Decentralized Multiplayer Puzzle Game

Carlo Di Raddo
carlo.diraddo@studio.unibo.it

Demetrio Andriani
demetrio.andriani@studio.unibo.it

Marzo 2022

Contents

List of Figures

1 Goal/Requirements

L'obiettivo principale è quello di realizzare un puzzle game completamente decentralizzato in modo distribuito sulla rete. I requisiti necessari per poter utilizzare il puzzle game sono quello di avere a disposizione:

1. un computer sul quale mandare in esecuzione l'applicativo (gioco),
2. una connessione internet per poter giocare contemporaneamente con gli altri giocatori.

Come risultato atteso del progetto ci si aspetta che ogni giocatore una volta avviato il gioco possa scegliere tra 3 opzioi:

1. creare una nuova partita.
2. partecipare ad una partita già in corso.
3. decide di uscire dal gioco.

Ogni giocatore partecipante alla stessa sessione di gioco, deve avere la possibilità di risolvere il puzzle in un modello di gioco concorrente (non ci sono turni di gioco). Al momento della risoluzione del puzzle, tutti i giocatori devono vedere sia il puzzle risolto allo stesso modo e contestualmente un messaggio che ne indichi l'avvenuta terminazione con la scritta "Puzzle Complete!".

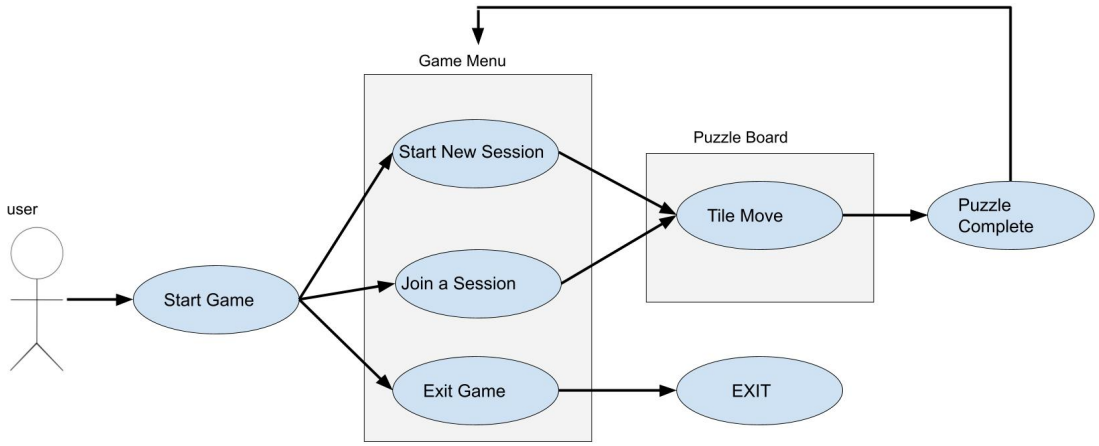

Figure 1: Use Case Diagram

Una possibile domanda relativa ad eventuali problematiche può essere, ad esempio "cosa succede se un nodo(giocatore) crasha?", la riposta è duplice:

- se il nodo non è un nodo host del cluster questo viene contattato ogni tot secondi per vedere se torna disponibile, se non torna disponibile viene tolto dal cluster
- se il nodo è un nodo host si ha una nuova elezione per il prossimo nodo host che prenderà il posto di quello vecchio.

1.1 Scenarios

Il gioco in modo distribuito è similare a quello del gioco da tavolo, infatti lo scopo è quello di ricomporre una figura suddivisa in tasselli (tile), con la finalità di rigenerare l'immagine iniziale in modo chiaro e definito.

Gli utenti hanno la possibilità di giocare in modo asincrono e concorrente (non c'è un modello a turni per la risoluzione del gioco).

Gli utenti per poter usufruire dell'applicazione inizialmente dovranno creare una partita oppure partecipare ad una già esistente. Successiavamente, dopo questa fase, agli utenti sarà possibile visualizzare il puzzle ed interagirvi per poter sostituire le varie tessere(tile) tramite un click con la finalità di risolvere il puzzle.

Quando il puzzle è risolto, la sessione finisce e tutti i partecipanti vedranno a schermo un messaggio che indica la risoluzione del puzzle ("Puzzle Completed!").

