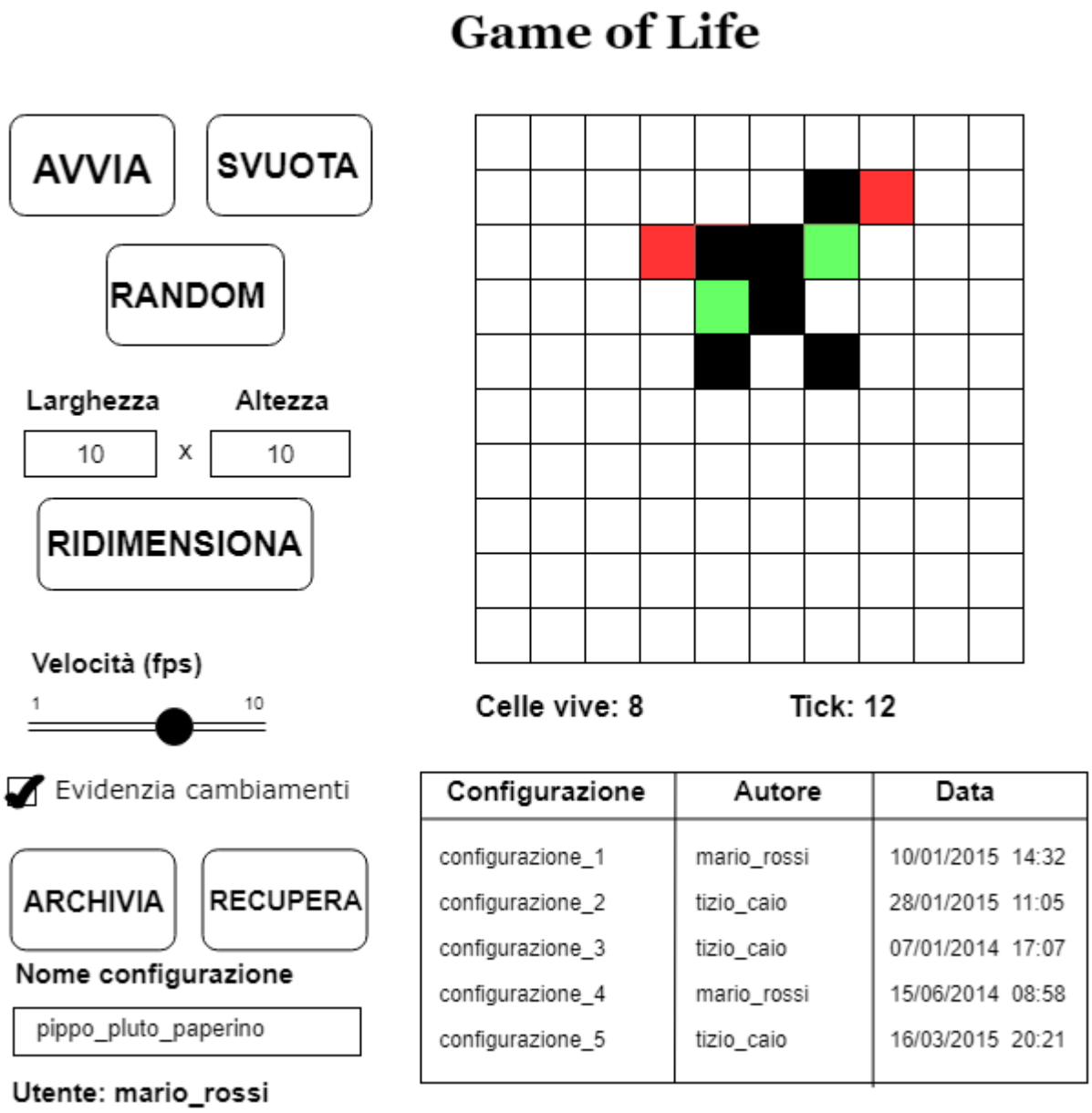
Documento di analisi

Vista dell'interfaccia

All'avvio l'applicazione si presenta all'utente come in figura. Sulla sinistra ci sono i comandi ed il pannello di configurazione, a destra invece c'è il campo. In basso è presente la tabella delle configurazioni salvate nel database, che l'utente può caricare selezionando la riga corrispondente. Le celle scure sono quelle vive, le bianche sono quelle morte. È possibile evidenziare con dei colori le celle diventate vive/morte nel corso dell'ultima iterazione.



Configurazione	Autore	Data
configurazione_1	mario_rossi	10/01/2015 14:32
configurazione_2	tizio_caio	28/01/2015 11:05
configurazione_3	tizio_caio	07/01/2014 17:07
configurazione_4	mario_rossi	15/06/2014 08:58
configurazione_5	tizio_caio	16/03/2015 20:21

Vista dinamica (scenario di utilizzo)

Di seguito viene specificato il comportamento dell'applicazione in un tipico caso di utilizzo.

