

Corso di Ingegneria del Software Deliverable di progetto	2021-2022
---	-----------

# “Ingegneria del Software” 2021-2022

**Docente: Prof. Angelo Furfaro**

## <Scale e Serpenti>

<b>Data</b>	<09/06/2022>
<b>Documento</b>	Documento Finale – D3

<b>Nome e Cognome</b>	<b>Matricola</b>	<b>E-mail address</b>
Giorgio Ubbriaco	209899	bbrgrg00h11d086x@studenti.unical.it

1	
---	--

Corso di Ingegneria del Software Deliverable di progetto	2021-2022
---	-----------

## Sommario

List of Challenging/Risky Requirements or Tasks.....	3
A. Stato dell'Arte .....	5
B. Raffinamento dei Requisiti .....	6
A.1 Servizi (con prioritizzazione) .....	6
A.2 Requisiti non Funzionali .....	8
A.3 Scenari d'uso dettagliati .....	8
A.4 Assunzioni .....	22
A.5 Use Case Diagrams.....	23
C. Architettura Software .....	25
C.1 The dynamic view of the software architecture: Sequence Diagram.	25
D. Dati e loro modellazione.....	27
E. Scelte Progettuali (Design Decisions).....	28
F. Progettazione di Basso Livello .....	30
G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs) .....	37
Appendix. Prototype .....	40

---

Corso di Ingegneria del Software Deliverable di progetto	<b>2021-2022</b>
---	------------------

### ***List of Challenging/Risky Requirements or Tasks***

<b>Challenging Task</b>	<b>Date the task is identified</b>	<b>Date the challenge is resolved</b>	<b>Explanation on how the challenge has been managed</b>
Inizializzazione del tabellone e assegnazione delle caselle all'interno della finestra principale	19-05-2022	21-05-2022	Viene inizializzata una matrice di caselle di tipologia "standard" di dimensione pari al numero delle righe e delle colonne specificate nel pannello configurazione. Successivamente, vengono inizializzate delle matrici booleane di controllo per ogni tipologia di casella speciale così da tenere traccia della posizione di ogni casella speciale all'interno del tabellone. Dopodiché, tramite un algoritmo randomico, vengono aggiunte le caselle speciali all'interno del tabellone secondo le regole specificate. Ovviamente, ogni casella avrà una grafica differente rispetto al resto delle caselle. Tanto è vero che all'interno della finestra principale sarà presente una legenda contenente i "caratteri" distintivi di ogni tipologia di casella. Infine, per evitare che il tabellone sia sovrappopolato di caselle speciali, esso, tramite un algoritmo, che prevede un certo numero massimo di caselle speciali per ogni riga della matrice, sfoltisce il tabellone re-assegnando tali caselle come caselle di tipologia standard.
Movimento della pedina da una casella ad un'altra	21-05-2022	25-05-2022	Si prende in considerazione la nuova casella che la pedina deve raggiungere. Se tale casella risulta avere un numero di casella maggiore del numero massimo di casella possibile all'interno del tabellone istanziato, allora tale pedina viene fatta retrocedere di un numero di caselle pari al numero di volte che effettivamente ha superato l'ultima casella possibile del tabellone. Altrimenti, per quanto riguarda il movimento generico della pedina, essa si sposterà in una nuova casella di cui sarà verificata la tipologia per gestirne eventualmente le regole. Inoltre, si tiene conto

<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

			se l'osservatore in fase di configurazione ha impostato la modalità "doppio sei".
Ripristino di una sessione di gioco salvata sul calcolatore	25-05-2022	25-05-2022	Nel pannello di scelta iniziale, l'osservatore può scegliere se inizializzare una nuova sessione di gioco o ripristinare una configurazione di gioco precedentemente salvata sul calcolatore. Pertanto, tramite l'ausilio di una finestra in cui vengono visualizzati file e directory, l'osservatore può selezionare il file di ripristino o al più digitarne il suo nome all'interno della directory che lo contiene. Quindi, viene verificata l'estensione del file (.properties) o l'esistenza del file nel caso in cui l'osservatore abbia digitato il nome di un file inesistente. In caso di errori di apertura, viene opportunamente segnalato all'utente la non apertura del file di ripristino considerato.
Salvataggio della sessione di gioco sul calcolatore	25-05-2022	25-05-2022	Nella finestra principale, l'osservatore può decidere di salvare la configurazione di gioco impostata nel pannello di configurazione. Pertanto, vengono salvati i parametri di configurazione in un'apposita struttura dati. Successivamente tali dati verranno salvati in un file con estensione .properties. Infine, sarà l'utente, tramite una finestra apposita, a decidere in quale directory salvare il file di ripristino.

## A. Stato dell'Arte

Scale e Serpenti è un gioco da tavolo tradizionale simile al gioco dell'oca, che prevede un percorso di forma bustrofedica, cioè che il numero delle caselle aumenta fino al prossimo bordo per poi continuare nel verso opposto secondo un percorso a "nastro". Solitamente sui siti Internet si trova la sua versione standard, cioè quella in cui sono presenti le scale e i serpenti su un tabellone per garantire spostamenti in avanti o a retroso su diverse righe, due dadi e un certo numero di pedine (solitamente che varia tra 4 e 6). Alcune di queste versioni prevedono il conseguimento della vittoria solo tramite una combinazione dei dadi esatta per l'ultima casella e, pertanto, il retrocedere di un certo numero di caselle della pedina nel caso in cui ottenga una combinazione maggiore, oppure versioni in cui una qualsiasi combinazione di dadi, anche maggiore dell'ultima casella, garantisce il conseguimento della vittoria.

Tale progetto, non solo prevede la presenza di scale e serpenti all'interno del tabellone, ma garantisce la presenza di caselle speciali che rendono il gioco più avvincente. Tra queste caselle speciali troviamo: "un solo dado", "pesca una carta", "sosta panchina", "sosta locanda", "premio dadi" e "premio molla". Inoltre, è prevista anche la modalità "doppio sei" che garantisce ad ogni giocatore, che ottiene un doppio sei nel lancio dei due dadi, un altro lancio di dadi per un ulteriore spostamento della pedina. Ovviamente, tutte le caselle speciali e le modalità aggiuntive sono a discrezione dell'osservatore, cioè lui stesso tramite un pannello di configurazione iniziale avente un'interfaccia grafica intuitiva ed user-friendly potrà configurare, secondo i suoi desideri, la nuova simulazione del gioco scegliendo quale tra le specialità aggiungere e quali rimuovere. Inoltre, è possibile salvare una configurazione di gioco scelta nel pannello di configurazione oppure ripristinarne una presente sul calcolatore così da effettuare una nuova simulazione con una configurazione usata in una sessione di gioco precedente. Oltretutto, molto particolare è la possibilità di pesca una carta da un mazzo nel caso in cui il giocatore finisca su una casella di tipologia "pesca una carta".

---

## B. Raffinamento dei Requisiti

### A.1 Servizi (con prioritizzazione)

Il sistema offre i seguenti servizi:

- **creazione di una nuova configurazione di gioco:** l'osservatore, nel "pannello scelte", potrà decidere di creare una nuova configurazione di gioco con cui eseguire una nuova sessione di gioco. Per la precisione, egli potrà scegliere il numero di giocatori (minimo 2 come prevede la modalità standard) da simulare all'interno del tabellone, il numero di righe e colonne (entrambi minimo 10 come prevede la modalità standard di gioco) di cui sarà composto il tabellone all'interno della "finestra principale", il numero di dadi (minimo uno e al più due dadi consentiti) utilizzabili dai giocatori per avanzare di casella, quali tipologie di caselle speciali dovrà contenere il tabellone e, infine, la possibilità di abilitare la modalità doppio sei. Più nel dettaglio, le tipologie di caselle possibili all'interno del tabellone sono: "un solo dado", disposte nelle caselle comprese tra i numeri N-6 ed N-1 (dove N è il numero di casella) che impongono all'utente di avanzare soltanto tramite il lancio di un solo dado, "premio dadi" che permette all'utente di lanciare nuovamente i dadi ed avanzare ulteriormente, "premio molla" che permette all'utente di avanzare nuovamente della stessa combinazione di dadi ottenuta, "sosta panchina" che impone all'utente di restare fermo per un turno nella suddetta casella, "sosta locanda" che impone all'utente di rimanere fermo per tre turni e, infine, "pesca una carta" che permette all'utente di pescare una carta tra le carte disponibili nel mazzo ("dadi", "molla", "panchina", "locanda", "divieto di sosta") aventi uguali effetti delle precedenti caselle sopra-citate. L'ultima carta citata è l'unica carta che può essere pescata dal mazzo (le altre, invece, devono essere inserite in fondo al mazzo) per essere utilizzata successivamente dal giocatore in questione nel caso in cui capiti nelle caselle di tipologia "sosta panchina" e "sosta locanda";
- **salvataggio di una configurazione di gioco:** nella "finestra principale", l'osservatore sarà in grado di salvare la configurazione di gioco precedentemente impostata nel "pannello configurazione". Il file potrà essere salvato con il nome desiderato dall'osservatore stesso. Pertanto, per sessioni di gioco future, l'osservatore sarà in grado di ripristinare tale file ed effettuare le dovute simulazioni di gioco con i parametri di configurazioni salvati sul file in questione;
- **ripristino di una configurazione di gioco:** nel "pannello scelte", l'osservatore potrà decidere di ripristinare una configurazione di gioco salvata precedentemente ed utilizzarla per una simulazione di una nuova sessione di gioco. Per la precisione, egli potrà selezionare il file di ripristino in questione o digitare il suo nome per una più veloce selezione;

- **avanzamento automatico o manuale della simulazione:** nella finestra di configurazione, l'osservatore potrà decidere quale modalità di esecuzione applicare alla simulazione della nuova sessione di gioco. Per la precisione, un avanzamento automatico impone all'utente di visualizzare direttamente le posizioni finali delle pedine ed il vincitore della sessione di gioco. Invece, per quanto riguarda l'avanzamento manuale, l'osservatore potrà osservare ogni turno di gioco effettuato dai giocatori in questione e, quindi, tutti gli spostamenti intermedi delle pedine;
- **traccia delle attività in un'apposita finestra:** per ogni esecuzione (che sia di tipologia automatica o manuale) il sistema tiene traccia delle attività svolte (spostamenti delle pedine, lancio dei dadi, ...) durante la simulazione della sessione di gioco. Ovviamente, un'esecuzione di tipologia automatica farà visualizzare direttamente l'intero storico delle attività svolte durante i turni di gioco fino alla vittoria di uno dei giocatori. Per quanto riguarda l'esecuzione manuale, essa permetterà all'osservatore di visualizzare volta per volta le attività svolte dai giocatori durante ogni singolo turno di gioco;
- **visualizzazione del vincitore della sessione di gioco:** alla fine di ogni simulazione (automatica e manuale) l'osservatore potrà visualizzare il vincitore della simulazione di gioco effettuata. Pertanto, l'osservatore potrà decidere se abbandonare direttamente l'applicazione oppure iniziare una nuova simulazione della sessione di gioco.

Servizio	Importanza	Complessità
<i>creazione di una nuova configurazione di gioco</i>	ALTA	BASSA
<i>salvataggio di una configurazione di gioco</i>	BASSA	BASSA
<i>ripristino di una configurazione di gioco</i>	BASSA	BASSA
<i>avanzamento automatico o manuale della simulazione</i>	ALTA	ALTA
<i>traccia delle attività in un'apposita finestra</i>	ALTA	MEDIA
<i>visualizzazione del vincitore della sessione di gioco</i>	MEDIA	BASSA

### A.2 Requisiti non Funzionali

Il sistema prevede il completamento della simulazione della sessione di gioco sia nella modalità automatica che manuale. Per la precisione, garantisce un vincitore per ogni sessione di gioco inizializzata. Inoltre, è garantito che ogni attività svolta durante la simulazione venga riportata all'interno della finestra apposita.

### A.3 Scenari d'uso dettagliati

#### *Esecuzione del gioco*

Caso d'uso	EsecuzioneAutomatica
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore sceglie come modalità di gioco la modalità automatica
Svolgimento normale	L'osservatore clicca sul pulsante "esegui automaticamente"
Postcondizione	Lo stato della sessione di gioco è aggiornato con i risultati della simulazione
Descrizione	L'osservatore cliccando sul pulsante "esegui automaticamente" visualizza il vincitore della sessione di gioco all'interno della finestra "Vittoria" e le posizioni finali delle pedine all'interno del tabellone contenuto nella finestra "Principale". Ogni turno viene descritto in maniera dettagliata all'interno della finestra "Terminale".

Caso d'uso	EsecuzioneManuale
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore sceglie come modalità di gioco la modalità manuale
Svolgimento normale	L'osservatore clicca sul pulsante "prossimo turno"
Postcondizione	Lo stato della sessione di gioco è aggiornato con i risultati del turno simulato
Descrizione	L'osservatore cliccando sul pulsante "prossimo turno" avanza con la simulazione del gioco osservando il movimento delle pedine verso nuove caselle, all'interno del tabellone contenuto nella finestra "Principale", e la descrizione dettagliata del turno, appena simulato, all'interno della finestra "Terminale".