1.2 Self-assessment policy

La qualità del software prodotto è stata determinata attraverso i vari feedback dei diversi utenti(giocatori) che hanno avuto la possibilità di poter utilizzare l'applicazione, mentre la sua efficacia è stata stabilita tramite diversi game test.

2 Requirements Analysis

Nello sviluppo del progetto sono presenti diversi requisiti.

I requisiti impliciti sono:

1. il giocatore che vuole creare la partita deve conoscere ed inserire il numero di porta che occuperà
2. il giocatore che vuole partecipare ad una partita, deve conoscere e digitare il numero di porta e l'indirizzo IP della sessione a cui vuole unirsi.

   Per quanto riguarda i requisti non funzionali il sistema fa affidamento ad un framework che gestisce in maniera "autonoma" eventuali errori quali

ad esempio: disconnessioni da parte dei giocatori connessi alla rete, ritardi nell invio e ricezione di messaggi e la loro eventuale non ricezione.

Il progetto si basa sul modello ad attori e in particolare sul framework Akka Cluster. Grazie all'utilizzo del framewrok Akka Cluster e del modello ad attori non ci sono particolari gap da colmare, in quanto queste tecnologie permettono di ottimizzare e di gestire in modo efficente la maggior parte delle problematiche che si presentano sia a livello di rete che a livello applicativo (comunicazione tra i componenti).

3 Design

Il design architetturale ruota intorna al concetto di Cluster e di Actor System. Ogni Actor System rappresenta un nodo (giocatore). Come si evince meglio dalla figura sottostante i giocatori possono comunicare tra di loro grazie all'Akka-Cluster.

