente cliccando con il tasto sinistro del mouse clicchi o meno una carta
  - Creazione del metodo FlipCard di MemoryTable.cs che viene chiamato da CardClicker se si clicka effettivamente su una carta: questo metodo fa girare una carta al gioctore di turno se essa non è già girata e se è la prima ad essere stata girata durante il turno, gli permette di girarne un'altra, salvando il valore di quella appena girata. Se invece quando viene girata la carta ne esiste una già girata, verrà chiamato il metodo IsAMatch di Card.cs. Se le due carte sono una coppia verrà incrementato il punteggio a video del giocatore di turno, cancellato il riferimento alla carta girata in precedenza, diminuito il numero di coppie di carte ancora in gioco e se questo numero arriva a 0 si conclude il gioco. Se invece le due carte non sono una coppia, verrà settato un timer che dopo

1.5 secondi, chiamerà il metodo ReFlip di MemoryTable.cs, che rimetterà le carte in posizione coperta e infine verrà cambiato il giocatore di turno

- Creazione del metodo IsAMatch di Card.cs per verificare se le due carte (quella dalla cui istanza viene chiamato il metodo e quella che viene passa come parametro) sono uguali sia nel seme che nel valore di carta. Se lo sono il metodo restituirà un valore booleano vero, altrimenti, oltre a restituire falso, resetterà i valori delle due carte e ne cancellerà i riferimenti salvati
- Creazione del metodo EndGame di MemoryTable.cs che viene chiamato nel momento in cui non ci sono più coppie coperte in gioco. Semplicemente in questo metodo se il punteggio del primo giocatore è superiore a quello del secondo faremo apparire una notifica che ci avvertirà della vittoria del primo giocatore; in caso opposto ci notificherà della vittoria del secondo; nel caso in cui i due punteggi si eguaglino verrà comunicato il pareggio. Dopo 2 secondi si tornerà al menù del memory
- **Fine Storia 2**
- **Release 1.1:** In questa release inseriamo il gioco del Memory completo e perfettamente funzionante, anche se mancano ancora delle accortezze per quanto riguarda l'interfaccia

- **Fine - 31/08/2020**
- **Inizio - 01/09/2020**
  - **Inizio Storia 3**
  - Nell'editor di Unity creiamo 4 pannelli con testo: due che indicano il punteggio dei due giocatori, uno che indica il numero di coppie rimaste e uno, invisibile di default che useremo per notificare la fine della partita
  - Creazione del metodo UpdateScore di MemoryTable.cs che viene chiamato quando uno dei due giocatori segna un punto. Questo metodo modifica il testo dei 3 pannelli testuali su schermo, aggiornando ad ogni chiamata i 3 valori (il punteggio del primo giocatore, il punteggio del secondo giocatore e il numero di coppie rimanenti).

- **Fine Storia 3**
- **Release 1.2:** Questa release non apporta modifiche di tipo logico o funzionale al gioco, ma lo rende più userfriendly mostrando a video i punteggi dei giocatori e le coppie di carte rimaste, rendendolo più simile ad un gioco
- **Inizio Storia 4**
- Creazione delle scene MemoryMenu e MainMenu dall'editor di Unity. Nella prima inseriamo un pulsante che permette di iniziare una partita a memory ed uno che permette di cambiare scena e tornare al menù principale. Nel secondo inseriamo due pulsanti che permettano di navigare da questo menù ai menù di Dama o Memory
- Creazione degli script GameMenagerMemo.cs e GameManagerMain.cs dove sono stati codificati i funzionamenti dei pulsanti rispettivamente del menù di Memory e del menù principale
- Modifica della scena Menu e dello script GameManager.cs, per inserire un nuovo pulsante che permetta di tornare al menù principale dal menù della Dama
- **Fine Storia 4**
- **Inizio Storia 5**
- Modifica delle scene MemoryMenu e MainMenu, dove andremo ad applicare lo stesso standard estetico del menù di Dama, inseriamo inoltre lo skybox e l'effetto acqua sulle nuove scene (Memory, MemoryMenu, MainMenu) e i modelli 3D dei giochi privati di script ed inseriti per scopi estetici
- **Fine Storia 5**
- **Release 1.3:** Questa release inserisce perfettamente un menù principale e lo imposta come prima scena del videogioco, permette la navigazione tra i vari menù e permette sia di giocare al memory che alla dama nella stessa sessione del software. Il progetto allo stato attuale è pronto per la sua pubblicazione ed è in uno stato tale in cui sarebbe possibile implementare ulteriori giochi senza troppo lavoro

- **Fine - 01/09/2020**

2.4 Testing

In parallelo al processo di sviluppo sono stati svolti gli unit test sul codice a mano a mano che veniva implementato. Utilizzando Unity Test Framework i test sono stati svolti in maniera automatizzata, secondo la filosofia XP, permettendo così di risparmiare tempo nei processi di debug e di testing manuale.

Gli script di testing sono comunemente divisi in 3 fasi:

1. Arrange: gli input e le condizioni iniziali vengono esplicitati
2. Act: il metodo sotto test viene chiamato per ottenere il risultato
3. Assert: verificare che il risultato ottenuto sia quello sperato

Così facendo è possibile verificare qualunque metodo di una classe, associando ad ognuno di questi un test che ne assicuri il corretto funzionamento in varie sezioni critiche.

Nello specifico gli unit test servono a testare singolarmente ogni aspetto del sistema, e questi sono quelli che sono stati svolti (sia in maniera automatizzata che non):

GameManagerMain

- Avvio corretto
- Tutti i pulsanti eseguono correttamente l'animazione su metodo MouseOver()
- MemoryButton() su metodo OnClick() esegue correttamente il cambio di scena, caricando la scena MemoryMenu
- CheckersButton() su metodo OnClick() esegue correttamente il cambio di scena, caricando la scena Menu

GameManagerMemo

- Avvio corretto
- Tutti i pulsanti eseguono correttamente l'animazione su metodo MouseOver()

- GameButton() su metodo OnClick() esegue correttamente il cambio di scena, caricando la scena Memory
- BackButton() su metodo OnClick() esegue correttamente il cambio di scena, caricando la scena MainMenu

GameManager

- Avvio corretto
- Tutti i pulsanti eseguono correttamente l'animazione su metodo MouseOver()
- LocalButton() su metodo OnClick() esegue correttamente il cambio di scena, caricando la scena Game
- MainButton() su metodo OnClick() esegue correttamente il cambio di scena, caricando la scena MainMenu
- HostButton() su metodo OnClick() cambia correttamente la visibilità dei Layer del Canvas nella scena Menu
- HostButton() crea correttamente una istanza di Server e Client
- ConnectButton() su metodo OnClick() cambia correttamente la visibilità dei Layer del Canvas nella scena Menu
- ConnectToServerButton() su metodo OnClick() connette correttamente al dispositivo corrispondente all'indirizzo IP, se esso accetta la connessione sulla socket predefinita, altrimenti non accade nulla
- BackButton() su metodo OnClick() cambia correttamente la visibilità dei Layer del Canvas della scena Menu e distrugge correttamente le istanze di Client e Server se esistono
- La scena Game viene caricata correttamente se ci si unisce per primi ad un Server o se almeno un Client si connette all'istanza Server sul dispositivo