1. L'utente specifica la Larghezza AND l'Altezza del campo AND preme Ridimensiona.
   1.1 Il sistema ridimensiona opportunamente il campo di gioco.
2. L'utente specifica la Velocità di aggiornamento del campo.
3. L'utente sceglie se colorare le celle appena evolute tramite Evidenzia cambiamenti
4. IF l'utente seleziona una riga della tabella AND preme Recupera
   4.1 il sistema carica la configurazione corrispondente alla riga selezionata dal database.
5. IF l'utente preme Randomizza
   5.1 il campo assume una configurazione di celle vive/morte casuale.
6. FOR EACH Cella cliccata dall'utente
   6.1 IF la Cella è viva
   6.1.1 la Cella diventa morta.
   ELSE
   6.1.2 la Cella diventa viva
7. IF l'utente preme Avvia
   7.2 il tasto Avvia diventa il tasto Pausa.
   7.1 il sistema disabilita tutti i pulsanti, ad eccezione di Pausa.
   7.3 il sistema avvia il gioco.
   ELSE
   7.4 GOTO 10.
8. REPEAT
   8.1 Il sistema aggiorna il campo secondo le regole del gioco:
   FOR EACH Cella
   8.1.1 IF la Cella è viva e ha due o tre Cella adiacenti vive
   8.1.1.1 la Cella rimane viva.
   ELSE IF la Cella è viva e ha meno di due Cella adiacenti vive
   8.1.1.2 la Cella diventa morta.
   ELSE IF la Cella è viva e ha più di tre Cella adiacenti vive
   8.1.1.3 la Cella diventa morta.
   ELSE IF la Cella è morta e ha esattamente tre Cella adiacenti vive
   8.1.1.4 la Cella diventa viva.
   ELSE
   8.1.1.5 la Cella rimane morta.
   8.2 Il sistema aggiorna i contatori di Tick e Celle vive.
   8.3 Il sistema attende un intervallo di tempo dipendente da Velocità.
   UNTIL l'utente preme Pausa
9. Il gioco si ferma, cioè il campo smette di evolversi.
   9.1 Il tasto Pausa diventa il tasto Avvia.
   9.2 Il sistema riabilita i pulsanti per modificare la configurazione.
10. IF l'utente inserisce il Nome configurazione AND preme Archivia
    10.1 Il sistema salva lo stato nel database, includendo il Nome configurazione.
11. IF l'utente preme Svuota
    11.1 Il campo assume una configurazione vuota (nessuna cella viva).
12. GOTO 1.

File di configurazione locale in XML
All'apertura dell'applicazione il sistema legge dal file di configurazione i seguenti dati:
- Larghezza di default del campo
- Larghezza minima del campo
- Larghezza massima del campo
- Altezza di default del campo
- Altezza minima del campo
- Altezza massima del campo
- Velocità (fps) di default dell'evoluzione
- Velocità (fps) minima dell'evoluzione
- Velocità (fps) massima dell'evoluzione
- Nome dell'utente
- IP del client
- IP del server
- Porta del server

Cache locale su file binario
All'avvio dell'applicazione, il sistema carica da file binario le seguenti informazioni, aggiornando di conseguenza la finestra:
- Larghezza del campo
- Altezza del campo
- Contatore dei tick
- Configurazione delle celle (vive/morte)

Viceversa, alla chiusura dell'applicazione, il sistema salva sul file binario i suddetti dati.

Archivio
Quando l'utente preme il tasto 'Archivia', il sistema archivia sul database:
- Nome della configurazione
- Nome dell'utente
- Data e ora del salvataggio
- Larghezza del campo
- Altezza del campo
- Configurazione delle celle (vive/morte)

Quando l'utente preme il tasto 'Recupera', il sistema recupera le suddette informazioni dal database e aggiorna la finestra.

File di log remoto in XML
Il sistema genera una riga di log in occasione dei seguenti eventi:
- Apertura dell'applicazione ("Avvio App")
- Pressione del tasto 'Avvia' ("Avvio Simulazione")
- Pressione del tasto 'Pausa' ("Pausa Simulazione")
- Salvataggio sul database ("Archivio")
- Caricamento dal database ("Recupero")
- Chiusura dell'applicazione ("Chiusura App")

La riga di log contiene il nome dell'evento, IP del client, data e ora dell'evento.

Responsabilità delle classi

Classe **GiocoDellaVita**: è la classe principale dell'applicazione; fornisce l'interfaccia grafica; istanzia le classi deputate all'esecuzione della logica applicativa; al verificarsi di un evento invoca gli opportuni metodi delle classi istanziate.

Classe **Mondo**: rappresenta dal punto di vista logico la griglia di gioco e le relative variabili di stato; restituisce il numero di tick trascorsi, le dimensioni del mondo e lo stato di ogni singola cella; permette di resettare la griglia, modificarne le dimensioni o le celle contenute; implementa i metodi per l'avanzamento dell'evoluzione nel gioco.

Classe **MondoVisuale**: rappresenta graficamente un Mondo; crea, colora e dispone dei quadrati per riprodurre visivamente la griglia di gioco; mostra il numero di tick trascorsi e il totale delle celle vive sullo schermo; aggiorna i suoi contenuti coerentemente con quelli del Mondo ad essa associato; risponde ai click sui quadrati invocando i metodi di Mondo sulla cella corrispondente; consente di cambiare il Mondo da rappresentare.

Classe **Cella**: mantiene e restituisce lo stato individuale di una cella, ossia se attualmente è viva o morta e se ha subito una transizione di stato nel corso dell'ultima iterazione dell'evoluzione; permette di modificare lo stato di vita della cella.