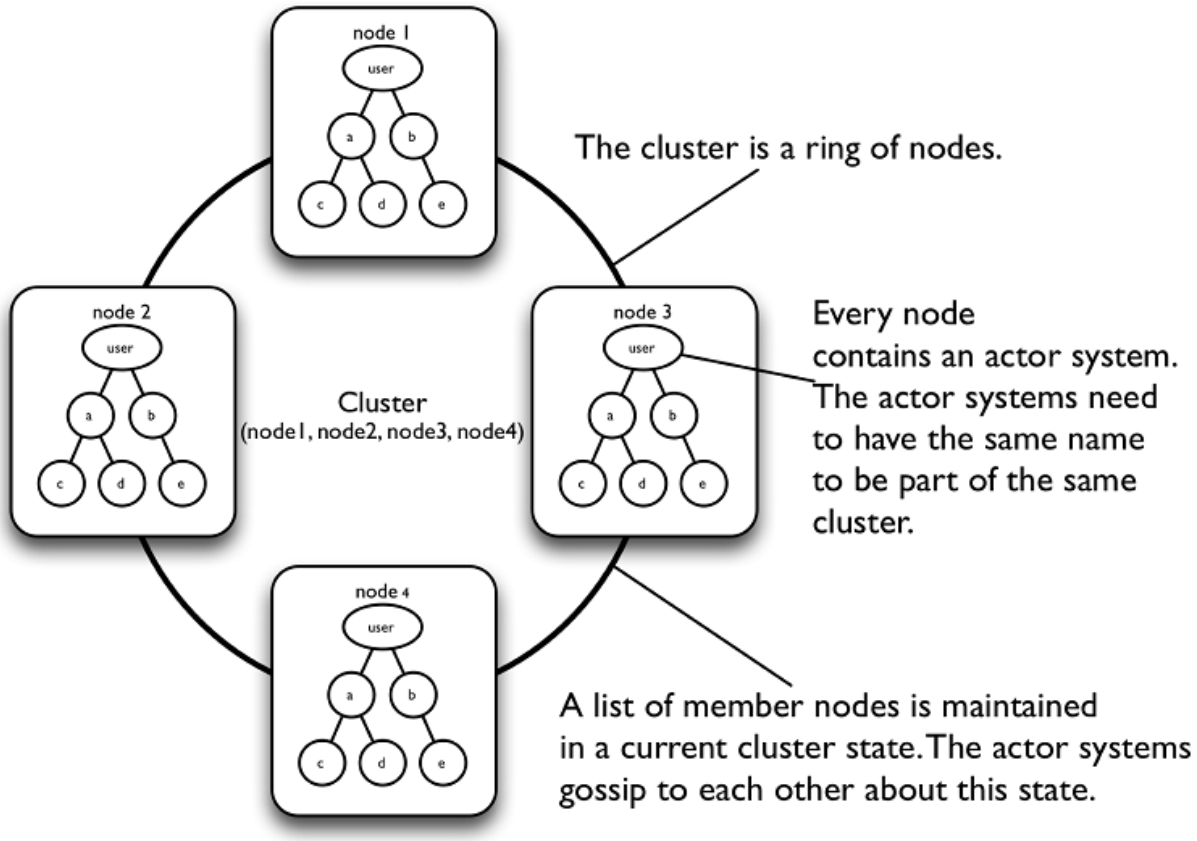

Figure 2: Akka Cluster-System

Grazie a questo Design Acrhitetturale e all'utilizzo della tecnologia Akka si riesce a far fronte a molti problemi che nascono nel distribuito, quali gestione e perdita di pacchetti durante la comunicazione dei nodi, gestione della connessione e dei relativi problemi alla sua perdita e infine grazie a questa architettura è possibile giocare in maniera completamente distribuita e decentralizzata tra giocatori senza necessità di avere un server centrale al quale connettersi.

3.1 Structure

Le entità che costituiscono il sistema sono espresse con le seguenti relazioni che si evincono nel diagramma:

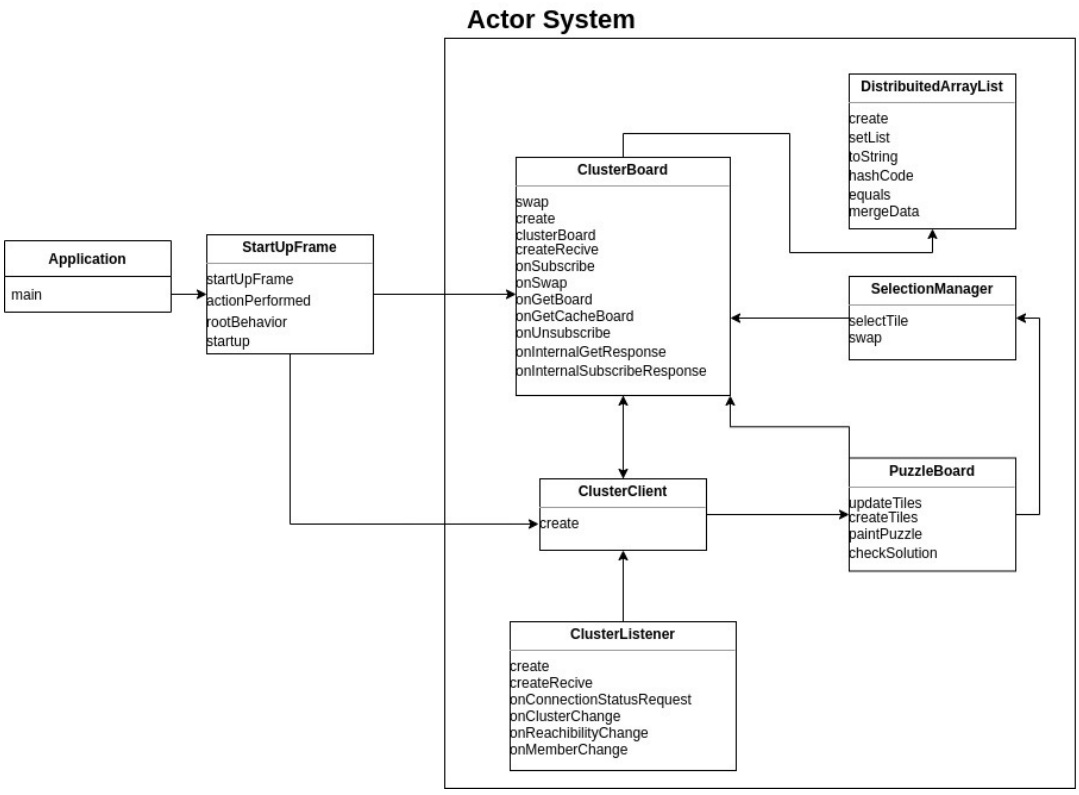

Figure 3: Class Diagram

3.2 Behaviour

Si può notare, tramite il diagramma, il comportamento del sistema:

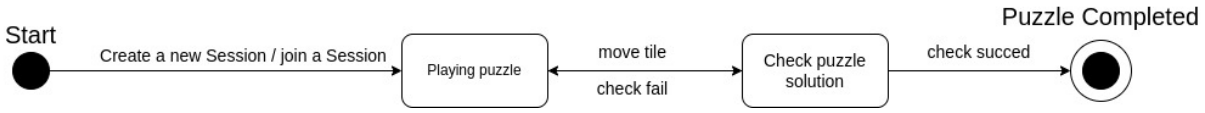

Figure 4: State Diagram

3.3 Interaction

Si può notare, tramite diagramma, come interagiscono tra di loro i vari componenti del sistema a livello locale:

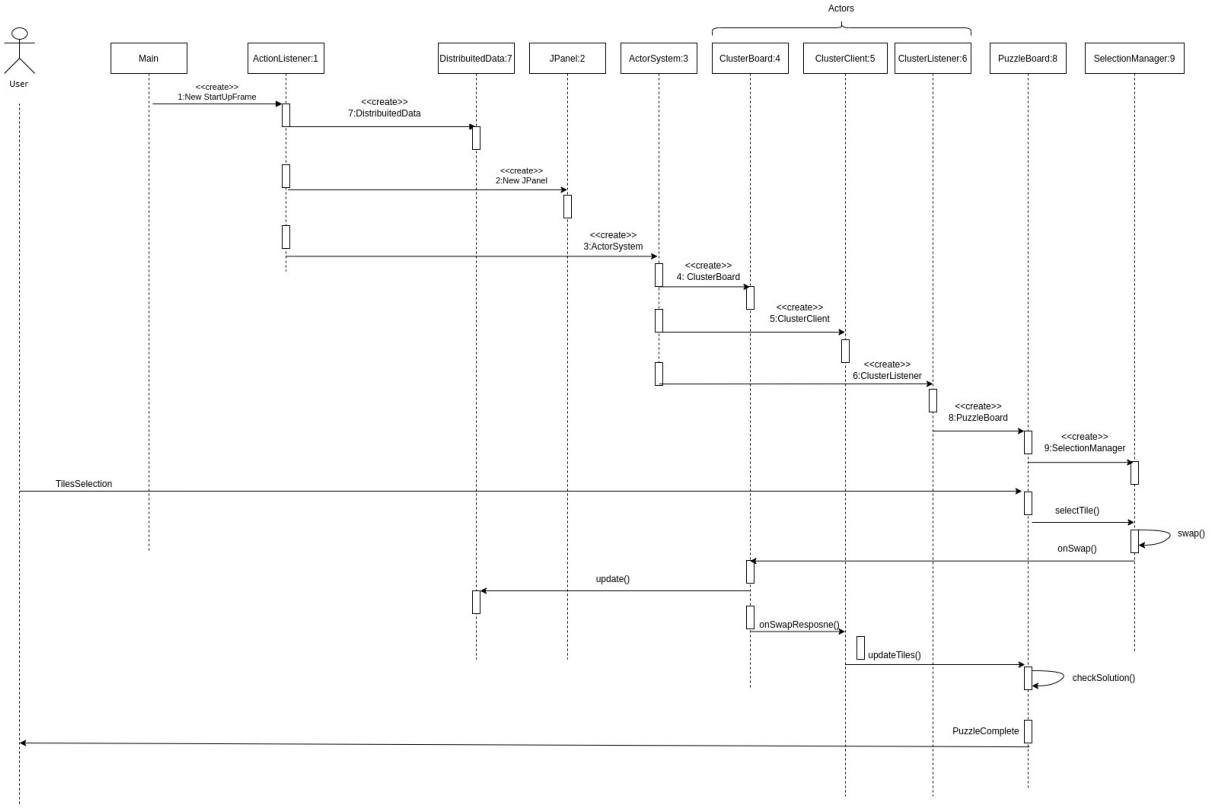

Figure 5: Sequence Diagram

Vista la complessità nel comprendere il flusso delle operazioni del framework Akka cluster, e di conseguenza l'interazione tra i vari Actors e tutto ciò che interagisce con essi, si è voluto descrivere nel modo più chiaro e comprensibile i vari passaggi che il sistema esegue.