<p align="center"><b>Corso di Ingegneria del Software</b>  <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
---	------------------

<b>Caso d'uso</b>	<b>Visualizza Vincitore Della Sessione Di Gioco</b>
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	Terminazione della simulazione della sessione di gioco
Svolgimento normale	Viene visualizzata la finestra "Vittoria" contenente il vincitore del gioco
Postcondizione	Il vincitore è visibile nella finestra "Vittoria"
Descrizione	L'osservatore, dopo aver effettuato la simulazione automatica o manuale, visualizza il vincitore della sessione di gioco simulata nella finestra "Vittoria" e può decidere se abbandonare la sessione di gioco corrente o inizializzarne una nuova.

<b>Caso d'uso</b>	<b>Inizializzazione Finestra Vittoria</b>
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	La simulazione della sessione di gioco ha trovato un vincitore
Svolgimento normale	Viene inizializzata la grafica della finestra "Vittoria"
Postcondizione	La finestra "Vittoria" è stata inizializzata in maniera opportuna
Descrizione	Questa operazione consiste nell'inizializzazione grafica della finestra "Vittoria" in modo da consentire la visualizzazione del vincitore della sessione di gioco e l'abbandono della sessione corrente o l'inizializzazione di una nuova sessione di gioco.

Corso di Ingegneria del Software Deliverable di progetto	2021-2022
---	-----------

*Salvataggio della configurazione del gioco*

Caso d'uso	SalvataggioDellaConfigurazioneDelGioco
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	La configurazione della sessione di gioco è completata
Svolgimento normale	L'osservatore clicca sul pulsante "Salva". Viene visualizzata una finestra per consentire il salvataggio dei parametri di configurazione in una directory che desidera
Postcondizione	I parametri della configurazione della sessione di gioco corrente sono salvati in un file (con estensione .properties) nella directory scelta dall'osservatore
Descrizione	L'osservatore cliccando sul pulsante "Salva" fa comparire una finestra che gli consente di salvare nella directory da lui desiderata e con il nome da lui desiderato i parametri della configurazione della sessione di gioco corrente. Pertanto, dopo aver deciso la directory, l'osservatore preme il pulsante per confermare ed il sistema salva tutti i parametri fondamentali all'interno del file. Per la precisione verrà salvato il numero di giocatori configurato, il numero di righe e colonne scelto, il numero di dadi e se è stata configurata la modalità doppio sei e, inoltre, tutte le caselle speciali che l'osservatore ha scelto per la configurazione della sessione di gioco.

Caso d'uso	VerificaPresenzaDelFile
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha cliccato sul pulsante "Salva"
Svolgimento normale	Viene verificato che il nome con cui intende salvare la configurazione della sessione di gioco non sia già presente nell'attuale directory da lui scelta
Postcondizione	Il nome è verificato per il salvataggio della configurazione della sessione di gioco
Descrizione	Questa operazione consente di verificare se il nome con cui l'osservatore intende salvare la configurazione della sessione di gioco non sia stato già utilizzato precedentemente per un altro file di salvataggio che ha la stessa estensione .properties.

Corso di Ingegneria del Software Deliverable di progetto	2021-2022
---	-----------

*Scelta della sessione di gioco*

Caso d'uso	NuovaSessioneDiGioco
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore richiede una nuova sessione del gioco Scale e Serpenti
Svolgimento normale	L'osservatore clicca sul pulsante "Nuovo" per configurare una nuova sessione di gioco
Postcondizione	Una nuova sessione di gioco è stata inizializzata per la configurazione corrispondente
Descrizione	L'osservatore cliccando sul pulsante "Nuovo" determina la chiusura della finestra delle scelte per inizializzare una nuova finestra: il pannello di configurazione. Tramite questo pannello, l'osservatore potrà configurare secondo i suoi desideri e secondo determinati limiti la nuova sessione di gioco (guarda i casi d'uso SceltaDiAggiungereCaselleUnSoloDado, SceltaDiAggiungereCaselleSosta, SceltaDiAggiungereCasellePremio, SceltaDiAggiungereCasellePescaUnaCarta, SceltaDiAggiungereScale, SceltaDiAggiungereSerpenti, SceltaNumeroDiGiocatori, SceltaNumeroRighe, SceltaNumeroColonne, SceltaNumeroDadi, SceltaModalitaDoppioSei ).

Caso d'uso	RipristinaSessioneDiGioco
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore richiede il ripristino di una configurazione di una sessione di gioco salvata precedentemente sul calcolatore
Svolgimento normale	L'osservatore clicca sul pulsante "Riprendi" e compare una finestra in cui vengono mostrate le directory affinché l'osservatore possa selezionare il file di ripristino
Descrizione	L'osservatore cliccando sul pulsante "Riprendi" determina la comparsa di una nuova finestra che fa visualizzare graficamente directory e file presenti sul calcolatore. Pertanto, l'osservatore decide quale file di ripristino utilizzare cliccando con il mouse o digitando il nome del file di ripristino in un'apposita casella di testo.

<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

<b>Caso d'uso</b>	<b>VerificaPresenzaDiFileDiRipristino</b>
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha selezionato il file di ripristino o ha digitato il nome del file nell'apposita casella di testo
Svolgimento normale	Il sistema verifica l'estensione del file selezionato o l'esistenza del file di cui è stato digitato il nome nella casella di testo e verificandone in caso l'estensione
Postcondizione	Il sistema ha ripristinato la configurazione della sessione di gioco selezionata dall'osservatore
Descrizione	Il sistema verifica che il file di ripristino selezionato dall'osservatore abbia un'estensione di tipo .properties (l'estensione che il sistema supporta per salvataggio e ripristino di configurazione di gioco). Nel caso in cui, invece, l'osservatore ha digitato il nome del file da ripristinare nell'apposita casella di testo, dopo essersi posizionato nella directory in cui pensa sia contenuto tale file di ripristino, il sistema verifica l'esistenza di tale file di ripristino. Se il file esiste allora viene verificato che tale file abbia la corretta estensione .properties. Pertanto, viene ripristinata la configurazione di gioco presente nel file di ripristino. Nel caso in cui il file non dovesse esistere oppure l'estensione non è valida, allora il sistema segnala all'osservatore l'errore nel ripristino di tale configurazione di gioco.

<b>Caso d'uso</b>	<b>InizializzazioneSessioneDiGioco</b>
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha configurato in maniera opportuna la nuova sessione di gioco o ne ha ripristinato una precedentemente salvata sul calcolatore
Svolgimento normale	La finestra "Principale" e la finestra "Terminale" vengono inizializzate secondo i parametri configurati nel pannello di configurazione
Svolgimento alternativo	La finestra "Principale" e la finestra "Terminale" vengono inizializzate secondo i parametri salvati sul file di ripristino presente sul calcolatore
Postcondizione	La finestra "Principale" e la finestra "Terminale" sono state inizializzate
Descrizione	Questa operazione consiste nell'inizializzazione della finestra "Principale" e della finestra "Terminale" secondo i parametri configurati nel pannello configurazione oppure secondo i parametri salvati su un file presente sul calcolatore in riferimento ad una configurazione inizializzata in una sessione di gioco precedente.

Corso di Ingegneria del Software Deliverable di progetto	2021-2022
---	-----------

*Configurazione del gioco*

Caso d'uso	SceltaDiAggiungereCaselleUnSoloDado
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore spunta la casella per inserire le caselle di tipologia "un solo dado" all'interno del tabellone
Postcondizione	Le caselle di tipologia "un solo dado" sono state aggiunte al tabellone e sono visibili all'osservatore
Descrizione	Questo caso d'uso descrive la possibile aggiunta da parte dell'osservatore delle caselle "un solo dado" all'interno del tabellone. Esse verranno aggiunte nelle posizioni comprese tra N-6 ed N-1 (dove N è il numero di caselle del tabellone della sessione di gioco configurata) e verranno visualizzate graficamente dall'osservatore. Esse saranno di colore grigio. I giocatori che capiteranno su tali caselle, nei prossimi turni, dovranno avanzare lanciando un solo dado.

Caso d'uso	SceltaDiAggiungereCaselleSosta
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore spunta la casella per inserire le caselle di tipologia "sosta" all'interno del tabellone
Postcondizione	Le caselle di tipologia "sosta" sono state aggiunte al tabellone e sono visibili all'osservatore
Descrizione	Questo caso d'uso descrive la possibile aggiunta da parte dell'osservatore delle caselle "sosta" all'interno del tabellone. Esse verranno aggiunte in maniera randomica all'interno del tabellone del gioco e verranno visualizzate graficamente dall'osservatore. Esse saranno di colore rosso. Tali caselle sono di due tipologie e hanno due funzioni differenti durante l'esecuzione del gioco. Pertanto, le caselle di tipologia "sosta panchina" verranno visualizzate con il colore rosso chiaro e faranno stare ferma per un turno la pedina che arriva su di essa. Mentre, le caselle di tipologia "sosta locanda" verranno visualizzate con il colore rosso scuro e faranno stare ferma per tre turni la pedina che arriva su di essa.

<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

<b>Caso d'uso</b>	<b>SceltaDiAggiungereCasellePremio</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore spunta la casella per inserire le caselle di tipologia "premio" all'interno del tabellone
Postcondizione	Le caselle di tipologia "premio" sono state aggiunte al tabellone e sono visibili all'osservatore
Descrizione	Questo caso d'uso descrive la possibile aggiunta da parte dell'osservatore delle caselle "premio" all'interno del tabellone. Esse verranno aggiunte in maniera randomica all'interno del tabellone del gioco e verranno visualizzate graficamente dall'osservatore. Esse saranno di colore verde. Tali caselle sono di due tipologie e hanno due funzioni differenti durante l'esecuzione del gioco. Pertanto, le caselle di tipologia "premio dadi" verranno visualizzate con il colore verde scuro e faranno rilanciare i dadi per una seconda volta al giocatore che arriva su di essa. Mentre, le caselle di tipologia "premio molla" verranno visualizzate con il colore verde chiaro e faranno avanzare la pedina che arriva su di essa della stessa combinazione dei dadi ottenuta precedentemente.

<b>Caso d'uso</b>	<b>SceltaDiAggiungereCasellePescaUnaCarta</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore spunta la casella per inserire le caselle di tipologia "pesca una carta" all'interno del tabellone
Postcondizione	Le caselle di tipologia "pesca una carta" sono state aggiunte al tabellone e sono visibili all'osservatore
Descrizione	Questo caso d'uso descrive la possibile aggiunta da parte dell'osservatore delle caselle "pesca una carta" all'interno del tabellone. Esse verranno aggiunte in maniera randomica all'interno del tabellone del gioco e verranno visualizzate graficamente dall'osservatore. Esse saranno di colore giallo. Tali caselle permetteranno alla pedina che arriva su di esse di pescare una carta. Esistono per la precisione cinque tipologie di carte: "panchina", "locanda", "dadi", "molla" e "divieto di sosta". Le prime quattro citate svolgeranno la stessa funzione delle omonime tipologie di caselle e dovranno essere "consumate" appena pescate mentre l'ultima citata potrà essere conservato dal giocatore che l'ha pescata per essere utilizzata in un secondo momento (nel caso in cui capiti su una casella di tipologia "sosta panchina" o "sosta locanda").

<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

<b>Caso d'uso</b>	<b>SceltaDiAggiungereScale</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore spunta la casella per inserire le "scale" all'interno del tabellone
Postcondizione	Le "scale" sono state aggiunte al tabellone e sono visibili all'osservatore
Descrizione	Questo caso d'uso descrive la possibile aggiunta da parte dell'osservatore delle scale all'interno del tabellone. Esse verranno aggiunte in maniera randomica all'interno del tabellone del gioco e verranno visualizzate graficamente dall'osservatore. Esse saranno di colore blu. Per la precisione, un giocatore che arriva tramite una combinazione dei dadi sulla casella contenente i "piedi" (la "coda") di una scala si sposterà sulla casella contenente la "testa" della scala senza ulteriore lancio di dadi.

<b>Caso d'uso</b>	<b>SceltaDiAggiungereSerpenti</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore spunta la casella per inserire i "serpenti" all'interno del tabellone
Postcondizione	I "serpenti" sono state aggiunti al tabellone e sono visibili all'osservatore
Descrizione	Questo caso d'uso descrive la possibile aggiunta da parte dell'osservatore dei serpenti all'interno del tabellone. Essi verranno aggiunti in maniera randomica all'interno del tabellone del gioco e verranno visualizzati graficamente dall'osservatore. Essi saranno di colore marrone. Per la precisione, un giocatore che arriva tramite una combinazione dei dadi sulla casella contenente la "testa" di un serpente dovrà spostarsi sulla casella contenente la "coda" del serpente e, pertanto, ritornando indietro di un certo numero di caselle.