CheckersBoard

- Avvio corretto

- La scacchiera ed il bordo vengono caricati correttamente
- Le coordinate della posizione del mouse sopra la scacchiera vengono lette correttamente dal metodo UpdateMouseOver()
- Le collisioni del click del mouse sono limitate alla scacchiera
- I click esterni alla scacchiera non vengono registrati
- Le pedine vengono create e posizionate in maniera corretta ed in numero corretto
- Al click sulla tessera della scacchiera su cui si trova una pedina la pedina cambierà correttamente posizione
- Al rilascio del click la pedina tornerà correttamente alla sua posizione originale se non sarà stata rilasciata in una posizione che risulti in una mossa "legale"
- Se la pedina viene rilasciata fuori dalla scacchiera tornerà alla propria posizione originale
- Se la pedina viene rilasciata in uno spazio vuoto diagonalmente adiacente alla sua posizione originale nella direzione corretta o in uno spazio vuoto diagonalmente distante 2 tessere dalla sua posizione originale nella direzione corretta con la posizione in mezzo occupata da una pedina avversaria o senza considerare la direzione corretta se la pedina è una regina, la mossa sarà considerata valida
- Se la pedina compie una mossa valida tornerà alla sua posizione veritcale precedente, nella nuova posizione raggiunta
- Se la pedina ha compiuto una mossa valida superando una pedina avversaria e spostandosi di due posizioni allora della pedina avversaria in mezzo (la pedina mangiata) viene cancellata correttamente l'istanza
- Non è possibile muovere durante il turno avversario o muovere le pedine avversarie
- Inizia il giocatore bianco
- Se la partita è online, il giocatore bianco è l'host

- Se un giocatore ha compiuto una mossa durante il suo turno non potrà agire fino a che non sarà nuovamente il proprio turno
- Se un giocatore ha concluso il suo turno il turno passa al giocatore successivo
- Se una o più pedine possono mangiare il giocatore sarà costretto a muovere una di quelle pedine
- Se una pedina appena mossa ha mangiato si trova in posizione di poter mangiare ancora dovrà farlo, rimandando di fatto la fine del turno del giocatore
- Se una pedina raggiunge l'altro capo della scacchiera viene promossa a regina
- Gli oggetti Highlight vengono caricati correttamente ad inizio partita nella loro posizione corretta
- Quando viene individuata una pedina costretta a mangiare viene cambiata correttamente la posizione dello/degli oggetti Highlight
- Al termine del turno o della mossa gli oggetti Highlight vengono riportati nella loro posizione iniziale
- La Chat viene caricata correttamente se si tratta di una partita online
- Nello spazio di input si può scrivere correttamente
- Il pulsante di invio dei messaggi funziona correttamente inviando un messaggio se ne è stato composto uno nello spazio di input
- I messaggi inviati e ricevuti vengono mostrati correttamente nel pannello di chat
- Il pannello di chat scorre correttamente in su ed in giù e mantiene costanti le proporzioni laterali dei pannelli contenenti i messaggi
- Il nome del mittente del messaggio viene mostrato correttamente
- L'opacità dell'oggetto AlertCanvas è settata correttamente all'avvio della partita

- Il testo dell'oggetto AlertCanvas viene modificato correttamente quando viene chiamato il metodo Alert()
- L'opacità dell'oggetto AlertCanvas viene modificata e resettata correttamente e nei tempi prefissati
- Quando l'ultima pedina di un dato colore viene mangiata, il giocatore che ha appena mosso vince
- Quando un giocatore vince viene visualizzato correttamente il messaggio di vittoria (o sconfitta) e viene caricata la scena Menu distruggendo tutte le istanze attuali dopo il tempo corretto
- Quando la partita viene caricata vengono notificati correttamente i nomi dei due giocatori
- Quando un giocatore finisce il turno viene notificato correttamente il turno del prossimo giocatore

Client

- Avvio corretto
- La richiesta di connessione al server funziona correttamente
- Il metodo Update() mantiene correttamente il client in costante attesa di messagi da parte del server
- Il client invia e riceve correttamente i messaggi dialogando con il server a cui è connesso
- Il metodo OnIncomingData() permette di ricevere e interpretare correttamente i messaggi ricevuti dal server e risponde con i messaggi corretti o le azioni corrette
- Se si connette un nuovo utente il client lo inserirà correttamente nella sua lista di client
- La connessione Tcp con il server del client viene chiusa correttamente nell'eventualità nell'eventualità della distruzione del client o della chiusura dell'applicazione

- La classe GameClient funziona in maniera adeguata allo scopo di rappresentare gli altri client connessi al server

Server

- Alla chiamata del metodo Init() il server si inizializza correttamente e si mette in ascolto sulla socket 6321
- Il server fa il check dei client connessi nel metodo Update() e se sono disconnessi ne chiude la connessione e poi li rimuove, altrimenti controlla se il client ha inviato dei messaggi e li legge
- Il metodo AcceptTcp() aggiunge correttamente il client che ha inviato la richiesta di connessione alla lista dei client connessi e notifica gli altri client della connessione
- Il server riceve ed invia correttamente messaggi dialogando con i client connessi
- Il metodo Broadcast ed il suo relativo override funzionano correttamente
- Il sistema di messaggi di OnIncomingData, permette di interpretare correttamente i messaggi ricevuti dal client e invia le risposte corrette
- La classe ServerClient funziona in maniera adeguata allo scopo di rappresentare i client connessi

MemoryTable

- Avvio corretto
- Il tavolo viene caricato correttamente
- Il mazzo di carte viene generato correttamente
- Le carte vengono posizionate correttamente sul tavolo
- Il BoxCollider funziona correttamente e il click del mouse seleziona correttamente ogni carta
- Al click la carta viene girata correttamente eseguendo il metodo Flip() e l'attributo isShown vien settato a true

- Se viene clickata una carta con isShown==true essa non eseguirà il metodo Flip()
- Chiamanfo il metodo Flip() viene correttamente modificata la posizione della carta, ruotandola sul fianco di 180 gradi
- Se il giocatore di turno clicka su una seconda carta invocandone il metodo Flip() ne verrà chiamato anche il metodo IsAMatch() passando come parametro la carta selezionata prima, restituendo correttamente true se le carte hanno uguale seme e uguale valore o false in caso contrario
- Se il metodo IsAMatch restituisce false la posizione delle due carte torna quella originale e il loro attributo isShown viene resettato a false
- Se le carte sono una coppia il punteggio del giocatore aumenta correttamente, mentre il numero di coppie rimanenti diminuisce correttamente
- Se le carte non sono una coppia il turno passa al giocatore successivo
- Se il numero di coppie rimanenti raggiunge lo 0 la partita si conclude
- Il giocatore col punteggio più alto verrà considerato vincitore e apparirà correttamente un alert che lo comunicherà
- A seguito dell'alert dopo un tempo corretto verrà caricata la scena MemoryMenu distruggendo ogni istanza dell'attuale scena

2.5 Class Diagram

Il diagramma delle classi (class diagram) serve a fornire una vista strutturale del sistema in termini di attributi e metodi delle classi e le relazioni che intercorrono tra loro. Il diagramma fa riferimento alle classi effettivamente realizzate e alle strutture dati effettivamente impiegate, infatti non lo definiamo uno schema concettuale, ma implementativo.