L'applicazione inizia creando un ActorSystem, il quale è il primo componente che viene creato. L'actorSystem, tuttavia, ha un rootBehavior che viene eseguito prima della sua attuale creazione. Tale rootBehavior consente di creare 3 attori che sono rispettivamente il ClusterBoard, ClusterClient, ClusterListener.

Essendo la programmazione ad attori uno stile di programmazione asincorno, l'esecuzione dei 3 attori generati dal rootBehavior avviene in maniera totalemente indipendete l'uno dall'altro. Vediamo nel dettaglio di cosa si occupano i 3 attori:

1. Il ClusterBoard svolge la funzione di Replicator. Il Relicator ha la funzione di propagare il cambiamentto verso gli altri replicator del cluster.
2. Il ClusterClient ha il compito di fornire un interfaccia Client per il nodo, cioè il giocatore.
3. Il ClusterListener ha il compito di gestire eventuali failure di rete o eventuali join o left da parte di nodi.

Il ClusterClient possiede un riferimento verso il ClusterBoard e viceversa, grazie a questo riferimento il client può comunicare i cambiamenti che avvengono in locale su un nodo al replicator il quale avrà il compito di prorogare tale cambiamento su tutti i nodi della rete.

Il ClusterClient all'avvio si occupa della creazione della puzzleBoard, la puzzleBoard è composta da oggetti di Tipo Tile che rappresentano i tasselli del puzzle, tali tasselli sono suscettibili ad eventuali azioni da parte dell'utente in quanto implementano un action listener.

L'utente a questo punto una volta selezionato un Tile procede a selezionare il Tile con cui desidera effettuare lo scambio. Questo implica l'invocazione di una funzione di Swap. Questo viene comunicato al Replicator che procede ad istanziare un oggetto di tipo DistribuitedArray (componente che estende la classe AbstractReplicatedData) questo Array è un oggetto che viene "condiviso" tra tutti i nodi presenti sulla rete.

Il DistribuitedArray viene "passato" usando la serializzazione / deserializzazione tramite file Json

Quando il Replicator di un altro nodo riceve il msg contenente il DistribuitedArray procede a creare una nuova puzzleBoard con i tasselli aggiornati e di conseguenza invertiti.

Quando il ClusterBoard viene creato fà un Subscription verso il DistribuitedArray in modo da essere sempre informato sugli eventuali aggiornamenti (vedi scambio Tile). In questo modo un Replicator su un nodo remoto viene constantemente notificato sui cambiamenti del DistribuitedArray e può procede in maniera quasi istantanea alla creazione di una nuova PuzzleBoard con i Tile nelle giuste posizioni.
In questo modo tutti i giocatori potranno visualizzare in contemporanea la puzzleBoard "corretta".

4 Implementation Details

I messaggi che vengono scambiati dai vari nodi sulla rete contengo la distribuitedArrayList, questo oggetto è contiente tutte le informazioni necessarie per far in modo che i nodi riceventi questo oggetto tramite messaggio possano ricostruire in locale la puzzleBoard che è stata aggiornata da un altro giocatore e continuare a giocare su di essa. la teconlogia Json è stata utilizzato per la serializzazione di questo oggetto in modo da renderlo "spedibile" sulla rete sotto forma di messaggio.

5 Self-assessment / Validation

Per vedere il funzionamento della Web Application è stato testato attarverso il play testing, cioè far interagire direttamente i giocatori con il gioco distribuito per vederne il corretto funzionamento ed in caso rilevare possibili criticità e/o errori.

6 Deployment Instructions

L'applicazione si presenta sotto formato java, per eseguirla basta una qualsiasi macchina dotata di JVM su cui mandare in esecuzione l'applicazione che non necissita di nessuna installazione o configurazione particolare.

7 Usage Examples

la Web Application all'avvio presenta un'interfaccia grafica dove è possibile inserire i dati per creare oppure accedere ad una partita.