Classe **CacheAssettoMondo**: legge e scrive da/su file binario; memorizza in fase di chiusura e ripristina all'avvio l'assetto complessivo di un mondo, ovvero la disposizione delle celle, le dimensioni della griglia e i tick trascorsi.

Classe **ParametriConfigurazione**: legge, deserializza e valida il file di configurazione XML; fornisce l'accesso ai parametri di configurazione alle altre classi.

Classe **ArchivioModelliSalvati**: interroga il database per il salvataggio e il caricamento dello stato di uno specifico mondo; restituisce una lista inclusiva di nome, autore e data dei modelli salvati nella base di dati.

Classe **ProspettoModelliSalvati**: fornisce una rappresentazione in forma tabellare dei modelli salvati nell'archivio. Permette la selezione manuale di una specifica riga nell'elenco per ripristinare il rispettivo mondo.

Classe **ModelloSalvato**: classe bean per realizzare la tabella di ProspettoModelliSalvati; identifica un modello con un nome unico, memorizzandone anche il nome dell'autore e la data di salvataggio.

Classe **TimerEvoluzioneMondo**: classe thread che coordina lo svolgimento del gioco in modo asincrono rispetto ai comandi dell'interfaccia. Si può avviare o fermare: se avviato, periodicamente determina l'avanzamento nell'evoluzione del mondo di un tick e conseguentemente invoca l'aggiornamento della grafica.

Classe **EventoUtilizzoGUI**: contiene le informazioni di un evento di log; si serializza in XML; permette la validazione di un evento tramite XML Schema.

Classe **LoggerEventiUtilizzoGUI**: crea un evento di log e, dopo averlo serializzato in XML, lo invia al server.

Classe **LogServerEventiUtilizzoGUI**: (server) riceve un evento di log XML; invoca la validazione XML; aggiunge l'evento ricevuto al file di log.

Classe **MonitorAvvisi**: riceve dalle altre classi messaggi e notifiche da mostrare all'utente; visualizza tali comunicazioni all'interno della schermata dell'applicazione.

Classe **NomeModelloNonConsentitoException**: eccezione lanciata quando si tenta di archiviare un modello senza aver specificato un nome valido.

Classe **NomeModelloDuplicatoException**: eccezione lanciata quando si tenta di archiviare un modello con lo stesso nome di uno già presente.

Classe **NessunModelloSelezionatoException**: eccezione lanciata quando si tenta di recuperare un modello senza averlo selezionato nell'apposita tabella.

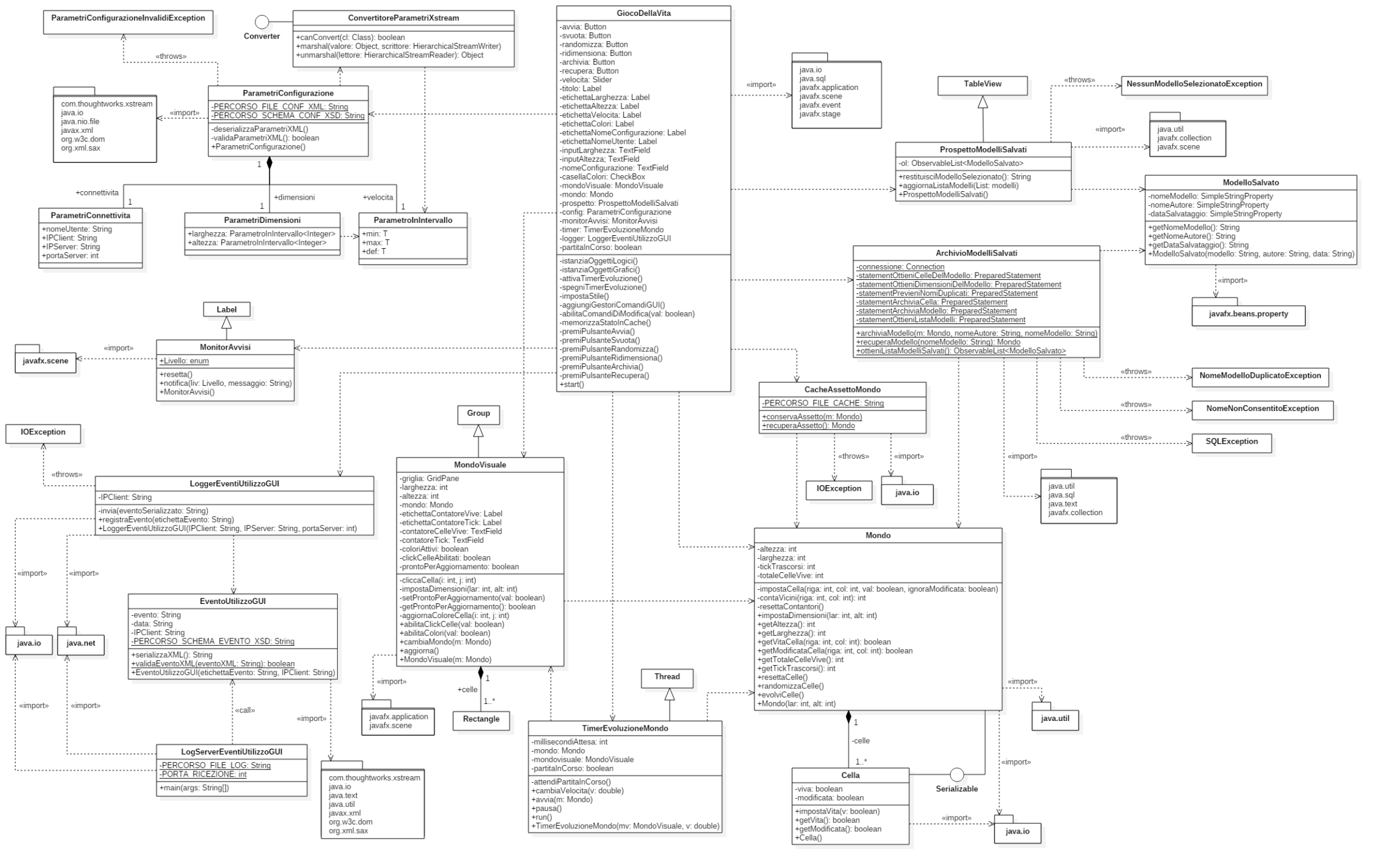
Classe **ParametriConfigurazioneInvalidiException**: eccezione lanciata quando si verifica un problema nel
recupero dei parametri di configurazione dal relativo file.