<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

<b>Caso d'uso</b>	<b>SceltaNumeroDiGiocatori</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore inserisce il numero dei giocatori che competeranno nella nuova sessione di gioco
Postcondizione	Lo stato del sistema è stato aggiornato con il numero dei giocatori inserito dall'osservatore
Descrizione	Questo caso d'uso descrive la configurazione da parte dell'osservatore del numero dei giocatori che dovranno competere all'interno della nuova sessione di gioco. Per la precisione, è consentito almeno un numero di giocatori pari a 2. Ovviamente, non inserire alcun numero oppure inserire caratteri comporta la comparsa di una finestra di errore (vedi casi d'uso VerificaInputNumerico e VerificaScelta).

<b>Caso d'uso</b>	<b>SceltaNumeroRighe</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore inserisce il numero delle righe che dovrà contenere il tabellone per la nuova sessione di gioco
Postcondizione	Il tabellone è stato inizializzato con il numero di righe inserito dall'osservatore
Descrizione	Questo caso d'uso descrive la configurazione da parte dell'osservatore del numero di righe di cui dovrà essere composto il tabellone per la nuova sessione di gioco. Per la precisione, è consentito un numero di righe maggiore o uguale a 10 (corrisponde al numero di righe del tabellone di Scale e Serpenti standard). Ovviamente, non inserire alcun numero oppure inserire caratteri comporta la comparsa di una finestra di errore (vedi casi d'uso VerificaInputNumerico e VerificaScelta).



<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

<b>Caso d'uso</b>	<b>SceltaNumeroColonne</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore inserisce il numero di colonne che dovrà contenere il tabellone per la nuova sessione di gioco
Postcondizione	Il tabellone è stato inizializzato con il numero di colonne inserito dall'osservatore
Descrizione	Questo caso d'uso descrive la configurazione da parte dell'osservatore del numero di colonne di cui dovrà essere composto il tabellone per la nuova sessione di gioco. Per la precisione, è consentito un numero di colonne maggiore o uguale a 10 (corrisponde al numero di colonne del tabellone di Scale e Serpenti standard). Ovviamente, non inserire alcun numero oppure inserire caratteri comporta la comparsa di una finestra di errore (vedi casi d'uso VerificaInputNumerico e VerificaScelta).

<b>Caso d'uso</b>	<b>SceltaNumeroDadi</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore inserisce il numero di dadi con cui verrà simulato ogni turno di ogni giocatore durante la sessione di gioco
Postcondizione	Lo stato del sistema è stato aggiornato con il numero di dadi inserito dall'osservatore
Descrizione	Questo caso d'uso descrive la configurazione da parte dell'osservatore del numero di dadi che ogni giocatore potrà lanciare durante il proprio turno di gioco. Per la precisione, è consentito un numero di dadi pari a 1 o pari a 2 (corrisponde al numero di dadi standard). Ovviamente, non inserire alcun numero oppure inserire caratteri comporta la comparsa di una finestra di errore (vedi casi d'uso VerificaInputNumerico e VerificaScelta).

<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

<b>Caso d'uso</b>	<b>SceltaModalitaDoppioSei</b>
Tipo	Primario (sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha richiesto la configurazione di una nuova sessione di gioco
Svolgimento normale	L'osservatore spunta la casella "modalità doppio sei" per configurare la nuova sessione di gioco con tale modalità
Postcondizione	Il sistema è configurato con la "modalità doppio sei"
Descrizione	Questo caso d'uso descrive la configurazione da parte dell'osservatore della "modalità doppio sei" di cui potranno usufruire i giocatori durante la simulazione della nuova sessione di gioco. Per la precisione, la modalità doppio sei permette ai giocatori che ottengono una combinazione di dadi 6 6 (per l'appunto doppio sei) di rilanciare i dadi e muoversi nuovamente verso un'altra casella. L'ulteriore lancio dei dadi è garantito dopo che la pedina ha effettuato tutti i movimenti tra le caselle e dopo che le regole del gioco sono state gestite. Ovviamente, configurare la nuova sessione di gioco con la modalità doppio sei e selezionando non un numero di dadi pari a 1 non ha senso poiché tale modalità verrebbe ignorata.

<b>Caso d'uso</b>	<b>VerificaInputNumerico</b>
Tipo	Secondario (non sollecitato dall'osservatore)
Precondizione	L'osservatore ha terminato la configurazione della nuova sessione di gioco
Svolgimento normale	Il sistema verifica che gli input numerici (numero di giocatori, numero di righe, numero di colonne, numero di dadi) inseriti dall'osservatore siano effettivamente input numerici e non lettere
Descrizione	Questa operazione consiste nella verifica degli input immessi dall'osservatore all'interno del pannello di configurazione per la nuova sessione di gioco. Per la precisione, verrà controllato che il numero di giocatori, il numero di righe, il numero di colonne ed il numero di dadi inserito dall'osservatore siano effettivamente input numeri che non contengono alcuna lettera.

<p align="center"><b>Corso di Ingegneria del Software</b> <b>Deliverable di progetto</b></p>	<b>2021-2022</b>
--	------------------

<b>Caso d'uso</b>	<b>VerificaScelte</b>
Tipo	Secondario (non sollecitato dall'osservatore)
Precondizione	L'osservatore ha terminato la configurazione della nuova sessione di gioco
Svolgimento normale	Il sistema verifica che l'osservatore ha configurato in maniera opportuna la nuova sessione di gioco
Postcondizione	La nuova sessione di gioco è stata inizializzata secondo i parametri configurati dall'osservatore
Descrizione	Questa operazione consiste nella verifica di una corretta configurazione da parte dell'osservatore. Una configurazione risulta essere corretta se l'osservatore ha inserito correttamente i settaggi, cioè ha inserito effettivamente numeri dove erano consentiti input numerici e non lettere o input vuoti.

<b>Caso d'uso</b>	<b>InizializzazioneFinestraPrincipale</b>
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha inizializzato una nuova sessione di gioco oppure ha ripristinato una sessione di gioco precedente
Svolgimento normale	Viene inizializzata graficamente la finestra "Principale"
Postcondizione	La finestra "Principale" è visibile all'osservatore ed è stata configurata in maniera opportuna
Descrizione	Questa operazione permette l'inizializzazione e la visualizzazione della finestra "Principale"

<b>Caso d'uso</b>	<b>InizializzazioneFinestraTerminale</b>
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha inizializzato una nuova sessione di gioco oppure ha ripristinato una sessione di gioco precedente
Svolgimento normale	Viene inizializzata graficamente la finestra "Terminale"
Postcondizione	La finestra "Terminale" è visibile all'osservatore ed è stata configurata in maniera opportuna
Descrizione	Questa operazione permette l'inizializzazione e la visualizzazione della finestra "Terminale"

Corso di Ingegneria del Software Deliverable di progetto	2021-2022
---	-----------

Caso d'uso	InizializzazioneFinestra
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	L'osservatore ha intenzione di inizializzare la sessione di gioco
Svolgimento normale	Inizializza la grafica generale della finestra in questione e implementa l'operazione di chiusura della finestra
Postcondizione	La finestra è stata inizializzata in maniera opportuna ed è visibile all'osservatore
Descrizione	Questa operazione permette l'inizializzazione grafica di una finestra e l'implementazione della possibile chiusura della stessa.

Caso d'uso	InizializzazioneLayoutFinestra
Tipo	Secondario (non sollecitato direttamente dall'osservatore)
Precondizione	È stata invocata l'inizializzazione della finestra
Svolgimento normale	Ogni layout viene inizializzato in maniera opportuna
Postcondizione	I layout della finestra sono stati inizializzati e sono visibili all'osservatore
Descrizione	Questa operazione permette l'inizializzazione grafica e logica di ogni layout (North, Center, South, East, West) di ogni finestra. Per la precisione ogni layout sarà provvisto di componenti user-friendly e interattivi adatti per il tipo di finestra inizializzata.

Caso d'uso	InizializzazioneLayoutNORTH
Tipo	Secondario (non sollecitato dall'osservatore)
Precondizione	È stata invocata l'inizializzazione del layout della finestra
Svolgimento normale	Il layout viene inizializzato graficamente e aggiunto nel layout North della finestra
Postcondizione	Il layout è visibile graficamente all'interno della finestra e, inoltre, è interattivo
Descrizione	Questa operazione permette l'inizializzazione grafica e logica del layout North di ogni finestra. Ad esempio, nel caso di una finestra principale sarà presente il pulsante di esecuzione.

Corso di Ingegneria del Software Deliverable di progetto	2021-2022
---	-----------

Caso d'uso	InizializzazioneLayoutCENTER
Tipo	Secondario (non sollecitato dall'osservatore)
Precondizione	È stata invocata l'inizializzazione del layout della finestra
Svolgimento normale	Il layout viene inizializzato graficamente e aggiunto nel layout Center della finestra
Postcondizione	Il layout è visibile graficamente all'interno della finestra e, inoltre, è interattivo
Descrizione	Questa operazione permette l'inizializzazione grafica e logica del layout Center di ogni finestra. Ad esempio, nel caso di una finestra principale sarà presente il tabellone su cui si sposteranno le pedine.

Caso d'uso	InizializzazioneLayoutSOUTH
Tipo	Secondario (non sollecitato dall'osservatore)
Precondizione	È stata invocata l'inizializzazione del layout della finestra
Svolgimento normale	Il layout viene inizializzato graficamente e aggiunto nel layout South della finestra
Postcondizione	Il layout è visibile graficamente all'interno della finestra e, inoltre, è interattivo
Descrizione	Questa operazione permette l'inizializzazione grafica e logica del layout South di ogni finestra. Ad esempio, nel caso di una finestra principale sarà presente il pannello della legenda relativo alle tipologie delle caselle.

Caso d'uso	InizializzazioneLayoutEAST
Tipo	Secondario (non sollecitato dall'osservatore)
Precondizione	È stata invocata l'inizializzazione del layout della finestra
Svolgimento normale	Il layout viene inizializzato graficamente e aggiunto nel layout East della finestra
Postcondizione	Il layout è visibile graficamente all'interno della finestra e, inoltre, è interattivo
Descrizione	Questa operazione permette l'inizializzazione grafica e logica del layout East di ogni finestra.

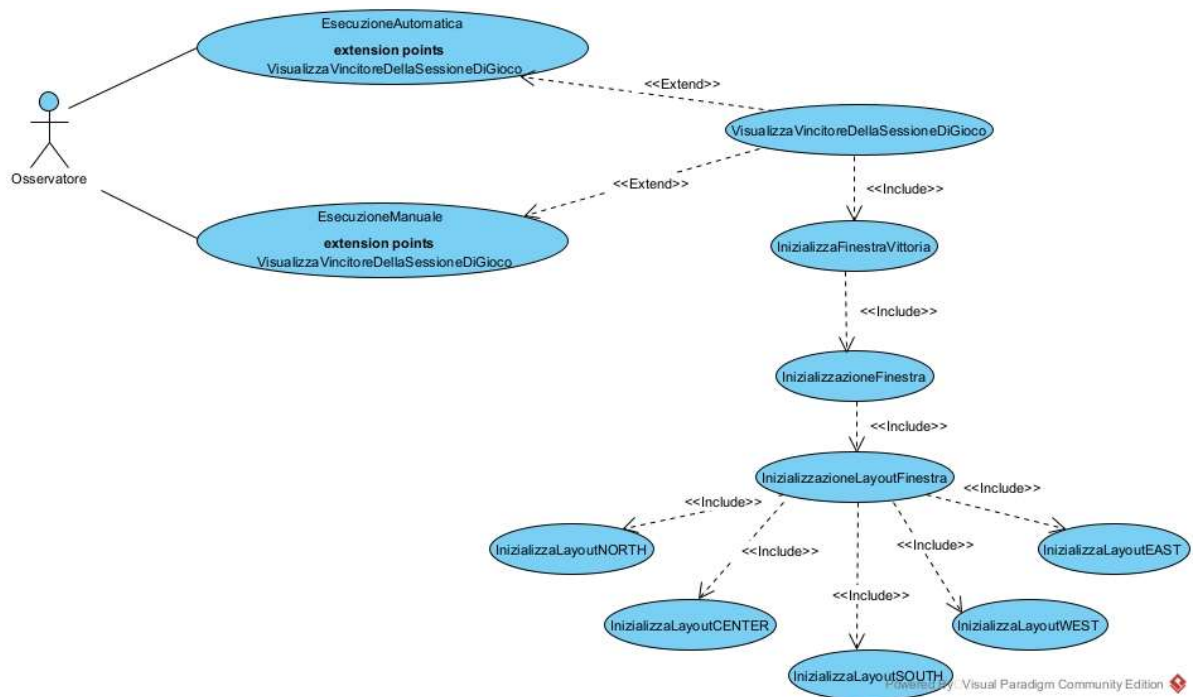
Caso d'uso	InizializzazioneLayoutWEST
Tipo	Secondario (non sollecitato dall'osservatore)
Precondizione	È stata invocata l'inizializzazione del layout della finestra
Svolgimento normale	Il layout viene inizializzato graficamente e aggiunto nel layout West della finestra
Postcondizione	Il layout è visibile graficamente all'interno della finestra e, inoltre, è interattivo
Descrizione	Questa operazione permette l'inizializzazione grafica e logica del layout West di ogni finestra.