Nelle classi gli attributi sono scritti nella forma:
NomeAttributo : TipoAttributo = ValoreDefault

E i metodi nella forma:
NomeMetodo(ListaTipoParametri) : TipoValoreRestituito

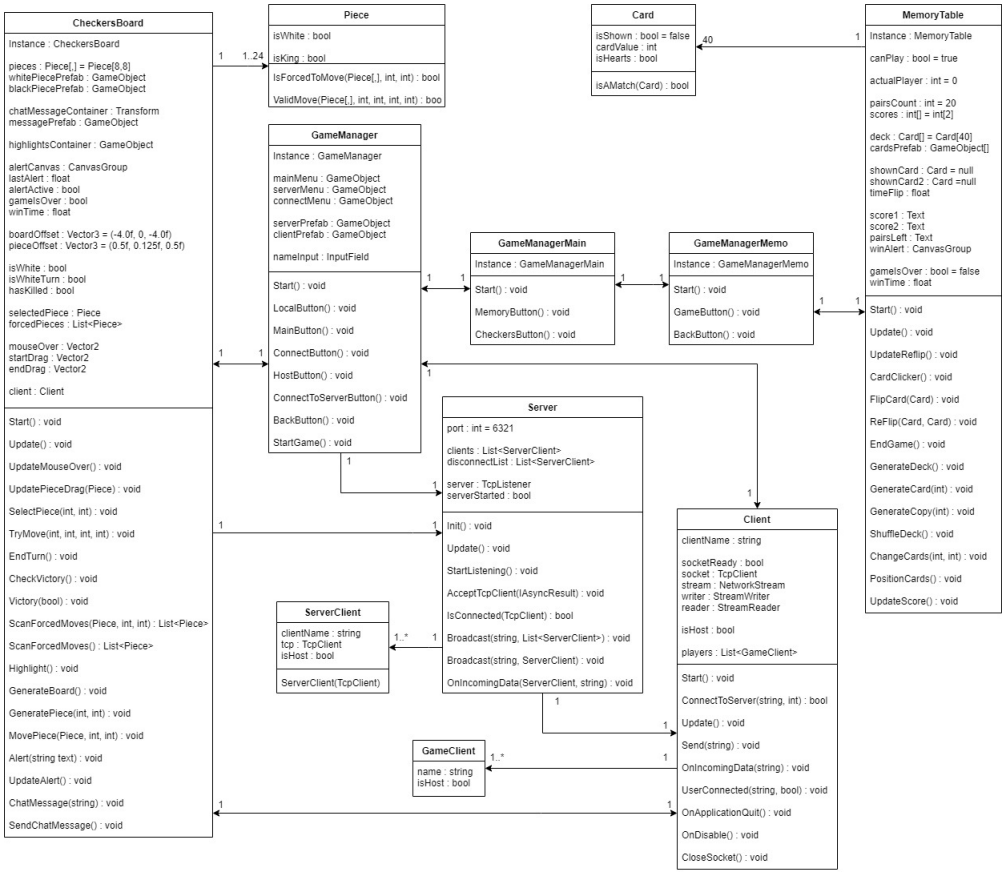

Figura 9: Diagramma delle classi del sistema

Nella figura 9 possiamo vedere il diagramma delle classi sviluppato a seguito della conclusione del progetto per dare una rappresentazione completa delle classi e delle loro relazioni. Come è possibile notare non esistono classi frutto di aggregazioni, questo perché sia nel caso delle pedine della dama che delle carte del memory, non mi era sembrato conveniente, a livello di sfruttamento delle risorse, creare una ipotetica superclasse di pezzi che avrei chiamato Team o una superclasse di carte che avrei chiamato Deck. Invece le classi Piece e Card vengono utilizzate per definire le istanze di oggetti fondamentali all'interno di CheckersBoard e MemoryTable, ovvero Pieces e Deck (definiti come array di oggetti di classe rispettivamente Piece e Card). Infatti le loro occorrenze all'interno dei due script sono numeri precisi: 24 (12 pedine bianche e 12 pedine nere) occorrenze di Piece dentro CheckersBoard e

40 (10 carte per seme ripetute due volte) occorrenze di Card dentro MemoryTable; va fatto notare inoltre che le occorrenze di Piece sono comprese tra 1 e 24 poiché esse possono variare nel tempo, visto che nel momento in cui una pedina viene mangiata durante una partita, la sua istanza come GameObject viene distrutta, però in nessun caso le pedine potranno essere di numero inferiore ad 1, quindi durante il ciclo di vita di un'istanza di CheckersBoard, il numero di istanze di Piece può diminuire fino ad un minimo di 1.

Salta inoltre subito all'occhio che in nessuna relazione è mai presente più di una occorrenza di CheckersBoard o di MemoryTable: questo perché in nessun caso esisterà contemporaneamente più di una istanza delle due classi.

La classe MemoryTable è legata solo da un'altra relazione con GameManagerMemo, in quanto esse richiamano il costruttore dell'altra classe nel momento in cui distruggono la propria istanza. Medesimo discorso vale per la relazione tra CheckersBoard e GameManager.

Sia GameManager che GameManagerMemo sono a loro volta in relazione con GameManagerMain, in quanto anche in questo caso esse chiamano vicendevolmente il costruttore dell'altra classe nel momento in cui vengono distrutte.

GameManager presenta poi una relazione con Server e Client, in quanto è lei stessa a chiamare dentro di se i metodi Init di Server e Start di Client.

Server a sua volta presenta una relazione con CheckersBoard, poiché sarà la stessa CheckersBoard, che prima di distruggere la propria istanza, distruggerà l'istanza di Server (così come quella di Client). Inoltre, la classe server è in relazione uno a molti con ServerClient, la classe che definisce l'oggetto che rappresenta i client connessi al server, oltre che con Client ovviamente, dal momento che essa è la classe con cui interagisce maggiormente tramite il metodo Broadcast e OnIncomingData.

Allo stesso modo Client, oltre che con Server, GameManager e CheckersBoard è in relazione con la classe GameClient, la classe definita per descrivere gli oggetti che rappresentano le istanze degli altri client connessi al server.

2.6 Deployment/Component Diagram

Il Deployment Diagram è un diagramma strutturale che mostra la configurazione del sistema durante la sua esecuzione, mettendone in risalto i nodi, le dipendenze e le associazioni. Questo tipo di diagrammi viene utilizzato per visualizzare e specificare le caratteristiche di sistemi embedded, distribuiti,

client/server. Potremmo definire il Deployment Diagram come un tipo di Class Diagram che si focalizza sui nodi di un sistema.

Il Component Diagram scompone il sistema analizzato nelle sue funzionalità ad alto livello. Come il Deployment Diagram, possiamo definire il Component Diagram come un tipo di Class Diagram, che in questo caso invece si focalizza sulle componenti di un sistema utilizzate per l'implementazione della visione statica statica di un sistema.

Solitamente è possibile creare un diagramma ibrido tra Deployment e Component, risultando in una visione che mostra la relazione tra le componenti Software (SW) ed Hardware (HW) del sistema.