Classe **ParametriConnettivita**: contiene e rende disponibili i parametri di configurazione relativi alle connessioni client-server e all'identificazione dell'utente.

Classe **ParametriDimensioni**: contiene e rende disponibili i parametri di configurazione relativi alle dimensioni della griglia di gioco.

Classe **ParametroInIntervallo**: tipo di dato astratto per rappresentare una grandezza che ammette un valore minimo, uno massimo e uno di default; rende disponibili questi valori alle altre classi.

Classe **ConvertitoreParametriXStream**: implementa l'interfaccia Converter di XStream; definisce le modalità di serializzazione e deserializzazione XML per la classe ParametroInIntervallo.


ParametriConfigurazioneInvalidiException
«throws»
Converter
ConvertitoreParametriXstream
+canConvert(cl: Class): boolean
+marshal(valore: Object, scrittore: HierarchicalStreamWriter)
+unmarshal(lettore: HierarchicalStreamReader): Object
com.thoughtworks.xstream
java.io
java.nio.file
javax.xml
org.w3c.dom
org.xml.sax
«import»
ParametriConfigurazione
-PERCORSO_FILE_CONF_XML: String
-PERCORSO_SCHEMA_CONF_XSD: String
-deserializzaParametriXML()
-validaParametriXML(): boolean
+ParametriConfigurazione()
1
+connettivita
1
+dimensioni
1
+velocita
1
ParametriConnettivita
+nomeUtente: String
+IPClient: String
+IPServer: String
+portaServer: int
ParametriDimensioni
+larghezza: ParametroInIntervallo<Integer>
+altezza: ParametroInIntervallo<Integer>
ParametroInIntervallo
+min: T
+max: T
+def: T
GiocoDellaVita
-avvia: Button
-svuota: Button
-randomizza: Button
-ridimensiona: Button
-archivia: Button
-recupera: Button
-velocita: Slider
-titolo: Label
-etichettaLarghezza: Label
-etichettaAltezza: Label
-etichettaVelocita: Label
-etichettaColori: Label
-etichettaNomeConfigurazione: Label
-etichettaNomeUtente: Label
-inputLarghezza: TextField
-inputAltezza; TextField
-nomeConfigurazione: TextField
-casellaColori: CheckBox
-mondoVisuale: MondoVisuale
-mondo: Mondo
-prospetto: ProspettoModelliSalvati
-config: ParametriConfigurazione
-monitorAvvisi: MonitorAvvisi
-timer: TimerEvoluzioneMondo
-logger: LoggerEventiUtilizzoGUI
-partitaInCorso: boolean
-istanziaOggettiLogici()
-istanziaOggettiGrafici()
-attivaTimerEvoluzione()
-spegniTimerEvoluzione()
-impostaStile()
-aggiungiGestoriComandiGUI()
-abilitaComandiDiModifica(val: boolean)
-memorizzaStatoInCache()
-premiPulsanteAvvia()
-premiPulsanteSvuota()
-premiPulsanteRandomizza()
-premiPulsanteRidimensiona()
-premiPulsanteArchivia()
-premiPulsanteRecupera()
+start()
«import»
java.io
java.sql
javafx.application
javafx.scene
javafx.event
javafx.stage
TableView
«throws»
NessunModelloSelezionatoException
ProspettoModelliSalvati
-ol: ObservableList<ModelloSalvato>
+restituisciModelloSelezionato(): String
+aggiornaListaModelli(List: modelli)
+ProspettoModelliSalvati()
«import»
java.util
javafx.collection
javafx.scene
ModelloSalvato
-nomeModello: SimpleStringProperty
-nomeAutore: SimpleStringProperty
-dataSalvataggio: SimpleStringProperty
+getNomeModello(): String
+getNomeAutore(): String
+getDataSalvataggio(): String
+ModelloSalvato(modello: String, autore: String, data: String)
«import»
javafx.beans.property
ArchivioModelliSalvati
-connessione: Connection
-statementOttieniCelleDelModello: PreparedStatement
-statementOttieniDimensioniDelModello: PreparedStatement
-statementPrevieniNomiDuplicati: PreparedStatement
-statementArchiviaCella: PreparedStatement
-statementArchiviaModello: PreparedStatement
-statementOttieniListaModelli: PreparedStatement
+archiviaModello(m: Mondo, nomeAutore: String, nomeModello: String)
+recuperaModello(nomeModello: String): Mondo
+ottieniListaModelliSalvati(): ObservableList<ModelloSalvato>
«throws»
NomeModelloDuplicatoException
«throws»
NomeNonConsentitoException