#### **A.4 Assunzioni**

Le assunzioni adottate a livello progettuale sono diverse. Per quanto riguarda l'inizializzazione del tabellone, si cerca di assegnare almeno una tipologia di casella speciale (se si riesce) per ogni riga. Tutto ciò grazie alla generazione di un numero random di casella compreso tra l'estremo numero sinistro di casella e l'estremo numero destro di casella presente in quella determinata riga. Se tale numero random rispecchia un numero di una casella in cui è già presente un'altra casella speciale, allora tale casella speciale, che si sta cercando di assegnare, non viene assegnata per quella determinata riga. Tale assunzione è valida per ogni tipologia di casella speciale presente all'interno del gioco. Ovviamente per tutte tranne le caselle di tipologia "un solo dado" le quali vengono assegnate in maniera predefinita nei numeri di caselle compresi tra N-6 ed N-1 (dove N è il numero di casella). Analogamente anche per l'assegnazione delle scale e dei serpenti all'interno del gioco dei quali però si cerca di assegnare prima la testa e poi, se si è riusciti ad assegnarla, si passa all'assegnazione della coda nelle righe sottostanti rimanenti del tabellone. Un'altra assunzione è relativa all'estensione del file prevista come .properties (per maggiori dettagli leggere la sezione "Dati e loro Modellazione").

## A.5 Use Case Diagrams

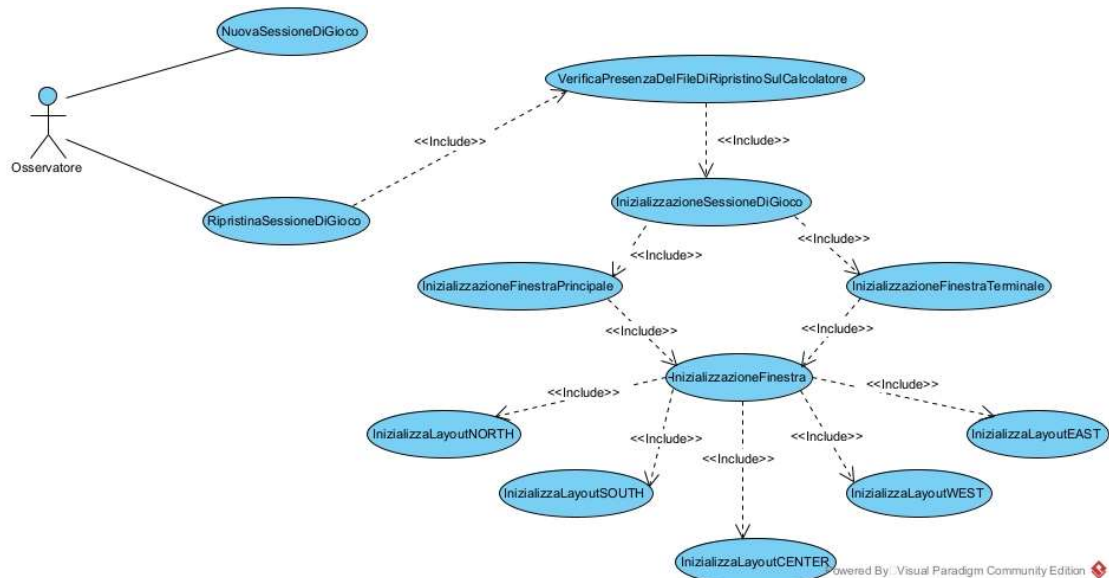
### Esecuzione del gioco



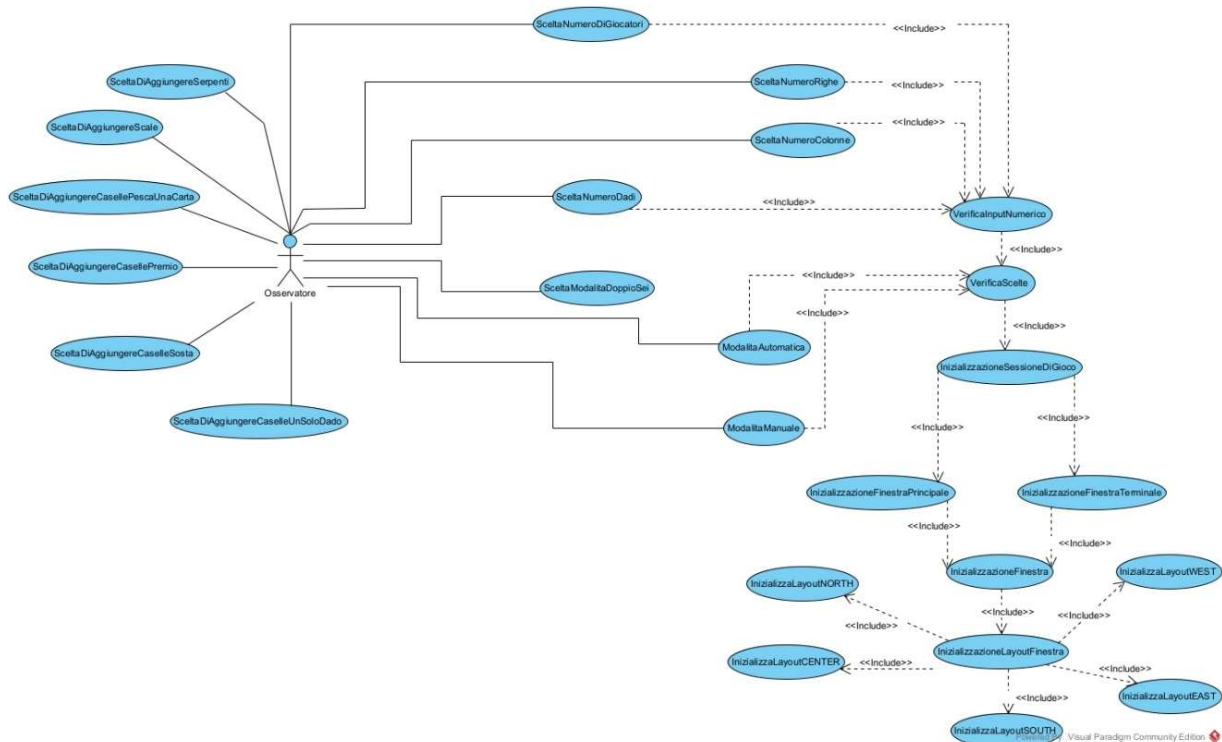
### Salvataggio della configurazione del gioco



*Scelta della sessione di gioco*



*Configurazione del gioco*



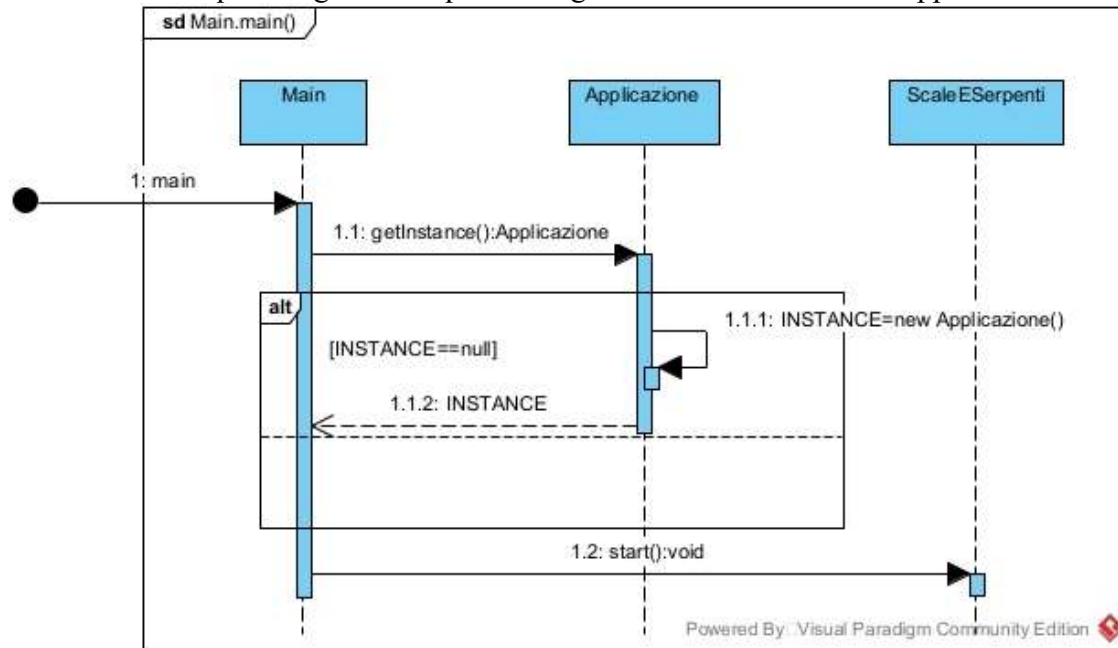


## C. Architettura Software

### C.1 The dynamic view of the software architecture: Sequence Diagram

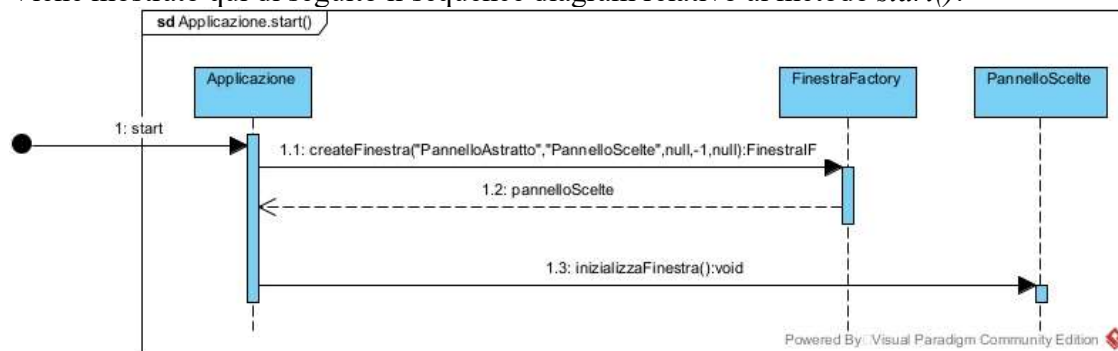
#### Main Sequence Diagram

Viene mostrato qui di seguito il sequence diagram relativo al main dell'applicazione:



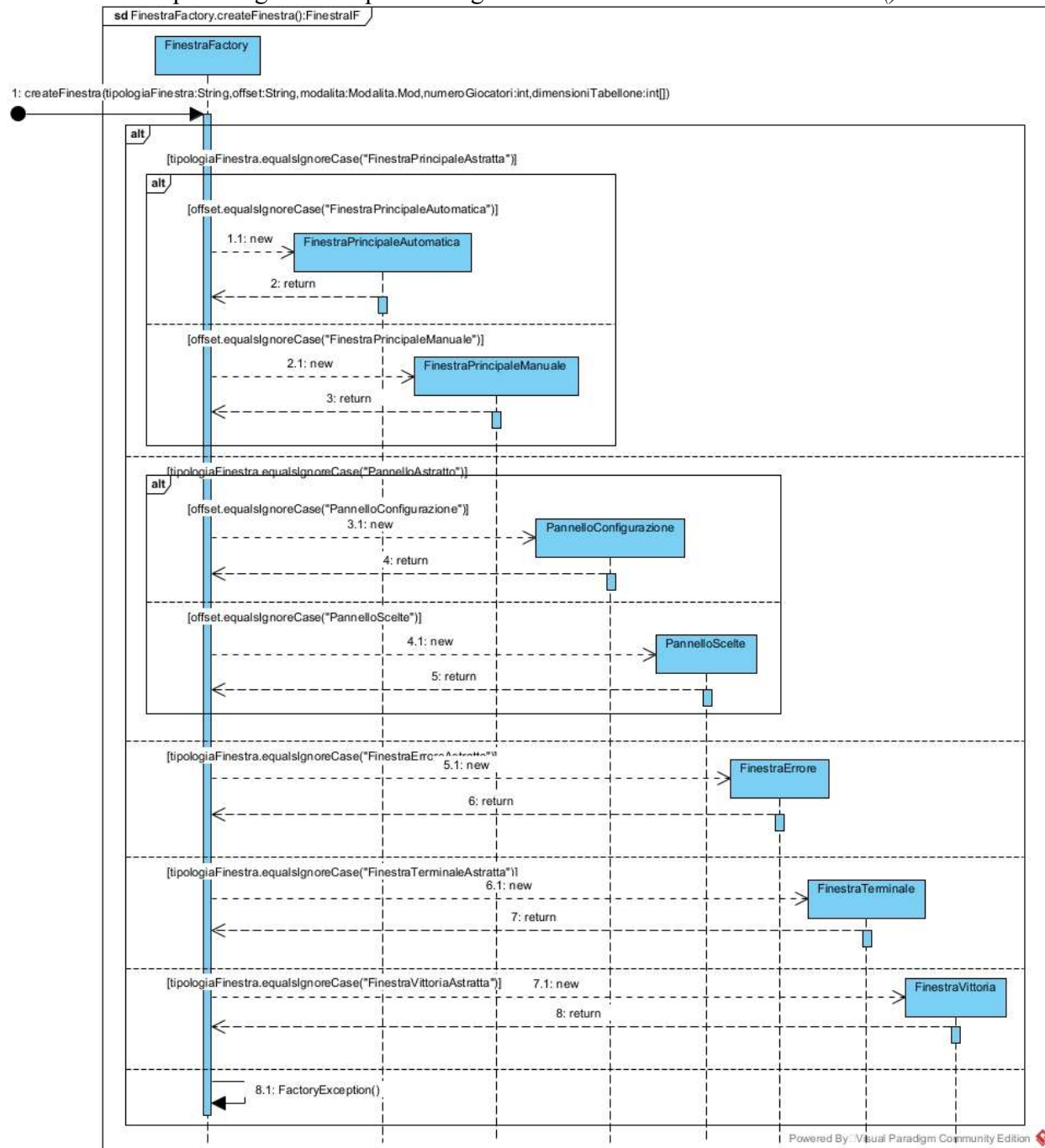
#### Applicazione Sequence Diagram

Viene mostrato qui di seguito il sequence diagram relativo al metodo *start()*:



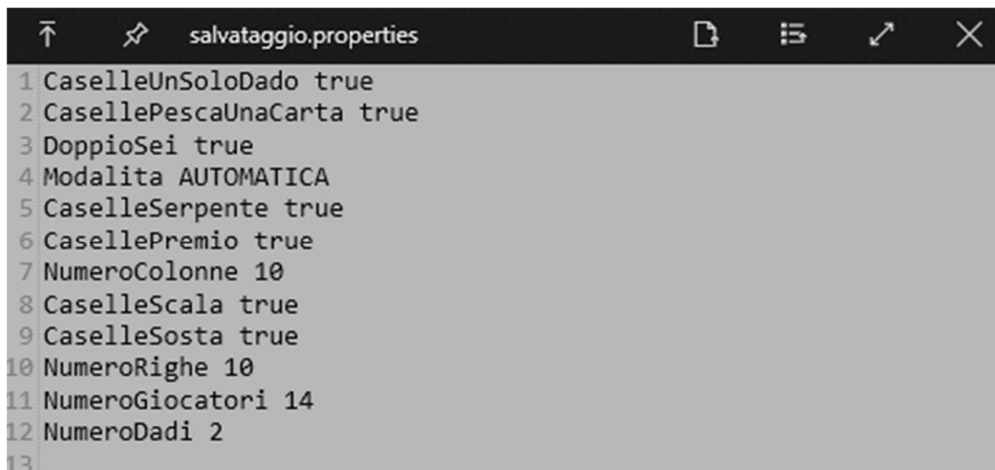
### *FinestraFactory Sequence Diagram*

Viene mostrato qui di seguito il sequence diagram relativo al metodo *createFinestra()*:



## D. Dati e loro modellazione

All'interno dell'applicazione è garantito il salvataggio della configurazione impostata nel "pannello configurazione" dall'osservatore e, inoltre, è garantito il ripristino di una configurazione di gioco precedentemente salvata sul calcolatore. Sia il salvataggio che il ripristino sono garantiti tramite file con estensione .properties. Tali tipologie di file conterranno coppie key-value che rispecchieranno per l'appunto parametro-valore (dove per parametro si intende un parametro di configurazione di gioco). Per quanto riguarda il salvataggio, verranno prima raccolti tutti i dati, utili ad un ripristino successivo di tale configurazione, in un'apposita struttura dati tale da mantenere l'accoppiamento key-value per ciascun parametro ed il suo corrispondente valore. Successivamente tali coppie verranno salvate per l'appunto su un file con estensione .properties. Per quanto riguarda il ripristino del file, esso sarà ottenuto tramite una lettura del file con estensione .properties che si intende ripristinare e con l'assegnazione di tali valori ai parametri corrispondenti all'interno dell'applicazione in esecuzione. Pertanto, si procederà al ripristino della configurazione di gioco in questione tale da garantire una nuova simulazione del gioco.



```
↑ ↗ salvataggio.properties
1 CaselleUnSoloDado true
2 CasellePescaUnaCarta true
3 DoppioSei true
4 Modalita AUTOMATICA
5 CaselleSerpente true
6 CasellePremio true
7 NumeroColonne 10
8 CaselleScala true
9 CaselleSosta true
10 NumeroRighe 10
11 NumeroGiocatori 14
12 NumeroDadi 2
13
```

*Esempio di salvataggio*

## E. Scelte Progettuali (Design Decisions)

I design pattern utilizzati durante la progettazione sono:

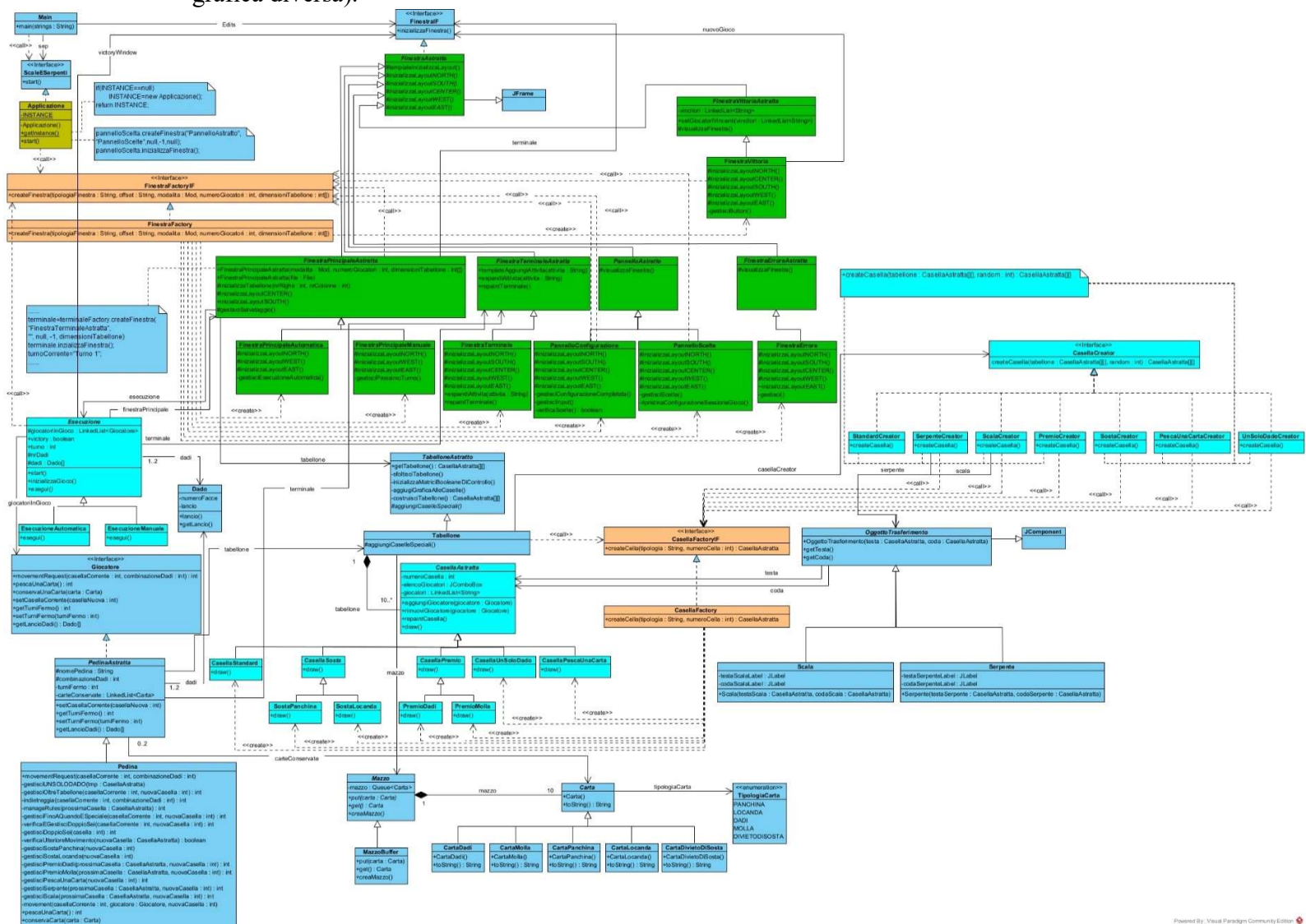
- Singleton
- Factory Method
- Template Method
- Strategy

Il design pattern **Singleton** (evidenziato nel Class Diagram con il colore marrone chiaro) è stato utilizzato per assicurare che la classe “Applicazione” (*app.game*) abbia una sola istanza, cioè tale da avere una sola istanza della sessione di gioco. Per evitare l’istanziamento arbitraria, il costruttore è stato reso privato tale che l’accesso all’unica istanza avviene tramite un metodo statico (*getInstance()*). Tale metodo utilizza la tecnica della *lazy initialization*, cioè se l’istanza non è stata ancora inizializzata allora la si inizializza tramite il costruttore privato altrimenti si restituisce quella già esistente.

Il design pattern **Factory Method** (evidenziato nel Class Diagram con il colore arancione) è stato utilizzato per definire un’interfaccia per la creazione di finestre e caselle, lasciando alle sotto-classi la decisione sulla classe concreta (rispettivamente sulla tipologia di finestra e di casella) da istanziare. Per la precisione, viene utilizzato in entrambi i casi il *Factory Method parametrizzato*. Tanto è vero che il metodo “*createFinestra*” implementa una finestra basandosi sugli input di tale metodo (principalmente vengono utilizzate due stringhe, una di base ed una come “offset”, per riconoscere la tipologia di finestra da implementare e ulteriori parametri utili per determinate tipologie di finestre). Stesso discorso vale per quanto riguarda il metodo “*createCella*” che si basa sulla stringa in input per istanziare caselle di tipologia diversa.

Il design pattern **Template Method** (evidenziato nel Class Diagram con il colore verde) è stato utilizzato per definire la struttura di un algoritmo all’interno di un metodo (*templateInizializzaLayout()*) di una classe base (*gui.window.FinestraAstratta*), delegando alcuni passi (*inizializzaLayoutNORTH()*, *inizializzaLayoutCENTER()*, *inizializzaLayoutSOUTH()*, *inizializzaLayoutWEST()*, *inizializzaLayoutEAST()*) alle sotto-classi. In questa maniera, le sotto-classi possono ridefinire i sotto-passi dell’algoritmo senza alterarne la struttura e senza la necessità di doverla re-implementare nuovamente. Per la precisione, il metodo template è stato dichiarato “*final*” così da non consentire alle sotto-classi di cambiare la struttura interna di quest’ultimo. Pertanto, le classi concrete dovranno fornire le implementazioni concrete dei metodi astratti definiti all’interno della classe astratta “*FinestraAstratta*”. Quindi, a seconda della classe concreta istanziata, verrà eseguito l’algoritmo template (*templateInizializzaLayout()*) con al suo interno ovviamente le operazioni concretizzate all’interno della classe istanziata.

Il design pattern **Strategy** (evidenziato nel Class Diagram con il colore azzurro chiaro) è stato utilizzato per definire una famiglia di algoritmi che sono in grado di risolvere lo stesso tipo di problema. Per la precisione, il metodo *createCasella()* nella classe “CasellaCreator” (*app.tabellone.casella.strategy*) verrà implementato secondo una strategia diversa a seconda della tipologia di casella istanziata. Stesso discorso vale per quanto riguarda il metodo *esegui()* nella classe “Esecuzione” (*app.esecuzione*) che verrà implementato secondo una strategia diversa a seconda della modalità di esecuzione scelta in fase di configurazione. Inoltre, tale design pattern è stato utilizzato per rappresentare graficamente ogni tipologia di casella all’interno del tabellone. Per la precisione, è stata definita una classe astratta “CasellaAstratta” (*app.tabellone.casella*) all’interno della quale è presente il metodo “*draw()*” che permette di rappresentare graficamente la tipologia di casella corrispondente. Tanto è vero, in base alla tipologia di casella istanziata, verrà utilizzato il metodo concretizzato in essa (essendo che ogni casella ha una rappresentazione grafica diversa).