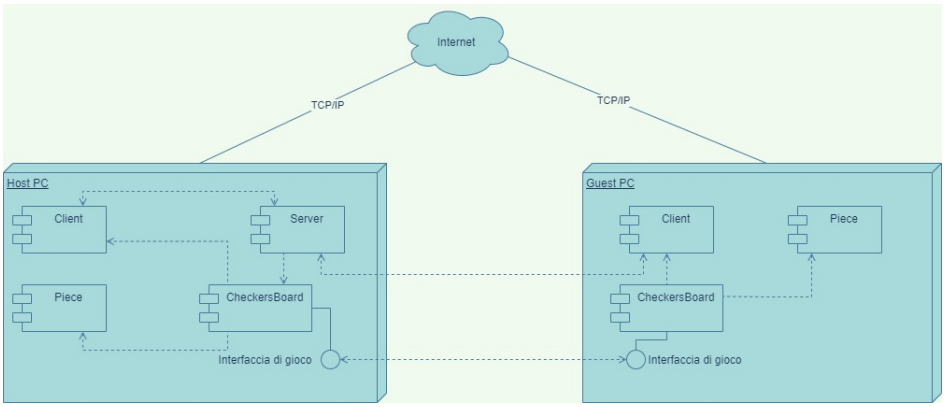

Figura 10: Deployment/Component Diagram di una sessione di una partita online a dama

Nella figura 10 possiamo vedere il deployment/component diagram del sistema durante una partita online di dama. Ho deciso di non includere la rappresentazione del resto dell'esecuzione del sistema in quanto non aggiungerebbe nulla di nuovo rispetto a quanto è stato già illustrato dal class diagram precedentemente mostrato nella figura 9.

A livello hardware identifichiamo due soli nodi principali: il PC dell'host e il PC del guest. All'interno dell'architettura del sistema essi sono di pari livello, connessi tra loro tramite internet con l'utilizzo di una connessione TCP; verrebbe quasi da interpretarla come una connessione P2P se non fosse per le componenti che vengono istanziate durante il run-time dell'applicazione. Infatti vediamo la differenziazione tra Client e Server tra i due PC non a livello HW ma a livello SW: l'host, oltre alla sua istanza di Client, identica a quella del guest, presenta la propria istanza Server, che ha un ruolo chiave nel

gestire la connessione a livello applicativo tra i due nodi. Le dipendenze tra i vari componenti e risorse SW del sistema li abbiamo già visti rapresentati in precedenza.

Notiamo inoltre che è necessario collegare tra loro nel diagramma anche le interfacce di gioco derivate dal componente CheckersBoard: ciò è dovuto alla connessione in real-time che permette di visualizzare una qualsiasi modifica ad una delle due componenti anche sull'altro dispositivo in tempo reale.

3 Conclusioni e codice

Il progetto è stato completato nei tempi previsti ed entro la data di consegna prefissata. La documentazione qui prodotta presenta un'analisi metodologica ed una descrizione del lavoro svolto abbastanza miticolosa, cercando di mantenere un linguaggio che per lo più risulti comprensibile anche ad un ipotetico cliente, o generico lettore, sprovvisto di competenze in ambito informatico. Il software risulta funzionante e privo di bug ed è stato testato per questo tipo di indagine più volte.

Possiamo quindi in conclusione affermare che lo sviluppo del progetto si è concluso con risultato positivo.

Qui di seguito verrà mostrato il codice sviluppato per il progetto, reperibile anche all'indirizzo: `https://github.com/MTheHead61/SoftwareEngProject`.