«throws»
SQLException
«import»
java.util
java.sql
java.text
javafx.collection
Label
MonitorAvvisi
+Livello: enum
+resetta()
+notifica(liv: Livello, messaggio: String)
+MonitorAvvisi()
«import»
javafx.scene
CacheAssettoMondo
-PERCORSO_FILE_CACHE: String
+conservaAssetto(m: Mondo)
+recuperaAssetto(): Mondo
«throws»
IOException
«import»
java.io
Group
MondoVisuale
-griglia: GridPane
-larghezza: int
-altezza: int
-mondo: Mondo
-etichettaContatoreVive: Label
-etichettaContatoreTick: Label
-contatoreCelleVive: TextField
-contatoreTick: TextField
-coloriAttivi: boolean
-clickCelleAbilitati: boolean
-prontoPerAggiornamento: boolean
-cliccaCella(i: int, j: int)
-impostaDimensioni(lar: int, alt: int)
-setProntoPerAggiornamento(val: boolean)
-getProntoPerAggiornamento(): boolean
-aggiornaColoreCella(i: int, j: int)
+abilitaClickCelle(val: boolean)
+abilitaColori(val: boolean)
+cambiaMondo(m: Mondo)
+aggiorna()
+MondoVisuale(m: Mondo)
«import»
javafx.application
javafx.scene
+celle
1
1..*
Rectangle
Thread
Mondo
-altezza: int
-larghezza: int
-tickTrascorsi: int
-totaleCelleVive: int
-impostaCella(riga: int, col: int, val: boolean, ignoraModificata: boolean)
-contaVicini(riga: int, col: int): int
-resettaContantori()
+impostaDimensioni(lar: int, alt: int)
+getAltezza(): int
+getLarghezza(): int
+getVitaCella(riga: int, col: int): boolean
+getModificataCella(riga: int, col: int): boolean
+getTotaleCelleVive(): int
+getTickTrascorsi(): int
+resettaCelle()
+randomizzaCelle()
+evolviCelle()
+Mondo(lar: int, alt: int)
1
-celle
1..*
«import»
java.util
IOException
«throws»
LoggerEventiUtilizzoGUI
-IPClient: String
-invia(eventoSerializzato: String)
+registraEvento(etichettaEvento: String)
+LoggerEventiUtilizzoGUI(IPClient: String, IPServer: String, portaServer: int)
«import»
«import»
java.io
java.net
EventoUtilizzoGUI
-evento: String
-data: String
-IPClient: String
-PERCORSO_SCHEMA_EVENTO_XSD: String
+serializzaXML(): String
+validaEventoXML(eventoXML: String): boolean
+EventoUtilizzoGUI(etichettaEvento: String, IPClient: String)
«import»
«import»
«call»
«import»
LogServerEventiUtilizzoGUI
-PERCORSO_FILE_LOG: String
-PORTA_RICEZIONE: int
+main(args: String[])
com.thoughtworks.xstream
java.io
java.text
java.util
javax.xml
org.w3c.dom
org.xml.sax
TimerEvoluzioneMondo
-millisecondiAttesa: int
-mondo: Mondo
-mondovisuale: MondoVisuale
-partitaInCorso: boolean
-attendiPartitaInCorso()
+cambiaVelocita(v: double)
+avvia(m: Mondo)
+pausa()
+run()
+TimerEvoluzioneMondo(mv: MondoVisuale, v: double)
Cella
-viva: boolean
-modificata: boolean
+impostaVita(v: boolean)
+getVita(): boolean
+getModificata(): boolean
+Cella()
Serializable
«import»
java.io

Documento di collaudo

1. All'avvio viene letto il file di configurazione *config/configurazioneXML.xml* (Fig.2), dal quale l'applicazione recupera alcuni parametri per inizializzare la GUI e altri per connettersi al server di log. Stabilita la connessione, viene generata l'interfaccia grafica.
   Se è la prima volta che si usa l'applicazione, questa appare come in (Fig.1a): il campo di gioco risulta vuoto e le sue dimensioni sono quelle di default, specificate nel file di configurazione.
   Altrimenti, la configurazione è esattamente quella presente quando è stata chiusa l'applicazione per l'ultima volta e memorizzata in *cache/mondo.bin* (Fig.1b).
   In entrambi i casi, il nome utente e lo slider della velocità vengono inizializzati in base al file di configurazione.
   Le celle del mondo vive appaiono di colore nero, quelle morte di colore bianco.
   La tabella in basso mostra le configurazioni presenti in archivio, cioè che sono state precedentemente salvate dai vari utenti.
   Infine una riga di log viene aggiunta al file *logs/log_eventi_GUI.xml* (Fig.3).