## F. Progettazione di Basso Livello

Per quanto riguarda l'algoritmo di assegnazione delle tipologie delle caselle all'interno del tabellone, sono state fatte determinate scelte progettuali. Infatti, tale algoritmo si basa sulla generazione di numeri random. Per la precisione, per l'assegnazione di tali caselle si itera sulle righe del tabellone per calcolare i *bounds* di ciascuna riga, cioè il numero di casella minore e quello maggiore corrispondente a tale riga. Successivamente viene generato un numero random per ogni tipologia di casella su un intervallo di valori avente come estremi (inclusi) i *bounds* appena citati. Quindi, si cerca di assegnare almeno una casella per ogni tipologia di casella (se possibile poiché ci sono casi in cui l'assegnazione di una determinata tipologia di casella non va a buon fine e, pertanto, per quella determinata tale tipologia non viene assegnata) controllando ogni volta se tale tipologia è stata selezionata al momento della configurazione da parte dell'osservatore. Sostanzialmente, ogni tipologia di casella cerca di assegnare essa stessa in quella determinata casella il cui numero è stato calcolato in maniera random. Nel caso in cui essa non viene assegnata allora si passa alla prossima tipologia da assegnare (quindi, non si calcola nuovamente un altro numero random così da assegnarla). Molto particolare, è l'assegnazione di scale e serpenti poiché di entrambi bisogna ogni volta assegnare due caselle: testa e coda. Infatti, inizialmente si cerca di assegnare la testa e se è stata assegnata si cerca allora di assegnare successivamente anche la coda generando un numero random per la riga che conterrà la coda. Se la coda viene assegnata allora si crea rispettivamente l'oggetto Scala e Serpente altrimenti viene de-assegnata la testa. Ovviamente, quando il numero di riga random viene generato, esso sarà sicuramente diverso dal numero di riga della rispettiva testa poiché scale e serpenti non possono avere testa e coda sulla stessa riga del tabellone. In generale, per ogni tipologia di casella che si cerca di assegnare, si controlla che il numero di casella random generato sulla base dei *bounds* sia un numero di casella che non corrisponda ad una casella a cui è stata assegnata un'altra tipologia altrimenti essa non verrà assegnata. Comunque, l'algoritmo fa in modo che per ogni riga ci sia un certo numero minimo di caselle speciali così da rendere *avvincente* la simulazione del gioco. Da precisare che le caselle di tipologia "Un Solo Dado" vengono assegnate all'inizio dell'algoritmo poiché per le regole del gioco esse hanno numeri di casella già pre-fissati.

Si può notare qui di seguito il codice relativo all'assegnamento delle caselle speciali all'interno del tabellone:

```
@Override protected void aggiungiCaselleSpeciali() {
    if (PannelloConfigurazione.caselleUnSoloDadoINSIDE) {
        casellaCreator = new UnSoloDadoCreator();
        tabellone = casellaCreator.createCasella(tabellone, -1);
    }

    for (int i=0; i<tabellone.length; ++i) {
        int[] bounds = boundsRigaIesima(tabellone[i]);
        int base = -1, limite = -1;
        if (bounds[0]>bounds[1]) {
            base = bounds[1];
            if (base==1)
                base=2;
            limite = bounds[0];
        } else {
```



```
        base = bounds[0];
        if (base==1)
            base=2;
        limite = bounds[1];
    }

    if (PannelloConfigurazione.caselleSostaINSIDE) {
        int randSostaPanchina = randomSosta.nextInt(base,limite+1);
        casellaCreator = new SostaPanchinaCreator(i, this);
        tabellone = casellaCreator.createCasella(tabellone, randSostaPanchina);
        int randSostaLocanda = randomSosta.nextInt(base, limite+1);
        casellaCreator = new SostaLocandaCreator(i, this);
        tabellone = casellaCreator.createCasella(tabellone, randSostaLocanda);
    }

    if (PannelloConfigurazione.casellePremioINSIDE) {
        int randPremioDadi = randomPremio.nextInt(base, limite+1);
        casellaCreator = new PremioDadiCreator(i, this);
        tabellone = casellaCreator.createCasella(tabellone, randPremioDadi);
        int randPremioMolla = randomPremio.nextInt(base, limite+1);
        casellaCreator = new PremioMollaCreator(i, this);
        tabellone = casellaCreator.createCasella(tabellone, randPremioMolla);
    }

    if (PannelloConfigurazione.casellePescaUnaCartaINSIDE) {
        int randPescaUnaCarta = randomPescaUnaCarta.nextInt(base, limite+1);
        casellaCreator = new PescaUnaCartaCreator(i, this);
        tabellone = casellaCreator.createCasella(tabellone, randPescaUnaCarta);
    }

    if (PannelloConfigurazione.scaleINSIDE) {
        int randScala = randomScala.nextInt(base, limite+1);
        casellaCreator = new ScalaCreator(i, this);
        if (randScala!=1)
            tabellone = casellaCreator.createCasella(tabellone, randScala);
        else;
    }

    if (PannelloConfigurazione.serpentiINSIDE) {
        int randSerpente = randomSerpente.nextInt(base, limite+1);
        casellaCreator = new SerpenteCreator(i, this);
        if (randSerpente!=1)
            tabellone = casellaCreator.createCasella(tabellone, randSerpente);
        else;
    }
}
}
```

Per quanto riguarda, invece, l'algoritmo di avanzamento della pedina sono state fatte alcune scelte progettuali determinate in base alle possibili regole del gioco che possono essere configurate dall'osservatore. Per la precisione, ogni volta che un giocatore deve muoversi da una casella ad un'altra all'interno del tabellone, viene fatta una richiesta di movimento (*app.esecuzione.giocatore.Pedina#movementRequest(int, int)*) specificando il numero della casella corrente e la combinazione dei dadi ottenuta dal lancio dei dadi o del dado. Tale metodo permette, quindi, alla pedina in questione di avanzare per un certo numero di caselle in base alla combinazione dei dadi ottenuta ed in base alle regole del gioco. Infatti, inizialmente si verifica se essa si trova in casella di tipologia "Un Solo Dado" e caso mai si applicano le corrispettive regole, altrimenti si verifica che il numero della nuova casella ottenuto non sia maggiore del numero di casella corrispondente al "traguardo" perché altrimenti il giocatore dovrà indietreggiare di un certo numero di caselle. Se non è questa la situazione, allora si verifica se la nuova casella è una casella di tipologia speciale e si gestiscono le regole finché il giocatore non capita in una casella di tipologia "Standard" oppure in una casella di tipologia "Sosta Panchina" o "Sosta

Locanda” facendo rimanere la pedina ferma rispettivamente per un turno o per tre turni. Infine, viene gestito a livello grafico lo spostamento della pedina all’interno del tabellone eliminando il giocatore dall’elenco dei giocatori presenti nella vecchia casella da cui era partito e aggiungendolo all’elenco dei giocatori della nuova casella in cui si posiziona. Ovviamente, ogni *attività* svolta durante il movimento del giocatore (spostamento della pedina, gestione delle regole di caselle speciali e combinazione dei dadi ottenuta) viene documentata e stampata in maniera dettagliata all’interno della finestra Terminale, istanziata inizialmente insieme alla finestra Principale successivamente alla configurazione del gioco.

Si può notare qui di seguito il codice relativo alla richiesta di movimento per una certa combinazione di dadi da parte di una pedina a partire da una determinata casella iniziale:

```
@Override public int movementRequest(int casellaCorrente, int combinazioneDadi) {
    this.setCombinazioneDadi(combinazioneDadi);
    this.setCasellaCorrente(casellaCorrente);
    CasellaAstratta tmp = getCasella(matriceTabellone, casellaCorrente);
    int nuovaCasella = gestisciUNSOLODADO(tmp);
    if (nuovaCasella <= matriceTabellone[0][0].getNumeroCasella()) {
        attivita = this.toString()+" si e' spostato nella casella"+nuovaCasella;
        terminale.esandiAttivita(attivita);
        terminale.repaintTerminale();
    }
    if ( nuovaCasella > matriceTabellone[0][0].getNumeroCasella() ) {
        return gestisciOltreTabellone(casellaCorrente, nuovaCasella);
    }
    else {
        tmp = getCasella(matriceTabellone, nuovaCasella);
        this.setCasellaCorrente(nuovaCasella);
        nuovaCasella = manageRules( tmp );
        this.setCasellaCorrente(nuovaCasella);
        if (nuovaCasella > matriceTabellone[0][0].getNumeroCasella()) {
            return gestisciOltreTabellone(casellaCorrente, nuovaCasella);
        }
        nuovaCasella = gestisciFinoAQuandoESpeciale(casellaCorrente,nuovaCasella);
        nuovaCasella = verificaEGestisciDoppioSei(casellaCorrente,nuovaCasella);
        this.setCasellaCorrente(nuovaCasella);
        movement( casellaCorrente, this, nuovaCasella );
        return nuovaCasella;
    }
}
```

Per quanto riguarda, invece, i design pattern utilizzati:

- **Singleton**

```
public final class Applicazione implements ScaleESerpenti {

    private static Applicazione INSTANCE = null;

    private Applicazione() {}

    public static synchronized Applicazione getInstance() {
        if(INSTANCE == null)
            INSTANCE = new Applicazione();
        return INSTANCE;
    }
}
```



Si può notare la *lazy initialization* all'interno del metodo *getInstance()* che restituisce un'istanza della classe in questione. Tale istanza se non esiste verrà creata tramite il costruttore privato altrimenti verrà restituita quella esistente.

- **Factory Method**

```
public class FinestraFactory implements FinestraFactoryIF {

    @Override public FinestraIF createFinestra(String tipologiaFinestra, String
    offset, Modalita.Mod modalita, int numeroGiocatori, int[] dimensioniTabellone) {

        if (tipologiaFinestra.equalsIgnoreCase("FinestraPrincipaleAstratta")) {

            if (offset.equalsIgnoreCase("FinestraPrincipaleAutomatica"))
                return new FinestraPrincipaleAutomatica(modalita,
                    numeroGiocatori, dimensioniTabellone);
            else if (offset.equalsIgnoreCase("FinestraPrincipaleManuale"))
                return new FinestraPrincipaleManuale(modalita,
                    numeroGiocatori, dimensioniTabellone);

        }
        else if (tipologiaFinestra.equalsIgnoreCase("PannelloAstratto")) {

            if (offset.equalsIgnoreCase("PannelloConfigurazione"))
                return new PannelloConfigurazione();
            else if (offset.equalsIgnoreCase("PannelloScelte"))
                return new PannelloScelte();

        }
        elseif (tipologiaFinestra.equalsIgnoreCase("FinestraErroreAstratta"))
            return new FinestraErrore();

        else if (tipologiaFinestra.equalsIgnoreCase("FinestraTerminaleAstratta"))
            return new FinestraTerminale();

        else if (tipologiaFinestra.equalsIgnoreCase("FinestraVittoriaAstratta"))
            return new FinestraVittoria();

        throw new FactoryException();

    }
}
```

Si può notare che il *Factory Method parametrico* tramite i parametri di input riesce ad instanziare tipologie di finestre diverse. Per la precisione, la tipologia di finestra generale corrisponderà alla stringa “*tipologiaFinestra*” mentre la tipologia più specifica alla stringa “*offset*”.

```
public class CasellaFactory implements CasellaFactoryIF {

    @Override public CasellaAstratta createCella(String tipologia, int numeroCella) {

        if (tipologia.equalsIgnoreCase("Standard"))
            return new CasellaStandard(numeroCella);
        else if (tipologia.equalsIgnoreCase("UnSoloDado"))
            return new CasellaUnSoloDado(numeroCella);
        else if (tipologia.equalsIgnoreCase("SostaPanchina"))
            return new CasellaSostaPanchina(numeroCella);
        else if (tipologia.equalsIgnoreCase("SostaLocanda"))
            return new CasellaSostaLocanda(numeroCella);
        else if (tipologia.equalsIgnoreCase("PremioDadi"))
            return new CasellaPremioDadi(numeroCella);
        else if (tipologia.equalsIgnoreCase("PremioMolla"))
            return new CasellaPremioMolla(numeroCella);

    }
}
```

```
        else if (tipologia.equalsIgnoreCase("PescaUnaCarta"))  
            return new CasellaPescaUnaCarta(numeroCella);  
        else if (tipologia.equalsIgnoreCase("Scala"))  
            return new CasellaScala(numeroCella);  
        else if (tipologia.equalsIgnoreCase("Serpente"))  
            return new CasellaSerpente(numeroCella);  
        throw new FactoryException();  
    }  
}
```

Si può notare che il *Factory Method parametrico* tramite i parametri di input riesce ad instanziare tipologie di caselle diverse. Per la precisione, la tipologia di casella generale corrisponderà alla stringa “tipologiaCasella” mentre il numero di casella corrisponderà al parametro “numeroCella”.