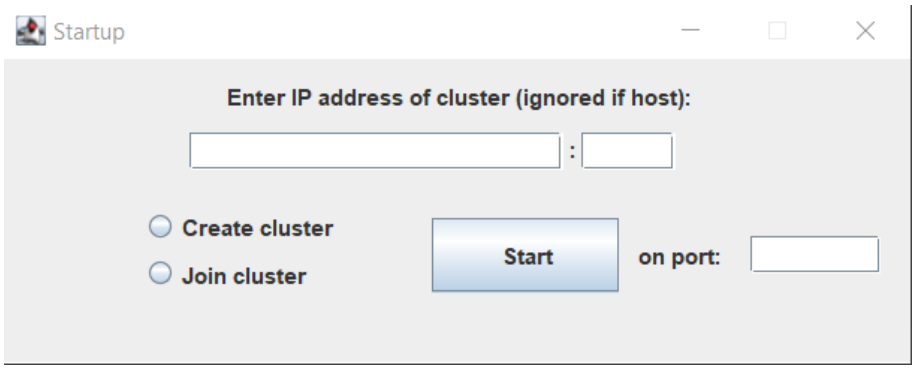

Figure 6: schermata iniziale

All' interno dell'interfaccia sarà possibile:

- indicare se si è host oppure partecipante

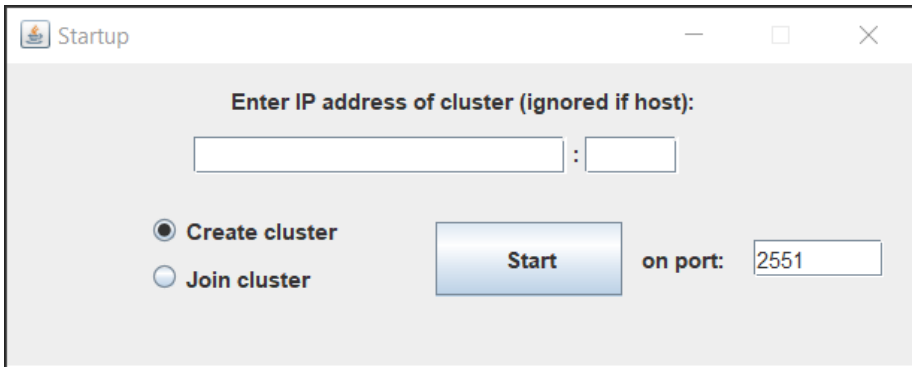

Figure 7: schermata per indicare host

- creare, nel caso di essere host, una partita indicando il numero di porta,

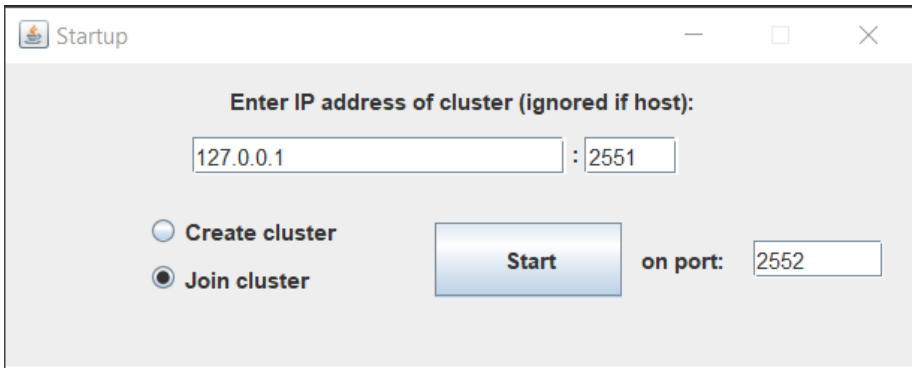

Figure 8: schermata per indicare chi partecipa

- unirsi, nel caso di essere partecipante, ad una già in corso, inserendo indirizzo ip dell'host, numero di porta dell'host e la propria porta (es.2552, l'operazione sarà iterativa per i successivi utenti).

Successivamente l'interfaccia grafica mostra la board con la quale potervi interagire per completare il puzzle in modo distribuito ed in modo cooperativo con tutti gli altri giocatori collegati alla partita, attraverso lo spostamento delle varie tessere(tail).

Figure 9: puzzle board

Quando un giocatore avrà risolto il puzzle, tutti i partecipanti vedranno a

schermo un messaggio che indica il termine del gioco
(“Puzzle Completed!”).