Fig. 1a

Fig. 1b

Fig. 2

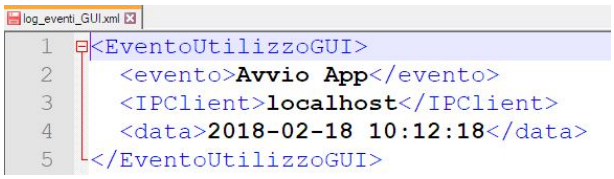


Fig. 3

2. Quando il gioco è fermo, l'utente può modificare le dimensioni del mondo, specificandone i valori negli appositi campi (Fig.4a) e poi cliccando sul pulsante Ridimensiona (Fig.4b): il mondo assume un aspetto come quello in (Fig.4c).
   Può inoltre regolare la velocità di evoluzione del gioco muovendo lo slider (Fig.5) e scegliere se evidenziare (Fig.6a) o meno (Fig.6b) con dei colori le celle che subiscono una transizione di stato (da vive diventano morte o viceversa).
   Infine, ha la possibilità di intervenire manualmente sulla configurazione attuale del mondo, cliccando sulle singole celle che vuole invertire o, più rapidamente, rendendola casuale tramite il tasto Randomizza (Fig.7). È presente anche un pulsante Svuota (Fig.8) per pulire completamente il campo di gioco (tutte le celle presenti diventano morte, come in (Fig.4c)).

Fig.4a

Fig.4b

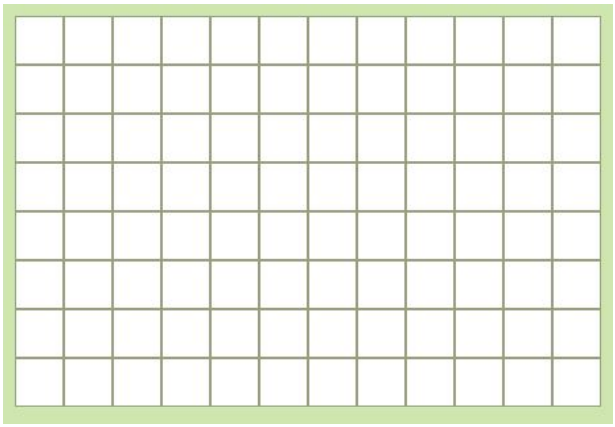


Fig.4c

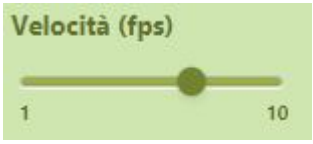


Fig.5

Fig.6a

Fig.6b

Fig.7a

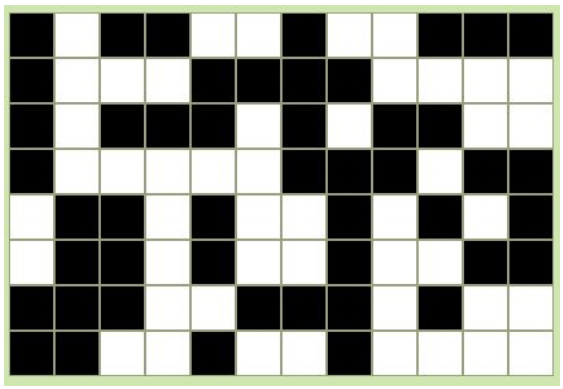


Fig.7b

Fig.8

3. Non appena l'utente è pronto per iniziare, preme il tasto Avvia (Fig.9): il mondo comincia la sua evoluzione e l'interfaccia assume un aspetto simile a quello in (Fig.10), con il tasto Avvia che si trasforma in Pausa (Fig.11).
   Una riga di log viene aggiunta al relativo file (Fig.12).