Listing 1: CheckersBoard.cs

```
1  using System.Collections.Generic;
2  using System.Collections;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using UnityEngine.SceneManagement;
6
7  public class CheckersBoard : MonoBehaviour
8  {
9    public static CheckersBoard Instance { set; get; }
10
11   public Piece[,] pieces = new Piece[8,8];
12   public GameObject whitePiecePrefab;
13   public GameObject blackPiecePrefab;
14
15   public Transform chatMessageContainer;
16   public GameObject messagePrefab;
17
```

```
18    public GameObject highlightsContainer;
19
20    public CanvasGroup alertCanvas;
21    private float lastAlert;
22    private bool alertActive;
23    private bool gameIsOver;
24    private float winTime;
25
26    private Vector3 boardOffset = new Vector3(-4.0f, 0, -4.0f
          );
27    private Vector3 pieceOffset = new Vector3(0.5f, 0.125f,
          0.5f);
28
29    public bool isWhite;
30    private bool isWhiteTurn;
31    private bool hasKilled;
32
33    private Piece selectedPiece;
34    private List<Piece> forcedPieces;
35
36    private Vector2 mouseOver;
37    private Vector2 startDrag;
38    private Vector2 endDrag;
39
40    private Client client;
41
42    private void Start()
43    {
44      Instance = this;
45
46      client = FindObjectOfType<Client>();
47
48      foreach(Transform t in highlightsContainer.transform)
49      {
50        t.position = Vector3.down * 100;
51      }
52
53      if(client)
54      {
55        isWhite = client.isHost;
56        Alert(client.players[0].name + " VS " + client.
              players[1].name);
57      }
58      else
59      {
```

```
60        Alert("Turno giocatore bianco");
61        Transform c = GameObject.Find("Canvas").transform;
62        foreach(Transform t in c)
63        {
64          t.gameObject.SetActive(false);
65        }
66        c.GetChild(0).gameObject.SetActive(true);
67      }
68
69      isWhiteTurn = true;
70      forcedPieces = new List<Piece>();
71      GenerateBoard();
72    }
73
74    private void Update()
75      {
76      if(gameIsOver)
77      {
78        if(Time.time - winTime > 3.0f)
79        {
80          Server server = FindObjectOfType<Server>();
81          Client client = FindObjectOfType<Client>();
82
83          if(server)
84            Destroy(server.gameObject);
85
86          if(client)
87            Destroy(client.gameObject);
88
89          SceneManager.LoadScene("Menu");
90        }
91        return;
92      }
93      foreach(Transform t in highlightsContainer.transform)
94      {
95        t.Rotate(Vector3.up * 90 * Time.deltaTime);
96      }
97
98      UpdateAlert();
99      UpdateMouseOver();
100
101         if((isWhite)?isWhiteTurn:!isWhiteTurn)
102         {
103       int x = (int)mouseOver.x;
104       int y = (int)mouseOver.y;
```

```
105
106       if (selectedPiece != null)
107         UpdatePieceDrag(selectedPiece);
108
109       if (Input.GetMouseButtonDown(0))
110         SelectPiece(x, y);
111
112       if (Input.GetMouseButtonUp(0))
113         TryMove((int)startDrag.x, (int)startDrag.y, x, y);
114     }
115
116   }
117
118   private void UpdateMouseOver()
119     {
120     if(!Camera.main)
121         {
122       Debug.Log("Camera main non trovata");
123       return;
124         }
125     RaycastHit hit;
126     if(Physics.Raycast(Camera.main.ScreenPointToRay(Input.
          mousePosition),out hit, 25.0f, LayerMask.GetMask("
          Board")))
127         {
128       mouseOver.x = (int)(hit.point.x - boardOffset.x);
129       mouseOver.y = (int)(hit.point.z - boardOffset.z);
130         }
131     else
132         {
133       mouseOver.x = -1;
134       mouseOver.y = -1;
135         }
136     }
137
138   private void UpdatePieceDrag(Piece p)
139     {
140     if (!Camera.main)
141     {
142       Debug.Log("Camera main non trovata");
143       return;
144     }
145     RaycastHit hit;
146     if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.
          mousePosition), out hit, 25.0f, LayerMask.GetMask("
```

```
          Board")))
147     {
148       p.transform.position = hit.point + Vector3.up;
149
150     }
151   }
152
153   private void SelectPiece(int x, int y)
154     {
155     if (x < 0 || x >= 8 || y < 0 || y >= 8)
156       return;
157
158     Piece p = pieces[x, y];
159     if(p != null && p.isWhite == isWhite)
160         {
161       if (forcedPieces.Count == 0)
162       {
163         selectedPiece = p;
164         startDrag = mouseOver;
165       }
166             else
167             {
168         if (forcedPieces.Find(fp => fp == p) == null)
169           return;
170
171         selectedPiece = p;
172         startDrag = mouseOver;
173             }
174       }
175     }
176
177   public void TryMove(int x1, int y1, int x2, int y2)
178     {
179     forcedPieces = ScanForcedMoves();
180
181     //Multiplayer
182     startDrag = new Vector2(x1, y1);
183     endDrag = new Vector2(x2, y2);
184     selectedPiece = pieces[x1, y1];
185
186     //Fuori limite
187     if (x2 < 0 || x2 >= 8 || y2 < 0 || y2 >= 8)
188         {
189       if (selectedPiece != null)
190         MovePiece(selectedPiece, x1, y1);
```

```
191
192         startDrag = Vector2.zero;
193         selectedPiece = null;
194         Highlight();
195         return;
196           }
197
198       if (selectedPiece != null)
199           {
200         //Nessuna mossa
201         if (endDrag == startDrag)
202               {
203           MovePiece(selectedPiece, x1, y1);
204           startDrag = Vector2.zero;
205           selectedPiece = null;
206           Highlight();
207           return;
208         }
209
210               //Controllo mossa valida
211               if (selectedPiece.ValidMove(pieces, x1, y1, x2,
                      y2))
212               {
213                   //Mangiato
214                   if (Mathf.Abs(x2 - x1) == 2)
215                   {
216             Piece p = pieces[(x1 + x2) / 2, (y1 + y2) / 2];
217                       if (p != null)
218                       {
219             pieces[(x1 + x2) / 2, (y1 + y2) / 2] = null;
220             DestroyImmediate(p.gameObject);
221             hasKilled = true;
222           }
223         }
224
225                   if (forcedPieces.Count != 0 && !hasKilled)
226                   {
227           MovePiece(selectedPiece, x1, y1);
228           startDrag = Vector2.zero;
229           selectedPiece = null;
230           Highlight();
231           return;
232         }
233
234         pieces[x2, y2] = selectedPiece;
```

```
235         pieces[x1, y1] = null;
236         MovePiece(selectedPiece, x2, y2);
237 
238         EndTurn();
239             }
240             else
241             {
242         MovePiece(selectedPiece, x1, y1);
243         startDrag = Vector2.zero;
244         selectedPiece = null;
245         Highlight();
246         return;
247       }
248         }
249 
250     }
251 
252   private void EndTurn()
253     {
254     int x = (int)endDrag.x;
255     int y = (int)endDrag.y;
256 
257     //Promozione
258         if (selectedPiece != null)
259         {
260             if (selectedPiece.isWhite && !selectedPiece.
                    isKing && y == 7)
261             {
262         selectedPiece.isKing = true;
263         selectedPiece.transform.Rotate(Vector3.right * 180)
                ;
264             }
265       else if (!selectedPiece.isWhite && !selectedPiece.
             isKing && y == 0)
266       {
267         selectedPiece.isKing = true;
268         selectedPiece.transform.Rotate(Vector3.right * 180)
                ;
269       }
270     }
271 
272     if(client)
273     {
274       string msg = "CMOV|";
275       msg += startDrag.x.ToString() + "|";
```

```
276         msg += startDrag.y.ToString() + "|";
277         msg += endDrag.x.ToString() + "|";
278         msg += endDrag.y.ToString();
279 
280       client.Send(msg);
281       }
282 
283       selectedPiece = null;
284       startDrag = Vector2.zero;
285 
286       if (ScanForcedMoves(selectedPiece, x, y).Count != 0 &&
              hasKilled)
287         return;
288 
289       isWhiteTurn = !isWhiteTurn;
290       hasKilled = false;
291       CheckVictory();
292 
293       if(!gameIsOver)
294       {
295         if(!client)
296         {
297           isWhite = !isWhite;
298           if(isWhite)
299             Alert("Turno giocatore bianco");
300           else
301             Alert("Turno giocatore nero");
302         }
303         else
304         {
305           if(isWhite)
306             Alert("Turno di " + client.players[0].name);
307           else
308             Alert("Turno di " + client.players[1].name);
309         }
310       }
311 
312       ScanForcedMoves();
313       }
314 
315     private void CheckVictory()
316       {
317       var ps = FindObjectsOfType<Piece>();
318       bool hasWhite = false, hasBlack = false;
319       for (int i=0; i < ps.Length; i++)
```

```
320         {
321       if (ps[i].isWhite)
322         hasWhite = true;
323       else
324         hasBlack = true;
325         }
326
327     if (!hasWhite)
328       Victory(false);
329     if (!hasBlack)
330       Victory(true);
331     }
332
333   private void Victory(bool isWhite)
334     {
335     winTime = Time.time;
336
337     if(isWhite)
338       Alert("Il bianco vince!");
339     else
340       Alert("Il nero vince!");
341
342     gameIsOver = true;
343   }
344
345   private List<Piece> ScanForcedMoves(Piece p, int x, int y
        )
346     {
347     forcedPieces = new List<Piece>();
348
349     if (pieces[x, y].IsForcedToMove(pieces, x, y))
350       forcedPieces.Add(pieces[x, y]);
351
352     Highlight();
353     return forcedPieces;
354   }
355
356   private List<Piece> ScanForcedMoves()
357     {
358     forcedPieces = new List<Piece>();
359
360     for (int i = 0; i < 8; i++)
361       for (int j = 0; j < 8; j++)
362         if (pieces[i, j] != null && pieces[i, j].isWhite ==
              isWhiteTurn)
```

```
363             if (pieces[i, j].IsForcedToMove(pieces, i, j))
364               forcedPieces.Add(pieces[i, j]);
365       Highlight();
366       return forcedPieces;
367       }
368
369     private void Highlight()
370     {
371       foreach(Transform t in highlightsContainer.transform)
372       {
373         t.position = Vector3.down * 100;
374       }
375
376       if (forcedPieces.Count > 0)
377       {
378         highlightsContainer.transform.GetChild(0).transform.
              position = forcedPieces[0].transform.position +
              Vector3.down * 0.1f;
379         if (forcedPieces.Count > 1)
380           highlightsContainer.transform.GetChild(1).transform
                .position = forcedPieces[1].transform.position +
                Vector3.down * 0.1f;
381       }
382     }
383
384     private void GenerateBoard()
385     {
386       //Genera pezzi bianchi
387       for (int y = 0; y < 3; y++)
388       {
389         bool oddRow = (y % 2 == 0);
390         for(int x = 0; x < 8; x += 2)
391         {
392           GeneratePiece((oddRow) ? x : x + 1, y);
393         }
394       }
395
396       //Genera pezzi neri
397       for (int y = 7; y > 4; y--)
398       {
399         bool oddRow = (y % 2 == 0);
400         for (int x = 0; x < 8; x += 2)
401         {
402           GeneratePiece((oddRow) ? x : x + 1, y);
403         }
```

```
404     }
405   }
406
407   private void GeneratePiece(int x, int y)
408   {
409     bool isPieceWhite = (y > 3) ? false : true;
410     GameObject go = Instantiate((isPieceWhite) ?
          whitePiecePrefab : blackPiecePrefab) as GameObject;
411     go.transform.SetParent(transform);
412     Piece p = go.GetComponent<Piece>();
413     pieces[x,y] = p;
414     MovePiece(p, x, y);
415
416   }
417
418   private void MovePiece(Piece p, int x, int y)
419     {
420     p.transform.position = (Vector3.right * x) + (Vector3.
          forward * y) + boardOffset + pieceOffset;
421     }
422
423   public void Alert(string text)
424   {
425     alertCanvas.GetComponentInChildren<Text>().text = text;
426     alertCanvas.alpha = 1;
427     lastAlert = Time.time;
428     alertActive = true;
429   }
430
431   public void UpdateAlert()
432   {
433     if(alertActive)
434     {
435       if(Time.time - lastAlert > 1.5f)
436       {
437
438         alertCanvas.alpha = 1 - ((Time.time - lastAlert) -
              1.5f);
439
440         if(Time.time - lastAlert > 2.5f)
441         {
442           alertActive = false;
443         }
444       }
445     }
```

```
446   }
447
448   public void ChatMessage(string msg)
449   {
450     GameObject go = Instantiate(messagePrefab) as
          GameObject;
451     go.transform.SetParent(chatMessageContainer);
452
453     go.GetComponentInChildren<Text>().text = msg;
454   }
455
456   public void SendChatMessage()
457   {
458     InputField i = GameObject.Find("MessageInput").
          GetComponent<InputField>();
459
460     if(i.text == "")
461       return;
462
463     client.Send("CMSG|" + i.text);
464
465     i.text = "";
466   }
467 }
```