- **Template Method**

```
protected final void templateInizializzaLayout() {  
    inizializzaLayoutNORTH();  
    inizializzaLayoutCENTER();  
    inizializzaLayoutSOUTH();  
    inizializzaLayoutWEST();  
    inizializzaLayoutEAST();  
}  
  
protected abstract void inizializzaLayoutNORTH();  
protected abstract void inizializzaLayoutCENTER();  
protected abstract void inizializzaLayoutSOUTH();  
protected abstract void inizializzaLayoutWEST();  
protected abstract void inizializzaLayoutEAST();  
  
public class FinestraVittoria extends FinestraVittoriaAstratta {  
  
    private JLabel labelTitle;  
  
    @Override protected void inizializzaLayoutNORTH() {  
        pNORTH = new JPanel();  
        pNORTH.setBackground(Color.LIGHT_GRAY);  
  
        labelTitle = new JLabel("Vincitore della sessione di gioco");  
        labelTitle.setBackground(Color.GREEN.darker());  
        labelTitle.setForeground(Color.BLACK);  
        labelTitle.setBorder(new RoundedBorder(10));  
        labelTitle.setOpaque(true);  
        pNORTH.add(labelTitle, BorderLayout.CENTER);  
  
        this.add(pNORTH, BorderLayout.NORTH);  
    }  
  
    @Override protected void inizializzaLayoutCENTER() {  
        pCENTER = new JPanel();  
        pCENTER.setBackground(Color.LIGHT_GRAY);  
  
        this.add(pCENTER, BorderLayout.CENTER);  
    }  
  
    private JButton esci, newGame;
```

```
@Override protected void inizializzaLayoutSOUTH() {
    pSOUTH = new JPanel();
    pSOUTH.setBorder(new RoundedBorder(5));

    esci = new JButton("ABBANDONA");
    esci.setForeground(Color.BLACK);
    esci.setBackground(Color.RED);
    esci.setBorder(new RoundedBorder(10));
    esci.setOpaque(true);
    pSOUTH.add(esci, BorderLayout.CENTER);

    newGame = new JButton("NUOVA SESSIONE DI GIOCO");
    newGame.setForeground(Color.BLACK);
    newGame.setBackground(Color.GREEN);
    newGame.setBorder(new RoundedBorder(10));
    newGame.setOpaque(true);
    pSOUTH.add(newGame, BorderLayout.CENTER);

    gestisciButton();

    this.add(pSOUTH, BorderLayout.SOUTH);
}

@Override protected void inizializzaLayoutWEST() {}
@Override protected void inizializzaLayoutEAST() {}
}
```

Si può notare che la struttura dell’algoritmo “templateInizializzaLayout” sia composta da sotto-passi che verranno implementati nelle sotto-classi. Infatti, possiamo notare come tali sotto-passi siano dichiarati “abstract” all’interno della classe base “FinestraAstratta”. Subito dopo lo screenshot del *template method* è stato allegato un ulteriore screenshot di esempio relativo alla concretizzazione di tali metodi all’interno della classe “FinestraVittoria”.

- **Strategy**

```
public interface CasellaCreator {

    CasellaAstratta[][] createCasella(CasellaAstratta[][] tabellone, int random);

}

public class PescaUnaCartaCreator implements CasellaCreator {

    private int nrRiga;
    private Tabellone t;

    private CasellaFactoryIF casellaFactory = new CasellaFactory();

    public PescaUnaCartaCreator(int nrRiga, Tabellone t) {
        this.nrRiga = nrRiga;
        this.t = t;
    }

    @Override public CasellaAstratta[][] createCasella(CasellaAstratta[][] tabellone,
        int random) {

        if(random == tabellone[0][0].getNumeroCasella())
            return tabellone;

        for(int j=0;j<tabellone[nrRiga].length;++j) {
```

```
        if( tabellone[nrRiga][j].getNumeroCasella()==random ) {

            if(tabellone[nrRiga][j].getNumeroCasella() >=
Tabellone.CELLE_UN_SOLO_DADO[0] &&
tabellone[nrRiga][j].getNumeroCasella() <=
Tabellone.CELLE_UN_SOLO_DADO[1])
                break;

            if( t.verificaCellaNonSpeciale(nrRiga, j) ) {
                t.pescaUnaCarta[nrRiga][j] = true;

                tabellone[nrRiga][j] = casellaFactory.createCella(
"PescaUnaCarta",
tabellone[nrRiga][j].getNumeroCasella());
                break;
            }
        }
    }

    return tabellone;
}
}
```

Si può notare che l'implementazione dell'algoritmo varia in base alla classe. Subito dopo lo screenshot dell'algoritmo è stato allegato un ulteriore screenshot di esempio relativo alla strategia scelta per la creazione e l'assegnazione della casella di tipologia "*Pesca Una Carta*".

## G. Spiegare come il progetto soddisfa i requisiti funzionali (FRs) e quelli non funzionali (NFRs)

Per quanto riguarda la creazione di una nuova configurazione di gioco, essa è stata possibile introducendo un pannello di configurazione dove l'osservatore è in grado di configurare la nuova sessione di gioco secondo i suoi desideri. Pertanto, tramite *JCheckBox*, che permettono la scelta delle tipologie di caselle da inserire nel tabellone, e tramite appositi *JTextField*, che permettono all'osservatore di inserire il numero di righe, di colonne e di giocatori desiderato, egli stesso è libero di configurare la nuova simulazione nel modo da lui desiderato. Inoltre, all'interno del "*PannelloConfigurazione*" è possibile scegliere tramite due *JRadioButton* quale modalità di esecuzione adottare per la nuova sessione di gioco, quindi se automatica o manuale. Per la precisione in base alla modalità di esecuzione scelta, se l'osservatore sceglierà un'esecuzione automatica allora visualizzerà nel *JPanel* "*pNORTH*" un *JButton* con il nome "*Esegui automaticamente*" che gli permetterà di eseguire automaticamente l'intera simulazione della sessione di gioco configurata. Cliccando sul pulsante permetterà all'osservatore di visualizzare le posizioni raggiunte dalle pedine nell'ultimo turno giocato da queste ultime, tale che il *JButton* "*Esegui automaticamente*" verrà disabilitato e verrà visualizzato il giocatore vincente in un'apposita finestra. Se, invece, sceglierà la modalità di esecuzione manuale allora l'osservatore visualizzerà nel pannello "*pNORTH*" il *JButton* con il nome "*Prossimo turno*" che gli permetterà di eseguire un solo turno di gioco. Pertanto, ogni volta che cliccherà su tale pulsante verrà simulato un singolo turno di gioco tale da fargli visualizzare la nuova posizione delle pedine all'interno del tabellone. Ovviamente, dopo un certo numero di turni qualche giocatore arriverà al traguardo, il *JButton* "*Prossimo turno*" verrà disabilitato proprio perché la simulazione della sessione di gioco è giunta al termine ed il nome del giocatore vincente verrà visualizzato in un'apposita finestra. Per la precisione il giocatore vincente verrà mostrato in una finestra (configurata sempre tramite il *factory method parametrizzato* come d'altronde le altre finestre) dove, inoltre, saranno presenti due *JButton*: "*Abbandona*" e "*Nuova Sessione di Gioco*". Rispettivamente, il primo permetterà all'osservatore di terminare l'applicazione in questione mentre il secondo gli permetterà di configurare ed inizializzare una nuova sessione di gioco. Molto interessante è la funzione di salvataggio della configurazione di gioco. Appena viene inizializzata una nuova "*FinestraPrincipale*" per una nuova sessione di gioco, nel pannello "*pNORTH*", oltre a visualizzare il *JButton* corrispondente alla modalità di esecuzione scelta, sarà presente un ulteriore *JButton* con il nome "*Salva*" che permetterà all'osservatore di salvare i parametri, configurati all'interno del "*PannelloConfigurazione*", così da essere ripristinati in seguito e inizializzare una nuova simulazione di gioco. Pertanto, all'interno di un file con estensione *.properties*, salvato in una directory sul calcolatore scelta dall'osservatore, verranno salvate le configurazioni dei parametri utili all'inizializzazione della simulazione. Per la precisione verrà salvata la modalità di esecuzione scelta, il numero di giocatori, le

dimensioni del tabellone, il numero dei dadi e se è stata abilitata la modalità *doppio sei* e le tipologie di caselle che dovranno essere presenti all'interno del tabellone. Il processo di salvataggio avverrà tramite un *JFileChooser* che permetterà all'osservatore per l'appunto di salvare il file con il nome da lui desiderato e, soprattutto, nella directory da lui desiderata. Per quanto riguarda il ripristino, esso sarà possibile all'interno del "*PannelloScelte*". Per la precisione, appena l'osservatore deciderà di eseguire l'applicazione, egli visualizzerà un pannello in cui potrà scegliere se inizializzare una nuova sessione di gioco e, quindi, procedere alla sua configurazione tramite il "*PannelloConfigurazione*" oppure ripristinare una precedente configurazione del gioco salvata sul proprio calcolatore. Tale file di ripristino, ovviamente, dovrà essere un file con estensione *.properties* altrimenti l'osservatore visualizzerà in un'apposita finestra l'errore di caricamento del file. Infatti, appena egli cliccherà sul *JButton* "*Ripristina*" visualizzerà un *JFileChooser* affinché possa selezionare il file in questione oppure, recandosi nella directory che contiene tale file, potrà direttamente digitare il suo nome così da ripristinarlo di conseguenza. Ovviamente, in questo ultimo caso, se viene inserito un nome di un file non esistente o non conforme all'estensione prevista verrà segnalato l'errore all'osservatore.

Per quanto riguarda, invece, l'avanzamento automatico e manuale della simulazione, questo è stato possibile configurando opportunamente l'algoritmo di avanzamento della pedina. Rispettivamente, la prima modalità citata permette tramite un *loop* continuo (fin quando un giocatore non risulta vincente) lo spostamento di casella in casella delle pedine gestendo per ogni movimento le regole previste dal gioco.

Per quanto riguarda la seconda modalità citata essa fondamentalmente si baserà sul click da parte dell'osservatore sul *JButton* "*Prossimo turno*". Infatti, ogni volta che egli cliccherà su tale pulsante, verrà eseguita una simulazione delle pedine per il singolo turno di gioco. Nel momento in cui un giocatore raggiunge il traguardo, l'esecuzione del turno si interrompe e dichiara il giocatore vincente. Per ogni turno giocato e per ogni giocatore, sia per l'esecuzione automatica che manuale, viene ogni volta calcolata una nuova combinazione di dadi ottenuta tramite il lancio dei dadi, istanziati in base alla configurazione dell'osservatore. Per entrambe le esecuzioni, ogni volta che avviene un evento (che possa essere il lancio dei dadi, lo spostamento di una pedina o la gestione di determinate regole) durante la simulazione, esso verrà *stampato* all'interno della "*FinestraTerminale*", istanziata insieme alla "*FinestraPrincipale*" non appena viene inizializzata la nuova configurazione di gioco imposta dall'osservatore. Per la precisione, viene utilizzata una *LinkedList<String>* alla quale ogni volta vengono aggiunti gli eventi (*attività*) appena accaduti nella simulazione. Ogni volta che un evento viene aggiunto, il terminale viene aggiornato tramite il metodo *repaintTerminale()* che lo aggiornerà graficamente.

Per quanto riguarda i requisiti non funzionali fondamentalmente il completamento della simulazione del gioco è garantito dal fatto che effettivamente il sistema software permette, a prescindere dalla modalità di esecuzione scelta, di completare la simulazione di gioco effettuata. Per la precisione, il completamento dell'esecuzione automatica è garantito dal singolo click sul *JButton* "*Esegui automaticamente*", contenuto nel *JPanel pNORTH* della *FinestraPrincipale*, che effettuerà l'intera simulazione del gioco, mentre nel caso

dell'esecuzione manuale, il completamento della simulazione sarà garantito in base ai click dell'osservatore, cioè se egli cliccherà il *JButton* "*Prossimo turno*" un certo numero di volte così da consentire il terminare della simulazione allora effettivamente la simulazione sarà portata al termine altrimenti la simulazione si fermerà al turno di gioco appena simulato.

Per quanto riguarda, invece, il vincitore del gioco, esso sarà garantito dai motivi appena citati riguardo il completamento della simulazione. Se la simulazione verrà completata allora effettivamente al suo termine verrà visualizzato un vincitore corrispondente alla simulazione effettuata.

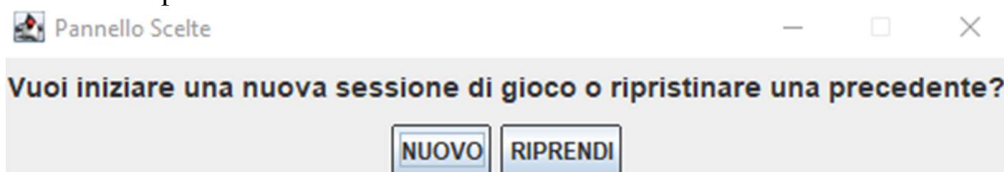
Per quanto riguarda le attività che vengono *svolte* durante la simulazione, ogni attività che può essere anche il semplice spostamento da una casella di tipologia "*Standard*" ad un'altra casella "*Standard*" verrà documentato in maniera dettagliata e automatica all'interno della "*Finestra Terminale*".

---

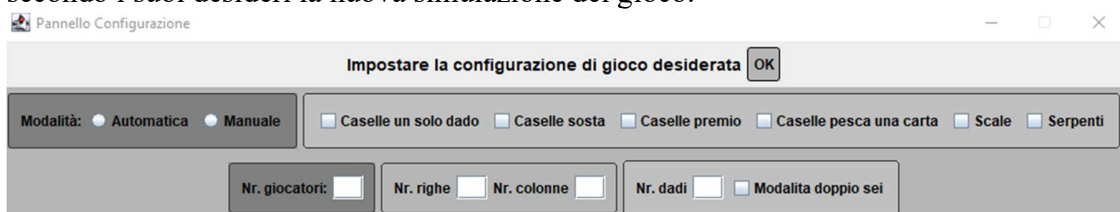
## Appendix. Prototype

Vengono mostrate qui di seguito le immagini relative all'applicazione Scale e Serpenti in funzione con le corrispettive descrizioni.