Fig.9

Fig.11

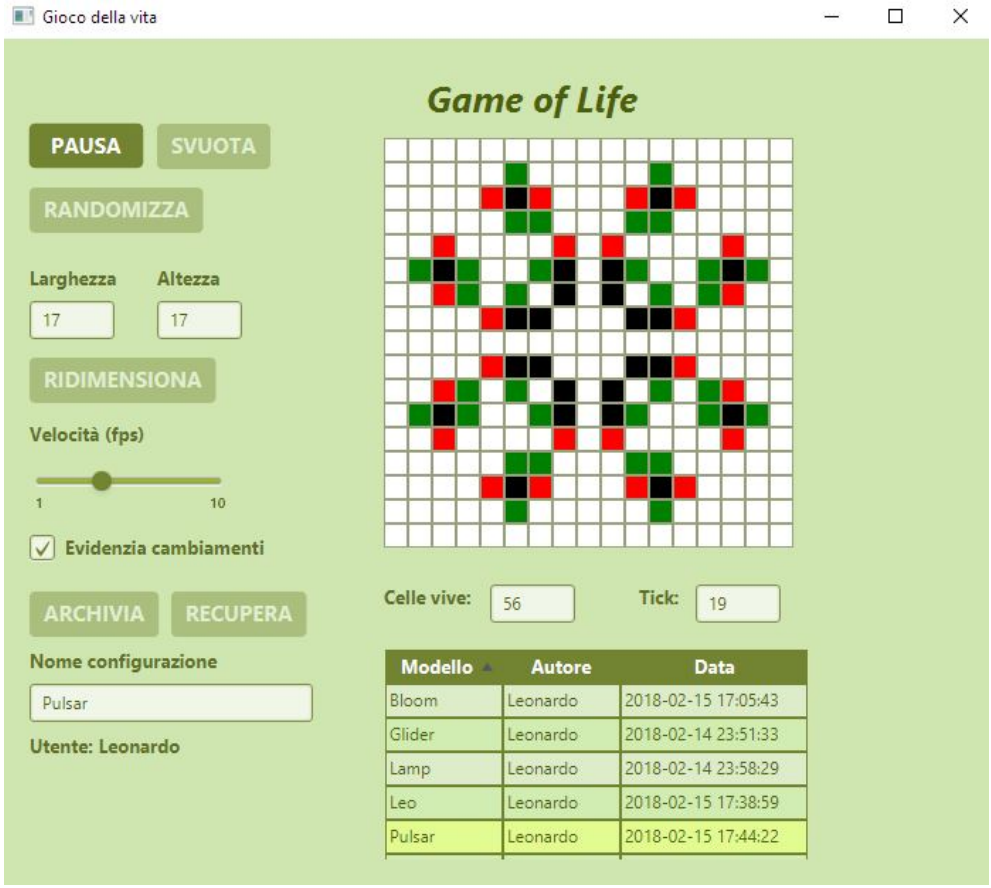


Fig.10

```
<EventoUtilizzoGUI>
  <evento>Avvio Simulazione</evento>
  <IPClient>localhost</IPClient>
  <data>2018-02-18 10:12:27</data>
</EventoUtilizzoGUI>
```

Fig.12

4. Come si vede in (Fig.10), mentre il gioco è in corso tutti i pulsanti risultano disabilitati, ad eccezione di Pausa.
   Rimane ancora possibile modificare dinamicamente la velocità e l'evidenziazione dei cambiamenti tramite colori, mentre l'evoluzione prosegue normalmente.
   Ad intervalli regolari (la cui durata dipende della velocità) le celle del mondo si aggiornano simultaneamente, in accordo alle regole del Gioco della Vita:
   - Una cella viva con due/tre celle adiacenti vive resta viva.
   - Una cella viva con meno di due adiacenti vive muore.
   - Una cella viva con più di tre adiacenti vive muore.
   - Una cella morta con esattamente tre celle adiacenti vive diventa anch'essa viva.

   Un esempio di iterazione è mostrato in (Fig.13), dove si può osservare uno stato (Fig.13a) e il successivo sia evidenziando i colori (Fig.13b) che non (Fig.13c).
   Ad ogni iterazione il contatore dei tick viene incrementato di uno, mentre quello delle celle

vive si aggiorna di conseguenza (Fig.14).

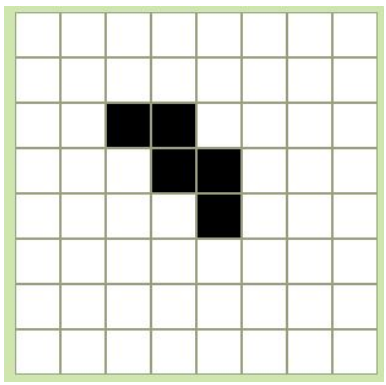


Fig.13a

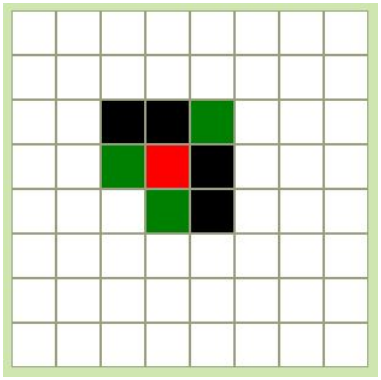


Fig.13b

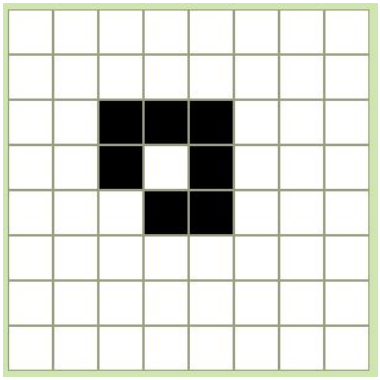


Fig.13c

Fig.14

5. Non appena l'utente preme il tasto Pausa (Fig.11) l'evoluzione si arresta: le celle smettono di aggiornarsi ed i pulsanti ritornano attivi come all'inizio (Fig.1).
   Una riga di log viene aggiunta al relativo file (Fig.15).

```
<EventoUtilizzoGUI>
  <evento>Pausa Simulazione</evento>
  <IPClient>localhost</IPClient>
  <data>2018-02-18 10:12:41</data>
</EventoUtilizzoGUI>
```

Fig.15

   Mentre il gioco è fermo, l'utente può anche salvare la configurazione attuale delle celle in archivio oppure caricarne una fra quelle mostrate nella tabella.

6. Per salvare una configurazione, l'utente deve prima specificare un nome nell'apposito campo (Fig.16), poi cliccare sul tasto Archivia (Fig.17).
   Se l'operazione ha successo, appare un messaggio di conferma in basso (Fig.19) e la nuova configurazione viene aggiunta nella tabella, altrimenti viene notificato un errore (Fig. 18).
   In (Fig.20) è mostrato lo stato del database in seguito all'operazione di archiviazione.
   Una riga di log viene aggiunta al relativo file (Fig.21).