Listing 2: Piece.cs

```
1  using System.Collections;
2  using UnityEngine;
3
4  public class Piece : MonoBehaviour
5  {
6      public bool isWhite;
7      public bool isKing;
8
9      public bool IsForcedToMove(Piece[,] board, int x, int y
           )
10     {
11         if (isWhite||isKing)
12         {
13             if (x >= 2 && y <= 5)
14             {
15                 Piece p = board[x - 1, y + 1];
16                 if (p != null && p.isWhite != isWhite)
17                 {
```

```
18                      if (board[x - 2, y + 2] == null)
19                          return true;
20                  }
21              }
22
23              if (x <= 5 && y <= 5)
24              {
25                  Piece p = board[x + 1, y + 1];
26                  if (p != null && p.isWhite != isWhite)
27                  {
28                      if (board[x + 2, y + 2] == null)
29                          return true;
30                  }
31              }
32          }
33          if (!isWhite||isKing)
34          {
35              if (x >= 2 && y >= 2)
36              {
37                  Piece p = board[x - 1, y - 1];
38                  if (p != null && p.isWhite != isWhite)
39                  {
40                      if (board[x - 2, y - 2] == null)
41                          return true;
42                  }
43              }
44
45              if (x <= 5 && y >= 2)
46              {
47                  Piece p = board[x + 1, y - 1];
48                  if (p != null && p.isWhite != isWhite)
49                  {
50                      if (board[x + 2, y - 2] == null)
51                          return true;
52                  }
53              }
54          }
55          return false;
56      }
57
58      public bool ValidMove(Piece[,] board, int x1, int y1,
            int x2, int y2)
59      {
60          //Muoversi su un'altra pedina
61          if (board[x2, y2] != null)
```

```
62              return false;
63
64          int deltaMove = Mathf.Abs(x1 - x2);
65          int deltaMoveY = y2 - y1;
66
67          if (isWhite || isKing)
68          {
69              if (deltaMove == 1)
70              {
71                  if (deltaMoveY == 1)
72                      return true;
73              }
74              else if (deltaMove == 2)
75              {
76                  if (deltaMoveY == 2)
77                  {
78                      Piece p = board[(x1 + x2) / 2, (y1 + y2
                            ) / 2];
79                      if (p != null && p.isWhite != isWhite)
80                          return true;
81                  }
82              }
83          }
84
85          if (!isWhite || isKing)
86          {
87              if (deltaMove == 1)
88              {
89                  if (deltaMoveY == -1)
90                      return true;
91              }
92              else if (deltaMove == 2)
93              {
94                  if (deltaMoveY == -2)
95                  {
96                      Piece p = board[(x1 + x2) / 2, (y1 + y2
                            ) / 2];
97                      if (p != null && p.isWhite != isWhite)
98                          return true;
99                  }
100             }
101         }
102
103         return false;
104
```

```
105     }
106 }
```

Listing 3: Server.cs

```
1  using System.Collections;
2  using UnityEngine;
3  using System.Net.Sockets;
4  using System;
5  using System.Collections.Generic;
6  using System.Net;
7  using System.IO;
8
9  public class Server : MonoBehaviour
10 {
11     public int port = 6321;
12
13     private List<ServerClient> clients;
14     private List<ServerClient> disconnectList;
15
16     private TcpListener server;
17     private bool serverStarted;
18
19     public void Init()
20     {
21         DontDestroyOnLoad(gameObject);
22     clients = new List<ServerClient>();
23     disconnectList = new List<ServerClient>();
24
25     try
26     {
27       server = new TcpListener(IPAddress.Any, port);
28       server.Start();
29
30       StartListening();
31       serverStarted = true;
32     }
33     catch(Exception e)
34     {
35       Debug.Log("Socket errore: " + e.Message);
36     }
37     }
38
39   private void Update()
40   {
```

```
41      if(!serverStarted)
42        return;
43
44      foreach(ServerClient c in clients)
45      {
46        //Client ancora connesso?
47        if(!IsConnected(c.tcp))
48        {
49          c.tcp.Close();
50          disconnectList.Add(c);
51          continue;
52        }
53        else
54        {
55          NetworkStream s = c.tcp.GetStream();
56          if(s.DataAvailable)
57          {
58            StreamReader reader = new StreamReader(s, true);
59            string data = reader.ReadLine();
60
61            if(data != null)
62              OnIncomingData(c, data);
63          }
64        }
65      }
66      for(int i = 0; i < disconnectList.Count - 1; i++)
67      {
68        //Avviso di disconnessione
69        clients.Remove(disconnectList[i]);
70        disconnectList.RemoveAt(i);
71      }
72    }
73
74    private void StartListening()
75    {
76      server.BeginAcceptTcpClient(AcceptTcpClient, server);
77    }
78
79    private void AcceptTcpClient(IAsyncResult ar)
80    {
81      TcpListener listener = (TcpListener)ar.AsyncState;
82
83      string allUsers = "";
84      foreach(ServerClient i in clients)
85      {
```

```
86          allUsers += i.clientName + "|" ;
87        }
88
89        ServerClient sc = new ServerClient(listener.
              EndAcceptTcpClient(ar));
90        clients.Add(sc);
91
92        StartListening();
93
94        Broadcast("SWHO|" + allUsers ,clients[clients.Count-1])
              ;
95      }
96
97      private bool IsConnected(TcpClient c)
98      {
99        try
100       {
101         if(c != null && c.Client != null && c.Client.
                Connected)
102         {
103           if(c.Client.Poll(0,SelectMode.SelectRead))
104             return !(c.Client.Receive(new byte[1],
                    SocketFlags.Peek) == 0);
105
106           return true;
107         }
108         else
109           return false;
110       }
111       catch
112       {
113         return false;
114       }
115     }
116     //Invio
117     private void Broadcast(string data, List<ServerClient> cl
          )
118     {
119       foreach(ServerClient sc in cl)
120       {
121         try
122         {
123           StreamWriter writer = new StreamWriter(sc.tcp.
                  GetStream());
124           writer.WriteLine(data);
```

```
125           writer.Flush();
126         }
127         catch(Exception e)
128         {
129           Debug.Log("Error: " + e.Message);
130         }
131       }
132     }
133
134     private void Broadcast(string data, ServerClient c)
135     {
136       List<ServerClient> sc = new List<ServerClient> { c };
137       Broadcast(data, sc);
138     }
139
140     //Lettura
141     private void OnIncomingData(ServerClient c, string data)
142     {
143       string[] aData = data.Split('|');
144
145       switch(aData[0])
146       {
147         case "CWHO":
148           c.clientName = aData[1];
149           c.isHost = (aData[2] == "0") ? false : true;
150           Broadcast("SCNN|" + c.clientName, clients);
151           break;
152         case "CMOV":
153           Broadcast("SMOV|" + aData[1] + "|" + aData[2] + "|"
                + aData[3] + "|" + aData[4], clients);
154           break;
155         case "CMSG":
156           Broadcast("SMSG|" + c.clientName + ": " + aData[1],
                clients);
157           break;
158       }
159     }
160 }
161
162 public class ServerClient
163 {
164     public string clientName;
165     public TcpClient tcp;
166   public bool isHost;
167
```

```
168     public ServerClient(TcpClient tcp)
169     {
170         this.tcp = tcp;
171     }
172 }
```