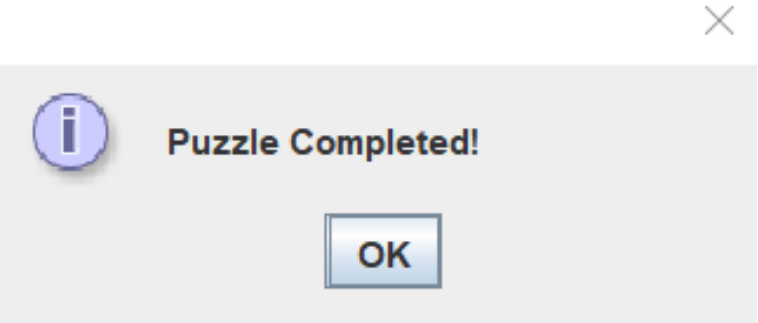

Figure 10: messaggio di risoluzione del puzzle

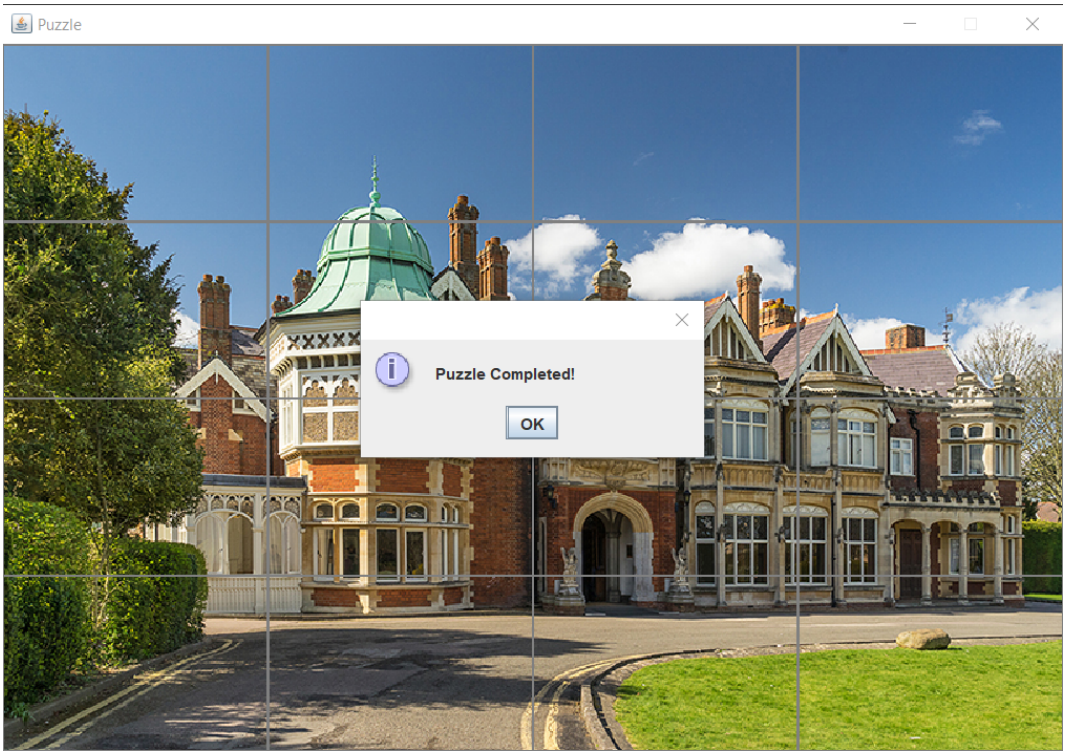

Figure 11: puzzle risolto con relativo messaggio

8 Conclusions

E' stato sviluppato, attraverso il modello ad attori e l'utilizzo del framework Akka Cluster, un puzzle game che riprende le regole dall'omonimo gioco ma con la caratteristica di essere decentralizzato sulla rete e peer-to-peer e non avendo un modello a turni. Tale progetto ci ha dato la possibilità di usare un framework di lavoro molto complesso dal punto di vista di utilizzo e di studio ma che una volta compreso si rivela essere un tool estremamente potente e versatile per la realizzazione di applicazioni distribuite e decentralizzate in quanto permette di gestire in maniera automatica molte problematiche di rete che nascono quando si sviluppano tali applicazioni.

8.1 Future Works

Per quanto riguarda gli sviluppi futuri, ecco un elenco delle possibile implementazioni:

1. dare la possibilita di poter selezionare un'immagine da un gruppo di figure.
2. visualizzare il nome dei giocatori che giocano nella stessa partita.
3. visualizzare il tempo di gioco.
4. dare la possibilità ai giocatori di avere più tile data un immagine.