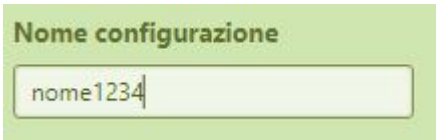


Fig.16

Fig.17

Fig.19

Fig.18

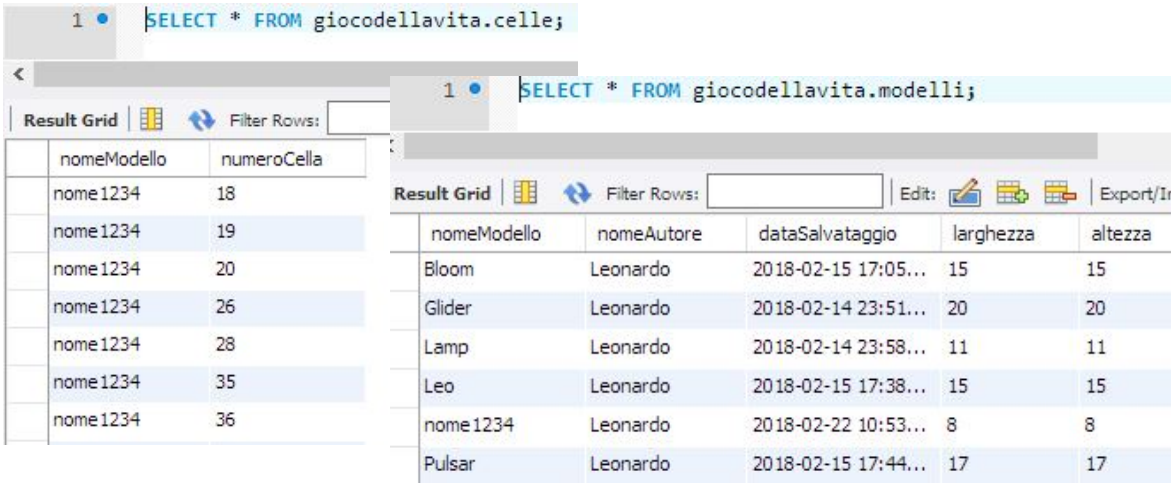


Fig.20

```
<EventoUtilizzoGUI>
  <evento>Archivio</evento>
  <IPClient>localhost</IPClient>
  <data>2018-02-19 19:15:48</data>
</EventoUtilizzoGUI>
```

Fig.21

7. Per caricare una configurazione, l'utente deve semplicemente cliccare sulla corrispondente riga della tabella, che verrà evidenziata (Fig.22), poi premere il tasto Recupera (Fig.23).
   In caso di successo, le celle del mondo assumono la configurazione appena caricata (Fig.24), altrimenti viene generato un messaggio di errore (Fig.25).
   Una riga di log viene aggiunta al relativo file (Fig.26).

Fig.22

Fig.23

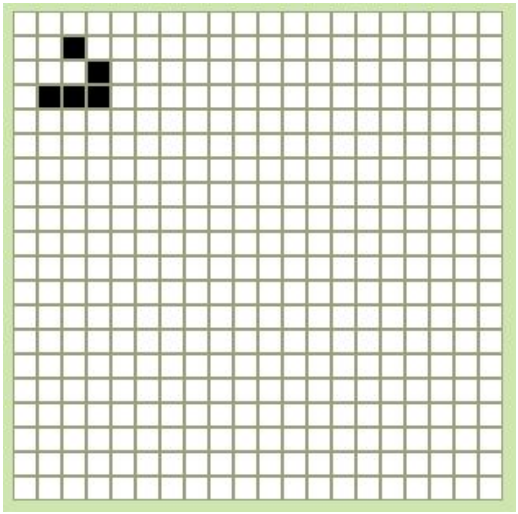


Fig.24

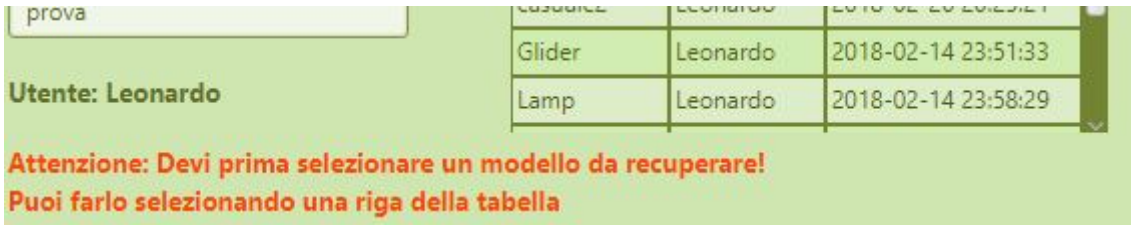


Fig.25

```
<EventoUtilizzoGUI>
  <evento>Recupero</evento>
  <IPClient>localhost</IPClient>
  <data>2018-02-19 19:16:27</data>
</EventoUtilizzoGUI>
```

Fig.26

8. Al momento della chiusura dell'applicazione, lo stato attuale del mondo viene memorizzato su un file binario di cache, così da poterlo ripristinare al prossimo avvio.
   La connessione con il server viene interrotta, ma non prima di aver registrato una riga di log (Fig. 27).

```
<EventoUtilizzoGUI>
  <evento>Chiusura App</evento>
  <IPClient>localhost</IPClient>
  <data>2018-02-19 19:16:44</data>
</EventoUtilizzoGUI>
```

Fig.27