Listing 4: Client.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Net.Sockets;
5  using System.IO;
6  using System;
7
8  public class Client : MonoBehaviour
9  {
10   public string clientName;
11
12   private bool socketReady;
13   private TcpClient socket;
14   private NetworkStream stream;
15   private StreamWriter writer;
16   private StreamReader reader;
17
18   public bool isHost;
19
20   public List<GameClient> players = new List<GameClient>();
21
22   private void Start()
23   {
24     DontDestroyOnLoad(gameObject);
25   }
26
27   public bool ConnectToServer(string host, int port)
28   {
29     if(socketReady)
30       return false;
31
32     try
33     {
34       socket = new TcpClient(host, port);
35       stream = socket.GetStream();
36       writer = new StreamWriter(stream);
37       reader = new StreamReader(stream);
```

```
38
39        socketReady = true;
40      }
41      catch(Exception e)
42      {
43        Debug.Log("Socket error: " + e.Message);
44      }
45
46      return socketReady;
47    }
48
49    private void Update()
50    {
51      if(socketReady)
52      {
53        if(stream.DataAvailable)
54        {
55          string data = reader.ReadLine();
56          if(data != null)
57            OnIncomingData(data);
58        }
59      }
60    }
61
62    //Send
63    public void Send(string data)
64    {
65      if(!socketReady)
66        return;
67
68      writer.WriteLine(data);
69      writer.Flush();
70    }
71
72    //Read
73    private void OnIncomingData(string data)
74    {
75      string[] aData = data.Split('|');
76
77      switch(aData[0])
78      {
79        case "SWHO":
80          for(int i=1; i < aData.Length - 1; i++)
81          {
82            UserConnected(aData[i], false);
```

```
83          }
84          Send("CWHO|" + clientName + "|" + ((isHost)?1:0).
                ToString());
85          break;
86        case "SCNN":
87          UserConnected(aData[1], false);
88          break;
89        case "SMOV":
90          CheckersBoard.Instance.TryMove(int.Parse(aData[1]),
                int.Parse(aData[2]), int.Parse(aData[3]), int.
                Parse(aData[4]));
91          break;
92        case "SMSG":
93          CheckersBoard.Instance.ChatMessage(aData[1]);
94          break;
95      }
96    }
97
98    private void UserConnected(string name, bool host)
99    {
100     GameClient c = new GameClient();
101     c.name = name;
102
103     players.Add(c);
104
105     if(players.Count == 2)
106       GameManager.Instance.StartGame();
107   }
108
109   private void OnApplicationQuit()
110   {
111     CloseSocket();
112   }
113
114   private void OnDisable()
115   {
116     CloseSocket();
117   }
118
119   private void CloseSocket()
120   {
121     if(!socketReady)
122       return;
123
124     writer.Close();
```

```
125     reader.Close();
126     socket.Close();
127     socketReady = false;
128   }
129
130 }
131
132 public class GameClient
133 {
134   public string name;
135   public bool isHost;
136 }
```

Listing 5: GameManager.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System;
6  using UnityEngine.SceneManagement;
7
8  public class GameManager : MonoBehaviour
9  {
10   public static GameManager Instance { set; get; }
11
12   public GameObject mainMenu;
13   public GameObject serverMenu;
14   public GameObject connectMenu;
15
16   public GameObject serverPrefab;
17   public GameObject clientPrefab;
18
19   public InputField nameInput;
20
21   private void Start()
22   {
23     Instance = this;
24     serverMenu.SetActive(false);
25     connectMenu.SetActive(false);
26     DontDestroyOnLoad(gameObject);
27   }
28
29   public void LocalButton()
30   {
```

```
31      SceneManager.LoadScene("Game");
32    }
33
34    public void ConnectButton()
35    {
36      mainMenu.SetActive(false);
37      connectMenu.SetActive(true);
38    }
39
40    public void HostButton()
41    {
42      try
43      {
44        Server s = Instantiate(serverPrefab).GetComponent<
              Server>();
45        s.Init();
46
47        Client c = Instantiate(clientPrefab).GetComponent<
              Client>();
48        c.clientName = nameInput.text;
49        c.isHost = true;
50        if(c.clientName == "")
51          c.clientName = "Host";
52        c.ConnectToServer("127.0.0.1", 6321);
53
54      }
55      catch(Exception e)
56      {
57        Debug.Log(e.Message);
58      }
59
60      mainMenu.SetActive(false);
61      serverMenu.SetActive(true);
62    }
63
64    public void ConnectToServerButton()
65    {
66      string hostAddress = GameObject.Find("HostInput").
            GetComponent<InputField>().text;
67      if(hostAddress == "")
68        hostAddress = "127.0.0.1";
69
70      try
71      {
72        Client c = Instantiate(clientPrefab).GetComponent<
```

```
            Client>();
73        c.clientName = nameInput.text;
74        if(c.clientName == "")
75          c.clientName = "Client";
76        c.ConnectToServer(hostAddress, 6321);
77        connectMenu.SetActive(false);
78      }
79      catch (Exception e)
80      {
81        Debug.Log(e.Message);
82      }
83    }
84
85    public void BackButton()
86    {
87      mainMenu.SetActive(true);
88      serverMenu.SetActive(false);
89      connectMenu.SetActive(false);
90
91      Server s = FindObjectOfType<Server>();
92      if(s != null)
93        Destroy(s.gameObject);
94
95      Client c = FindObjectOfType<Client>();
96      if(c != null)
97        Destroy(c.gameObject);
98    }
99
100   public void StartGame()
101   {
102     SceneManager.LoadScene("Game");
103   }
104
105 }
```