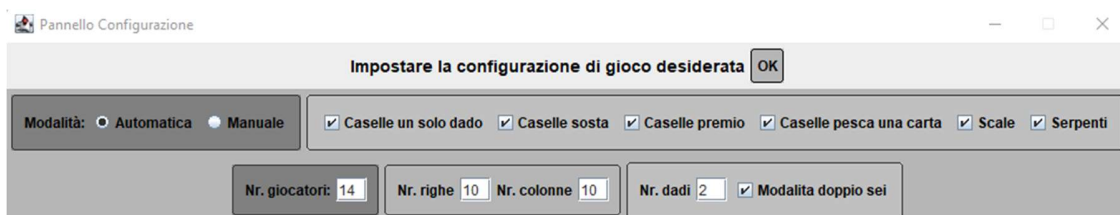
Si può notare il “*PannelloScelte*”, mostrato inizialmente all'avvio dell'applicazione, che permette all'osservatore di scegliere se inizializzare una nuova sessione di gioco o ripristinarne una precedente:



Nel caso in cui l'osservatore scelga di inizializzare una nuova sessione di gioco, verrà mostrato successivamente il “*PannelloConfigurazione*” che gli permetterà di configurare secondo i suoi desideri la nuova simulazione del gioco:

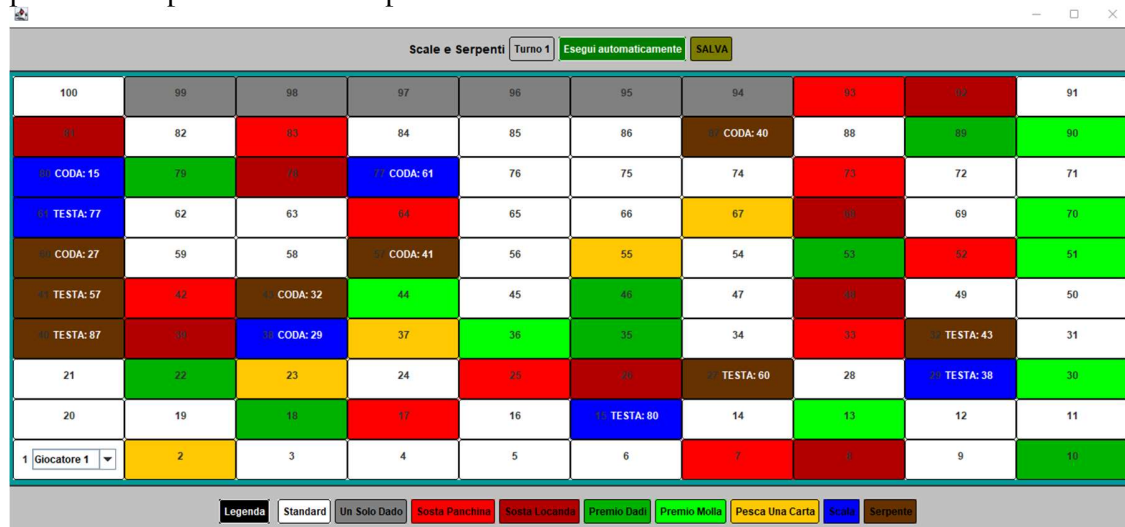


Qui, di seguito viene mostrata una possibile configurazione della nuova sessione di gioco:





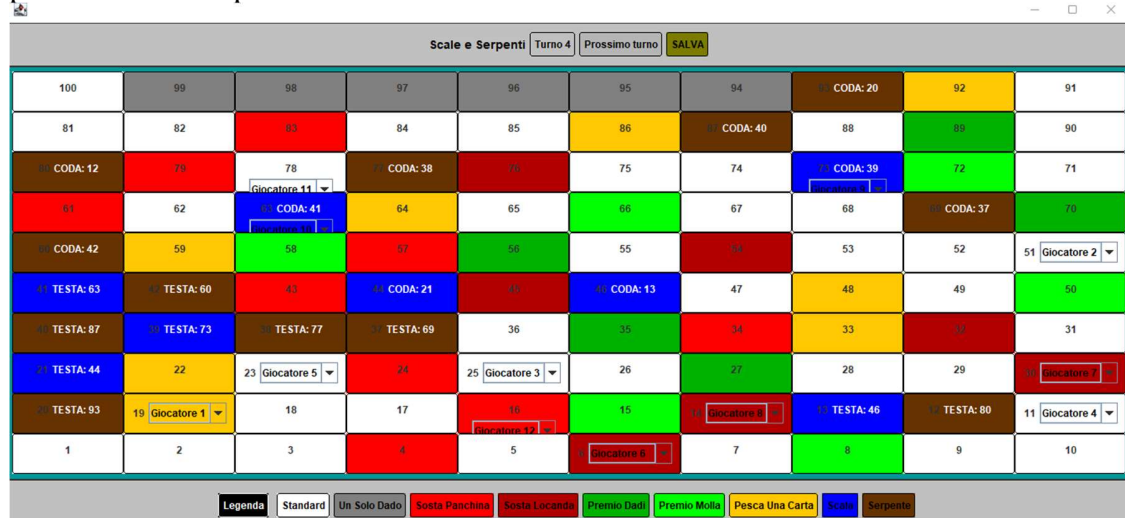
Successivamente, dopo che l'osservatore ha configurato la nuova sessione di gioco in maniera opportuna, visualizza la “FinestraPrincipale” dove risulta essere visibile il tabellone avente percorso di forma bustrofedica, la legenda per le varie tipologie di caselle, il turno corrente e il *JButton* “Esegui automaticamente” trattandosi per l'appunto di una simulazione inizializzata con la modalità di esecuzione *Automatica*. Molto particolare è la rappresentazione delle scale e dei serpenti dove per ognuna delle caselle viene indicato un riferimento rispettivamente alla testa e alla coda. Si può notare che inizialmente tutte le pedine sono posizionate nella prima casella del tabellone:



Viene mostrato, qui di seguito, come il terminale viene inizializzato all'inizio di una sessione di gioco che sia nuova o ripristinata. Si può notare come inizialmente vengono *caricati* tutti i giocatori all'interno della partita:



Qui di seguito viene mostrata una possibile esecuzione in modalità *Manuale* tale che tutte le pedine si troveranno in una determinata casella del tabellone. Si può notare, inoltre, che ogni casella, nel momento in cui quest'ultima viene occupata da una o più pedine, viene mostrato, di conseguenza, al suo interno una *JComboBox* contenente i giocatori presenti in quella casella. Ovviamente ogni volta che un giocatore si sposta di casella, la grafica di quest'ultima viene aggiornata di conseguenza tale che una casella non contenente alcuna pedina allora non presenterà nessuna *JComboBox* al suo interno:



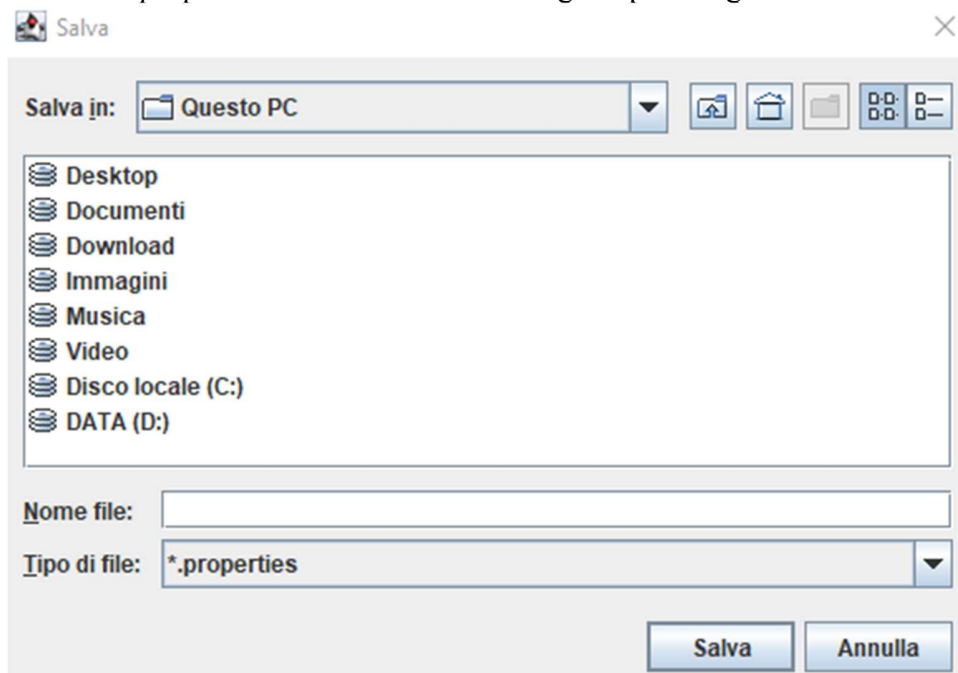
Qui di seguito viene mostrata una possibile descrizione dettagliata delle attività durante un turno della sessione di gioco. Si può notare come, ad esempio, il giocatore 10 partendo dalla casella 31 ed effettuando un lancio di dadi pari a 5 5 si sposta di 10 caselle finendo nella casella avente numero 41 però, essendo che in tale casella è presente la coda di una scala, allora tale giocatore salirà lungo la scala finendo nella testa della scala, cioè nella casella avente numero 63 (il tutto può essere osservato nel tabellone sopra mostrato):



Nel caso in cui un giocatore riesce a raggiungere il traguardo con un lancio di dadi esatto (come viene specificato nelle regole del gioco) allora esso sarà dichiarato vincitore e verrà mostrato nella “FinestraVittoria”. In tale finestra, inoltre, sarà possibile abbandonare la sessione di gioco corrente o iniziarne una nuova:



Nel caso in cui, durante una sessione di gioco si volesse salvare la configurazione del gioco corrente allora basterà cliccare sul *JButton* “Salva” e, pertanto, verrà mostrato un *JFileChooser* che permetterà all’osservatore di scegliere la directory in cui salvare il file e il nome del file stesso (la procedura di ripristino mostrerà un *JFileChooser* analogo a quello qui di seguito con la differenza che l’operazione non permetterà un salvataggio ma la selezione di un file di configurazione con estensione *.properties* per ripristinare una configurazione di gioco precedentemente salvata sul calcolatore). L’unica estensione del file consentita è *.properties* come mostrato nella figura qui di seguito:



Inoltre, per confermare la correttezza di alcuni tra i requisiti, viene implementato, tramite la libreria Java JUnit, uno unit testing:

```
@Test(timeout=TIMEOUT)
public void test() {
    Path path = FileSystems.getDefault().getPath("");
    String directoryName = path.toAbsolutePath().toString();
    fileRipristino = new File(directoryName+"\\src\\main\\Configurazione.properties");
    ripristino();

    Properties p = new Properties();
    try(FileInputStream in = new FileInputStream(fileRipristino.getAbsolutePath())){
        p.load(in);
        numeroGiocatori = p.getProperty("NumeroGiocatori");
        numeroRighe = p.getProperty("NumeroRighe");
        numeroColonne = p.getProperty("NumeroColonne");
        modalita = p.getProperty("Modalita");
        finestraAbstract = (FinestraPrincipaleAstratta)finestra;
    }catch(IOException e) {
        e.printStackTrace();
    }

    testRipristino();
    testTabellone();
}

public static void testTabellone() {
    Tabellone tabellone = new Tabellone(nrRighe, nrColonne);
    CasellaAstratta[][] matriceTabellone = tabellone.getTabellone();

    assertSame("Numero dell'ultima casella non corretto!", (nrRighe*nrColonne),
        matriceTabellone[0][0].getNumeroCasella());
}

public static void testRipristino() {
    assertEquals("Numero giocatori ripristinato non correttamente!", numeroGiocatori,
        String.valueOf(finestraAbstract.getNumeroGiocatori()));

    assertEquals("Numero righe ripristinato non correttamente!", numeroRighe,
        String.valueOf(finestraAbstract.getNrRighe()));

    assertEquals("Numero colonne ripristinato non correttamente!", numeroColonne,
        String.valueOf(finestraAbstract.getNrColonne()));

    assertEquals("Modalita ripristinata non correttamente!", modalita,
        String.valueOf(finestraAbstract.getModalita().toString()));
}
```

Si può notare come viene testata la correttezza del ripristino di un file presente nella stessa directory dello script di testing e, inoltre, viene verificata la correttezza dell'inizializzazione del tabellone all'interno della finestra principale. Il test è stato configurato affinché possa fallire dopo un certo periodo di tempo pari a *TIMEOUT*. Si può notare come nel caso del *testTabellone()* sia stato utilizzato il metodo *assertSame(expected, actual)* dove viene verificato che il valore dell'ultima casella, cioè quella del traguardo, sia effettivamente pari al prodotto tra il numero delle righe e il numero delle colonne. Mentre si può notare come nel caso del *testRipristino()* siano stati utilizzati dei metodi *assertEquals(expected, actual)* per verificare se effettivamente i valori più importanti (*numeroGiocatori*, *numeroRighe*, *numeroColonne* e *modalita*) per la configurazione siano stati ripristinati correttamente.