Listing 6: MemoryTable.cs

```
1 using System;
2 using System.Threading;
3 using System.Threading.Tasks;
4 using System.Collections;
5 using System.Collections.Generic;
6 using UnityEngine;
7 using UnityEngine.UI;
8 using UnityEngine.SceneManagement;
```

```
9
10 public class MemoryTable : MonoBehaviour
11 {
12   public static MemoryTable Instance { set; get; }
13
14   public bool canPlay=true;
15
16   //public int numPlayers=2;
17
18   public int actualPlayer=0;
19
20   public int pairsCount = 20;
21   public int[] scores = new int[2];
22
23   public Card[] deck = new Card[40];
24   public GameObject[] cardsPrefab;
25
26   public Card shownCard=null;
27   public Card shownCard2=null;
28   private float timeFlip;
29
30   public Text score1;
31   public Text score2;
32   public Text pairsLeft;
33   public CanvasGroup winAlert;
34
35   private bool gameIsOver=false;
36   private float winTime;
37
38   private void Start()
39   {
40     Instance = this;
41
42     GenerateDeck();
43   }
44
45   private void Update()
46   {
47     CardClicker();
48     UpdateReflip();
49     if(gameIsOver)
50     {
51       if(Time.time - winTime > 2.0f)
52       {
53         SceneManager.LoadScene("MemoryMenu");
```

```
54        }
55      }
56    }
57
58    private void UpdateReflip()
59    {
60      if(shownCard2)
61      {
62        if(Time.time - timeFlip > 1.5f)
63        {
64          ReFlip(shownCard2, shownCard);
65          if(actualPlayer==0)
66          {
67            actualPlayer=1;
68          }
69          else
70          {
71            actualPlayer=0;
72          }
73          shownCard=null;
74          shownCard2=null;
75          canPlay=true;
76        }
77      }
78    }
79
80    private void CardClicker()
81    {
82      if(!Camera.main)
83      {
84        Debug.Log("Camera main non trovata");
85        return;
86      }
87      if(Input.GetMouseButtonDown(0))
88      {
89        RaycastHit hit;
90        Ray ray = Camera.main.ScreenPointToRay(Input.
              mousePosition);
91        if(Physics.Raycast(ray, out hit, 100.0f))
92        {
93          if(hit.transform!=null)
94          {
95            Card c;
96
97            if(c=hit.transform.GetComponent<Card>())
```

```
98            {
99              if(canPlay){
100               if(!c.isShown)
101                 FlipCard(c);
102             }
103           }
104         }
105       }
106     }
107   }
108
109   private void FlipCard(Card c)
110   {
111     if(!c.isShown)
112     {
113       canPlay=false;
114       c.transform.Rotate(Vector3.right * 180);
115       c.isShown = true;
116     }
117     else
118     {
119       return;
120     }
121     if(shownCard)
122     {
123       if(c.IsAMatch(shownCard))
124       {
125         scores[actualPlayer]++;
126         //Debug.Log("Il giocatore " + (actualPlayer+1) + "
              ha " + scores[actualPlayer] + " punti.");
127         pairsCount--;
128         //Debug.Log("Restano " + pairsCount + " coppie.");
129         UpdateScore();
130         shownCard = null;
131         canPlay=true;
132         if(pairsCount==0)
133           EndGame();
134       }
135       else
136       {
137         shownCard2=c;
138         timeFlip=Time.time;
139       }
140     }
141     else
```

```
142     {
143       shownCard = c;
144       canPlay=true;
145     }
146   }
147
148   private void ReFlip(Card c1, Card c2)
149   {
150     c1.transform.Rotate(Vector3.right * 180);
151     c2.transform.Rotate(Vector3.right * 180);
152   }
153
154   private void EndGame()
155   {
156     if(scores[0]>scores[1])
157     {
158       winAlert.GetComponentInChildren<Text>().text = "Vince
              il player 1!";
159       //Debug.Log("Il giocatore 1 vince!");
160     }
161     else if(scores[0]<scores[1])
162     {
163       winAlert.GetComponentInChildren<Text>().text = "Vince
              il player 2!";
164       //Debug.Log("Il giocatore 2 vince!");
165     }
166     else
167     {
168       winAlert.GetComponentInChildren<Text>().text = "
             Pareggio!";
169     }
170     winAlert.alpha = 1;
171     gameIsOver=true;
172     winTime=Time.time;
173   }
174
175   private void GenerateDeck()
176   {
177     for(int i=0;i<20;i++)
178     {
179       GenerateCard(i);
180     }
181     for(int i=20;i<40;i++)
182     {
183       GenerateCopy(i);
```

```
184     }
185     ShuffleDeck();
186     PositionCards();
187   }
188
189   private void GenerateCard(int i)
190   {
191     GameObject go = Instantiate(cardsPrefab[i]) as
          GameObject;
192     go.transform.SetParent(transform);
193     Card c = go.GetComponent<Card>();
194     c.cardValue = (i%10)+1;
195     c.isHearts = (i%10==i)?false:true;
196     deck[i] = c;
197   }
198
199   private void GenerateCopy(int i)
200   {
201     int i_up=i-20;
202     GameObject go = Instantiate(cardsPrefab[i_up]) as
          GameObject;
203     go.transform.SetParent(transform);
204     Card c = go.GetComponent<Card>();
205     c.cardValue = (i_up%10)+1;
206     c.isHearts = (i_up%10==i_up)?false:true;
207     deck[i] = c;
208   }
209
210   private void ShuffleDeck()
211   {
212     System.Random r = new System.Random();
213     for(int i=0;i<40;i++)
214     {
215       int j = (i+r.Next())%40;
216       ChangeCards(i,j);
217     }
218   }
219
220   private void ChangeCards(int i, int j)
221   {
222     Card c = deck[i];
223     deck[i] = deck[j];
224     deck[j] = c;
225   }
226
```

```
227   private void PositionCards()
228   {
229     float x = 0.24f;
230     float y = 0.2f;
231     int i = 0;
232     foreach(Card c in deck)
233     {
234       c.transform.position = (Vector3.right * x) + (Vector3
              .forward * y) + (Vector3.up * 0.42f);
235       c.transform.Rotate(Vector3.right * 180);
236       x -= 0.07f;
237       i ++;
238       if(i%8==0)
239       {
240         y-=0.1f;
241         x=0.24f;
242       }
243     }
244   }
245
246   private void UpdateScore()
247   {
248     score1.text = scores[0].ToString();
249     score2.text = scores[1].ToString();
250     pairsLeft.text = "Coppie rimaste: " + pairsCount.
          ToString();
251   }
252 }
```

Listing 7: Card.cs

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Card : MonoBehaviour
6 {
7   public bool isShown=false;
8   public int cardValue;
9   public bool isHearts;
10
11  public bool IsAMatch(Card c)
12  {
13    if(!(this.isHearts ^ c.isHearts)&&(this.cardValue == c.
          cardValue))
```

```
14      {
15        return true;
16      }
17      else
18      {
19        this.isShown=false;
20        c.isShown=false;
21        return false;
22      }
23    }
24 }
```

Listing 8: GameManagerMain.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System;
6  using UnityEngine.SceneManagement;
7
8  public class GameManagerMain : MonoBehaviour
9  {
10   public static GameManagerMain Instance { set; get; }
11
12   private void Start()
13   {
14     Instance = this;
15   }
16
17   public void MemoryButton()
18   {
19     SceneManager.LoadScene("MemoryMenu");
20   }
21
22   public void CheckersButton()
23   {
24     SceneManager.LoadScene("Menu");
25   }
26 }
```

Listing 9: GameManagerMemo.cs

```
1 using System.Collections;
```

```
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System;
6  using UnityEngine.SceneManagement;
7
8  public class GameManagerMemo : MonoBehaviour
9  {
10   public static GameManagerMemo Instance { set; get; }
11
12   private void Start()
13   {
14     Instance = this;
15   }
16
17   public void GameButton()
18   {
19     SceneManager.LoadScene("Memory");
20   }
21
22   public void BackButton()
23   {
24     SceneManager.LoadScene("MainMenu");
25   }
26 }
```